智能工程

课程报告

2212266 张恒硕

南開大學
Nankai University

目 录

返回目录

图 片

表 格

返回目录

1 目的

1. 掌握《智能工程》课程内容，深入理解移动机器人建模方法、控制算法、传感器模型（误差传导模型），点云直线特征提取和配准，以及基于卡尔曼滤波的定位算法。
2. 为下图1和下表1描述的三轮叉车移动机器人建立运动学模型，设计控制算法，求取里程计模型和误差传导模型。（使用符号而非数据求解通用模型）

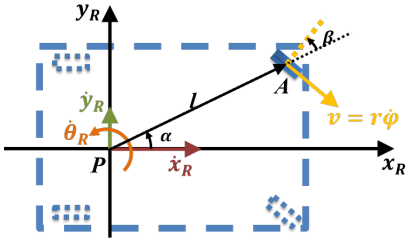


图 1: 三轮叉车移动机器人模型

轮子	类型	驱动方式	α	β	l
A	固定标准轮	随动	$-\pi/2$	π	1
B	固定标准轮	随动	$\pi/2$	0	1
C	转向标准轮	主动	0	$\pi/2$	2

表 1: 三轮叉车移动机器人车轮参数

3. 基于激光雷达的2D点云，采用多种方法提取直线特征，并进行基于ICP的点云配准。
4. 实现基于卡尔曼滤波的机器人定位算法。
5. 使用matlab实现各模型、算法的仿真（附录代码进行了大幅删节，完整请见源码），并对结果进行分析。

2 系统建模

2.1 原理

2.1.1 瞬心（*ICR*）法

刚体上各点角速度相同，如下图2。

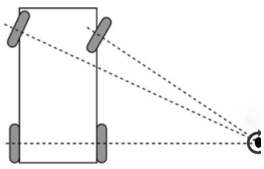


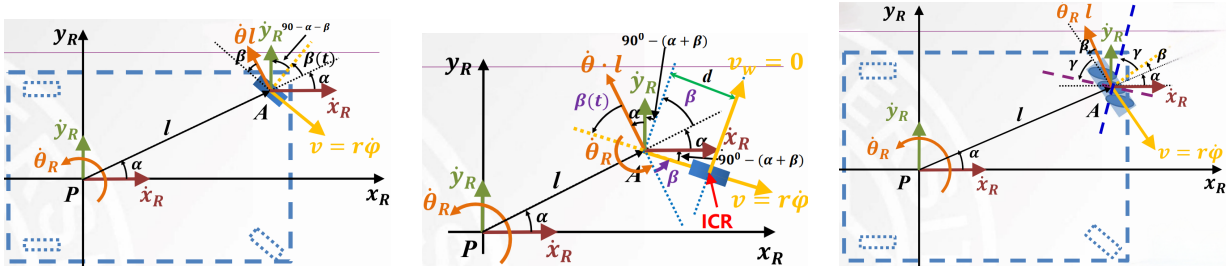
图 2: 瞬心

步骤

1. 坐标系变换。
2. 确定约束。
3. 确定瞬心：各轮轮轴到该点距离与速度成正比。
4. 求解$\dot{\xi}_R = \begin{bmatrix} \dot{x}_R & \dot{y}_R & \dot{\theta}_R \end{bmatrix}^T$。

返回目录

2.1.2 运动约束法



(a) 标准轮　(b) 脚轮　(c) 瑞典轮

图 3: 车轮约束示意图

类型	约束	约束方程	主动轮	随动轮
标准轮	纯滚动	$\begin{bmatrix}\sin(\alpha+\beta(t)) & -\cos(\alpha+\beta(t)) & -l\cos\beta(t)\end{bmatrix} R\theta\dot{\xi}_I = r\dot{\varphi}$	√	x
	无滑动	$\begin{bmatrix}\cos(\alpha+\beta(t)) & \sin(\alpha+\beta(t)) & l\sin\beta(t)\end{bmatrix} R\theta\dot{\xi}_I = 0$	√	√
脚轮	纯滚动	$\begin{bmatrix}\sin(\alpha+\beta) & -\cos(\alpha+\beta) & -l\cos\beta\end{bmatrix} R\theta\dot{\xi}_I = r\dot{\varphi}$	√	x
	无滑动	$\begin{bmatrix}\cos(\alpha+\beta) & \sin(\alpha+\beta) & d+l\sin\beta\end{bmatrix} R\theta\dot{\xi}_I = -d\dot{\beta}$	√	x
瑞典轮	纯滚动	$\begin{bmatrix}\cos(\alpha+\beta+\gamma) & \sin(\alpha+\beta+\gamma) & l\sin(\beta+\gamma)\end{bmatrix} R\theta\dot{\xi}_I = r\dot{\varphi}\sin\gamma + r_{sw}\dot{\varphi}_{sw}$	√	x
	无滑动	$\begin{bmatrix}\sin(\alpha+\beta+\gamma) & -\cos(\alpha+\beta+\gamma) & -l\cos(\beta+\gamma)\end{bmatrix} R\theta\dot{\xi}_I = r\dot{\varphi}\cos\gamma$	√ 小轮	x

表 2: 车轮运动约束方程

参考上图3和上表2，根据各轮主/随动状态列运动约束方程，得到最多三个独立约束方程（对应平面三维位姿）。

2.2 推导

2.2.1 瞬心法

构建如下图4 on the next page的模型，其参数如右侧列表：

返回目录

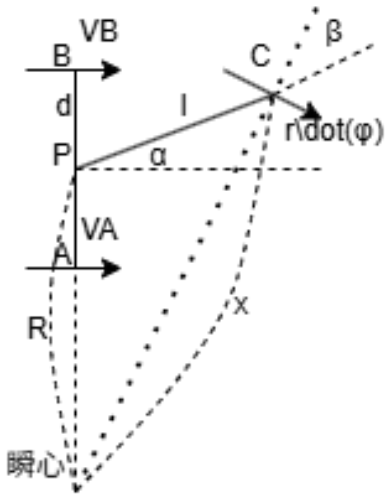


图 4: 瞬心法模型

- l：轮C到P距离。
- d：轮A,B到P距离。
- x：轮C到瞬心距离。
- R：P到瞬心距离。
- r：轮子半径。
- α：轮C偏角。
- β：轮C舵轮角度。
- $\dot{\varphi}$：轮C转速。

在△{瞬心, P, C}中，有正弦定理：

$$\frac{x}{\sin(\alpha+\frac{\pi}{2})}=\frac{x}{\cos\alpha}=\frac{l}{\sin(\frac{\pi}{2}-\alpha-\beta)}=\frac{l}{\cos(\alpha+\beta)}=\frac{R}{\sin\beta}\Rightarrow\begin{cases}x=\frac{\cos\alpha}{\cos(\alpha+\beta)}l\\R=\frac{\sin\beta}{\cos(\alpha+\beta)}l\end{cases}$$

各轮绕瞬心的角速度一致：

$$\dot{\theta}=\frac{V_A}{R-d}=\frac{V_B}{R+d}=\frac{V_C}{x}=\frac{r\dot{\varphi}}{x}$$

代入几何关系：

$$\dot{\theta}=\frac{V_A}{\frac{\sin\beta}{\cos(\alpha+\beta)}l-d}=\frac{V_B}{\frac{\sin\beta}{\cos(\alpha+\beta)}l+d}=\frac{r\dot{\varphi}}{\frac{\cos\alpha}{\cos(\alpha+\beta)}l}=\frac{\cos(\alpha+\beta)}{\cos\alpha l}r\dot{\varphi}$$

有：

$$\dot{x}_R=\frac{1}{2}(V_A+V_B)=\frac{\sin\beta}{\cos(\alpha+\beta)}\dot{\theta}l=\frac{\sin\beta}{\cos\alpha}r\dot{\varphi}$$

2.2.2 运动约束法

构建如下图5 on the following page的模型，其参数如右侧列表：

返回目录

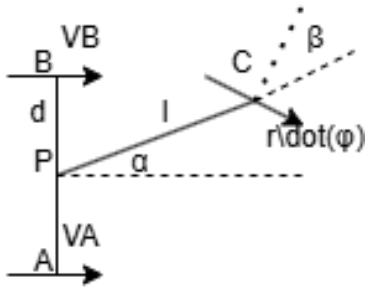


图 5: 运动约束法模型

- l：轮C到P距离。
- d：轮A,B到P距离。
- r：轮子半径。
- α：轮C偏角。
- β：轮C舵轮角度。
- $\dot{\varphi}$：轮C转速。

对轮C：

$$\begin{cases} \begin{bmatrix} \sin(\alpha+\beta) & -\cos(\alpha+\beta) & -l\cos(\beta) \end{bmatrix} \dot{\xi}_R = r\dot{\varphi} \\ \begin{bmatrix} \cos(\alpha+\beta) & \sin(\alpha+\beta) & l\sin(\beta) \end{bmatrix} \dot{\xi}_R = 0 \end{cases} \Rightarrow \begin{cases} \sin(\alpha+\beta)\dot{x}_R - \cos(\alpha+\beta)\dot{y}_R - l\cos\beta\dot{\theta}_R = r\dot{\varphi} \\ \cos(\alpha+\beta)\dot{x}_R + \sin(\alpha+\beta)\dot{y}_R + l\sin\beta\dot{\theta}_R = 0 \end{cases}$$

对轮A,B：

$$\begin{cases} \begin{bmatrix} \sin\frac{\pi}{2} & -\cos\frac{\pi}{2} & -d\cos\pi \end{bmatrix} \dot{\xi}_R = V_A \\ \begin{bmatrix} \cos\frac{\pi}{2} & \sin\frac{\pi}{2} & d\sin\pi \end{bmatrix} \dot{\xi}_R = 0 \end{cases} \Rightarrow \begin{cases} \dot{x}_R + d\dot{\theta}_R = V_A \\ \dot{y}_R = 0 \end{cases}$$

$$\begin{cases} \begin{bmatrix} \sin\frac{\pi}{2} & -\cos\frac{\pi}{2} & -d\cos 0 \end{bmatrix} \dot{\xi}_R = V_B \\ \begin{bmatrix} \cos\frac{\pi}{2} & \sin\frac{\pi}{2} & d\sin 0 \end{bmatrix} \dot{\xi}_R = 0 \end{cases} \Rightarrow \begin{cases} \dot{x}_R - d\dot{\theta}_R = V_B \\ \dot{y}_R = 0 \end{cases}$$

向轮C约束式代入无摩擦约束$\dot{y}_R = 0$，有:

$$\begin{cases} \sin(\alpha+\beta)\dot{x}_R - l\cos\beta\dot{\theta}_R = r\dot{\varphi} \quad (1) \\ \cos(\alpha+\beta)\dot{x}_R + l\sin\beta\dot{\theta}_R = 0 \quad (2) \end{cases}$$

$\sin\beta(1) + \cos\beta(2)$，有：

$$\sin\beta\sin(\alpha+\beta)\dot{x}_R - l\sin\beta\cos\beta\dot{\theta}_R + \cos\beta\cos(\alpha+\beta)\dot{x}_R + l\cos\beta\sin\beta\dot{\theta}_R = r\dot{\varphi}\sin\beta$$

$$\Rightarrow \cos\alpha\dot{x}_R = r\dot{\varphi}\sin\beta \Rightarrow \dot{x}_R = \frac{\sin\beta}{\cos\alpha} r\dot{\varphi}$$

代回解得：

$$\dot{\theta}_R = \frac{\cos(\alpha+\beta)}{l\cos\alpha} r\dot{\varphi}$$

返回目录

2.2.3 系统模型

以上两种方法结果一致：

$$\dot{\xi}_R = \begin{bmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta}_R \end{bmatrix} = \frac{r\dot{\varphi}}{\cos\alpha} \begin{bmatrix} \sin\beta \\ 0 \\ \frac{\cos(\alpha+\beta)}{l} \end{bmatrix} \overset{\alpha=0}{\Longrightarrow} \dot{\xi}_R = \begin{bmatrix} \dot{x_R} \\ \dot{y_R} \\ \dot{\theta_R} \end{bmatrix} = r\dot{\varphi} \begin{bmatrix} \sin\beta \\ 0 \\ \frac{\cos\beta}{l} \end{bmatrix}$$

3 控制

3.1 原理

图 6: 运动控制器框图

上图6给出了运动控制的基本框图，根据目标可进行如下划分：

- 定点（镇定）控制（Regulation Control）：以指定姿态到达指定位置。
- 轨迹跟踪控制（Trajectory Tracking Control）：跟随给定轨迹。
- 路径跟踪控制（Path Tracking Control）：跟随给定路线。

3.1.1 定点控制器

控制目标 机器人系下误差$e = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$，设计控制阵$K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \end{bmatrix}$，其中$k_{ij} = k(t,e)$，得到控制输入$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = Ke$，使$\lim_{t\to\infty} e(t) = 0$。

误差信号转换

返回目录

惯性系下，实际状态$q=\begin{bmatrix}x & y & \theta\end{bmatrix}^T$与参考状态$q_r\begin{bmatrix}x_r & y_r & \theta_r\end{bmatrix}^T$之差为开环误差$\tilde{q}=\begin{bmatrix}\tilde{x} & \tilde{y} & \tilde{\theta}\end{bmatrix}^T=\begin{bmatrix}x-x_r & y-y_r & \theta-\theta_r\end{bmatrix}^T$。

1. 转换到机器人系

$$e=\begin{bmatrix}e_1\\e_2\\e_3\end{bmatrix}=R(\theta)^T\tilde{q}=\begin{bmatrix}\cos\theta & \sin\theta & 0\\-\sin\theta & \cos\theta & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}\tilde{x}\\\tilde{y}\\\tilde{\theta}\end{bmatrix}\overset{\text{闭环}}{\Longrightarrow}\dot{e}=\begin{bmatrix}\dot{e}_1\\\dot{e}_2\\\dot{e}_3\end{bmatrix}=R(\theta)\dot{\tilde{q}}+\dot{R}(\theta)\tilde{q}=\begin{bmatrix}v_1+v_2e_2\\-v_2e_1\\v_2\end{bmatrix}$$

2. 转换到极坐标系

$$\begin{cases}\rho &= \sqrt{\tilde{x}^2+\tilde{y}^2}\\\beta &= -\arctan 2(-\tilde{y},-\tilde{x})\\\alpha &= -\beta-\tilde{\theta}\end{cases}\overset{\text{闭环}}{\Longrightarrow}\begin{bmatrix}\dot{\rho}\\\dot{\beta}\\\dot{\alpha}\end{bmatrix}=\begin{bmatrix}-\cos\alpha & 0\\\frac{\sin\alpha}{\rho} & 0\\\frac{\sin\alpha}{\rho} & -1\end{bmatrix}\begin{bmatrix}v_1\\v_2\end{bmatrix}$$

机器人系非线性控制器

设计控制器$\begin{bmatrix}v_1\\v_2\end{bmatrix}=\begin{bmatrix}-k_1e_1\\-k_2e_2+e_2^2\sin(t)\end{bmatrix}$，代入得$\begin{bmatrix}\dot{e}_1\\\dot{e}_2\\\dot{e}_3\end{bmatrix}=\begin{bmatrix}-k_1e_1+v_2e_2\\-v_2e_1\\-k_2e_2+e_2^2\sin(t)\end{bmatrix}$，其有误差时扰动，效果不佳。

极坐标系线性控制器

设计控制器$\begin{cases}v_1 &= k_\rho\rho\\v_2 &= k_\alpha\alpha+k_\beta\beta\end{cases}$，代入得

$$\begin{bmatrix}\dot{\rho}\\\dot{\beta}\\\dot{\alpha}\end{bmatrix}=\begin{bmatrix}-k_\rho\rho\cos\alpha\\-k_\rho\sin\alpha\\k_\rho\sin\alpha-k_\alpha\alpha-k_\beta\beta\end{bmatrix}\overset{\alpha\to0}{\Longrightarrow}\begin{bmatrix}-k_\rho\rho\\-k_\rho\alpha\\k_\rho\alpha-k_\alpha\alpha-k_\beta\beta\end{bmatrix}$$

其中前两行非线性耦合，在$\alpha\to0$时指数性稳定，非全局稳定。

极坐标系线性控制器

返回目录

设计控制器$\begin{cases} v_1 &= k_\rho \rho \cos\alpha \\ v_2 &= k_\alpha \alpha + \frac{k_\rho \sin\alpha\cos\alpha}{\alpha}(\alpha - k_\beta \beta) \end{cases}$，代入得

$$\begin{bmatrix} \dot{\rho} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} -k_\rho \rho \cos^2\alpha \\ -k_\rho \cos\alpha \sin\alpha \\ k_\rho \cos\alpha \sin\alpha - k_\alpha \alpha - \frac{k_\rho \sin\alpha\cos\alpha}{\alpha}(\alpha - k_\beta \beta) \end{bmatrix} \overset{\alpha\to 0}{\Longrightarrow} \begin{bmatrix} -k_\rho \rho \\ -k_\rho \alpha \\ -k_\alpha \alpha + k_\rho k_\beta \beta \end{bmatrix}$$

其全局渐近稳定。

3.1.2 轨迹跟踪控制器

控制目标与误差变换

惯性系下，实际轨迹$q(t) = \begin{bmatrix} x(t) & y(t) & \theta(t) \end{bmatrix}^T$与参考轨迹$q_r \begin{bmatrix} x_r(t) & y_r(t) & \theta_r(t) \end{bmatrix}^T$之差为开环误差$\tilde{q}(t) = \begin{bmatrix} \tilde{x}(t) & \tilde{y}(t) & \tilde{\theta}(t) \end{bmatrix}^T = \begin{bmatrix} x(t) - x_r(t) & y(t) - y_r(t) & \theta(t) - \theta_r(t) \end{bmatrix}^T$，控制目标为$\lim_{n\to\infty} \tilde{q}(t) = 0$。

辅助误差信号：

$$e = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{\theta} \end{bmatrix} \overset{\text{闭环}}{\Longrightarrow} \dot{e} = \begin{bmatrix} v_1 + v_2 e_2 - v_{1r}\cos e_3 \\ -v_2 e_1 + v_{1r}\sin e_3 \\ v_2 - v_{2r} \end{bmatrix}$$

控制器

设计控制器$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} -k_1 e_1 + v_{1r}\cos e_3 \\ -v_{1r}\frac{\sin e_3}{e_3} e_2 - k_2 e_3 + v_{2r} \end{bmatrix}$，代入得$\begin{bmatrix} \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \end{bmatrix} = \begin{bmatrix} -k_1 e_1 + v_2 e_2 \\ -v_2 e_1 + v_{1r}\sin e_3 \\ -k_2 e_3 - v_{1r}\frac{\sin e_3}{e_3} e_2 \end{bmatrix}$。

$e_3 \to 0$时，控制器简化为$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} -k_1 e_1 + v_{r1} \\ -v_{r1} e_2 + v_{r2} \end{bmatrix}$，代入得$\begin{bmatrix} \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \end{bmatrix} = \begin{bmatrix} -k_1 e_1 + v_2 e_2 \\ -v_2 e_1 \\ -k_2 e_3 - v_{r1} e_2 \end{bmatrix}$。

3.1.3 路径跟踪控制器

控制目标与误差变换

返回目录

惯性系下，实际路径$q(s) = \begin{bmatrix} x(s) & y(s) & \theta(s) \end{bmatrix}^T$与参考路径$q_r \begin{bmatrix} x_r(s) & y_r(s) & \theta_r(s) \end{bmatrix}^T$之差为开环误差$\tilde{q}(s) = \begin{bmatrix} \tilde{x}(s) & \tilde{y}(s) & \tilde{\theta}(s) \end{bmatrix}^T = \begin{bmatrix} x(s)-x_r(s) & y(s)-y_r(s) & \theta(s)-\theta_r(s) \end{bmatrix}^T$，其中$s \in [0,1]$为路径参考变量，控制目标为$\lim_{n\to\infty} \tilde{q}(s) = 0$。

作变换$\begin{cases} y_1 = x + b\cos\theta \\ y_2 = y + b\sin\theta \end{cases}$，得到闭环误差$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = T(\theta) \begin{bmatrix} v \\ \omega \end{bmatrix}$。

逆运算得到$\begin{bmatrix} v \\ \omega \end{bmatrix} = T^{-1}(\theta) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\frac{\sin\theta}{b} & \frac{\cos\theta}{b} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$，故$\begin{cases} \dot{y}_1 = u_1 \\ \dot{y}_2 = u_2 \\ \dot{\theta} = \frac{u_2\cos\theta - u_1\sin\theta}{b} \end{cases}$。

控制器

设计控制器$\begin{cases} u_1 = \dot{y}_{1d} + k_1(y_{1d} - y_1) \\ u_2 = \dot{y}_{2d} + k_2(y_{2d} - y_2) \end{cases}$，有$\begin{cases} \dot{\tilde{y}}_1 = -k_1\tilde{y}_1 \\ \dot{\tilde{y}}_2 = -k_2\tilde{y}_2 \end{cases}$，系统指数性收敛。

3.2 仿真

3.2.1 定点控制器

代码逻辑 见附录代码1。

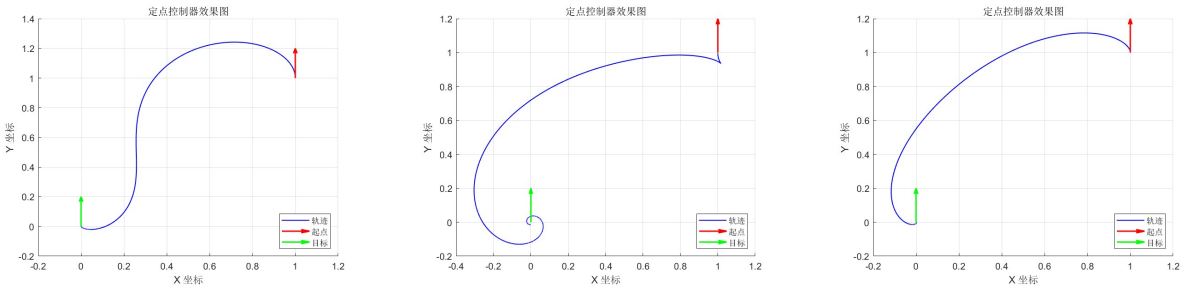
1. 设定参数，并指定惯性坐标系下的目标和起点。
2. 经旋转阵，将相对误差由惯性系转换到目标系，再转换到极坐标系（β是逆时针的）。
3. 设计线性和非线性控制器，二者可以单独使用或同时使用，在角小时可以简化。
4. 将控制器得到的增量从极坐标系经目标系还原到惯性系，更新位姿。
5. 循环3-4步，直到达到指定次数。

仿真结果 使用下表3的参数，得到如下图7 on the next page的仿真结果。

k_ρ	k_α	k_β
3	8	-1.5

表 3: 定点控制器控制参数

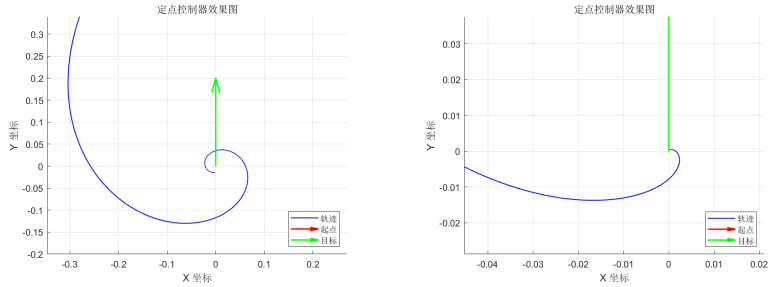
返回目录



(a) 线性　(b) 非线性　(c) 线性+非线性

图 7: 定点控制器仿真结果

分析　三种控制器的轨迹都很平滑，但线性控制器和二者结合的轨迹要比非线性控制器的轨迹更具可实现性。后者由于三角函数项引入了无限循环的螺线，如图8左图。在搭配线性控制器时得到抑制，如8右图，使螺旋的起始半径变得小到可以忽略，使轨迹更具实际意义。



(a) 非线性　(b) 线性+非线性

图 8: 定点控制器螺线抑制

参数作用

- k_ρ：影响系统的收敛速度和稳态误差。
- k_α：影响系统调整方向对准目标的速度。
- k_β：影响系统的俯仰调整速度和路径平滑性。

3.2.2 轨迹跟踪控制器

代码逻辑　见附录代码2。

1. 设定参数，并指定起点。

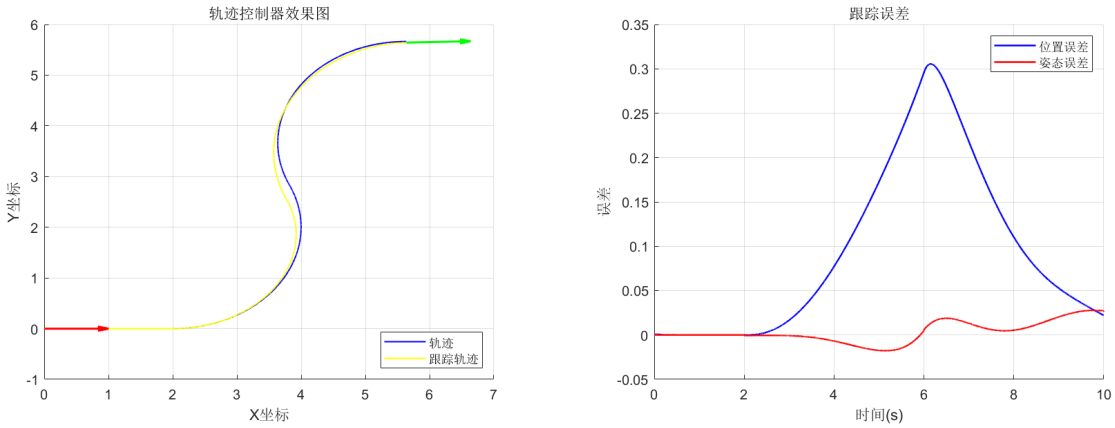
返回目录

2. 由小步长步进，生成惯性系下的平滑期望轨迹。
3. 将误差信号由惯性系转换到机器人系。
4. 设计轨迹控制器，在角小时可以简化。
5. 计算增量，更新位姿。
6. 循环3-5步，直到达到指定次数。

仿真结果 使用下表4的参数，得到如下图9左图的仿真结果。

k_1	k_2
5	1

表 4: 轨迹控制器控制参数



(a) 轨迹 **(b)** 误差

图 9: 轨迹控制器仿真结果

分析 跟踪轨迹和期望轨迹基本重合，机器人实现了平滑的跟踪。然而，如上图9右图，如果期望轨迹剧烈变动（如斜率绝对值较大处），机器人无法快速追踪，会造成相对大的误差，这是因为参数的增益较小。

参数作用

- k_1：控制位置误差的收敛速度，影响轨迹跟踪精度。
- k_2：控制角度误差的收敛速度，影响轨迹跟踪过程中转向的平滑程度。

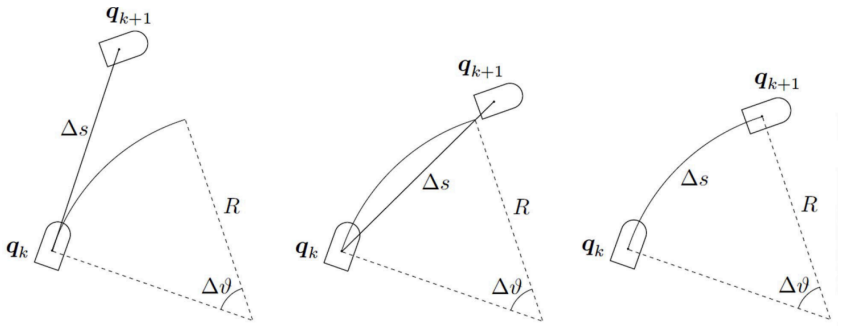
返回目录

4 里程计和误差传导模型

4.1 原理

4.1.1 里程计

计算短时间T_s内位姿变化，增量式更新，根据精度和计算量，有如下图组10的数值积分方法。



(a) 欧拉法　(b) RK2法　(c) 解析积分法

图 10: 数值积分方法

- 欧拉法

$$\begin{cases} x_{k+1} = x_k + v_k T_s \cos\theta_k \\ y_{k+1} = y_k + v_k T_s \sin\theta_k \\ \theta_{k+1} = \theta_k + \omega_k T_s \end{cases}$$

- RK2法（二阶Runge-Kutta法）

$$\begin{cases} x_{k+1} = x_k + v_k T_s \cos(\theta_k + \frac{\omega_k T_s}{2}) \\ y_{k+1} = y_k + v_k T_s \sin(\theta_k + \frac{\omega_k T_s}{2}) \\ \theta_{k+1} = \theta_k + \omega_k T_s \end{cases}$$

- 解析积分法

$$\begin{cases} x_{k+1} = x_k + \frac{v_k}{\omega_k}(\sin\theta_{k+1} - \sin\theta_k) \\ y_{k+1} = y_k - \frac{v_k}{\omega_k}(\cos\theta_{k+1} - \cos\theta_k) \\ \theta_{k+1} = \theta_k + \omega_k T_s \end{cases} \xRightarrow{\omega_k = 0} \begin{cases} x_{k+1} = x_k + v_k T_s \\ y_{k+1} = y_k \end{cases}$$

4.1.2 误差传导

误差来源

- 数值积分误差。

返回目录

- 运动学参数误差：速度不恒定，半径误差。
- 打滑。

误差传导 在运动时，位姿误差增量式更新。其分为位姿变化和控制输入量变化两部分，需对位姿更新公式求关于各变化量的偏导。

$$\sum_{p'} = \nabla_p f \cdot \sum_{p} \cdot \nabla_p f^T + \nabla_{rl} f \cdot \sum_{\Delta} \cdot \nabla_{rl} f^T$$

可视化 将误差传播协方差矩阵转化成椭圆来展示。

取$\sum_p$的左上二阶子阵$\sum_{p_{xy}} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{xy} & \sigma_{yy} \end{bmatrix}$，计算特征值$\lambda_1, \lambda_2$和特征向量$\vec{p}_1, \vec{p}_2$，其分别表示长短轴大小（开方后）和方向。椭圆的圆心是(x_k, y_k)。

4.2 推导

里程计码盘信息

$$\dot{\xi}_R = \begin{bmatrix} \dot{x_R} \\ \dot{y_R} \\ \dot{\theta_R} \end{bmatrix} = r\dot{\varphi} \begin{bmatrix} \sin\beta \\ 0 \\ \frac{\cos\beta}{l} \end{bmatrix} = \begin{bmatrix} v \\ 0 \\ \omega \end{bmatrix} \xrightarrow{\text{转换到惯性系}} \begin{cases} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \end{cases}$$

码盘读数为：

$$\begin{cases} \Delta s = r\Delta\varphi\sin\beta \\ \Delta\theta = \frac{r\Delta\varphi\cos\beta}{l} \end{cases} \xrightarrow{\text{小时间变化}} \begin{cases} \Delta s &= v_k T_s \\ \Delta\theta &= \omega_k T_s \end{cases}$$

误差传导（RK2法）

位姿更新公式为：

$$p' = f(x, y, \theta, \Delta\varphi, \beta) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s\cos(\theta_k + \frac{\Delta\theta}{2}) \\ \Delta s\sin(\theta_k + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} r\Delta\varphi\sin\beta\cos(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) \\ r\Delta\varphi\sin\beta\sin(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) \\ \frac{r\Delta\varphi\cos\beta}{l} \end{bmatrix}$$

返回目录

对位姿x, y, θ求偏导得：

$$\nabla_p f = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial f}{\partial \theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta s \sin(\theta_k + \frac{\Delta\theta}{2}) \\ 0 & 1 & \Delta s \cos(\theta_k + \frac{\Delta\theta}{2}) \\ 0 & 0 & 1 \end{bmatrix}$$

对控制输入量$\Delta\varphi, \beta$求偏导得：

$$\nabla_{rl} f_1 = \frac{\partial f}{\partial \Delta\varphi} = \begin{bmatrix} r\sin\beta\cos(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) - \frac{r^2\Delta\varphi\cos\beta\sin\beta}{2l}\sin(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) \\ r\sin\beta\sin(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) + \frac{r^2\Delta\varphi\cos\beta\sin\beta}{2l}\cos(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) \\ \frac{r\cos\beta}{l} \end{bmatrix}$$

$$\nabla_{rl} f_2 = \frac{\partial f}{\partial \beta} = \begin{bmatrix} r\Delta\varphi\cos\beta\cos(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) + \frac{r^2\Delta\varphi^2\sin^2\beta}{2l}\sin(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) \\ r\Delta\varphi\cos\beta\sin(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) - \frac{r^2\Delta\varphi^2\sin^2\beta}{2l}\cos(\theta_k + \frac{r\Delta\varphi\cos\beta}{2l}) \\ \frac{-r\Delta\varphi\sin\beta}{l} \end{bmatrix}$$

控制输入量的协方差矩阵为：

$$\sum_{\Delta} = Covar(\Delta\varphi, \beta) = \begin{bmatrix} k_\varphi\|\Delta\varphi\| & 0 \\ 0 & k_\beta \end{bmatrix}$$

按位姿误差更新公式进行迭代。

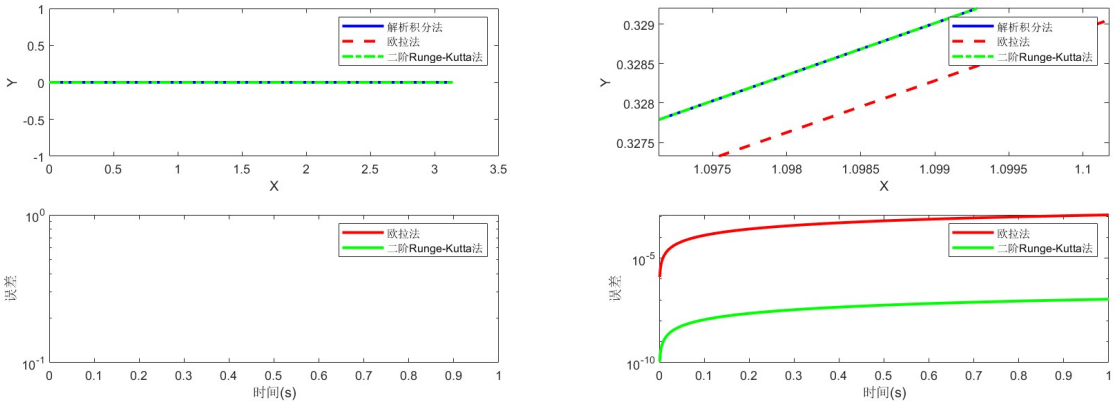
4.3 仿真

代码逻辑 见附录代码3。

1. 设定参数（$\beta = \frac{\pi}{2} + k\pi$时直线前进），并指定起点。
2. 采用解析积分法进行求解，在角速度为0时可以化简。
3. 采用欧拉法进行求解。
4. 采用RK2法进行求解，并计算误差传导。

里程计模型仿真结果 以下图组11 on the following page分别给出了直线前进和曲线前进的仿真结果。

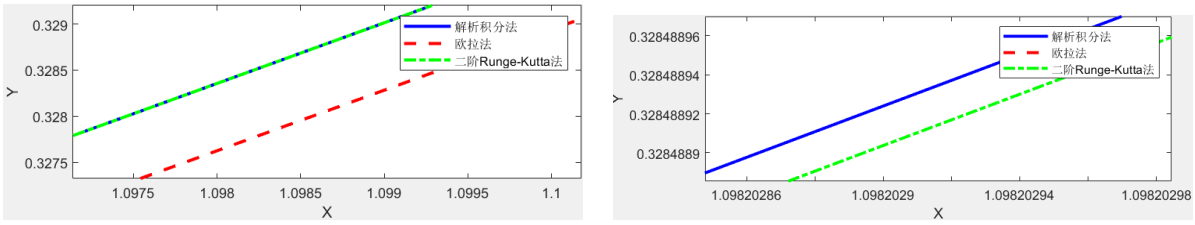
返回目录



(a) 直线前进　　(b) 曲线前进

图 11: 里程计模型

- 直线前进：两种方法的结果皆与解析积分的结果一致，这是因为更新公式中引起差异的转角部分无效。
- 曲线前进：两种方法的结果与解析积分的结果都有一定差距，如下图组13 on the next page。RK2法明显优于欧拉法，这是因为前者粗略地考虑了角度变化（引入一半的变化角），而后者在每一小段移动中，是固定原始角度的。



(a) 欧拉法误差放大　　(b) RK2法误差放大

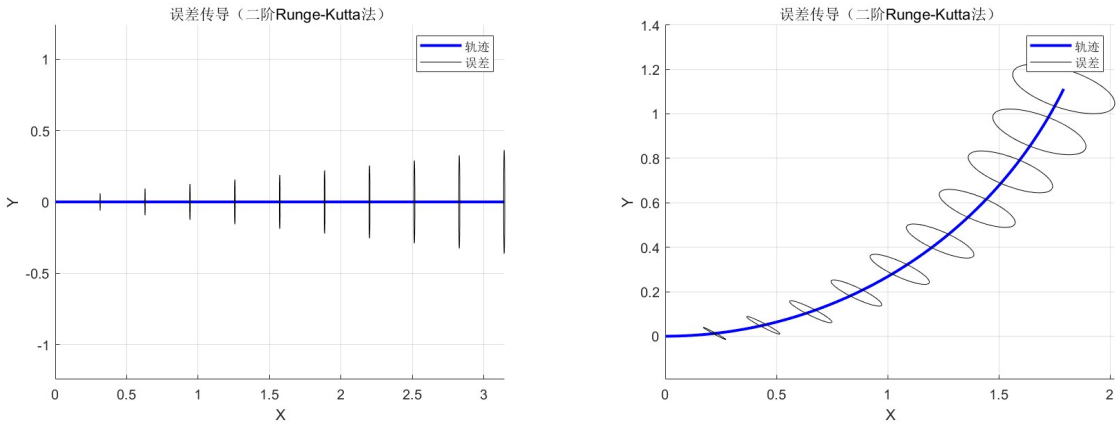
图 12: 里程计误差放大

误差传导仿真结果　使用下表5的参数，得到以下图组13 on the next page的结果。

车轮转速误差系数k_φ	舵机角度误差系数k_β
0.001	0.001

表 5: 里程计模型误差系数

返回目录



(a) 直线前进误差传导　　**(b)** 曲线前进误差传导

图 13: 里程计误差传导

可以发现，在前进过程中，误差都在累积变大。并且，与课程内容相符，直线前进的误差垂直于前进方向，而曲线前进的有一定偏离。

4.4 思考

编写仿真代码时，由于对转化协方差矩阵为椭圆形式的意义不甚了解，一开始没有取得较好的实验结果。以下记录对该部分的研究与实验过程。

协方差矩阵初始化　$\sum_p$的初始值会一定程度上影响结果，尤其是在其与$\sum_\Delta$差距过大时。代码中采用零阵初始化。

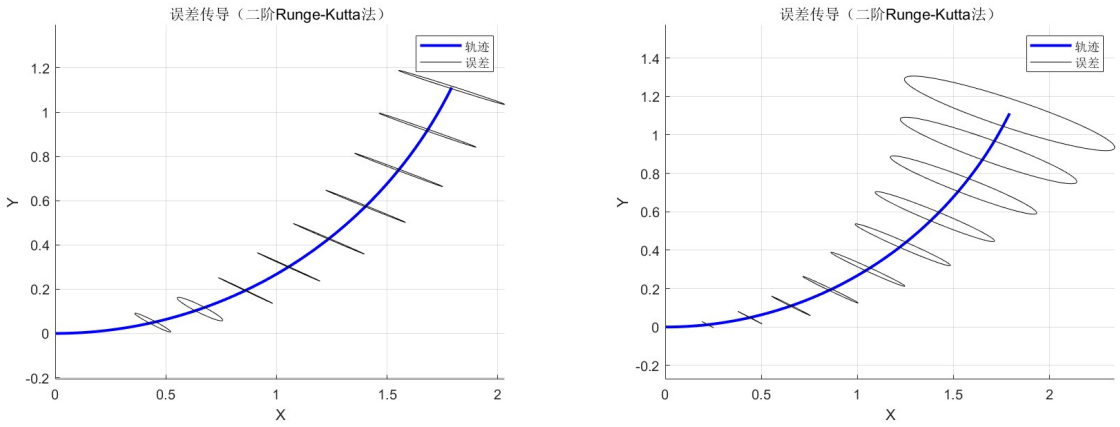
长短轴计算

迭代计算$\sum_p$得到如下三阶协方差矩阵：

$$\sum_p = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{y\theta} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta\theta} \end{bmatrix}$$

直接对其求解特征根和特征向量，再取前两个，结果与取二阶子阵再求解的结果不同，前者结果如下图14 on the following page左图。

返回目录



(a) 三阶矩阵特征　　(b) 不开根号

图 14: 里程计误差传导对比

长短轴比例　利用求得的特征根直接绘制椭圆可能会过大，因此乘以相应系数。然而长短轴的比例可能会不协调，查阅资料，选择对其开根号，减小二者比例，使椭圆不过分扁平。不开根号的结果如上图14右图。

椭圆意义

- 长短轴表示误差在不同方向上的大小。
- 方向角表示误差传播的主要方向，体现了各向异性。
- 椭圆大小反映了系统的误差范围，可按置信度缩放。

5 点云直线提取

5.1 原理

5.1.1 最小二乘法（*Least Squares Method*）

在求解拟合直线时，最小化拟合误差平方和，目标式为：

$$\min \sum_{i=1}^{n} d_i^2 = \min \sum_{i=1}^{n} [y_i - f(x_i)]^2$$

其中$f(x) = ax + b$是拟合直线，(x, y)是待拟合的点坐标。

返回目录

求解方法

1. 求偏导：求目标式关于a, b的偏导，得到如下极值条件：

$$\begin{cases} \frac{\partial \Pi}{\partial a} = 2\sum_{i=1}^{n}[y_i - (a + bx_i)] = 0 \\ \frac{\partial \Pi}{\partial b} = 2\sum_{i=1}^{n} x_i[y_i - (a + bx_i)] = 0 \end{cases}$$

2. 矩阵形式：将拟合直线$f(x) = ax + b$增广为矩阵形式$Y = X\beta$，在误差$d = Y - X\beta$趋于0时，有$Y = X\beta$，因此：

$$Y = X\beta \Rightarrow X^T Y = X^T X\beta \Rightarrow \beta = (X^T X)^{-1} X^T Y$$

5.1.2 *Split-and-Merge*（分割与合并）

1. 设定参数。
2. 分裂（Split）：将图像划分为小的子区域，直到所有子区域都满足特定的同质性条件。
   - 以全集作为初始点集。
   - 对当前点集拟合直线（采用端点拟合），计算到最远点的距离。
   - 距离大于阈值则在该最远点处将点集分裂为两个子集。
   - 为防止过度分裂，当点集涵盖的点数足够少时，不再进行分裂。
3. 合并（Merge）：基于相邻区域间的相似度进行合并，持续到没有更多的相邻区域可以合并为止。
   - 检查相邻线段是否满足合并要求（合并后是否有过远点）。
   - 若满足要求，则合并并拟合新的直线。
   - 最后一个线段直接保存。
4. 为减少外点参与的拟合线段，可设置线段点数阈值。

5.1.3 *Line-Regression*（线性回归）

1. 设定参数，选取初始窗口。
2. 对窗口内的点拟合直线（采用最小二乘法），计算其参数。
3. 滑动窗口拟合新的直线。
4. 检查相邻线段是否满足合并的角度和距离要求，满足则合并并拟合新的直线。
5. 直到所有线段不可再合并。

返回目录

5.1.4 RANSAC（Random Sample Consensus，随机抽样一致性算法）

- 外点（outliers）：异常值。
- 内点（inliers）：符合模型的数据点。

1. 设定参数。
2. 根据内点比例w和找到一个完全由内点组成的样本的希望概率p计算迭代次数：

$$k = \frac{\log(1-p)}{\log(1-w^2)}$$

3. 从所有数据点中随机选择最小数量（直线2点，平面3点）的数据点子集。为减小扰动，需保证选取点样本间有足够的距离。
4. 用数据点子集确定唯一的模型参数。
5. 计算剩余数据点与该模型的误差，小于设定阈值的为内点。
6. 重复迭代次数次采样，一般需避免重复选取点样本（代码未实现）。
7. 选取包含最多内点的模型。

5.1.5 Hough-Transform

- 点到线转换：图像空间中的一个点对应Hough空间中的一条线。激光定位任务中，常用极坐标$\rho = x\cos\theta + y\sin\theta$表示。在定距下，误差呈正态分布；而在变距下，误差增长与距离正相关。
- 投票机制：每个边缘点为符合的Hough空间参数投票，多点共线会产生累积高峰。

1. 设定参数，计算数据范围。
2. 初始化累加器，用于记录不同参数组合的支持程度。
3. 遍历边缘点，计算可能的参数组合，并在对应位置进行投票。
4. 在累加器中寻找峰值（可能不唯一），获得相应Hough空间参数。
5. 转换回图像空间，确定直线。

返回目录

5.1.6 性能对比

各算法的性能对比如下表6。

算法	复杂度	速度(Hz)	假阳性率FPR(%)	精度	多直线检测	点云属性
Split-and-Merge	$N \log N$	1500	10	+++	适用	
Line-Regression	NN_f	400	10	+++	适用	有序
RANSAC	SNk	30	30	++++	需调整	大量外点
Hough-Transform	$SNN_C + SN_RN_C$	10	30	++++	适用	

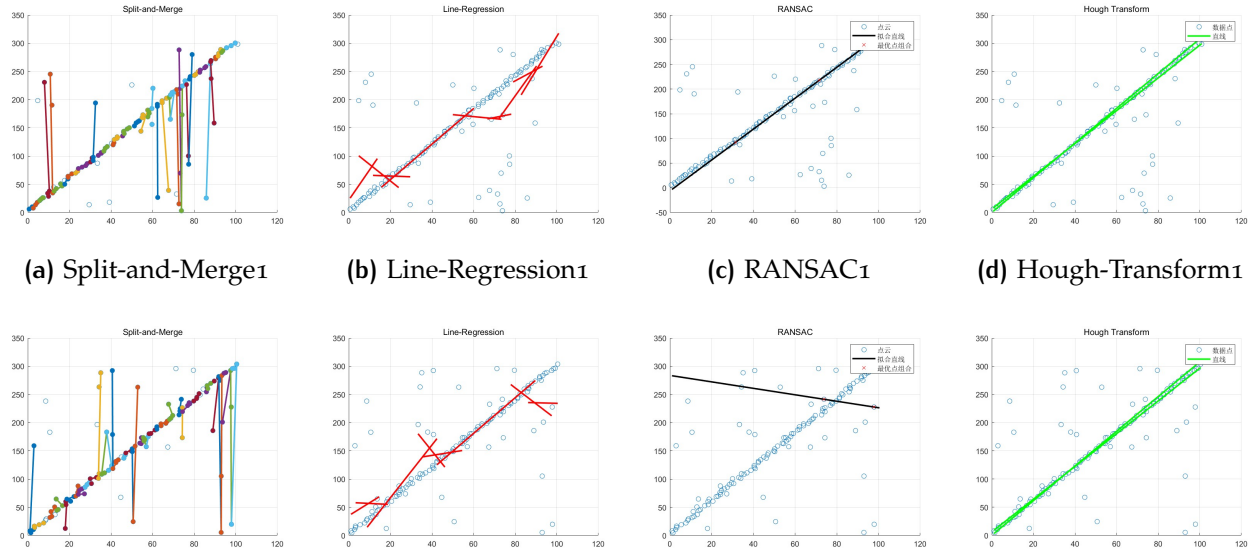
表 6: 直线特征提取算法性能对比

5.2 仿真

代码逻辑 见附录代码4。

1. 生成带噪音的目标直线，生成一定比例的外点，将所有点排序。
2. 执行各算法，记录时间，并输出可视化结果。

仿真结果 以下图组15展示了两组拟合结果，其中序号一致的是同一数据的检测结果。



(a) Split-and-Merge1 (b) Line-Regression1 (c) RANSAC1 (d) Hough-Transform1

(e) Split-and-Merge2 (f) Line-Regression2 (g) RANSAC2 (h) Hough-Transform2

图 15: 拟合结果

返回目录

耗时分析 各算法的耗时情况如下表7。由于不同算法用的函数的效率存在差异，可能无法与以上理论耗时相对应。仅就数据来看，Hough-Transform最慢，这可能是因为其在投票时要进行大量拟合。

算法	Split-and-Merge	Line-Regression	RANSAC	Hough-Transform
耗时(s)	0.0031	0.0033	0.0015	0.0274

表 7: 算法耗时对比

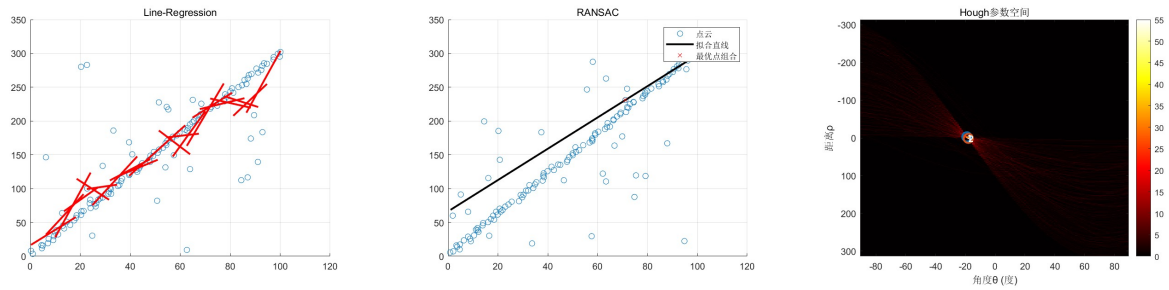
性能分析

1. Split-and-Merge：小线段拟合较好，但是无法连成长直线，也会存在外点线段。前者并无大碍，后者在内点率提高后可以得到改善。

2. Line-Regression：在窗口较小时，可以得到较好的小线段拟合结果，但是会有很多毛刺，如下图16左图。在增大窗口并放宽合并标准后，得到如上结果，仍是不十分理想。

3. RANSAC：正确结果精度最高，但是有较大概率无法获得正确结果。这是因为设置的外点比例较高，达到了$\frac{3}{13}$，在计算得到的6次迭代中，仍有一定概率无法选取全内点组合，如上图组15 on the preceding page中的图g。另外，临近的内点组合可能偏离很大，如下图16中图，但这一情况已通过限制取点距离得到极大改善。

4. Hough-Transform：结果最稳定而准确，但是一般得到临近的一簇直线而非一条直线，在Hough空间中显示为两个临近的点，如下图16右图。这一问题可以通过提高阈值比例或限制输出线长来改善。



(a) Line-Regression错误样例 **(b)** RANSAC错误样例 **(c)** Hough空间可视化

图 16: 拟合结果补充

返回目录

5.3 思考

这四种方法可以概括性地划分为三个思路，一是目标函数最小化，以最小二乘法的应用为主；二是积少成多，以合并小线段为主要方式，投票的形式也可以类比为这样；三是采样，以RANSAC为例。

6 点云配准

6.1 原理

6.1.1 奇异值分解（*Singular Value Decomposition*，*SVD*）

根据$A_{m\times n}$计算$(A^TA)_{n\times n}$ 和$(AA^T)_{m\times m}$两个对称矩阵，求二者的特征值和特征向量：

$$A^TAv_i = \lambda_i v_i, i = 1,2,\dots,n \quad AA^Tu_i = \mu_i u_i, i = 1,2,\dots,m$$

其中，λ_i, μ_i是特征值，v_i, u_i是其对应的特征向量，分别组成$V = [v_1, v_2, \dots, v_n]$和$U = [u_1, u_2, \dots, u_m]$。

奇异值σ_i是λ_i, μ_i的平方根，将其按降序排列，并构造对角矩阵$\Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_r)$，得到SVD分解$A = U\Sigma V^T$。

6.1.2 基于*SVD*的定位算法

条件与目标

在二维平面上，有基于世界坐标系的点云$P = \{p_1, p_2, \dots, p_n\}$和基于激光坐标系的点云$Q = \{q_1, q_2, \dots, q_n\}$，它们按下标顺序匹配。求解它们之间的刚体变换（旋转阵R和平移量t）。

以误差平方加权和的形式建模，得到目标式：

$$(R, t) = \arg\min \sum_{i=1}^{n} w_i \|(Rp_i + t) - q_i\|^2$$

其中w_i表示匹配点对(p_i, q_i)的权重，代码中默认相等，可以取为距离的倒数$w_i = \frac{1}{\sigma_i(\rho_i)}$。

返回目录

加权平均

为求极值，对目标式求关于t的偏导：

$$2t(\sum_{i=1}^{n} w_i) + 2R(\sum_{i=1}^{n} w_i p_i) - 2\sum_{i=1}^{n} w_i q_i = 0 \overset{\text{等号两边同除}\sum_{i=1}^{n} w_i}{\Longrightarrow} t + R\frac{(\sum_{i=1}^{n} w_i p_i)}{\sum_{i=1}^{n} w_i} - \frac{\sum_{i=1}^{n} w_i q_i}{\sum_{i=1}^{n} w_i} = 0$$

取两个点云的加权中心点$\hat{p} = \frac{\sum_{i=1}^{n} w_i p_i}{\sum_{i=1}^{n} w_i}, \hat{q} = \frac{\sum_{i=1}^{n} w_i q_i}{\sum_{i=1}^{n} w_i}$，上式可化简为$t = \hat{q} - R\hat{p}$。
该式描述了平移量和旋转阵的关系，将其带回目标式，得到单变量最值问题：

$$R = \arg\min \sum_{i=1}^{n} w_i \|R(p_i - \hat{p}) - (q_i - \hat{q})\|^2$$

去中心化

取去中心化$x_i = p_i - \hat{p}, y_i = q_i - \hat{q}$，目标式可化简为：

$$R = \arg\min \sum_{i=1}^{n} w_i \|Rx_i - y_i\|^2$$

将平方项变成矩阵相乘的形式：

$$\|Rx_i - y_i\|^2 = (Rx_i - y_i)^T(Rx_i - y_i) = x_i^T(R^TR)x_i - y_i^TRx_i - x_i^TR^Ty_i + y_i^Ty_i$$

- 旋转阵为标准正交阵，$R^{-1} = R^T$，因此$R^TR = E$。
- $x_i^TR^Ty_i$是一个1×1标量，转置后不变，$x_i^TR^Ty_i = y_i^TRx_i$。

所以:

$$\|Rx_i - y_i\|^2 = x_i^Tx_i - 2y_i^TRx_i + y_i^Ty_i$$

其中仅有负号项与R相关，其它项都是定值，目标可改写为：

$$R = \arg\max \sum_{i=1}^{n} w_i y_i^TRx_i$$

SVD分解

返回目录

将其写成对角阵的迹的形式：

$$\begin{aligned}\sum_{i=1}^{n} w_i y_i^T R x_i &= tr(diag(w_1 y_1^T R x_1, w_2 y_2^T R x_2, \dots, w_n y_n^T R x_n)) \\ &= tr(diag(w_1, w_2, \dots, w_n) \begin{bmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_n^T \end{bmatrix} R \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}) \\ &= tr(WY^T RX)\end{aligned}$$

因为矩阵的迹满足$tr(AB) = tr(BA)$，所以$tr(WY^T RX) = tr(RXWY^T)$。令$S = XWY^T$，基于SVD原理，$S = U\Sigma V^T$，其中U, V是单位正交阵，Σ为对角阵。所以$tr(RXWY^T) = tr(RU\Sigma V^T) = tr(\Sigma V^T RU)$，后三者都是单位正交阵，它们的积$M$也是单位正交阵。目标改写为：

$$R = argmaxtr(\Sigma M)$$

求解R和t　由于单位正交阵的最大迹是在单位阵下取得的，最大值条件为$M = E$，即$R = VU^T$。这里要注意保证R是单位正交阵。代回R和t的关系式，可确定t。

6.1.3 基于*ICP*（*Iterative Closest Point*，迭代最近点）的点云匹配算法

1. 条件与目标：二维平面上有检测点云$P = \{p_1, p_2, \dots, p_{N_p}\}$和目标点云$X = \{x_1, x_2, \dots, x_{N_x}\}$，求解它们的匹配。
2. 计算最近点集：采取采样方法（均匀采样、随机采样、基于特征的采样、法向量空间采样等）获得目标点云，并为检测点云数据点匹配最近的目标点云数据点。
3. 变换：应用基于SVD的定位算法求解齐次变换矩阵并应用于检测点云。
4. 目标函数计算：统计对齐误差，如果达到阈值则停止迭代，否则重复上述操作。

6.1.4 评价方法

- 平移量：向量，使用欧几里得范数。
- 旋转阵：矩阵，使用Frobenius范数（所有元素的平方和的平方根）。

返回目录

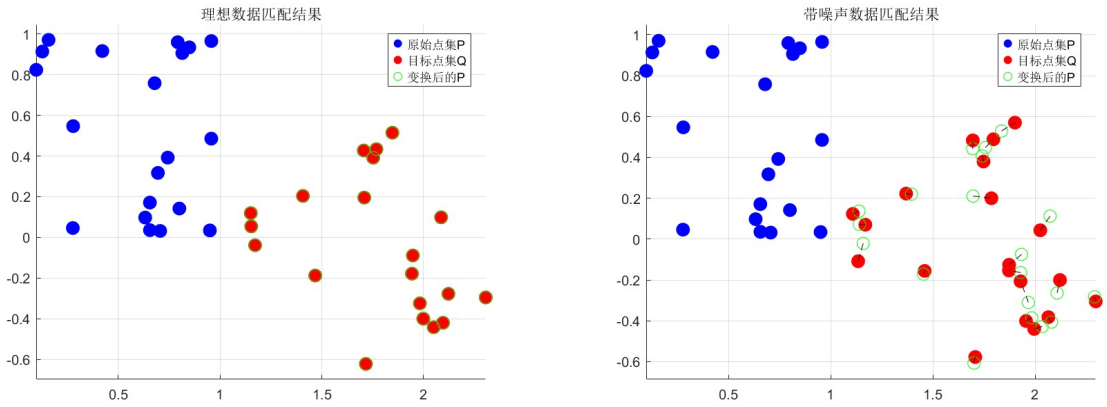
6.2 仿真

6.2.1 基于SVD的定位算法

代码逻辑 见附录代码5。

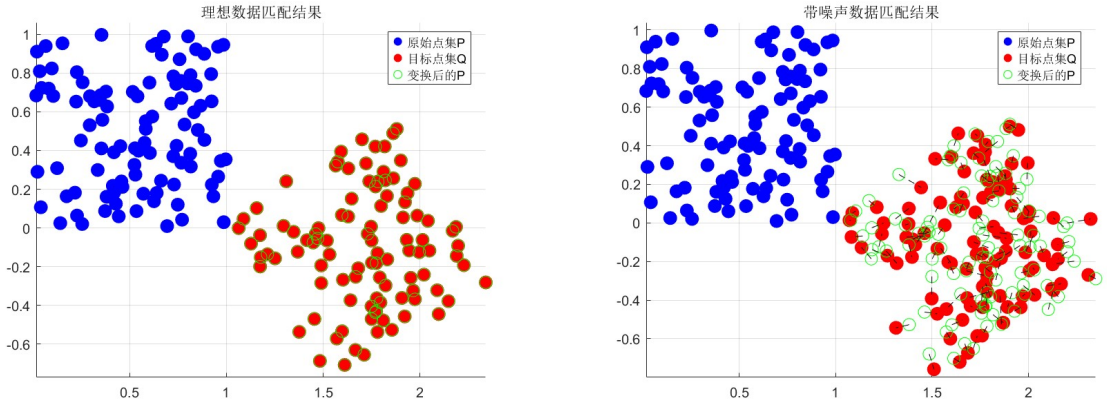
1. 生成点云数据，分有无噪音两种。
2. 利用基于SVD的定位算法计算旋转阵和平移量。
3. 分析误差并可视化。

仿真结果 在没有噪声时，基于SVD的定位算法可以准确获得点云变化，如下图17左侧二图。



(a) 无噪音（20个点）

(b) 有噪音（20个点）



(c) 无噪音（100个点）

(d) 有噪音（100个点）

图 17: 基于SVD的定位算法仿真结果

返回目录

加入随机噪音后，会对算法结果产生较小扰动，但基本仍能获得正确结果，如上图17 on the preceding page右侧二图和下表8，这里使用连线进行数据点的匹配。

数据	旋转角度	平移量	旋转矩阵误差	平移向量误差
真实变换	30.00°	[1.50, 0.80]	-	-
20个点	29.76°	[1.48, -0.78]	0.0059	0.0231
100个点	28.98°	[1.50, -0.79]	0.0251	0.0067

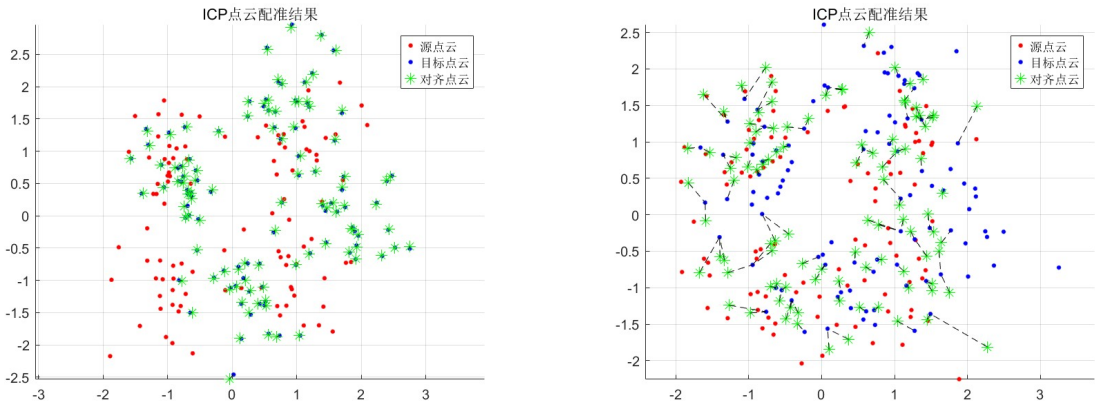
表 8: 有噪音SVD定位结果与误差

6.2.2 基于*ICP*的点云匹配算法

代码逻辑 见附录代码6。

1. 生成点云数据并添加扰动，生成和扰动采用不同尺度（前者时后者50倍）的随机。
2. 定义最近邻搜索函数和基于SVD的定位函数。
3. 利用基于ICP的点云匹配算法进行对齐。
4. 分析误差并可视化。

仿真结果 基于ICP的点云匹配算法得到如下图18左图和下表9 on the next page的结果。



(a) 成功 **(b)** 失败

图 18: 基于ICP的点云匹配算法仿真结果

返回目录

数据	旋转角度	平移量	旋转矩阵误差	平移向量误差
真实变换	30.00°	[0.50, 0.30]	-	-
100个点	30.13°	[0.50, 0.30]	0.0032	0.0013

表 9: ICP点云匹配结果误差

但是这种方法的结果并不一直稳定，有时也会出现匹配失败的结果，如上图18 on the preceding page右图。

7 卡尔曼滤波定位

7.1 原理

7.1.1 应用贝叶斯定理

新信息出现后的概率 = 概率 × 新信息带来的调整：

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

两次独立测量的概率均服从正态分布$p_1(q) = N(\hat{q}_1, \sigma_1^2), p_2(q) = N(\hat{q}_2, \sigma_2^2)$，它们整合得到的最终分布也服从正态分布：

$$\begin{aligned}
p(q) = p_1(q) \cdot p_2(q) &= \frac{1}{\sigma_1\sqrt{2\pi}} exp(-\frac{(q-\hat{q}_1)^2}{2\sigma_1^2}) \cdot \frac{1}{\sigma_2\sqrt{2\pi}} exp(-\frac{(q-\hat{q}_2)^2}{2\sigma_2^2}) \\
&= \frac{1}{2\sigma_1\sigma_2\pi} \exp[-\frac{(q-\hat{q}_1)^2}{2\sigma_1^2} - \frac{(q-\hat{q}_2)^2}{2\sigma_2^2}] \\
&= \frac{1}{2\sigma_1\sigma_2\pi} \exp\{-\frac{1}{2}[\frac{q^2(\sigma_1^2+\sigma_2^2) - 2q(\hat{q}_1\sigma_2^2 + \hat{q}_2\sigma_1^2) + (\hat{q}_1^2\sigma_2^2 + \hat{q}_2^2\sigma_1^2)}{\sigma_1^2\sigma_2^2}]\} \\
&= \frac{1}{2\sigma_1\sigma_2\pi} \exp\{-\frac{1}{2}[\frac{q^2 - \frac{2q(\hat{q}_1\sigma_2^2+\hat{q}_2\sigma_1^2)}{\sigma_1^2+\sigma_2^2} + \frac{(\hat{q}_1^2\sigma_2^2+\hat{q}_2^2\sigma_1^2)}{\sigma_1^2+\sigma_2^2}}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}]\} \\
&= \underbrace{\frac{1}{2\sigma_1\sigma_2\pi} \exp[-\frac{1}{2}\frac{\frac{\hat{q}_1^2\sigma_2^2+\hat{q}_2^2\sigma_1^2}{\sigma_1^2+\sigma_2^2} - (\frac{\hat{q}_1\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2})^2}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}]}_{\text{常数}} \underbrace{\exp[-\frac{1}{2}\frac{(q-\frac{\hat{q}_1\sigma_2^2+\hat{q}_2\sigma_1^2}{\sigma_1^2+\sigma_2^2})^2}{\frac{\sigma_1^2\sigma_2^2}{\sigma_1^2+\sigma_2^2}}]}_{\text{正态分布}} \\
&= N(\hat{q}, \sigma^2)
\end{aligned}$$

返回目录

有：

$$\hat{q} = \underbrace{\frac{\hat{q}_1\sigma_2^2 + \hat{q}_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}}_{\text{方差小的项权重大}} \Leftrightarrow \hat{q} = \hat{q}_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}\underbrace{(\hat{q}_2 - \hat{q}_1)}_{\text{校正量}}$$

$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \Leftrightarrow \sigma^2 = \underbrace{\sigma_1^2 - \frac{\sigma_1^4}{\sigma_1^2 + \sigma_2^2}}_{<\sigma_1^2,\sigma_2^2}$$

以$P = \sigma_1^2, Q = \sigma_2^2, R = \sigma^2$，记卡尔曼增益$K = P(P+Q)^{-1}$和创新协方差$\Sigma_{IN} = P + Q$：

$$\hat{q} = \hat{q}_1 + P(P+Q)^{-1}(\hat{q}_2 - \hat{q}_1) = \hat{q}_1 + K(\hat{q}_2 - \hat{q}_1)$$
$$R = P - P(P+Q)^{-1}P = P - K \cdot \Sigma_{IN} \cdot K^T$$

7.1.2 卡尔曼滤波算法

过程（推算）方程：$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}$

测量方程：$z_k = Hx_k + v_k$

其中$x_k \in R^n$为系统状态，$z_k \in R^m$为测量输出，$u_k \in R^l$为系统输入。$w_k \in R^n$为过程噪声（Process Noise）$p(w) = N(0, Q)$，$v_k \in R^m$为测量噪声（Measurement Noise，白噪声）$p(v) = N(0, R)$。

有先/后验估计$\hat{x}_k^-, \hat{x}_k$，先/后验估计误差$e_k^- = x_k - \hat{x}_k^-, e_k = x_k - \hat{x}_k$，先/后验估计方差$P_k^- = E[e_k^- e_k^{-T}], P_k = E[e_k e_k^T]$。其中，优化目标是$\mathrm{argmin}_K P_k$。

先后验转化关系为：

$$\hat{x}_k = \hat{x}_k^- + \underbrace{K}_{\text{卡尔曼增益}}\underbrace{(z_k - H\hat{x}_k^-)}_{\text{新息}}$$

P_k可进行转化：

$$\begin{aligned}
P_k &= E[e_k e_k^T] = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T](\text{误差展开}) \\
&= E[\{x_k - [\hat{x}_k^- + K(z_k - H\hat{x}_k^-)]\}\{x_k - [\hat{x}_k^- + K(z_k - H\hat{x}_k^-)]\}^T](\text{代入先后验转化关系}) \\
&= E\{[(x_k - \hat{x}_k^-) - K(Hx_k + v_k - H\hat{x}_k^-)][(x_k - \hat{x}_k^-) - K(Hx_k + v_k - H\hat{x}_k^-)]^T\}(\text{代入测量方程}) \\
&= E\{[(I - KH)e_k^- - Kv_k][(I - KH)e_k^- - Kv_k]^T\}(\text{误差重构}) \\
&= (I - KH)\underbrace{E[e_k^- e_k^{-T}]}_{P_n^-}(I - KH)^T + K\underbrace{E[v_k v_k^T]}_{R}K^T - K\underbrace{E[v_k e_k^{-T}]}_{\text{不相干，0}}(I - KH)^T - (I - KH)\underbrace{E[e_k^- v_k^T]}_{\text{不相干，0}}K^T \\
&= (I - KH)P_k^-(I - KH)^T + KRK^T
\end{aligned}$$

返回目录

令偏导为0：

$$\frac{\partial P_k}{\partial K} = -2P_k^- H^T + 2KHP_k^- H^T + 2KR = 0$$

有$K = P_k^- H^T (HP_k^- H^T + R)^{-1}$，在方差趋于0时，$\lim_{R\to 0} K = H^{-1}, \lim_{P_k^- \to 0} K = 0$。

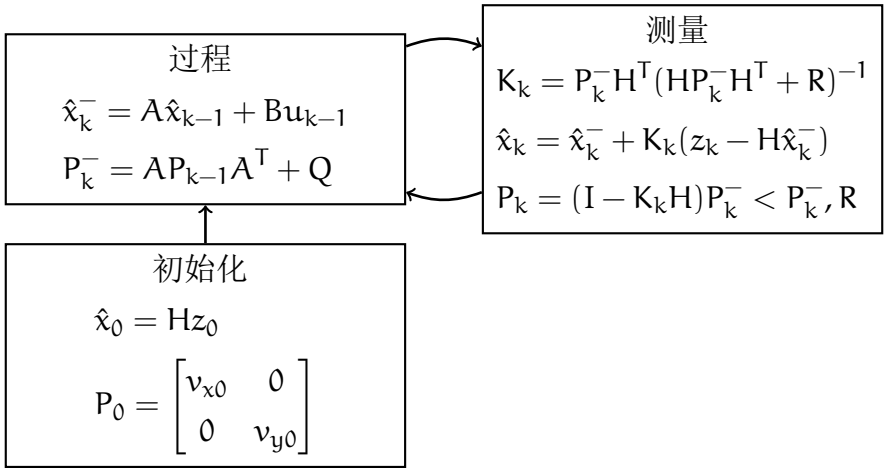


图 19: 卡尔曼滤波算法框图

步骤

状态	状态转移A	测量	测量矩阵H	性能
$\begin{bmatrix} x_k \\ y_k \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} u_k \\ v_k \end{bmatrix}$	$\begin{bmatrix} u_k \\ v_k \end{bmatrix}$	动态误差大 静止误差小 适合缓慢移动
$\begin{bmatrix} x \\ y \\ \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$		$\begin{bmatrix} H_x & 0 & 0 & 0 \\ 0 & H_y & 0 & 0 \end{bmatrix}$ 未测量速度	动态误差小 静止误差大 低维估计高维

表 10: 卡尔曼滤波算法描述维度

描述维度

7.2 仿真

代码逻辑 动态环境下的状态升维卡尔曼滤波定位，见附录代码7。

返回目录

1. 设定参数。
2. 按照位置和状态的升维模型，设计状态转移矩阵、控制输入矩阵、测量矩阵，并给出过程噪声和测量噪声。
3. 初始化状态和协方差矩阵，设定随时间变化的加速度。
4. 进行卡尔曼滤波，迭代进行预测和更新操作。
5. 可视化结果并输出误差。

仿真结果 下图20展示了卡尔曼滤波定位的结果，可以发现，虽然观测值（红色圆点）偏离了真实轨迹（蓝色曲线），但是卡尔曼估计（绿色曲线）与之基本重合。

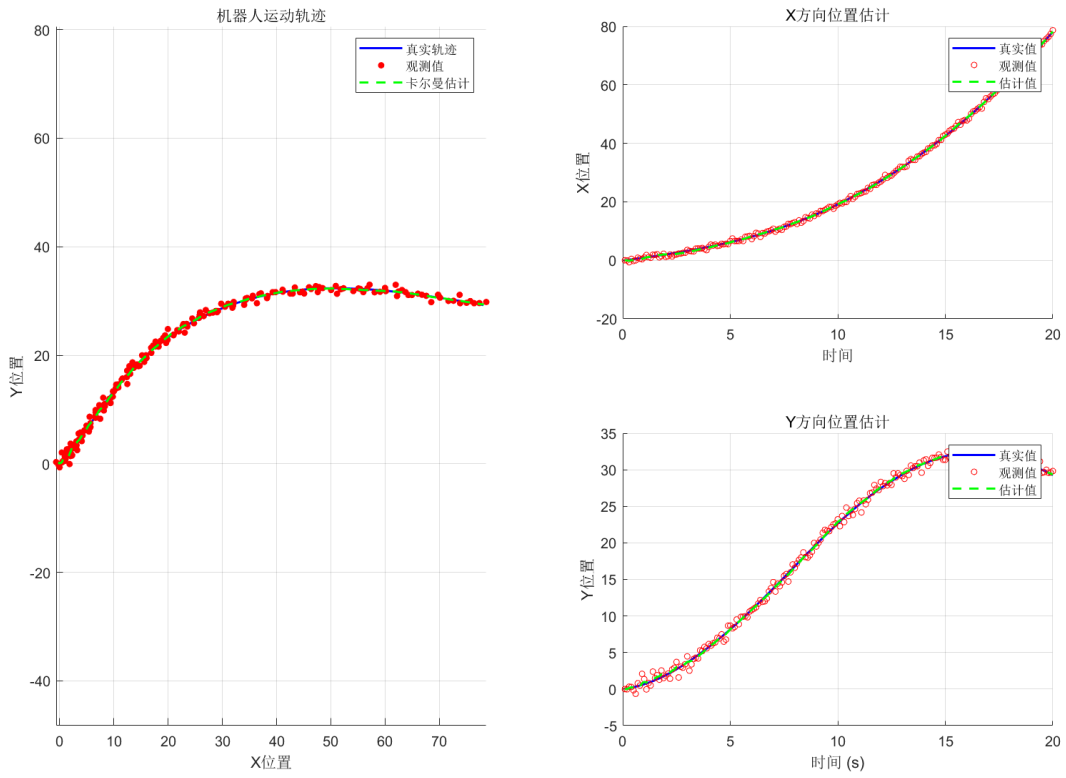


图 20: 卡尔曼滤波定位结果

分析 下图21 on the following page展示了卡尔曼滤波定位的误差变化曲线，经统计，平均位置误差为0.1618。观察曲线的变化趋势，在开始时误差略大，之后一直保持在较小范围内，说明状态升维的卡尔曼滤波算法在动态系统中具有较好的定位性能。

返回目录

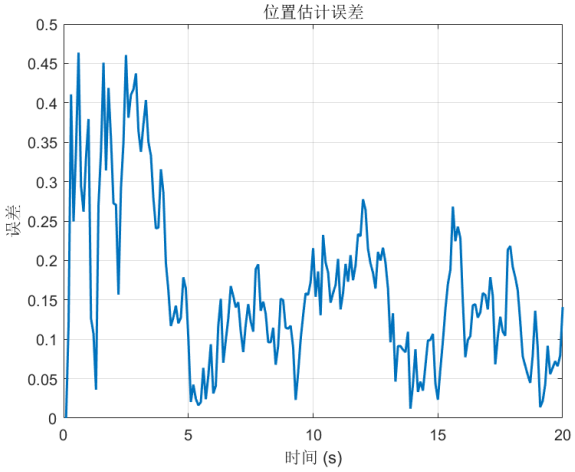
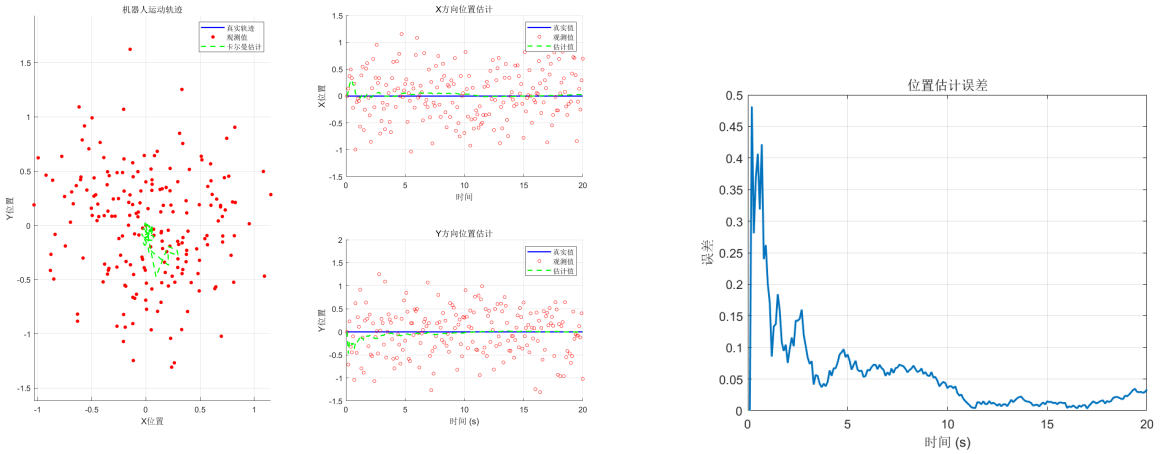


图 21: 卡尔曼滤波定位误差

对比 为验证不同状态维度下的定位性能，进行了以下对比，与PPT结果一致：

1. 静态环境，状态升维：如下图组22，观测值发散，定位偏差，导致出现移动，效果不佳。

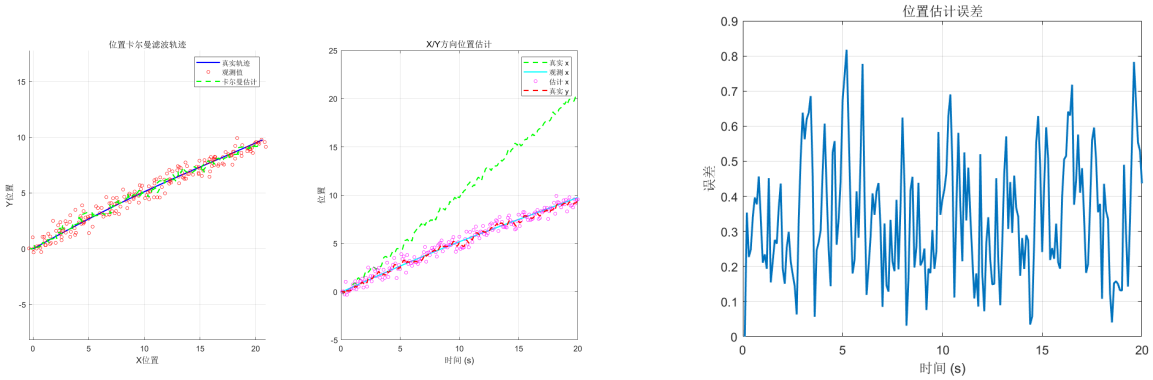


(a) 定位结果 **(b)** 定位误差

图 22: 静态环境，状态升维

2. 动态环境，状态不升维：如下图组23 on the following page，观测值同样在真实轨迹周围，但是卡尔曼估计发生偏离，误差居高不下，效果不佳。

返回目录



(a) 定位结果　　(b) 定位误差

图 23: 动态环境，状态不升维

8　总结

在本次作业的学习中，进行了机器人建模、控制算法、里程计模型（误差传导模型）、激光点云直线特征提取和配准、卡尔曼滤波算法定位的实践，全流程地实现了移动机器人的建模与分析。作业针对三轮叉车移动机器人和激光点云展开，在原理分析、模型推导、编程验证的交叉下，得到了完整的结果。

返回目录

附录

Listing 1: setGoal

```
%% 参数：略
%% 误差转化
% 旋转阵（目标到惯性）
R = [cos(theta_e), -sin(theta_e), 0;
     sin(theta_e),  cos(theta_e), 0;
     0,             0,            1];
% 相对误差：惯性系->目标系->极坐标系
q = [x_s - x_e; y_s - y_e; theta_s - theta_e];
e = R' * q;
rho = sqrt(e(1)^2 + e(2)^2);
beta = -atan2(-e(2), -e(1));
alpha = -beta - e(3);
%% 控制器：存储略
for i = 1:n
    if alpha >= 0.1
        % 线性控制器
        rho = rho + DELTA * (-k_rho*rho*cos(alpha));
        beta = beta + DELTA * (-k_rho*sin(alpha));
        alpha = alpha + DELTA * (k_rho*sin(alpha) - k_alpha*alpha - k_beta*beta);
        % 非线性控制器
        rho = rho + DELTA * (-k_rho*rho*cos(alpha)*cos(alpha));
        beta = beta + DELTA * (-k_rho*cos(alpha)*sin(alpha));
        alpha = alpha + DELTA * (k_rho*cos(alpha)*sin(alpha) - k_alpha*alpha -
            k_rho*sin(alpha)*cos(alpha)/alpha*(alpha-k_beta*beta));
    else % 角小时
        % 线性控制器
        rho = rho + DELTA * (-k_rho*rho);
        beta = beta + DELTA * (-k_rho*alpha);
        alpha = alpha + DELTA * (k_rho*alpha - k_alpha*alpha - k_beta*beta);
        % 非线性控制器
        rho = rho + DELTA * (-k_rho*rho);
        beta = beta + DELTA * (-k_rho*alpha);
```

返回目录

```
        alpha = alpha + DELTA * (-k_alpha*alpha + k_rho*k_beta*beta);
    end
    % 增量转化: 极坐标系->目标系->惯性系（beta逆时针）
    x_d = rho * -cos(-beta);
    y_d = rho * -sin(-beta);
    theta_d = -alpha - beta;
    p_i = R * [x_d; y_d; theta_d];
end
```

Listing 2: setTrajectory

```
%% 参数: 略
%% 生成期望轨迹（惯性系）: 略
%% 轨迹跟踪: 存储略
for i = 1:n
    % 旋转阵（机器人到惯性）
    R = [cos(theta), -sin(theta), 0;
         sin(theta), cos(theta),  0;
         0,          0,           1];
    % 误差信号: 惯性系->机器人系
    q_d = [x - X(i); y - Y(i); theta - Theta(i)];
    e = R' * q_d;
    % 控制器
    if e(3) >= 0.0001
        v = [-k_1*e(1) + dot_X(i)*cos(e(3));
             -dot_X(i)*sin(e(3))/e(3)*e(2) - k_2*e(3) + dot_Theta(i)];
        dot_e = [-k_1*e(1) + v(2)*e(2);
                 -v(2)*e(1) + dot_X(i)*sin(e(3));
                 -k_2*e(3) - dot_X(i)*sin(e(3))/e(3)*e(2)];
    else % 角小时
        v = [-k_1*e(1) + dot_X(i);
             -dot_X(i)*e(2) + dot_Theta(i)];
        dot_e = [-k_1*e(1) + v(2)*e(2);
                 -v(2)*e(1);
                 -k_2*e(3) - dot_X(i)*e(2)];
    end
    e = e + dot_e .* DELTA;
```

返回目录

```
    % 更新
    dot_x = v(1) * cos(theta);
    dot_y = v(1) * sin(theta);
    x = x + dot_x * DELTA;
    y = y + dot_y * DELTA;
    theta = theta + v(2) * DELTA;
end
```

Listing 3: odometer

```
%% 参数: 略
%% 解析积分法
for k = 1:(length(t)-1)
    if abs(omega) == 0 % 角速度为0
        x_exact(k+1) = x_exact(k) + v * dt;
        y_exact(k+1) = y_exact(k);
    else
        theta_exact(k+1) = theta_exact(k) + omega * dt;
        x_exact(k+1) = x_exact(k) + v / omega * (sin(theta_exact(k+1)) -
            sin(theta_exact(k)));
        y_exact(k+1) = y_exact(k) - v / omega * (cos(theta_exact(k+1)) -
            cos(theta_exact(k)));
    end
end
%% 欧拉法
for k = 1:(length(t)-1)
    x_euler(k+1) = x_euler(k) + v * dt * cos(theta_euler(k));
    y_euler(k+1) = y_euler(k) + v * dt * sin(theta_euler(k));
    theta_euler(k+1) = theta_euler(k) + omega * dt;
end
error_euler = sqrt((x_euler - x_exact).^2 + (y_euler - y_exact).^2);
%% 二阶Runge-Kutta法
for k = 1:(length(t)-1)
    s = v * dt;
    theta_rk = theta_rk2(k) + dt / 2 * omega;
    x_rk2(k+1) = x_rk2(k) + s * cos(theta_rk);
    y_rk2(k+1) = y_rk2(k) + s * sin(theta_rk);
```

返回目录

```
    theta_rk2(k+1) = theta_rk2(k) + dt * omega;
    % 误差传导矩阵
    F_p(:, :, k) = [
        1, 0, -s * sin(theta_rk);
        0, 1, s * cos(theta_rk);
        0, 0, 1];
    F_rl(:, :, k) = [
        r * sin(beta) * cos(theta_rk) - r^2 * dot_phi * cos(beta) * sin(beta) / (2 *
            l) * sin(theta_rk), ...
        r * dot_phi * cos(beta) * cos(theta_rk) + r^2 * dot_phi^2 * sin(beta)^2 / (2
            * l) * sin(theta_rk);
        r * sin(beta) * sin(theta_rk) + r^2 * dot_phi * cos(beta) * sin(beta) / (2 *
            l) * cos(theta_rk), ...
        r * dot_phi * cos(beta) * sin(theta_rk) - r^2 * dot_phi^2 * sin(beta)^2 / (2
            * l) * cos(theta_rk);
        r * cos(beta) / l, -r * dot_phi * sin(beta) / l];
end
error_rk2 = sqrt((x_rk2 - x_exact).^2 + (y_rk2 - y_exact).^2);
%% 协方差矩阵
P = zeros(3, 3, length(t));
delta = [k_phi * dt * dot_phi, 0;
    0, k_beta];
for k = 1:(length(t)-1)
    F_p_k = squeeze(F_p(:, :, k));
    F_rl_k = squeeze(F_rl(:, :, k));
    P(:, :, k+1) = F_p_k * P(:, :, k) * F_p_k' + F_rl_k * delta * F_rl_k';
end
```

Listing 4: line extraction

```
%% 数据生成：略
%% 法1：split-and-merge
% 参数：略
% Split
split_done = false;
while ~split_done
    split_done = true;
```

返回目录

```
    new_segments = {};
    for i = 1:length(segments)
        current_segment = segments{i};
        % 防止过度分割
        if size(current_segment, 2) < min_points
            new_segments{end + 1} = current_segment;
            continue;
        end
        % 端点拟合
        p1 = current_segment(:, 1);
        p2 = current_segment(:, end);
        dx = p2(1) - p1(1);
        dy = p2(2) - p1(2);
        % 点集到直线距离最远点
        distances = abs(dy * current_segment(1, :) - dx * current_segment(2, :) +
            (p2(1) * p1(2) - p2(2) * p1(1))) / sqrt(dx^2 + dy^2);
        [max_dist, max_idx] = max(distances);
        % 大于阈值分裂
        if max_dist > dist_threshold1
            split_done = false;
            new_segments{end + 1} = current_segment(:, 1:max_idx);
            new_segments{end + 1} = current_segment(:, max_idx:end);
        else
            new_segments{end + 1} = current_segment;
        end
    end
    segments = new_segments;
end
% Merge
merge_done = false;
while ~merge_done
    merge_done = true;
    new_segments = {};
    i = 1;
    while i <= length(segments)
        % 处理最后一个段
```

返回目录

```
        if i == length(segments)
            new_segments{end + 1} = segments{i};
            break;
        end
        % 尝试合并当前段和下一段
        seg1 = segments{i};
        seg2 = segments{i + 1};
        combined_seg = [seg1, seg2];
        % 端点拟合
        p1 = combined_seg(:, 1);
        p2 = combined_seg(:, end);
        dx = p2(1) - p1(1);
        dy = p2(2) - p1(2);
        % 点集到直线距离最远点
        distances = abs(dy * combined_seg(1, :) - dx * combined_seg(2, :) + (p2(1) *
            p1(2) - p2(2) * p1(1))) / sqrt(dx^2 + dy^2);
        [max_dist,~] = max(distances);
        % 判断是否执行合并
        if max_dist < dist_threshold1
            merge_done = false;
            new_segments{end + 1} = combined_seg;
            i = i + 2;
        else
            new_segments{end + 1} = seg1;
            i = i + 1;
        end
    end
    segments = new_segments;
end
%% 法2: Line-Regression
% 参数: 略
% 滑动窗口拟合
count = 0;
for i = 1:step:(length(x_sorted) - window_size + 1)
    % 提取当前窗口数据
    window_idx = i : (i + window_size - 1);
```

返回目录

```
    x_window = x_sorted(:, window_idx)';
    % 最小二乘法拟合
    A = [x_window(:,1), ones(window_size,1)];
    beta = (A'*A) \ (A'*x_window(:,2));
    % 保存线段
    count = count + 1;
    segments(count).a = beta(1);
    segments(count).b = beta(2);
    segments(count).x_range = [x_window(1,1), x_window(end,1)];
    segments(count).points = x_window;
end
% 合并
changed = true;
while changed % 外层循环：反复从头尝试合并
    changed = false;
    i = 1;
    n = length(segments);
    while i < n - 1 % 内层循环：相邻尝试合并
        current = segments(i);
        next = segments(i + 1);
        % 计算角度差
        angle_diff = rad2deg(abs(atan(current.a) - atan(next.a)));
        % 计算中点到对方直线的距离
        mid_x = mean(current.x_range);
        mid_y = current.a * mid_x + current.b;
        dist = abs(next.a * mid_x - mid_y + next.b) / sqrt(next.a^2 + 1);
        % 合并条件判断
        if angle_diff < merge_angle_threshold && dist < merge_dist_threshold
            % 合并两个线段
            merged_points = [current.points; next.points];
            A = [merged_points(:,1), ones(size(merged_points,1),1)];
            beta = (A'*A) \ (A'*merged_points(:,2));
            segments(i).a = beta(1);
            segments(i).b = beta(2);
            segments(i).x_range = [current.x_range(1), next.x_range(2)];
            segments(i).points = merged_points;
```

返回目录

```
            segments(i + 1) = [];
            n = n - 1;
            changed = true;
        else % 不合并移动到下一个
            i = i + 1;
        end
    end
end
%% 法3: RANSAC
% 参数: 略
for i = 1:iter_num
    count = 0;
    count_old = 0;
    % 随机选点（未防止重复选择）
    x1 = x_obs(:, randi(length(x_obs)));
    x2 = x_obs(:, randi(length(x_obs)));
    while norm(x1 - x2) < 1 % 防止选点过近（确保斜率）
        x2 = x_obs(:, randi(length(x_obs)));
    end
    % 统计内点数
    for j = 1:length(x_obs)
        distances = abs((x2(2) - x1(2)) * x_obs(1,:) - (x2(1) - x1(1)) * x_obs(2,:) +
            x2(1) * x1(2) - x2(2) * x1(1)) ...
            ./ sqrt((x2(2) - x1(2))^2 + (x2(1) - x1(1))^2);
        count = sum(distances < dist_threshold2);
    end
    % 更新最佳点组合
    if count > count_old
        count_old = count;
        x1_best = x1;
        x2_best = x2;
    end
end
%% 法4: Hough-Transform
% 参数: 略
% 投票
```

返回目录

```
[THETA, RHO] = meshgrid(theta, rho);
for i = 1:all_points
    rho_vals = x_obs(1,i)*cos(THETA) + x_obs(2,i)*sin(THETA);
    rho_diff = abs(rho_vals - RHO);
    [~, idx] = min(rho_diff, [], 1);
    for t = 1:length(theta)
        accumulator(idx(t), t) = accumulator(idx(t), t) + 1;
    end
end
% 寻找峰值
threshold = peak_thresh * max(accumulator(:));
[rho_idxs, theta_idxs] = find(accumulator >= threshold);
```

Listing 5: SVD

```
function [R, t] = svd_localization(P, Q, w)
    % 计算加权中心点
    sum_w = sum(w);
    hat_p = (P * w') / sum_w;
    hat_q = (Q * w') / sum_w;
    % 去中心化
    X = P - hat_p;
    Y = Q - hat_q;
    % 构造S矩阵并进行SVD分解
    W = diag(w); % 权重对角阵
    S = X * W * Y';
    [U, ~, V] = svd(S);
    % 旋转矩阵
    R = V * U';
    if det(R) < 0 % 保证单位阵
        V(:, end) = -V(:, end);
        R = V * U';
    end
    % 平移向量
    t = hat_q - R * hat_p;
end
```

返回目录

Listing 6: ICP

```
%% 数据生成：略
%% ICP
% 参数：略
for iter = 1:maxIter
    [idx, matched_dst] = Neighbor(aligned, dst); % 最近邻匹配
    [R, t] = SVD(aligned, matched_dst); % SVD变换
    % 应用变换
    T = [R t; 0 0 1] * T;      % 累积变换
    aligned = R * aligned + t; % 更新对齐点云
    % 误差
    dist = sqrt(sum((aligned - matched_dst).^2, 1));
    meanError = mean(dist);
    fprintf('Iter %d: Error = %.4f\n', iter, meanError);
    % 判敛
    if abs(prevError - meanError) < tolerance
        break;
    end
    prevError = meanError;
end
%% 最近邻搜索函数：略
%% 基于SVD的机器人定位函数：略
```

Listing 7: Kalman

```
%% 参数：略
%% 卡尔曼滤波参数矩阵：略
%% 初始化：略
%% 主循环
for k = 2:steps
    % 随机加速度扰动
    acc_noise = sigma_a * randn(2,1);
    u = acceleration(:,k-1) + acc_noise;
    % 更新真实状态
    true_state(:,k) = F * true_state(:,k-1) + B * u;
    % 生成带噪声的测量
    meas_noise = sigma_meas * randn(2,1);
```

返回目录

```
    meas(:,k) = H * true_state(:,k) + meas_noise;
    % 预测
    pred_state = F * est_state(:,k-1) + B * u;
    P_pred = F * P * F' + Q;
    % 更新
    K = P_pred * H' / (H * P_pred * H' + R);
    est_state(:,k) = pred_state + K * (meas(:,k) - H * pred_state);
    P = (eye(4) - K * H) * P_pred;
end
```

返回目录