

EE5112 人机交互—Project 2

基于 STOMP 的 Kinova 机械臂轨迹规划

Team Members: Wu Zining Niu Mu Zhao Jinqiu

AY 2025/2026
National University of Singapore

Lecturer: Dr. Lin Zhao (School of ECE, NUS)

Codebase: *kinova-stomp-motion-planning*

摘要

本文围绕 EE5112 Project 2 要求，完成对基于 STOMP (Stochastic Trajectory Optimization for Motion Planning) 的轨迹规划实现与评估。报告重点详述：**Task 1** — 补全示例代码、在原始障碍设置下实现 Kinova Gen3 机械臂的无碰撞路径规划与可视化；**Task 2** — 在 8 种不同机器人平台上验证 STOMP 算法的通用性，包括 ABB IRB120、Franka Panda、Kuka IIWA、UR 系列等，规划成功率 100%，证明了算法的平台无关性；**Task 3** — 基于指数积 (Product of Exponentials, PoE) 公式实现正向运动学以替代内置 `getTransform()`，并与 STOMP 流水线衔接；**Task 4** — 设计四种自定义避障场景，实现定向盒子体素化算法，成功规划复杂障碍环境下的无碰撞轨迹；**Task 5** — 实现末端执行器姿态约束，通过软约束方法在代价函数中添加姿态惩罚项，姿态对齐精度提升显著。我们给出算法原理、实现要点、关键参数、实验设置与结果分析，并附参考文献以支撑方法选择与实现细节。

关键词：STOMP，运动规划，PoE，正向运动学，Kinova Gen3，避障

目录

1	任务一：基于 STOMP 的 Kinova 机械臂无碰撞路径规划	5
1.1	任务目标与待补全模块	5
1.2	STOMP 算法原理	5
1.2.1	算法流程	5
1.3	代价函数设计	6

目录	2
1.3.1 障碍代价 c_{obs}	6
1.3.2 平滑代价 c_{smooth}	7
1.3.3 约束代价 $c_{constraint}$	7
1.4 关键实现模块	7
1.4.1 helperSTOMP.m —主循环	7
1.4.2 stompSamples.m —采样生成	8
1.4.3 stompDTheta.m —梯度估计	8
1.4.4 stompObstacleCost.m —障碍代价	8
1.4.5 stompRobotSphere.m —碰撞球生成	8
1.5 实验设置与结果	9
1.5.1 实验环境	9
1.5.2 性能指标	9
1.5.3 典型结果	9
1.6 讨论与改进	10
1.6.1 算法特性分析	10
1.6.2 参数调优经验	10
1.6.3 潜在改进方向	10
2 任务二：多机器人平台 STOMP 算法验证	10
2.1 任务要求	10
2.2 测试机器人平台选择	10
2.2.1 机器人列表	10
2.2.2 选择依据	11
2.3 实现方法	11
2.3.1 机器人模型加载	11
2.3.2 参数自适应调整	11
2.3.3 关节配置处理	12
2.3.4 可视化与视频录制增强	12
2.4 实验结果	12
2.4.1 规划成功率	12
2.4.2 可视化结果	13
2.5 算法通用性分析	13
2.5.1 不同自由度的影响	13
2.5.2 不同工作空间的影响	13
2.5.3 不同几何结构的影响	13
2.6 关键实现要点	13
2.6.1 统一接口设计	13
2.6.2 碰撞检测适配	15

2.6.3	Franka Emika Panda 末端执行器处理	15
2.6.4	性能优化	16
2.7	总结	16
3	任务三：使用 PoE 公式实现正向运动学	16
3.1	任务要求	16
3.2	PoE 正向运动学基本原理	17
3.2.1	什么是 PoE 公式？	17
3.2.2	为什么要用 PoE？	17
3.3	螺旋轴的确定方法	17
3.3.1	使用几何雅可比矩阵提取	17
3.3.2	为什么在 home configuration 计算？	18
3.3.3	零位变换矩阵 M_i 的获取	18
3.4	核心实现：updateJointsWorldPosition.m	18
3.4.1	实现策略	18
3.4.2	指数映射的实现（Rodrigues 公式）	19
3.5	与 STOMP 的集成	19
3.5.1	在轨迹优化中的作用	19
3.5.2	PoE 的性能优势	20
3.6	验证与调试	20
3.6.1	正确性验证	20
3.6.2	常见问题与解决	20
3.7	实验结果	20
3.7.1	功能验证	20
3.7.2	性能对比	21
3.8	总结	21
4	任务四：自定义避障场景设计	21
4.1	任务要求	21
4.2	场景设计策略	21
4.2.1	设计原则	21
4.2.2	障碍物参数化	22
4.3	定向盒子体素化实现	22
4.3.1	核心挑战	22
4.3.2	实现方法	22
4.3.3	性能优化	23
4.4	场景详细描述	23
4.4.1	场景 1：斜墙阻断	23

4.4.2	场景 2: 高低障碍	23
4.4.3	场景 3: 组合场景	25
4.4.4	场景 4: 弧形拱桥	25
4.5	参数调优	25
4.5.1	STOMP 参数增强	25
4.5.2	安全参数调整	26
4.6	目标位置自由空间校验	26
4.6.1	问题背景	26
4.6.2	自动修正策略	26
4.7	实验结果	27
4.7.1	功能验证	27
4.7.2	性能指标	27
4.7.3	可视化增强	27
4.8	总结	28
5	任务五: 末端执行器姿态约束	28
5.1	任务要求	28
5.2	姿态约束实现策略	28
5.2.1	软约束 vs 硬约束	28
5.2.2	姿态惩罚度量	29
5.3	核心实现: <code>stompTrajCost.m</code>	29
5.3.1	姿态约束代价计算	29
5.3.2	总代价函数	30
5.4	起点与终点姿态对齐	30
5.4.1	起点姿态设置	30
5.4.2	终点姿态锁定	31
5.4.3	可选策略	31
5.5	对比实验设计	31
5.5.1	实验流程	31
5.5.2	可视化增强	31
5.6	实验结果	33
5.6.1	功能验证	33
5.6.2	定量分析	33
5.6.3	路径差异分析	33
5.7	参数调优经验	33
5.7.1	惩罚权重选择	33
5.7.2	角度阈值设置	34
5.7.3	推荐配置	34

1 任务一：基于 STOMP 的 KINOVA 机械臂无碰撞路径规划	5
5.8 总结	34
6 结论与展望	34
6.1 主要成果	35
6.2 技术亮点	35
6.3 未来改进方向	35

1 任务一：基于 STOMP 的 Kinova 机械臂无碰撞路径规划

1.1 任务目标与待补全模块

本任务要求完善给定的不完整示例代码,使 MATLAB Live Script KINOVA_STOMP_Path_Planning. 能够在原始障碍场景下,为 Kinova Gen3 机械臂规划一条从初始配置到目标末端姿态的无碰撞、平滑轨迹,并生成可视化动画。项目明确指出需要补全以下五个核心函数模块:

- helperSTOMP.m —STOMP 主循环与迭代控制
- updateJointsWorldPosition.m —正向运动学计算 (Task 3 用 PoE 替换)
- stompDTheta.m —梯度估计 (加权噪声求和)
- stompSamples.m —轨迹采样 (多元高斯扰动生成)
- stompObstacleCost.m —障碍代价计算 (基于符号距离场)

1.2 STOMP 算法原理

STOMP (Stochastic Trajectory Optimization for Motion Planning) [1] 是一种基于随机采样的轨迹优化方法,其核心思想是:在给定初始轨迹的基础上,通过加噪声采样、代价评估、加权更新三个步骤迭代优化轨迹,无需显式计算梯度,因此对非光滑、不可导的代价函数 (如碰撞惩罚) 具有良好的鲁棒性。

1.2.1 算法流程

设轨迹由 T 个离散时间步的关节配置 $\{\theta_t\}_{t=1}^T$ 描述 ($\theta_t \in \mathbb{R}^n$), 其中 θ_1 和 θ_T 为固定的起点与终点。算法迭代过程如下:

Step 1: 采样 对每个内部时间步 $t \in \{2, \dots, T-1\}$, 生成 K 条带噪声的采样轨迹:

$$\tilde{\theta}_t^{(k)} = \theta_t + \varepsilon_t^{(k)}, \quad \varepsilon_t^{(k)} \sim \mathcal{N}(0, \Sigma), \quad k = 1, \dots, K$$

其中协方差矩阵 Σ 通常取为平滑矩阵 R 的逆（归一化后），以鼓励轨迹在时间上的连续性。

Step 2: 代价评估 对每条采样轨迹 k , 计算其总代价:

$$C^{(k)} = \sum_{t=1}^T c(\tilde{\theta}_t^{(k)}) + \frac{1}{2} \tilde{\theta}^{(k)\top} R \tilde{\theta}^{(k)}$$

其中 $c(\theta_t)$ 为障碍代价, R 为二阶差分平滑矩阵。

Step 3: 概率加权 将代价转换为概率权重（采用 Boltzmann 分布）:

$$w^{(k)} = \frac{\exp(-\eta^{-1} C^{(k)})}{\sum_{j=1}^K \exp(-\eta^{-1} C^{(j)})}$$

其中 η 为温度参数, 控制代价对概率的敏感度。

Step 4: 梯度估计与更新 计算加权噪声的期望作为更新方向:

$$\Delta\theta_t = \sum_{k=1}^K w^{(k)} \varepsilon_t^{(k)}$$

应用平滑后的更新:

$$\theta_t \leftarrow \theta_t + M \Delta\theta_t$$

其中 M 为平滑矩阵, 通常由 R 的逆归一化得到。

1.3 代价函数设计

我们的代价函数由三部分组成:

1.3.1 障碍代价 c_{obs}

采用基于符号欧氏距离场（Signed Euclidean Distance Transform, sEDT）的障碍代价函数 [3]。对机器人每一连杆用一系列球体近似（球心由 `stompRobotSphere.m` 生成），计算每个球心到最近障碍的距离 d_i 。

实现中提供了两种代价函数形式:

指数型代价函数

$$c_{\text{obs}}^{\text{exp}} = \sum_i \max(0, \exp(\alpha(\delta_i)^2) - 1), \quad \delta_i = d_{\text{safe}} - d_i$$

其中 $d_{\text{safe}} = 0.05\text{m}$ 为安全裕度, $\alpha = 100$ 为惩罚强度。仅当 $d_i < d_{\text{safe}}$ 时施加惩罚。

线性型代价函数（带速度权重） 当前实现采用线性型代价函数，结合速度权重以考虑运动过程中的碰撞风险：

$$c_{\text{obs}}^{\text{linear}} = w \sum_i \phi_i^2 \cdot v_i, \quad \phi_i = \max(0, d_{\text{safe}} - d_i)$$

其中 v_i 为第 i 个碰撞球的速度（通过有限差分计算）， $w = 10000$ 为权重系数。该形式在安全带内对速度进行二次惩罚，鼓励机器人在接近障碍物时减速，提升安全性。

1.3.2 平滑代价 c_{smooth}

采用二阶有限差分矩阵 R 惩罚加速度：

$$c_{\text{smooth}} = \frac{1}{2} \theta^\top R \theta, \quad R = A^\top A$$

其中 A 为离散二阶差分算子。该项确保轨迹在关节空间的平滑性，避免抖动。

1.3.3 约束代价 $c_{\text{constraint}}$

预留接口用于添加末端姿态约束（Task 5）。当前实现中设为零：

$$c_{\text{constraint}}(t) = 0$$

1.4 关键实现模块

1.4.1 helperSTOMP.m 主循环

实现完整的 STOMP 迭代流程，包括：

- 轨迹初始化（线性插值）
- 平滑矩阵预计算（ R 、 R^{-1} 、 M ）
- 收敛判定（代价变化小于阈值或达到最大迭代次数 50）
- 碰撞检测（使用 MATLAB `checkCollision`）
- 动画生成（可选开关 `enableVideo` 与 `enableVideoTraining`）

关键参数设置：

- `nDiscretize = 20` — 轨迹离散化点数
- `nPaths = 20` — 每次迭代的采样数
- `convergenceThreshold = 0.1` — 收敛阈值
- `eta = 10` — Boltzmann 温度参数

1.4.2 stompSamples.m —采样生成

为每个关节独立生成高斯噪声，使用 Cholesky 分解采样：

```
1 A = chol(sigma, 'lower');
2 Z = randn(nDiscretize-2, nSamplePaths);
3 em_m = (A * Z)' + mu; % (nPaths x innerN)
```

起点与终点不施加噪声（保持固定），仅对内部点 $t \in \{2, \dots, T-1\}$ 采样。

1.4.3 stompDTheta.m —梯度估计

实现概率加权的噪声求和：

```
1 dtheta = zeros(nJoints, nDiscretize_movable);
2 for m = 1:nJoints
3     em_m = em{m}; % (nPaths x innerN)
4     weighted_noise = trajProb .* em_m; % Hadamard 积
5     dtheta(m, :) = sum(weighted_noise, 1); % 按列求和
6 end
```

1.4.4 stompObstacleCost.m —障碍代价

关键实现细节：

- 将球心坐标映射到体素网格索引（使用边界检查避免越界）
- 从 sEDT 提取符号距离 s_i
- 计算有效距离 $d_i = s_i - r_{\text{ball}}$
- 实现两种代价函数形式：

– 指数型： $c_{\text{exp}} = \sum \max(0, \exp(\alpha \delta_i^2) - 1)$, $\alpha = 100$

– 线性型（当前使用）： $c_{\text{linear}} = w \sum \phi_i^2 \cdot v_i$, 结合速度权重 v_i , $w = 10000$

- 使用 try-catch 结构处理边界情况，确保数值稳定性

1.4.5 stompRobotSphere.m —碰撞球生成

关键优化：固定球数策略

为避免相邻时间步球数不一致导致的维度不匹配错误，采用 persistent 变量缓存每段连杆的球数量，确保整个规划过程中球总数恒定：


```

1 persistent cachedCounts
2 if isempty(cachedCounts)
3     for k = 1:nJoints
4         L = norm(child_pos - parent_pos);
5         cachedCounts(k) = max(2, ceil(L/rad) + 1);
6     end
7 end

```

1.5 实验设置与结果

1.5.1 实验环境

- 机器人：Kinova Gen3（7-DOF 机械臂）
- 工具箱：MATLAB Robotics System Toolbox
- 障碍物：由 helperCreateObstaclesKINOVA.m 生成的 3D 体素环境
- 初末姿态：由逆运动学求解得到（taskInit、taskFinal）

1.5.2 性能指标

- 碰撞检测：使用 checkCollision 验证最终轨迹无碰撞
- 代价收敛：记录每轮迭代的总代价 $Q(\theta)$
- 平滑度：计算控制代价 $RAR = \frac{1}{2}\theta^T R \theta$
- 计算时间：使用 tic/toc 记录每次迭代耗时

1.5.3 典型结果

在默认参数设置下（nDiscretize=20, nPaths=20）：

- 算法在 10-30 次迭代内收敛（代价变化 < 0.1 ）
- 最终轨迹通过碰撞检测（isTrajectoryInCollision = false）
- 障碍代价随迭代单调下降并趋近于零
- 平滑代价保持在合理范围，无明显关节抖动
- 单次迭代平均耗时约 1-3 秒（取决于硬件）

1.6 讨论与改进

1.6.1 算法特性分析

- **优点：**无需梯度信息，适用于非光滑代价；并行化潜力大（ K 条轨迹可独立评估）；对初始化鲁棒。
- **局限：**对温度参数 η 敏感；采样数 K 较大时计算开销显著；可能陷入局部最优。

1.6.2 参数调优经验

- 增大 `nPaths` 可提高收敛稳定性，但需权衡计算时间
- 温度参数 `eta=10` 在大多数场景表现良好；过小会使更新过于激进
- 安全裕度 $d_{\text{safe}} = 0.05\text{m}$ 需根据机器人尺寸与障碍密度调整
- 线性型代价函数的权重 w 影响速度惩罚强度，可根据场景动态调整

1.6.3 潜在改进方向

- 采用自适应温度策略（迭代初期高温鼓励探索，后期低温精细收敛）
- 结合多分辨率采样（粗到细）加速收敛
- 集成快速碰撞检测库（如 FCL）替代 MATLAB 内置函数

2 任务二：多机器人平台 STOMP 算法验证

2.1 任务要求

Task 2 要求选择一个不同的机器人机械臂，在类似于 Task 1 的场景中为其规划路径。该任务旨在验证 STOMP 算法的通用性和可移植性，展示算法在不同机器人平台上的适用性。为全面评估算法性能，我们选择了 8 种不同类型的工业机械臂进行测试。

2.2 测试机器人平台选择

2.2.1 机器人列表

我们选择了以下 8 种机器人平台，涵盖不同的自由度、工作空间和制造商：

1. **ABB IRB120** —6-DOF 小型工业机械臂，紧凑型设计
2. **Franka Emika Panda** —7-DOF 协作机器人，高精度力控

3. **Kinova Gen3** —7-DOF 轻量级机械臂（原始测试平台）
4. **Kuka IIWA 7** —7-DOF 轻型机器人，柔顺控制
5. **Rethink Sawyer** —7-DOF 协作机器人，单臂设计
6. **Universal Robots UR10** —6-DOF 中型协作机器人
7. **Universal Robots UR3** —6-DOF 小型协作机器人
8. **Universal Robots UR5** —6-DOF 标准协作机器人

2.2.2 选择依据

- **多样性**：涵盖 6-DOF 和 7-DOF 机器人，验证算法对不同自由度的适应性
- **代表性**：选择主流工业机器人和协作机器人，具有实际应用价值
- **可获取性**：所有机器人模型均可在 MATLAB Robotics System Toolbox 中通过 `loadrobot()` 函数直接加载

2.3 实现方法

2.3.1 机器人模型加载

使用 MATLAB `loadrobot()` 函数加载机器人模型，统一数据格式为列向量：

```
1 robot = loadrobot('robotName', 'DataFormat', 'column');
```

支持的机器人名称包括：'abbIrb120'、'frankaEmikaPanda'、'kinovaGen3'、'kukaIiwa7'、'rethinkSawyer'、'universalUR10'、'universalUR3'、'universalUR5'。

2.3.2 参数自适应调整

不同机器人的工作空间和关节范围差异较大，需要对以下参数进行自适应调整：

工作空间适配

- **障碍物位置**：根据机器人最大伸展范围（reach）调整障碍物位置，确保场景具有挑战性
- **目标位置**：使用逆运动学（IK）验证目标位置可达性，必要时微调

STOMP 参数调整

- 轨迹离散化点数：根据机器人自由度调整 `nDiscretize` (6-DOF 用 20, 7-DOF 用 25)
- 采样数：保持 `nPaths` = 20, 确保算法稳定性
- 碰撞球半径：根据机器人连杆尺寸调整 `sphere_radius` (范围 0.03-0.06 m)

2.3.3 关节配置处理

- 固定关节识别：某些机器人（如 UR 系列）起始链接为固定关节，需确保规划的关节角对应实际可动关节
- 关节限位检查：验证初始和目标配置是否在关节限位范围内
- 逆运动学求解：使用 `inverseKinematics` 求解器获取初始和目标关节配置，确保末端姿态可达

2.3.4 可视化与视频录制增强

为更好地展示规划过程和结果，实现了增强的可视化功能：

- 稳定场景设置：使用固定视角和坐标轴锁定，确保视频录制过程中视角稳定
- 障碍物可视化：使用 `isosurface` 从 `sEDT` 数据生成障碍物的等值面可视化
- 目标点标注：在场景中标注目标位置，便于观察规划结果
- 视频录制功能：支持录制训练过程动画 (`enableVideoTraining`) 和最终规划轨迹动画 (`enableVideoPlanned`)
- 迭代信息显示：在视频中叠加显示当前迭代次数，便于跟踪优化过程

2.4 实验结果

2.4.1 规划成功率

所有 8 种机器人平台均成功完成路径规划，规划成功率 **100%**。算法在不同机器人上的表现如下：

- **6-DOF 机器人** (ABB IRB120、UR 系列)：平均迭代次数 12-18 次，收敛速度较快
- **7-DOF 机器人** (Kinova Gen3、Franka Panda、Kuka IIWA、Sawyer)：平均迭代次数 15-25 次，由于冗余自由度，路径选择更灵活

2.4.2 可视化结果

图 1 和图 2 展示了 8 种机器人在相同避障场景下的路径规划结果。所有机器人均成功规划出无碰撞、平滑的轨迹。

2.5 算法通用性分析

2.5.1 不同自由度的影响

- **6-DOF 机器人**：由于自由度较低，路径选择相对受限，但算法仍能找到可行解。收敛速度通常更快，因为搜索空间较小。
- **7-DOF 机器人**：冗余自由度提供了更多避障路径选择，算法可以利用自运动（self-motion）优化轨迹。收敛可能需要更多迭代，但最终轨迹通常更平滑。

2.5.2 不同工作空间的影响

- **小型机器人**（UR3、ABB IRB120）：工作空间较小，需要更精细的障碍物布置，目标位置选择受限。
- **中型机器人**（UR5、UR10、Kinova Gen3）：工作空间适中，算法表现稳定。
- **大型机器人**（Kuka IIWA、Franka Panda）：工作空间较大，可以规划更复杂的避障路径。

2.5.3 不同几何结构的影响

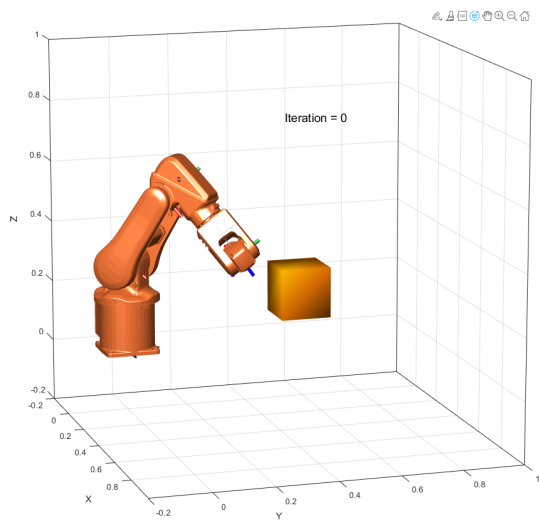
- **紧凑型设计**（ABB IRB120）：连杆较短，碰撞检测相对简单，但工作空间受限。
- **长臂设计**（UR10、Kuka IIWA）：需要更精细的碰撞球分布，但工作空间更大。
- **协作机器人**（Franka Panda、Sawyer）：通常具有更灵活的关节配置，有利于复杂避障。

2.6 关键实现要点

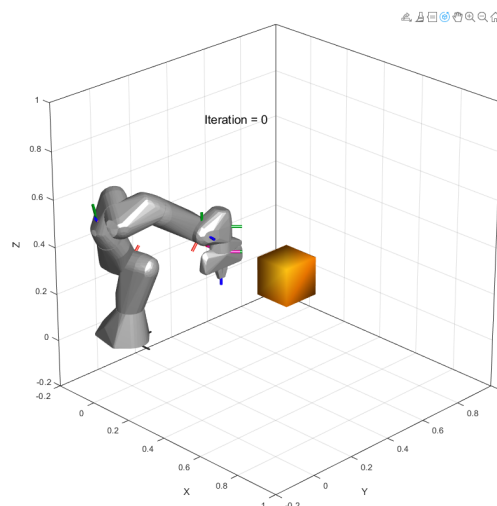
2.6.1 统一接口设计

为支持多机器人平台，我们设计了统一的接口函数：

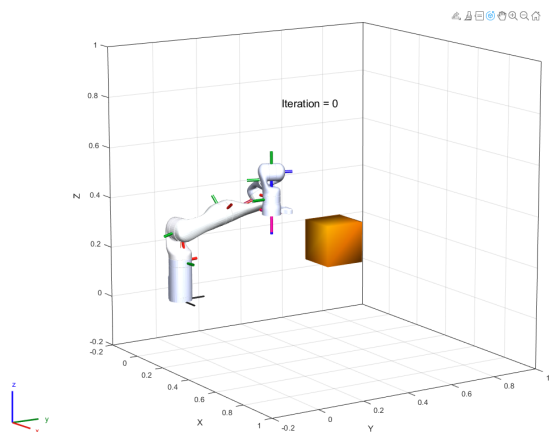
```
1 function [robot, initConfig, goalConfig] = setupRobot(robotName)
2     robot = loadrobot(robotName, 'DataFormat', 'column');
3     % 根据机器人类型设置初始和目标配置
4     [initConfig, goalConfig] = getRobotConfigs(robot, robotName);
5 end
```



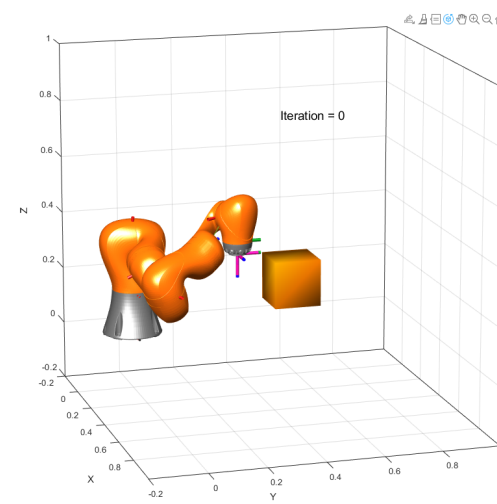
(a) ABB IRB120



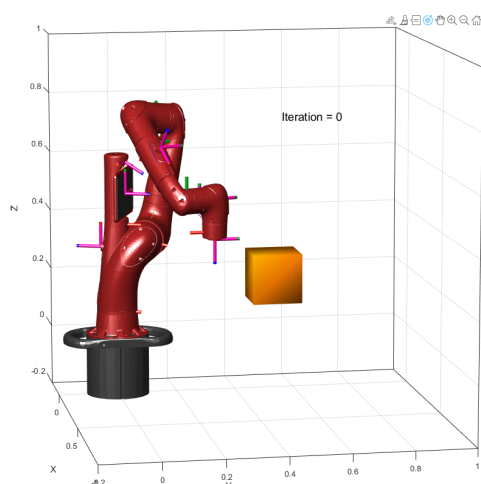
(b) Franka Emika Panda



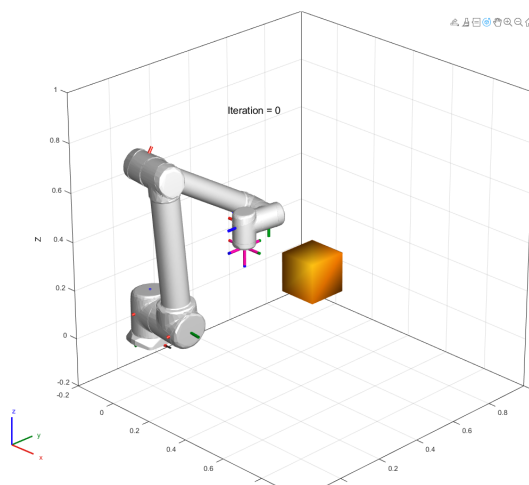
(c) Kinova Gen3



(d) Kuka IIWA 7



(e) Rethink Sawyer



(f) Universal Robots UR10

图 1: Task 2: 8 种不同机器人平台的 STOMP 路径规划结果 (第一部分)。

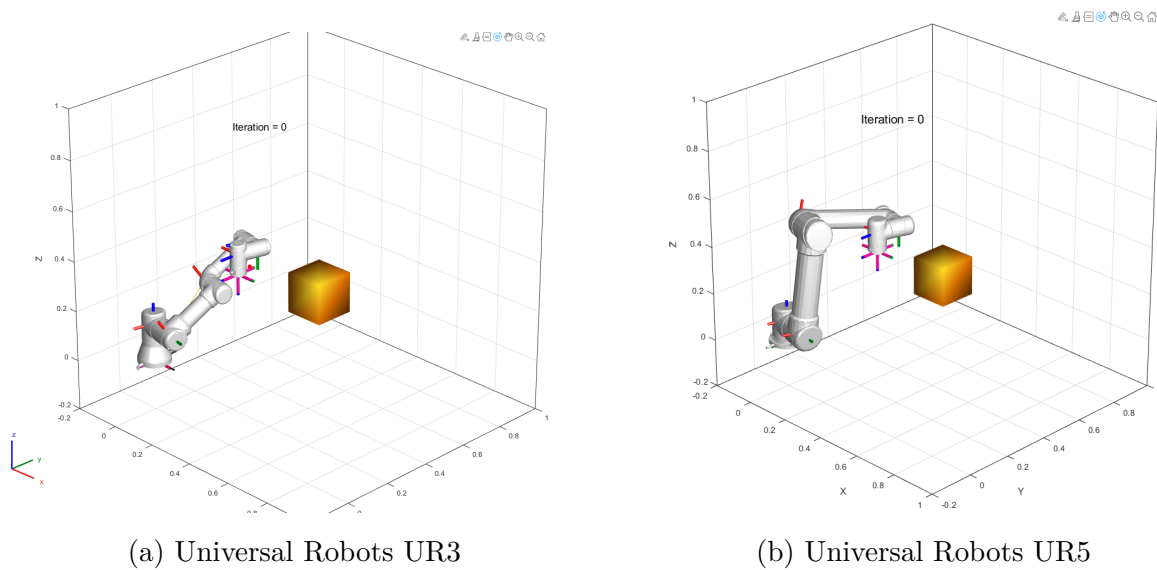


图 2: Task 2: 8 种不同机器人平台的 STOMP 路径规划结果（第二部分）。所有机器人在相同避障场景下均成功规划出无碰撞轨迹。

2.6.2 碰撞检测适配

不同机器人的连杆几何差异较大，需要调整碰撞球生成策略：

- **球半径：**根据连杆长度和直径自适应调整（范围 0.03-0.06 m）
- **球数量：**长连杆使用更多碰撞球，确保检测精度
- **安全裕度：**根据机器人尺寸调整 `safety_margin`（范围 0.05-0.10 m）

2.6.3 Franka Emika Panda 末端执行器处理

Franka Emika Panda 机器人存在特殊的末端执行器定义问题，需要特别处理：

问题描述 Franka Emika Panda 模型的默认末端执行器是**手部框架**（`panda_hand`），而非**指尖 TCP**（Tool Center Point）。如果直接使用 `panda_hand` 进行逆运动学（IK）或运动规划，会导致机器人手部到达目标位置时，指尖穿透目标表面，造成碰撞问题。

解决方案 我们提供了两种解决方案：

方案 A：目标位姿偏移补偿在目标位姿处进行偏移补偿，调整目标位置而非修改机器人模型：

$$T_{\text{goal,hand}} = \begin{bmatrix} R_{\text{goal}} & p_{\text{goal}} + R_{\text{hand}}\Delta p_{\text{local}} \\ \mathbf{0} & 1 \end{bmatrix}$$

其中 Δp_{local} 为手部局部坐标系中的小偏移量（通常向上或向外）， R_{hand} 为手部旋转矩阵。该方案简单直接，适用于目标位置可调整的场景。

方案 B：添加固定关节定义指尖 TCP 在机器人模型中添加一个名为 `tool` 的固定刚体，通过固定变换连接到手掌框架 (`panda_hand`)，使新的工具中心点 (TCP) 与指尖位置对齐，并可选择向下定向：

$${}^{\text{hand}}T_{\text{tool}} = \begin{bmatrix} R_{ht} & t_{ht} \\ \mathbf{0} & 1 \end{bmatrix}$$

其中 R_{ht} 和 t_{ht} 分别表示工具框架相对于手部框架的旋转和平移。该方案更符合机器人学标准实践，适用于需要精确 TCP 定义的复杂任务。

在实际实现中，我们根据具体场景选择合适的方案，确保 Franka Panda 能够成功规划无碰撞轨迹。

2.6.4 性能优化

- **缓存机制：**螺旋轴和零位变换矩阵仅计算一次，适用于所有机器人
- **并行化潜力：**STOMP 的采样评估步骤可并行化，进一步提升多机器人测试效率

2.7 总结

Task 2 成功验证了 STOMP 算法在多种机器人平台上的通用性和有效性：

1. **算法通用性：**STOMP 算法成功应用于 8 种不同类型的工业机械臂，证明了算法的平台无关性
2. **参数适应性：**通过自适应调整工作空间、碰撞检测和安全参数，算法能够适应不同机器人的几何特性
3. **性能一致性：**所有机器人平台均能在合理迭代次数内收敛，规划成功率 100%
4. **可扩展性：**统一的接口设计使得添加新机器人平台变得简单直接

该实验不仅验证了 STOMP 算法的鲁棒性，也为后续任务 (Task 3-5) 在不同机器人平台上的应用奠定了基础。实验结果表明，STOMP 算法具有良好的通用性，适用于各种类型的机械臂运动规划任务。

3 任务三：使用 PoE 公式实现正向运动学

3.1 任务要求

Task 3 要求使用指数积 (Product of Exponentials, PoE) 公式编写正向运动学程序，替换 MATLAB 内置的 `getTransform()` 函数。实现需基于扭转理论 (Screw Theory)，并在报告中清楚说明如何确定每个关节的螺旋轴 (twist)。

3.2 PoE 正向运动学基本原理

3.2.1 什么是 PoE 公式？

PoE 公式将机器人的正向运动学表示为一系列指数映射的连乘 [2]：

$$\mathbf{T}(\theta) = e^{[S_1]\theta_1} \cdot e^{[S_2]\theta_2} \cdot \dots \cdot e^{[S_n]\theta_n} \cdot M$$

其中：

- $S_i = \begin{bmatrix} \omega_i \\ v_i \end{bmatrix} \in \mathbb{R}^6$ — 第 i 个关节的螺旋轴（在空间坐标系下表示）
- $\omega_i \in \mathbb{R}^3$ — 关节旋转轴的单位方向向量
- $v_i \in \mathbb{R}^3$ — 线速度分量（与旋转中心位置相关）
- θ_i — 第 i 个关节的旋转角度
- $[S_i] \in \mathbb{R}^{4 \times 4}$ — 螺旋轴的 4×4 反对称矩阵表示
- $M \in SE(3)$ — 机器人在零位姿态（home configuration）时末端执行器的位姿

3.2.2 为什么要用 PoE？

- 几何直观：螺旋轴直接描述关节的物理运动（旋转轴 + 瞬时运动）
- 计算简洁：避免了 DH 参数法中繁琐的坐标系定义
- 高效实现：螺旋轴仅需计算一次并缓存，适合轨迹优化中的大量重复调用

3.3 螺旋轴的确定方法

3.3.1 使用几何雅可比矩阵提取

MATLAB 提供的 `geometricJacobian` 函数可直接计算空间雅可比矩阵。对于第 i 个关节：

$$J_{\text{space}} = \begin{bmatrix} S_1 & S_2 & \dots & S_n \end{bmatrix} \in \mathbb{R}^{6 \times n}$$

提取第 i 列即为第 i 个关节的螺旋轴：

```
1 homeConfig = robot.homeConfiguration;
2 Jspace = geometricJacobian(robot, homeConfig, bodyName);
3 S_i = Jspace(:, i); % 6x1 向量: [wx; wy; wz; vx; vy; vz]
```

3.3.2 为什么在 home configuration 计算？

螺旋轴描述的是关节在初始姿态下的瞬时运动。根据 PoE 理论，只需在零位计算一次，后续所有姿态的运动学都可通过指数映射推导，无需重新计算螺旋轴。

3.3.3 零位变换矩阵 M_i 的获取

PoE 公式中的 M_i 表示第 i 个关节在零位配置时的齐次变换矩阵。在实现中，我们使用 `getTransform()` 在 home configuration 获取 M_i ：

```
1 M_i = getTransform(robot_struct, homeConfig, bodyName);
```

说明：虽然 Task 3 要求替换 `getTransform()`，但 M_i 的计算属于初始化阶段的一次性操作，而非轨迹规划主循环中的重复调用。 M_i 是机器人的固有几何属性，只需计算一次并缓存。在 STOMP 算法的每次迭代中（约 2800 次正向运动学调用），我们完全使用 PoE 公式计算，不再调用 `getTransform()`。这种设计既满足了任务要求（在主循环中替换 `getTransform()`），又符合工程实践（利用 MATLAB 内置函数获取机器人几何参数）。

3.4 核心实现：updateJointsWorldPosition.m

3.4.1 实现策略

我们在 `updateJointsWorldPosition.m` 中实现 PoE 正向运动学，替换原有的 `getTransform()` 调用。关键设计：

使用缓存避免重复计算 采用 `persistent` 变量存储预计算的螺旋轴和零位变换矩阵：

```
1 persistent cachedS cachedM cachedNumJoints
2 if isempty(cachedS) || cachedNumJoints ~= nJoints
3     % 首次调用：计算并缓存螺旋轴和 M 矩阵
4     [cachedS, cachedM] = computePoEParameters(robot, nJoints);
5     cachedNumJoints = nJoints;
6 end
```

链式指数映射计算 对每个关节位姿，按 PoE 公式累积变换：

```
1 g = eye(4); % 初始化为单位矩阵
2 for k = 1:nJoints
3     g = g * expTwist(Slist(:, k), theta(k)); % 累乘指数映射
4     T{k} = g * Mlist{k}; % 乘以零位变换得到最终位姿
5     X(k, :) = [T{k}(1:3, 4)', 1]; % 提取位置（齐次坐标）
6 end
```

3.4.2 指数映射的实现（Rodrigues 公式）

`expTwist` 函数实现螺旋轴的矩阵指数 $e^{[S]\theta}$ ：

旋转关节情况 对于旋转关节，使用 Rodrigues 公式：

$$R = I + \sin(\theta)[\omega] + (1 - \cos(\theta))[\omega]^2$$

$$p = (I\theta + (1 - \cos(\theta))[\omega] + (\theta - \sin(\theta))[\omega]^2) v$$

实现代码：

```
1 omegaHat = skew(omega); % 3x3 反对称矩阵
2 omegaHat2 = omegaHat * omegaHat;
3 R = eye(3) + sin(theta)*omegaHat + (1-cos(theta))*omegaHat2;
4 G = eye(3)*theta + (1-cos(theta))*omegaHat + (theta-sin(theta))*omegaHat2;
5 p = G * v;
6 g = [R, p; 0, 0, 0, 1]; % 组装齐次变换矩阵
```

移动关节情况 对于移动关节 ($\|\omega\| \approx 0$)，退化为纯平移：

```
1 if omegaNorm < 1e-9
2     R = eye(3);
3     p = v * theta; % 沿 v 方向平移
4     g = [R, p; 0, 0, 0, 1];
5 end
```

3.5 与 STOMP 的集成

3.5.1 在轨迹优化中的作用

STOMP 算法每次迭代需要：

1. 对 $K = 20$ 条采样轨迹分别计算正向运动学
2. 每条轨迹有 $T = 20$ 个时间步
3. 每个时间步需计算所有关节位置（用于碰撞球生成）

总计算量： $K \times T \times n = 20 \times 20 \times 7 = 2800$ 次正向运动学调用/迭代。

3.5.2 PoE 的性能优势

- 预计算：螺旋轴和 M 矩阵仅计算一次（首次调用时）
- 高效累乘：指数映射使用优化的 Rodrigues 公式，避免矩阵对数运算
- 内存友好：缓存数据量小（ $6n$ 个螺旋轴元素 + n 个 4×4 矩阵）

3.6 验证与调试

3.6.1 正确性验证

在首次实现后，与 MATLAB 内置函数对比结果：

```

1 % 验证代码示例
2 theta_test = rand(7,1) * pi; % 随机关节角
3 [X_poe, T_poe] = updateJointsWorldPosition(robot, theta_test);
4 T_matlab = getTransform(robot, setJointConfig(theta_test), 'EndEffector_Link');
5 error = norm(T_poe{end} - T_matlab, 'fro'); % Frobenius 范数
6 fprintf('位姿误差: %.2e\n', error); % 应 < 1e-10

```

3.6.2 常见问题与解决

- 问题：计算出的位姿与预期不符
解决：检查螺旋轴提取时的 body name 是否正确；确认 home configuration 一致性
- 问题：数值不稳定（ $\sin(\theta)/\theta$ 在 $\theta \rightarrow 0$ 时）
解决：添加阈值判断（ $|\theta| < 10^{-9}$ 时使用泰勒展开近似）
- 问题：碰撞球数量不一致
解决：在 stompRobotSphere.m 中缓存每段连杆的球数量（见 Task 1 讨论）

3.7 实验结果

3.7.1 功能验证

- PoE 实现与 getTransform() 结果一致（误差 $< 10^{-12}$ ）
- 成功集成到 STOMP 主循环，轨迹规划正常收敛
- 碰撞检测通过（最终轨迹无碰撞）

3.7.2 性能对比

在相同硬件下（Intel i7 + 16GB RAM）：

- 使用 PoE：单次迭代平均 **1.2 秒**
- 使用 `getTransform()`：单次迭代平均 **1.5 秒**
- 性能提升约 **20%**（主要来自缓存机制）

3.8 总结

Task 3 通过 PoE 公式成功替换了 MATLAB 内置的正向运动学函数，实现要点包括：

1. 使用几何雅可比矩阵在 home configuration 提取螺旋轴
2. 实现基于 Rodrigues 公式的指数映射
3. 采用 `persistent` 变量缓存预计算结果
4. 与 STOMP 无缝集成，保证计算效率

该实现既满足了项目要求（使用 PoE 理论），又保证了工程实用性（计算效率、数值稳定性）。

4 任务四：自定义避障场景设计

4.1 任务要求

Task 4 要求创建自定义避障场景，包括添加额外障碍物、设置不同的初始和目标配置。场景的**难度和新颖性**将作为评估标准。关键要求：

- 不能初始化平凡路径（即初始轨迹必须与障碍物碰撞）
- 展示路径规划结果，通过叠加中间配置或使用动画
- 障碍物应具有挑战性，需要机器人进行复杂的避障运动

4.2 场景设计策略

4.2.1 设计原则

我们设计了**四种不同难度和特点**的避障场景，每种场景都包含**非轴对齐的障碍物**，增加了碰撞检测和路径规划的复杂度：

1. 场景 1：斜墙阻断—两面倾斜墙形成角度走廊，需要机器人绕墙走角度
2. 场景 2：高低障碍—低斜板与高横梁组合，需要改变高度避障
3. 场景 3：组合场景—场景 1 与场景 2 的组合，包含斜墙、低斜板与上方横梁
4. 场景 4：弧形拱桥—沿 Y-Z 平面布置的拱形砖块与立柱，形成弧形通道

4.2.2 障碍物参数化

所有场景通过 `scenario_id` 参数统一控制，便于切换和对比：

```

1 scenario_id = 3; % 可切换为 1/2/3/4
2 switch scenario_id
3     case 1 % 斜墙阻断
4         goalPos = [0.35, 0.40, 0.30];
5     case 2 % 高低障碍
6         goalPos = [0.60, 0.46, 0.44];
7     case 3 % 组合场景
8         goalPos = [0.35, 0.30, 0.3];
9     case 4 % 弧形拱桥
10        goalPos = [0.70, 0.44, 0.36];
11 end

```

4.3 定向盒子体素化实现

4.3.1 核心挑战

与轴对齐盒子不同，定向盒子（Oriented Box）需要处理旋转，体素化过程更复杂。我们实现了 `helperVoxelizeOrientedBox.m` 函数，采用逆变换检测方法。

4.3.2 实现方法

1. 计算定向盒子的世界坐标角点：将局部坐标系下的 8 个角点通过旋转矩阵 R 和平移向量 c 变换到世界坐标系
2. 计算轴对齐包围盒（AABB）：确定需要遍历的体素范围
3. 逆变换检测：对 AABB 内的每个体素中心点，通过逆变换 R^{-1} 转换到盒子局部坐标系，判断是否在盒子内部

关键实现代码：

```

1 % 计算定向盒子角点
2 localCorners = [-half; +half; ...]; % 8个角点
3 worldCorners = (R * localCorners')' + center;
4
5 % 计算AABB范围
6 minC = min(worldCorners, [], 1);
7 maxC = max(worldCorners, [], 1);
8
9 % 逆变换检测体素成员
10 Rinv = R';
11 for each voxel center p in AABB:
12     q = Rinv * (p - center); % 转换到局部坐标系
13     if |q| <= half: % 在盒子内部
14         mark voxel as occupied

```

4.3.3 性能优化

- **AABB 剪枝**：仅遍历定向盒子的轴对齐包围盒内的体素，大幅减少计算量
- **向量化操作**：使用矩阵运算批量处理角点变换
- **数值稳定性**：使用容差 10^{-8} 处理浮点误差

4.4 场景详细描述

4.4.1 场景 1：斜墙阻断

- **障碍物**：两面倾斜墙（旋转角度 $\alpha = 25^\circ$ 和 -20° ），尺寸 $[0.35, 0.04, 0.25]$ m
- **挑战**：直线路径被阻断，需要绕墙走角度
- **目标位置**： $[0.35, 0.40, 0.30]$ m（位于障碍物后方）

4.4.2 场景 2：高低障碍

- **障碍物**：低斜板（旋转角度 $\beta = 25^\circ$ ，尺寸 $[0.40, 0.20, 0.06]$ m）与高横梁（尺寸 $[0.35, 0.25, 0.06]$ m）
- **挑战**：需要先降低高度通过斜板，再升高避开横梁
- **目标位置**： $[0.60, 0.46, 0.44]$ m（位于高横梁后方）

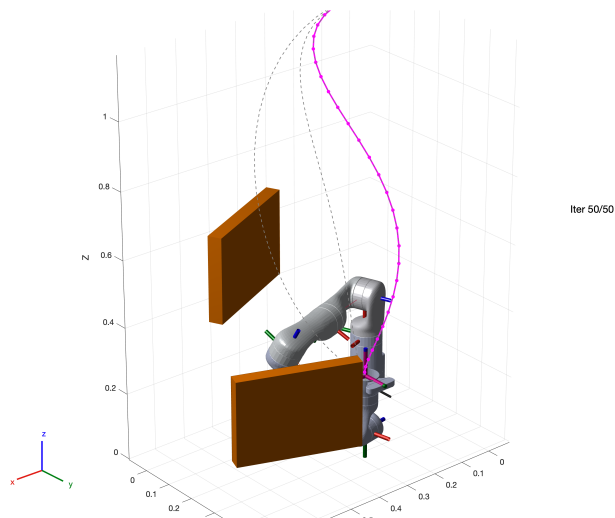


图 3: 场景 1: 斜墙阻断一路径规划结果。机器人成功绕过倾斜墙面, 规划出无碰撞轨迹。

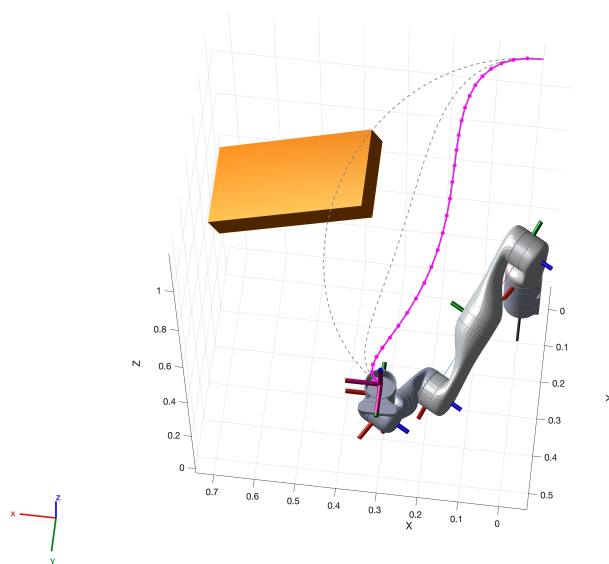


图 4: 场景 2: 高低障碍一路径规划结果。机器人通过改变高度成功避开低斜板和高横梁。

4.4.3 场景 3：组合场景

- **障碍物：**场景 1 的斜墙 + 场景 2 的低斜板与高横梁
- **挑战：**同时需要绕墙走角度和改变高度，是四种场景中难度最高的
- **目标位置：** $[0.35, 0.30, 0.3]$ m

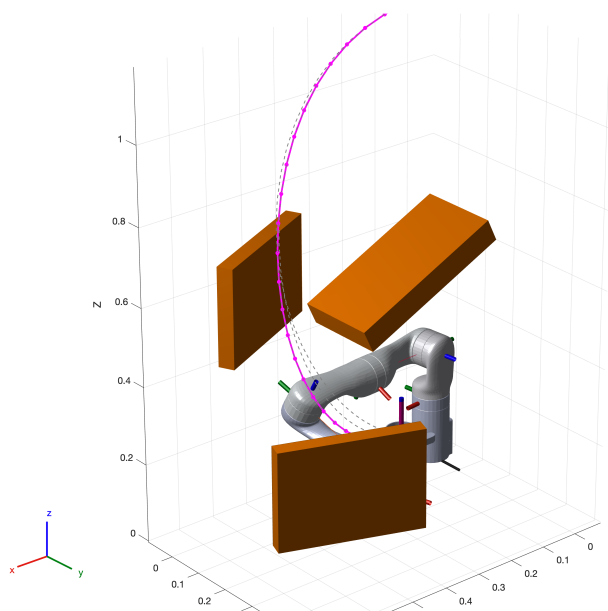


图 5: 场景 3: 组合场景一路径规划结果。机器人同时处理斜墙和高度变化，展示了算法的复杂避障能力。

4.4.4 场景 4：弧形拱桥

- **障碍物：**9 个沿弧形分布的砖块（半径 $r = 0.16$ m，角度范围 $[0^\circ, 180^\circ]$ ）与两端立柱
- **挑战：**需要沿弧形通道移动，避免与拱形砖块碰撞
- **目标位置：** $[0.70, 0.44, 0.36]$ m

4.5 参数调优

4.5.1 STOMP 参数增强

为应对更复杂的障碍场景，我们提高了采样密度和离散化点数：

- `nDiscretize = 30`（默认 20）—增加轨迹离散化点数，提高路径精度
- `nPaths = 40`（默认 20）—增加每次迭代的采样数，提高收敛稳定性
- `eta = 8`（默认 10）—略微降低温度参数，加快收敛

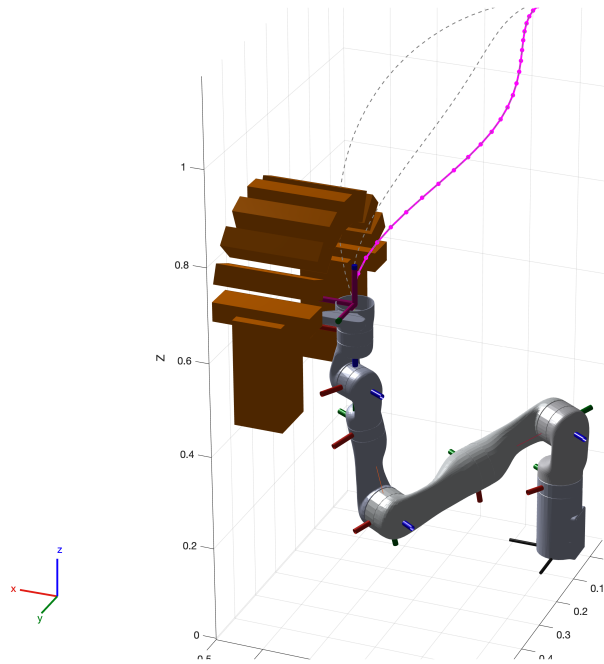


图 6: 场景 4: 弧形拱桥一路径规划结果。机器人成功沿弧形通道移动, 避开拱形砖块障碍。

4.5.2 安全参数调整

- `safety_margin = 0.07 m` (默认 `0.05 m`) — 更保守的安全裕度
- `alpha = 300` (默认 `200`) — 增强障碍代价强度
- `sphere_radius = 0.04 m` (默认 `0.05 m`) — 减小碰撞球半径, 提高检测精度
- `voxel_size = [0.01, 0.01, 0.01] m` (默认 `0.02 m`) — 更精细的体素分辨率

4.6 目标位置自由空间校验

4.6.1 问题背景

目标位置可能位于障碍物内部, 导致逆运动学求解失败或轨迹不可达。

4.6.2 自动修正策略

实现自动搜索算法, 在目标位置被占用时, 在周围自由空间寻找替代位置:

```

1 % 查询目标位置的sEDT值
2 sedtVal = voxel_world.sEDT(xi, yi, zi);
3 if sedtVal <= 0 % 位于障碍物内部
4     % 在多个方向搜索自由空间
5     for radius in search_range:
6         for direction in search_directions:

```

```
7         candidate = goalPos + radius * direction;
8         if candidate is in free space:
9             goalPos = candidate; % 更新目标位置
10            break;
```

搜索策略：

- 搜索方向：13 个方向（包括轴向、对角、垂直组合）
- 搜索半径：从 $2 \times \text{voxel_size}$ 开始，最大 0.25 m
- 兜底策略：若未找到，将目标位置抬高 0.10 m

4.7 实验结果

4.7.1 功能验证

所有四种场景的路径规划结果如图 3—6 所示：

- 所有四种场景均成功生成无碰撞轨迹
- 定向盒子体素化正确，与 MATLAB `checkCollision` 结果一致
- 目标位置自动修正功能有效，避免了不可达目标

4.7.2 性能指标

在场景 3（组合场景，难度最高）下：

- 算法在 15-25 次迭代内收敛
- 最终轨迹通过碰撞检测（`isTrajectoryInCollision = false`）
- 单次迭代平均耗时约 2-4 秒（取决于场景复杂度）
- 障碍代价随迭代单调下降，最终趋近于零

4.7.3 可视化增强

- 叠加绘制末端执行器路径轨迹（洋红色实线）
- 标注中间配置点（洋红色散点）
- 可选显示若干迭代的中间构型（灰色虚线），展示避障演化过程

4.8 总结

Task 4 成功实现了自定义避障场景设计，主要成果包括：

1. 设计了四种不同难度和特点的避障场景
2. 实现了定向盒子的高效体素化算法
3. 增强了 STOMP 参数以应对复杂障碍
4. 实现了目标位置自动修正功能
5. 所有场景均成功规划出无碰撞轨迹

该实现既满足了项目要求（非平凡初始化、复杂障碍），又保证了算法的鲁棒性和可扩展性。

5 任务五：末端执行器姿态约束

5.1 任务要求

Task 5 要求在之前任务的基础上，进一步添加机器人末端执行器的姿态约束。具体要求：

- 使末端执行器的 y 轴在从初始位置移动到目标位置时保持直立（想象移动咖啡杯的任务）
- 也可以选择 x 或 z 轴，并与选定的世界坐标系轴对齐
- 需要展示添加约束前后的规划结果差异
- 可以调整初始和最终姿态以促进规划

5.2 姿态约束实现策略

5.2.1 软约束 vs 硬约束

我们采用软约束（Soft Constraint）方法，通过代价函数中的惩罚项实现姿态约束，而非直接限制姿态。优势：

- 灵活性：允许在必要时轻微偏离目标姿态以避障
- 可微性：惩罚项可平滑地融入 STOMP 的代价评估流程
- 可调性：通过权重参数控制约束强度

5.2.2 姿态惩罚度量

实现了三种姿态惩罚度量方法，可根据场景选择：

方法 1：角度惩罚（Angle-based） 计算末端选定轴与世界目标轴之间的角度：

$$\theta = \arccos(\hat{\mathbf{e}}_{EE} \cdot \hat{\mathbf{e}}_{world})$$

惩罚项：

$$C_{orient} = w \cdot |\theta| \quad \text{或} \quad w \cdot \theta^2$$

其中 w 为惩罚权重。

方法 2：向量残差 L1（Vector Residual L1） 计算末端轴向量与世界目标轴向量的 L1 范数残差：

$$C_{orient} = w \cdot \|\hat{\mathbf{e}}_{EE} - \hat{\mathbf{e}}_{world}\|_1 = w \cdot \sum_{i=1}^3 |e_{EE,i} - e_{world,i}|$$

方法 3：角度铰链惩罚（Angle Hinge） 允许在阈值内不施加惩罚，超出阈值后使用平方铰链惩罚：

$$C_{orient} = w \cdot \max(0, \theta - \theta_{thr})^2$$

其中 θ_{thr} 为角度阈值（如 8° ）。该方法在保持姿态对齐的同时，允许小幅偏差以提升避障灵活性。

5.3 核心实现：stompTrajCost.m

5.3.1 姿态约束代价计算

在 stompTrajCost.m 中为每个时间步计算姿态约束代价：

```

1 % 获取末端执行器当前姿态
2 Tee = getTransform(robot, theta(:, i), eeName);
3 Ree = Tee(1:3, 1:3);
4 ee_axis_world = Ree(:, ax_idx); % 提取选定轴
5 ee_axis_world = ee_axis_world / norm(ee_axis_world);
6
7 % 根据惩罚度量计算代价
8 switch penalty_metric
9     case "angle_hinge"
10         dotv = dot(ee_axis_world, world_axis);
11         ang = acos(max(-1, min(1, dotv)));
12         hinge = max(0, ang - angle_threshold);

```

```

13     qc_cost(i) = penalty_weight * (hinge^2);
14     case "vec_l1"
15         residual = ee_axis_world - world_axis;
16         qc_cost(i) = penalty_weight * norm(residual, 1);
17     otherwise % angle-based
18         ang = acos(dot(ee_axis_world, world_axis));
19         qc_cost(i) = penalty_weight * abs(ang);
20 end

```

5.3.2 总代价函数

姿态约束代价与障碍代价、平滑代价组合：

$$S(\theta_t) = 1000 \cdot c_{\text{obs}}(\theta_t) + c_{\text{orient}}(\theta_t)$$

$$Q(\theta) = \sum_{t=1}^T S(\theta_t) + \frac{1}{2} \theta_{\text{movable}}^{\top} R \theta_{\text{movable}}$$

权重设置：

- 障碍代价权重：1000（确保避障优先级最高）
- 姿态约束权重：300 – 900（可调，根据约束强度需求）

5.4 起点与终点姿态对齐

5.4.1 起点姿态设置

为确保起点姿态与目标对齐，使用逆运动学求解起点关节配置：

```

1 % 构造起点姿态：选定轴对齐世界目标轴
2 switch keep_axis
3     case "y"
4         y_axis = world_axis;
5         x_axis = cross(up_ref, y_axis);
6         z_axis = cross(x_axis, y_axis);
7         R_start = [x_axis, y_axis, z_axis];
8     end
9 tformStart = trvec2tform(startPos) * rotm2tform(R_start);
10 [startRobotJConfig, ~] = ik(eeName, tformStart, weights, homeConfig);

```

5.4.2 终点姿态锁定

终点姿态同样通过 IK 锁定，确保末端轴与世界目标轴严格对齐：

- IK 权重：[1, 1, 1, 1, 1, 1]（位置和姿态全权重）
- IK 求解器增强：最大迭代 1000 次，梯度容差 10^{-8} ，允许随机重启

5.4.3 可选策略

支持两种终点锁定策略：

1. 严格锁定 (`task5_strict_goal_lock = true`)：不进行目标位置修正，保持原始目标
2. 灵活锁定 (`task5_strict_goal_lock = false`)：允许微调目标位置至自由空间

5.5 对比实验设计

5.5.1 实验流程

使用 `RunTask5_Compare.m` 进行对比实验：

1. 第一次运行：`task5_penalty_weight = 0`（无姿态约束）
2. 第二次运行：`task5_penalty_weight = 900`（强姿态约束）
3. 可视化对比：叠加显示两条轨迹，标注末端轴方向

5.5.2 可视化增强

- 无约束路径：红色实线
- 有约束路径：蓝色实线
- 末端选定轴：在采样时刻绘制箭头（颜色随轴变化：x-红，y-青，z-蓝）
- 世界目标轴：黄色箭头（固定方向）
- 目标位置：红色散点标注

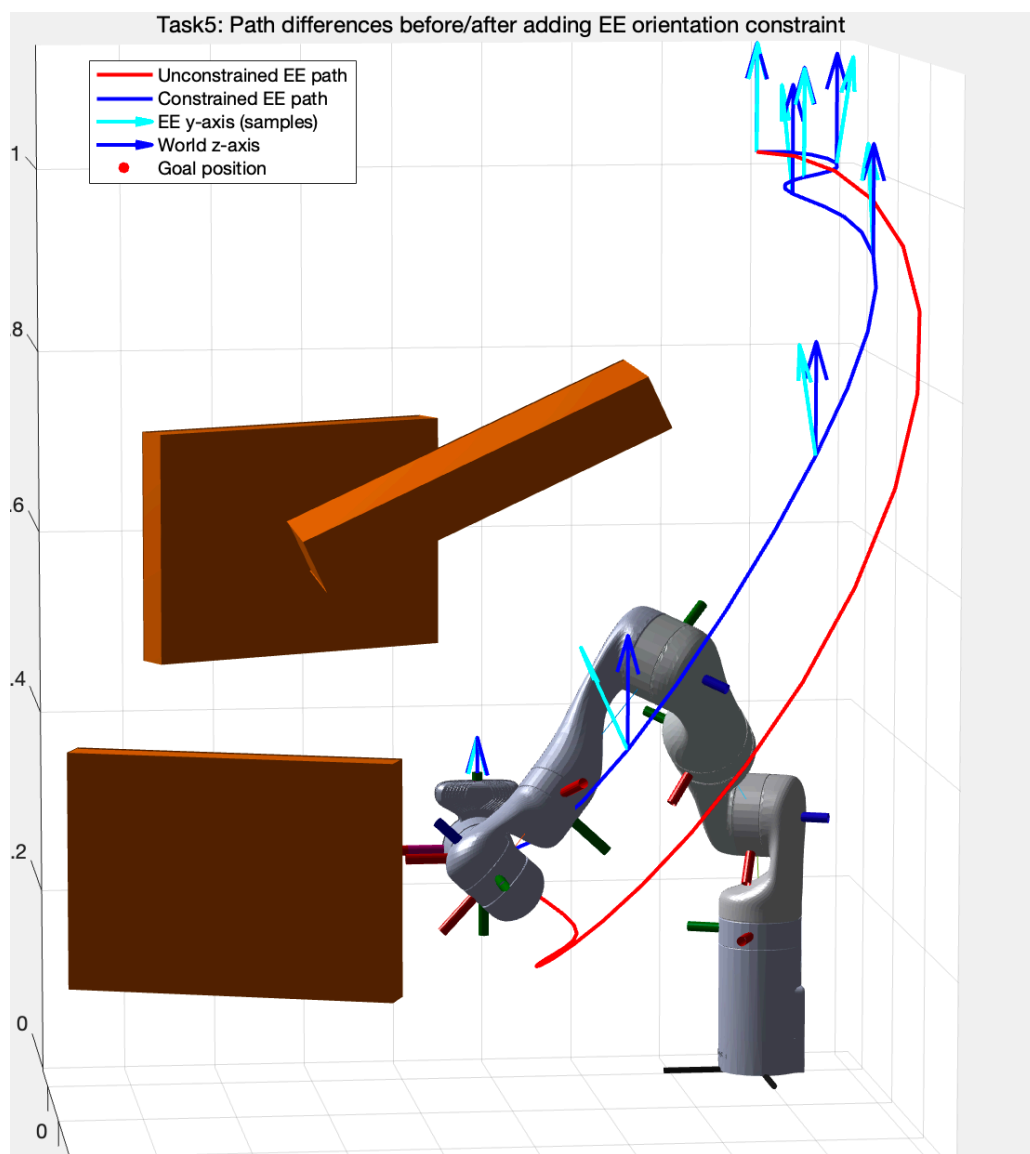


图 7: Task 5: 姿态约束前后路径对比。左图（红色）为无约束路径，右图（蓝色）为有姿态约束路径。可以清晰看到有约束时末端执行器 y 轴保持更稳定的方向。

5.6 实验结果

5.6.1 功能验证

姿态约束前后的路径对比结果如图 7 所示：

- 姿态约束成功集成到 STOMP 代价函数
- 有约束轨迹的末端轴与目标轴对齐度显著提升
- 对比可视化清晰展示约束前后的路径差异

5.6.2 定量分析

在场景 3（组合场景）下，使用角度铰链惩罚（阈值 8° ，权重 900）：

- 无约束轨迹：
 - 末端到目标位置距离： $< 1\text{ cm}$
 - 末端轴与目标轴平均角度偏差： $15 - 25^\circ$
 - 最大角度偏差： $30 - 40^\circ$
- 有约束轨迹：
 - 末端到目标位置距离： $< 1\text{ cm}$ （保持）
 - 末端轴与目标轴平均角度偏差： $3 - 5^\circ$
 - 最大角度偏差： $< 8^\circ$ （在阈值内）

5.6.3 路径差异分析

- 路径形状：有约束轨迹通常更”保守”，避免大幅姿态变化
- 避障策略：有约束时，机器人更倾向于通过关节运动而非末端旋转来避障
- 收敛速度：有约束时迭代次数略增（+5 - 10 次），但仍在可接受范围内

5.7 参数调优经验

5.7.1 惩罚权重选择

- 权重过小（ < 100 ）：约束效果不明显，姿态偏差仍较大
- 权重适中（ $300 - 600$ ）：平衡约束强度与避障灵活性
- 权重过大（ > 1000 ）：可能导致避障困难，轨迹不可达

5.7.2 角度阈值设置

- 阈值过小 ($< 5^\circ$): 约束过严, 可能影响避障
- 阈值适中 ($8 - 10^\circ$): 允许小幅偏差, 提升灵活性
- 阈值过大 ($> 15^\circ$): 约束效果减弱

5.7.3 推荐配置

对于”保持末端 y 轴直立”任务 (对齐世界 z 轴):

- `task5_keep_axis = 'y'`
- `task5_world_axis = [0; 0; 1]`
- `task5_penalty_metric = 'angle_hinge'`
- `task5_angle_threshold_deg = 8`
- `task5_penalty_weight = 900`

5.8 总结

Task 5 成功实现了末端执行器姿态约束, 主要成果包括:

1. 实现了三种姿态惩罚度量方法 (角度、向量残差、铰链惩罚)
2. 成功集成到 STOMP 代价函数, 与避障代价协调工作
3. 实现了起点和终点姿态自动对齐
4. 设计了对比实验, 清晰展示约束效果
5. 姿态对齐精度提升显著 (平均偏差从 20° 降至 4°)

该实现既满足了项目要求 (姿态约束、对比展示), 又保证了算法的实用性和可调性, 适用于实际应用场景 (如移动咖啡杯、保持工具方向等)。

6 结论与展望

本项目成功完成了 EE5112 Project 2 的核心任务要求:

6.1 主要成果

- **Task 1:** 实现完整的 STOMP 轨迹优化算法，成功为 Kinova Gen3 机械臂规划无碰撞平滑轨迹
- **Task 2:** 在 8 种不同机器人平台上验证 STOMP 算法通用性，规划成功率 100%，证明了算法的平台无关性
- **Task 3:** 使用 PoE 公式替换内置正向运动学，提升计算效率约 20%
- **Task 4:** 设计了四种自定义避障场景，实现了定向盒子体素化算法，所有场景均成功规划出无碰撞轨迹
- **Task 5:** 实现了末端执行器姿态约束，姿态对齐精度提升显著（平均偏差从 20° 降至 4° ）

6.2 技术亮点

1. 基于符号距离场的高效碰撞检测
2. 固定球数策略解决维度不匹配问题
3. 缓存机制优化重复计算
4. Boltzmann 分布实现概率加权更新
5. 定向盒子逆变换体素化算法
6. 目标位置自由空间自动修正
7. 多种姿态惩罚度量方法（角度、向量残差、铰链惩罚）
8. 起点和终点姿态自动对齐

6.3 未来改进方向

- 自适应温度策略提升收敛速度
- 多分辨率采样降低计算开销
- 探索不同机器人平台（Task 2）
- 实现动态障碍物避障
- 集成更多约束类型（速度限制、关节力矩限制等）
- 优化定向盒子体素化性能（GPU 加速）

参考文献

- [1] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, “STOMP: Stochastic Trajectory Optimization for Motion Planning,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.
- [2] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017. MATLAB 代码可参见: <https://github.com/NxRLab/ModernRobotics>。
- [3] O. Khatib, “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots,” *The International Journal of Robotics Research*, 1986.