机器人技术与实践

实验 2：SCARA 机械臂运动仿真

姓名: 徐屹寒

时间: 2025/10

目录

实验目的

本实验旨在通过 CoppeliaSim 机器人仿真软件实现 SCARA 机械臂的运动学仿真,主要目标包括:

1. 将实验 1 中建立的 SCARA 机械臂简化模型导入 CoppeliaSim 仿真软件
2. 在 CoppeliaSim 中定义机械臂的各项控制参数
3. 编写控制脚本实现 SCARA 机械臂的运动控制
4. 图形化显示 SCARA 机械臂的运动轨迹和关节速度

实验原理

2.1. CoppeliaSim 仿真平台

CoppeliaSim 是一款功能强大的机器人仿真软件,本实验使用 CoppeliaSim 进行 SCARA 机械臂的运动学仿真。

2.2. 运动学模式

CoppeliaSim 提供两种关节控制模式:

模式	特点	适用场景
Kinematics Mode	纯运动学计算,无动力学效应	轨迹规划、运动学仿真
Dynamic Mode	考虑力、力矩、惯性等	动力学分析、碰撞仿真

本实验采用 Kinematics Mode,避免动力学计算带来的不稳定性,专注于运动控制和轨迹规划。

实验内容

3.1. 模型导入

3.1.1. STL 网格文件导入

将实验 1 中建立的 SCARA 机械臂各零件 STL 文件导入 CoppeliaSim。导入步骤如下:

Important

STL 导入关键步骤:

1. **File → Import → Mesh** 选择 STL 文件
2. 取消 **Auto Scaling** 选项
3. 设置缩放因子为 **0.001**

4. 逐个零件独立导入

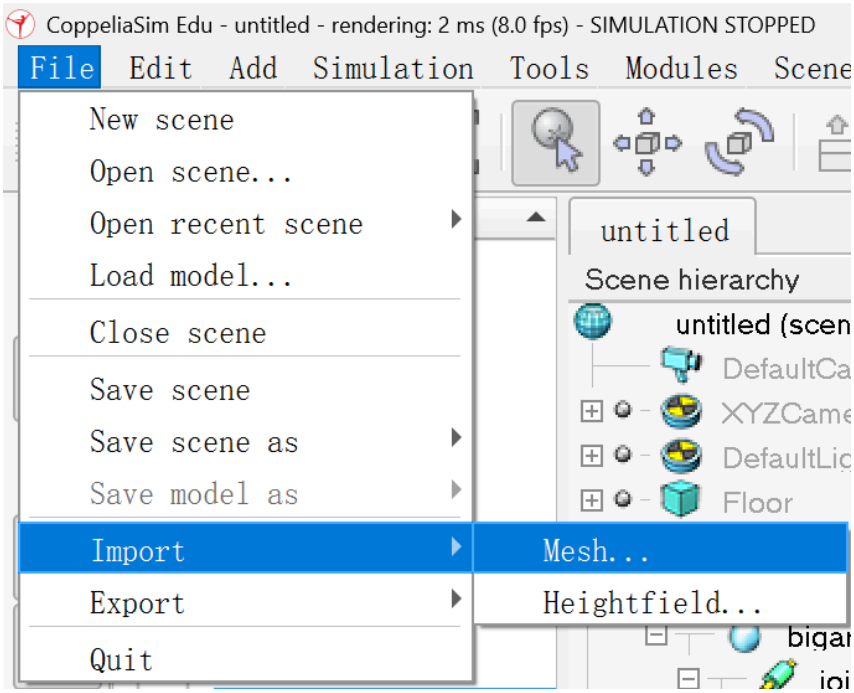

Figure 1: STL 文件导入

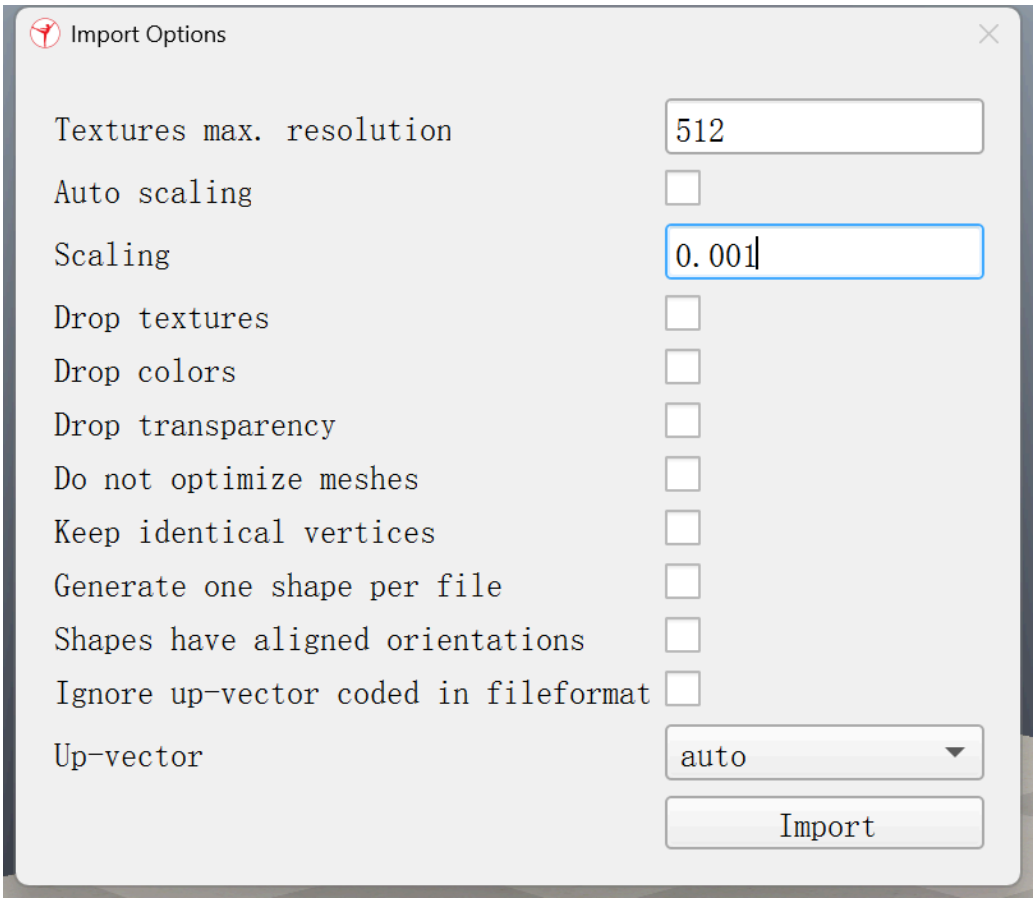

Figure 2: 导入设置

Note

导入时务必注意单位转换,CoppeliaSim 使用米(m)作为基准单位,若 CAD 模型以毫米(mm)为单位,缩放因子必须设为 0.001。

3.1.2. 坐标系与姿态调整

导入后需要对每个零件进行坐标系和姿态的规范化处理:

Important

坐标系调整步骤**:**

1. 右键 → **Edit** → **shape reference frame** → **relocate to mesh center**，重定向局部坐标系至模型质心
2. 姿态归零处理
   - 先将 Position (X, Y, Z)全部置零
   - 再将 Orientation (α, β, γ)全部置零
   - 最后统一调整各零件的相对位置

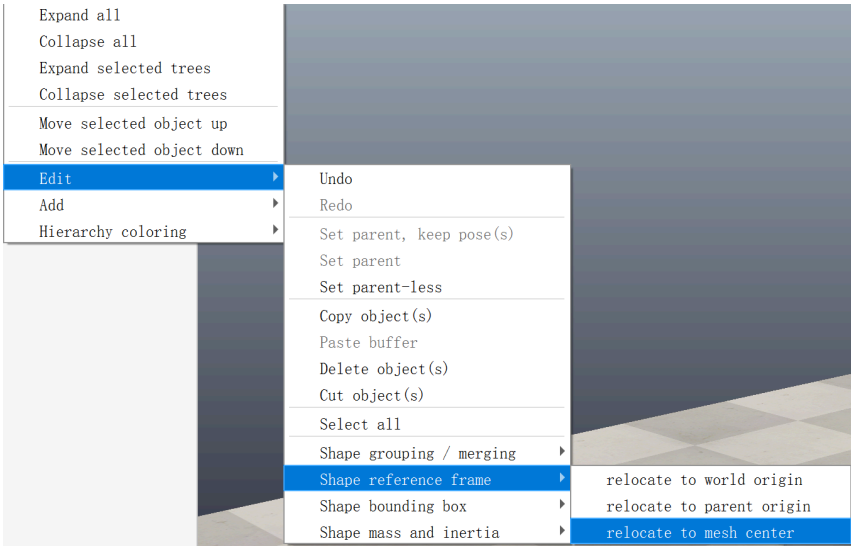

Figure 3: 重定向坐标系

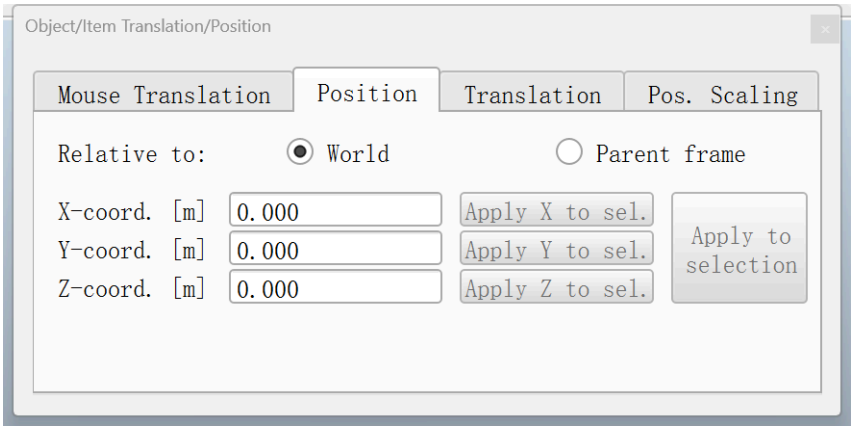

Figure 4: Position 归零

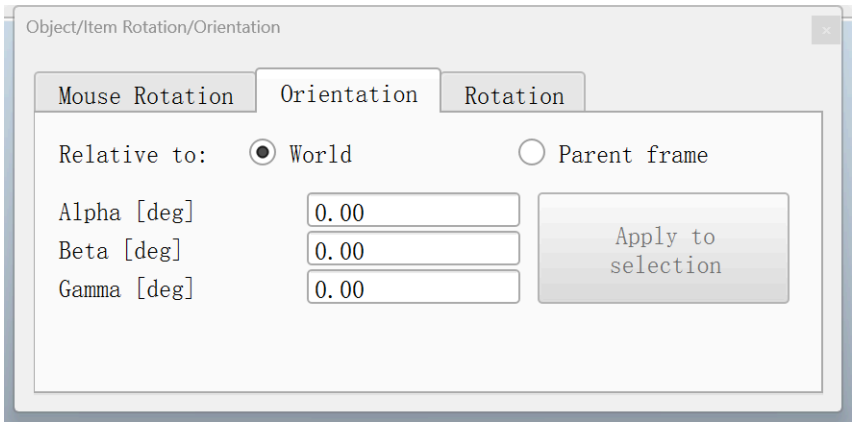

Figure 5: Orientation 归零

3.2. 机械结构组装

完成各零件的导入和姿态调整后,需要通过关节将它们连接成完整的机械臂系统。

3.2.1. 添加关节

Important

关节添加步骤:

1. 选择关节类型
   - **Revolute Joint**: 旋转关节(joint1、joint2、joint4)
   - **Prismatic Joint**: 直线移动关节(joint3)
2. 添加关节对象
3. 调整关节位置
4. 设置关节限位

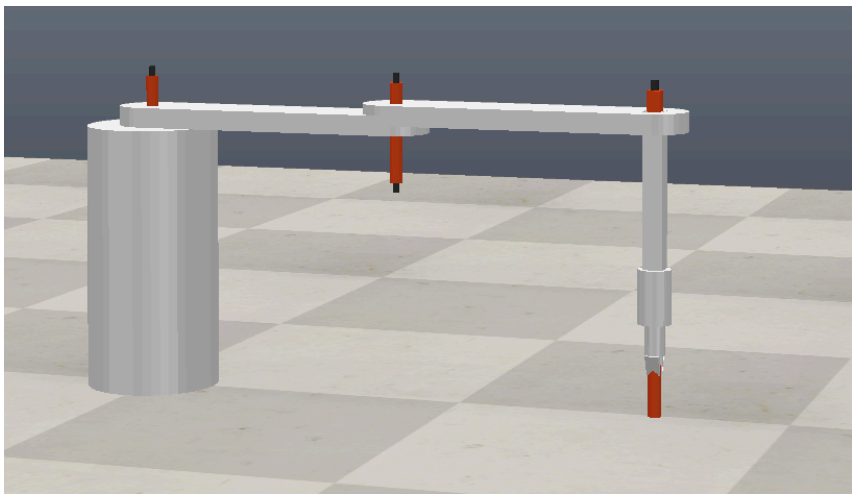

Figure 6: 机械臂装配图

3.2.2. 建立层级结构

CoppeliaSim 使用树状结构管理对象的父子关系。

Important

层级结构建立:

1. 拖拽建立父子关系

   在 Scene hierarchy 面板中，将子对象拖拽到父对象下，实现用关节连接机械臂组成部分

2. **SCARA** 结构

```
base
└── joint1
    └── bigarm
        └── joint2
            └── forearm
                └── joint3
                    └── hand
                        └── joint4
                            └── finger
```

3.2.3. 关节模式配置

双击关节打开属性面板,选择 **Kinematics mode**。

3.3. 控制参数定义

在 CoppeliaSim 中定义了以下控制参数和对象:

Important

关节对象:

- joint1: 底座旋转关节
- joint2: 大臂旋转关节
- joint3: 直线移动关节
- joint4: 末端旋转关节

传感器对象:

- finger: 末端执行器位置传感器
- graph: 数据图表显示对象

监测参数:

- 末端位置(X, Y, Z 坐标)
- 各关节速度(joint1/2/3)
- 运动轨迹实时可视化

3.4. 运动控制编程

3.4.1. 关键代码实现

Analysis

1. 初始化 - 获取对象句柄

```
def sysCall_init():
    sim = require('sim')

    # 使用sim.getObject()获取关节句柄
    self.joint1 = sim.getObject('/joint1')
    self.joint2 = sim.getObject('/joint2')
    self.joint3 = sim.getObject('/joint3')
    self.joint4 = sim.getObject('/joint4')

    # 获取末端执行器用于轨迹跟踪
    self.finger = sim.getObject('/finger')
```

- `sim.getObject()`: 通过对象路径获取句柄

Analysis

2. 主控制循环 - 设置关节位置

```
def sysCall_actuation():
    # 获取当前仿真时间作为控制输入
    time = sim.getSimulationTime()

    # 根据时间分段控制关节
    if time < 4:
        # 调用sim.setJointTargetPosition()设置目标角度
        sim.setJointPosition(self.joint1, math.pi * time / 2)
    elif time < 5.5:
        sim.setJointPosition(self.joint2, math.pi * (time - 4) * 0.5)
```

- `sim.getSimulationTime()`: 获取当前仿真时间(秒)
- `sim.setJointPosition()`: 直接设置关节位置(Kinematics 模式)
- `sim.setJointTargetPosition()`: 设置目标位置(Dynamic 模式)

Analysis

3. 传感器读取 - 位置和速度

```
def sysCall_sensing():
    # 获取末端执行器位置
    pos = sim.getObjectPosition(self.finger)

    # 获取关节速度
    vel1 = sim.getJointVelocity(self.joint1)
    vel2 = sim.getJointVelocity(self.joint2)
```

```
    # 更新图表显示
    sim.setGraphStreamValue(self.graph, self.objectPosX, pos[0])
```

- `sim.getObjectPosition()`: 返回[x, y, z]坐标数组
- `sim.getJointVelocity()`: 返回关节速度(rad/s 或 m/s)

详细代码见附录。

运动仿真与数据采集

4.1. 运动空间可视化

通过 finger 对象的位置传感器,实时记录末端执行器在三维空间中的运动轨迹。系统记录了 X、Y、Z 三个方向的位置数据,并在画面中显示。

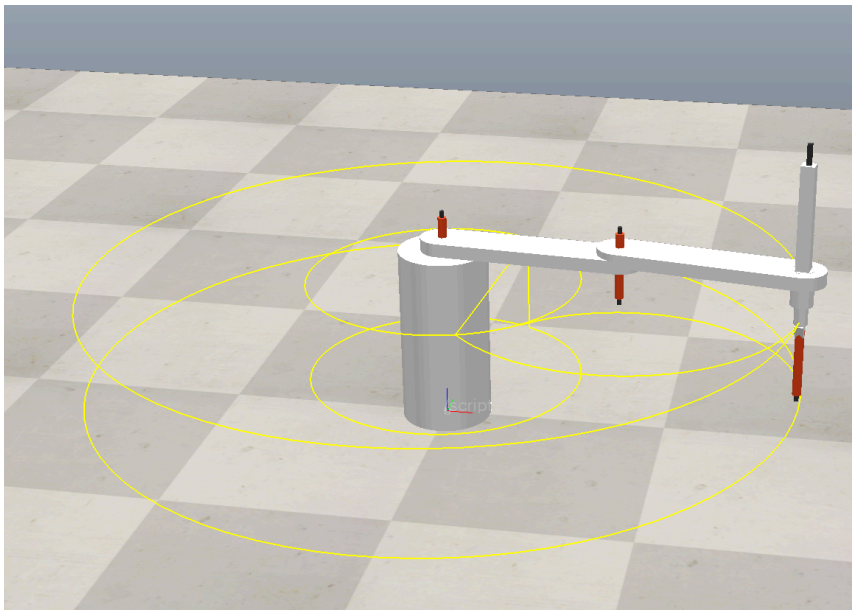

Figure 7: 运动空间可视化

4.2. 关节速度分析

系统实时监测 joint1、joint2、joint3 三个关节的角速度,并转换为度/秒单位显示。通过图表可以观察各关节在不同时间段的速度变化。

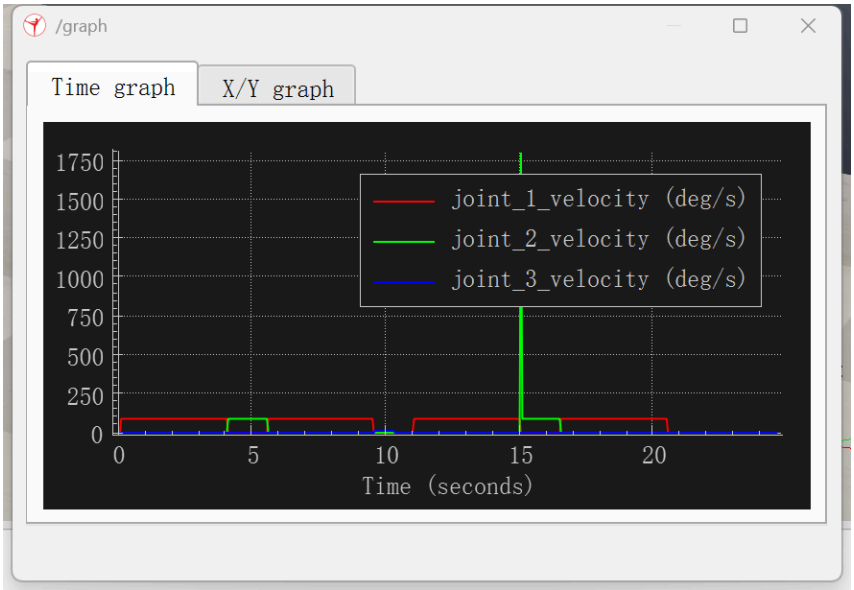

Figure 8: 关节速度图表

实验结果与分析

5.1. 仿真结果

通过 CoppeliaSim 的仿真运行,成功实现了 SCARA 机械臂的运动控制,主要结果如下:

1. 模型导入: SCARA 机械臂模型成功导入 CoppeliaSim,所有关节和连杆正确装配
2. 数据可视化: 末端位置和关节速度数据实时采集并显示。

附录: CoppeliaSim 控制脚本

以下为完整的 SCARA 机械臂控制脚本代码:

File: scara_simulation.py (CoppeliaSim Script)

```
import math
def sysCall_init():

    sim = require('sim')

    self.joint1 = sim.getObject('/joint1')
    self.joint2 = sim.getObject('/joint2')
    self.joint3 = sim.getObject('/joint3')
    self.joint4 = sim.getObject('/joint4')


    self.finger = sim.getObject('/finger')
    self.graph = sim.getObject('/graph')
```

```
    self.objectPosX = sim.addGraphStream(self.graph, 'object pos x', 'm', 1)
    self.objectPosY = sim.addGraphStream(self.graph, 'object pos y', 'm', 1)
    self.objectPosZ = sim.addGraphStream(self.graph, 'object pos z', 'm', 1)

    self.joint1Vel = sim.addGraphStream(self.graph, 'joint 1 velocity', 'deg/s',
0, [1, 0, 0])
    self.joint2Vel = sim.addGraphStream(self.graph, 'joint 2 velocity', 'deg/s',
0, [0, 1, 0])
    self.joint3Vel = sim.addGraphStream(self.graph, 'joint 3 velocity', 'deg/s',
0, [0, 0, 1])

    sim.addGraphCurve(self.graph, 'object pos x/y/z', 3, [self.objectPosX,
self.objectPosY, self.objectPosZ], [0,0,0],'m by m by m')


def sysCall_actuation():
    time = sim.getSimulationTime()

    if time < 4:
        sim.setJointPosition(self.joint1, math.pi * time / 2)
    elif time < 5.5:
        sim.setJointPosition(self.joint2, math.pi * (time - 4) * 0.5)
    elif time < 9.5:
        sim.setJointPosition(self.joint1, math.pi * (time - 5.5) * 0.5)
    elif time < 11:
        sim.setJointPosition(self.joint3, math.pi * (time - 9.5) * 0.1)
    elif time < 15:
        sim.setJointPosition(self.joint1, math.pi * (time - 11) * 0.5)
    elif time < 16.5:
        sim.setJointPosition(self.joint2, math.pi * (time - 16.5) * 0.5)
    elif time < 20.5:
        sim.setJointPosition(self.joint1, math.pi * (time - 16.5) * 0.5)

def sysCall_sensing():

    pos = sim.getObjectPosition(self.finger)

    sim.setGraphStreamValue(self.graph, self.objectPosX, pos[0])
    sim.setGraphStreamValue(self.graph, self.objectPosY, pos[1])
    sim.setGraphStreamValue(self.graph, self.objectPosZ, pos[2])

    sim.setGraphStreamValue(self.graph, self.joint1Vel,
180*sim.getJointVelocity(self.joint1)/math.pi)
    sim.setGraphStreamValue(self.graph, self.joint2Vel,
180*sim.getJointVelocity(self.joint2)/math.pi)
    sim.setGraphStreamValue(self.graph, self.joint3Vel,
180*sim.getJointVelocity(self.joint3)/math.pi)

def sysCall_cleanup():
    # do some clean-up here
```

```
    pass

# See the user manual or the available code snippets for additional callback
functions and details
```