机器人刚度建模

机器人在进行曲面加工时，铣削加工的铣削力数值较大且方向经常变化，机器人各个关节在抵御铣削力时容易发生形变，累积到末端产生变形误差，从而使机器人加工精度下降。因此，如何提高机器人的刚度性能是目前研究的重点。

1 机器人雅可比矩阵

除了关节角度和机器人末端执行器位置之间的关系外，还需要研究关节和末端执行器速度之间的关系。由机器人运动学公式，当给定一组关节角$\boldsymbol{\theta}=[\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6]^{\mathrm{T}}$时，可以确定机器人末端的位置与姿态。

$$\boldsymbol{X}=F\left(\boldsymbol{\theta}\right)$$

其中，$\boldsymbol{X}=[X,Y,Z,A,B,C]^{\mathrm{T}}$，将公式两边进行求导，可以得到：

$$\boldsymbol{V}=\dot{\boldsymbol{X}}=\boldsymbol{J}\left(\boldsymbol{\theta}\right)\dot{\boldsymbol{\theta}}$$

式中，$\boldsymbol{J}\left(\boldsymbol{\theta}\right)$为雅可比矩阵。

一般情况下，雅可比矩阵通过矢量差乘积法、微分变换法得到。而使用指数积公式表示正运动学，可以更加明确、优雅的推导出雅可比矩阵。在机器人正运动学的指数积公式下，末端的速度$[\boldsymbol{V}]$的表达式为：

$$[\boldsymbol{V}]=\dot{\boldsymbol{T}}\boldsymbol{T}^{-1}=\sum_{i=1}^{6}\left(\frac{\partial\boldsymbol{T}}{\partial\boldsymbol{\theta}_i}\dot{\boldsymbol{\theta}}_i\right)\boldsymbol{T}^{-1}=\sum_{i=1}^{6}\left(\frac{\partial\boldsymbol{T}}{\partial\boldsymbol{\theta}_i}\boldsymbol{T}^{-1}\right)\dot{\boldsymbol{\theta}}_i$$

带入机器人正运动学公式，并使用伴随映射将上式写成向量形式，即：

$$\boldsymbol{V}=\underbrace{\boldsymbol{\xi}_1}_{\boldsymbol{J}_1}\dot{\theta}_1+\underbrace{Ad_{e^{\xi_1\dot{\theta}_1}}\left(\boldsymbol{\xi}_2\right)}_{\boldsymbol{J}_2}\dot{\theta}_2+\underbrace{Ad_{e^{\xi_1\dot{\theta}_1}e^{\xi_2\dot{\theta}_2}}\left(\boldsymbol{\xi}_3\right)}_{\boldsymbol{J}_3}\dot{\theta}_3+\cdots$$

$$\boldsymbol{V}=\begin{bmatrix}\boldsymbol{J}_1 & \boldsymbol{J}_2 & \cdots & \boldsymbol{J}_6\end{bmatrix}\begin{bmatrix}\dot{\theta}_1\\\dot{\theta}_2\\\vdots\\\dot{\theta}_6\end{bmatrix}=J\left(\boldsymbol{\theta}\right)\dot{\boldsymbol{\theta}}$$

式中，对于任意$\boldsymbol{\xi}\in\boldsymbol{R}^6$，与$\boldsymbol{T}=(\boldsymbol{R},\boldsymbol{P})\in\boldsymbol{SE}\left(3\right)$相关联的伴随映射为:

$$\boldsymbol{\xi}'=Ad_{\boldsymbol{T}}\left(\boldsymbol{\xi}\right)=\begin{bmatrix}\boldsymbol{R} & 0\\[\boldsymbol{P}]\boldsymbol{R} & \boldsymbol{R}\end{bmatrix}\boldsymbol{\xi}$$

由虚功原理，关节处的功率消耗为机器人运动的功率消耗和末端执行器的功率消耗之和，假设机器人处于静平衡状态，没有用于机器人运动的功率消耗，关节处功率消耗等于末端执行器的功率消耗，用公式表示为：

$$\boldsymbol{\tau}^{\mathrm{T}}\dot{\boldsymbol{\theta}}=\boldsymbol{F}^{\mathrm{T}}\boldsymbol{V}$$

式中，$\boldsymbol{\tau}$为关节力矩的列向量形式，$\dot{\boldsymbol{\theta}}$为关节角速度，**F**为末端所受外力，**V**为末端空间速度，利用公式$\boldsymbol{V}=\boldsymbol{J}\left(\boldsymbol{\theta}\right)\dot{\boldsymbol{\theta}}$可得：

$$\boldsymbol{\tau}=\boldsymbol{J}^{\mathrm{T}}\left(\boldsymbol{\theta}\right)\boldsymbol{F}$$

上式表示的是关节力矩与末端所受外力之间的关系，若一个外力作用在末端器上以平衡各关节力矩，使用上式便可用于计算该力矩，以产生反作用力使机器人处于平衡状态。$\boldsymbol{J}^{\mathrm{T}}$与雅可比矩阵一样，也具有相似的特性。由公式可知，当机器人处于奇异形位时，关节产生的驱动力矩不能在机器人末端产生得到相应的平衡力矩，也就是丧失了对末端力矩的控制，使机器人变得难以控制，在加工过程中应尽量避免。

2 机器人刚度建模

KUKA KR60-3机器人臂架的刚性远大于关节的刚性，因此我们将仅考虑关节柔性对末端的偏移造成的影响，同时将关节的刚度映射至机器人末端的刚度。将各个关节间的驱动装置的刚度用一个弹簧来近似，各关节的受外力矩$\boldsymbol{\tau}$和关节变形$\Delta\boldsymbol{\theta}$的关系可以表示为：

$$\boldsymbol{\tau}=\boldsymbol{K}_{\theta}\boldsymbol{\Delta\theta}$$

式中，K_{θ}为关节刚度矩阵，其对角元素为6个关节的刚度值，

由前文机器人力雅可比矩阵可知外力与关节力矩的关系，带入上述公式为：

$$\boldsymbol{\tau}=\boldsymbol{J}^{\mathrm{T}}\left(\boldsymbol{\theta}\right)\boldsymbol{F}$$

末端变形为：

$$\boldsymbol{\Delta X}=\boldsymbol{J}\boldsymbol{\Delta\theta}$$

根据胡克定律，机器人末端执行器在空间中受到外力发生变形，可以使用公式表示为：

$$\boldsymbol{F}=\boldsymbol{K}\boldsymbol{\Delta X}=\boldsymbol{K}\boldsymbol{J}\boldsymbol{\Delta\theta}$$

式中，$\boldsymbol{F}$为末端执行器所受的空间力-力矩向量，$\boldsymbol{X}$为末端受外力产生的空间位移-转角向量。

将上述公式联合可得：

$$\boldsymbol{K}=\boldsymbol{J}^{-\mathrm{T}}\boldsymbol{K}_{\theta}\boldsymbol{J}^{-1}$$

式中，$\boldsymbol{J}$为雅可比矩阵。

当机器人处于奇异形位，雅可比矩阵不可逆，导致上述公式不成立，尽管知道末端受力也无法预测末端变形情况。因此，在加工过程中需要避免使机器人处于奇异形位。关节刚度矩阵为对角矩阵，由矩阵的运算性质可知，末端笛卡尔刚度矩阵$\boldsymbol{K}$为对称矩阵，其形式如下：

$$\boldsymbol{K}=\begin{bmatrix}k_{11} & k_{12} & \cdots & k_{16}\\ k_{21} & k_{22} & \cdots & k_{26}\\ \vdots & \vdots & \ddots & \vdots\\ k_{61} & k_{61} & \cdots & k_{66}\end{bmatrix}$$

3 刚度性能指标

上述运算涉及雅可比矩阵的求逆，这会导致运算过程中出现误差，尤其是在靠近机器人奇异形位的情况，针对上述问题，定义柔度矩阵为：

$$\boldsymbol{C}=\boldsymbol{K}^{-1}=\boldsymbol{J}\boldsymbol{K}_{\theta}^{-1}\boldsymbol{J}^{\mathrm{T}}$$

由公式可以看出，机器人末端笛卡尔柔度矩阵的计算未涉及到雅可比矩阵的求逆运算。机器人末端变形$\boldsymbol{\Delta X}$和所受外力$\boldsymbol{F}$的关系可以表示为$\boldsymbol{\Delta X}=\boldsymbol{CF}$，用矩阵形式表示为：

$$\begin{bmatrix}\boldsymbol{d}\\ \boldsymbol{\delta}\end{bmatrix}=\begin{bmatrix}\boldsymbol{C}_{fd} & \boldsymbol{C}_{f\delta}\\ \boldsymbol{C}_{md} & \boldsymbol{C}_{m\delta}\end{bmatrix}\begin{bmatrix}\boldsymbol{F}\\ \boldsymbol{M}\end{bmatrix}$$

式中，$\boldsymbol{C}_{fd}$为平移柔度矩阵，$\boldsymbol{C}_{m\delta}$为旋转柔度矩阵，$\boldsymbol{C}_{f\delta}$和$\boldsymbol{C}_{md}$为耦合柔度矩阵。

在实际加工过程中，末端的扭转变形与平移变形相比可以忽略不计，为了简化问题，只考虑末端的平移变形，将6×6的笛卡尔柔度矩阵简化为3×3平移柔度矩阵。

刚度是影响机器人末端变形的重要因素，为了研究刚度对机器人铣削加工变形的影响，需要引入一个刚度性能指标对机器人的刚度进行评价，而柔度矩阵$\boldsymbol{C}$是一个张量，无法直观地反映机器人的刚度性能，通常使用柔度椭球来描述机器人的刚度性能，末端平移变形为：

$$\boldsymbol{d}=\boldsymbol{C}_{fd}\boldsymbol{f}$$

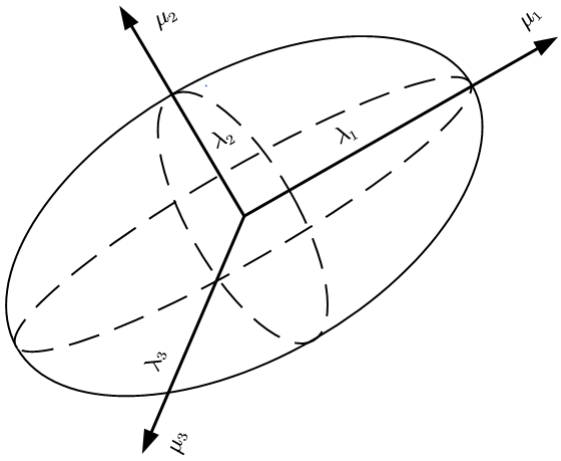
其中，f为末端所受单位力。

设机器人末端受外力的平移变形为单位向量：

$$\|\boldsymbol{d}\|^2=\boldsymbol{d}^{\mathrm{T}}\boldsymbol{d}=\boldsymbol{f}^{\mathrm{T}}{\boldsymbol{C}_{fd}}^{\mathrm{T}}\boldsymbol{C}_{fd}\boldsymbol{f}=1$$

上述公式定义了一个三维笛卡尔柔度椭球，如图所示。半长轴长度为λ_1、λ_2、λ_3的平方根，其中λ_1、λ_2、λ_3为$C_{fd}^TC_{fd}$的特征值。椭球的各个半长轴代表了机器人在各个方向上的刚度性能，该半轴越短，代表在该方向上的受力变形就越小，刚度性能就越优。因此，柔度椭球的体积V与半长轴的乘积成正比，柔度椭球的体积也代表了当前位置的综合刚度性能，将$C_{fd}^TC_{fd}$的特征值。椭球的各个半长轴代表了机器人在各个方向上的刚度性能，该半轴越短，代表在该方向上的受力变形就越小，刚度性能就越的条件数作为刚度性能评价指标k，即：

$$k=\sqrt{\lambda_1\lambda_2\lambda_3}=\sqrt{det\left({\boldsymbol{C}_{fd}}^{\mathrm{T}}\boldsymbol{C}_{fd}\right)}$$



以下为机器人刚度性能指标计算代码：

```
#每个姿态下机器人受沿刀轴方向单位力的末端变形向量
def Stiffness(thetalist):
    Jb =  mr.JacobianBody(Blist, thetalist)
    try:
        A = np.dot(Jb, Jb.T)
        K_theta = np.diag([5.7e7, 6.2e6, 1.2e7, 2.1e6, 2.3e6, 2.3e6])  # 关节刚度矩阵,打单位是N/MM
        StiffnessMatrix = np.linalg.inv(Jb).T @ K_theta @ np.linalg.inv(Jb)  # 笛卡尔刚度矩阵
        transStiffnessMatrix = StiffnessMatrix[3:6, 0:3]  # 平移刚度矩阵
        ComplianceMatrix = Jb @ np.linalg.inv(K_theta) @ Jb.T  # 柔度矩阵
        transComplianceMatrix = ComplianceMatrix[3:6, 0:3]  # 平移柔度矩阵
        A = np.dot( transComplianceMatrix.T,  transComplianceMatrix)
        eigenvalue, featurevector = np.linalg.eig(A)
    except:
```

```
            print('该姿态处于奇异形位！')
        return eigenvalue
```

在实际铣削加工中，加工效果的评价往往是针对整个曲面或者某一个区域，而不是某一个加工点。机器人刚度评价指标是对某一个机器人姿态进行评价，一个位姿的刚度良好并不能代表整个区域的刚度符合要求，因而需要提出一个评价区域的整体刚度的指标，使区域整体符合加工需求。使用加工区域内所有轨迹点刚度指标的统计特征整体刚度指标*H*来描述区域加工效果，整体刚度指标*H*体现的是全局的加工效果，而不是针对某个点，可以使姿态优化更为全面，使整个区域满足优化目标。

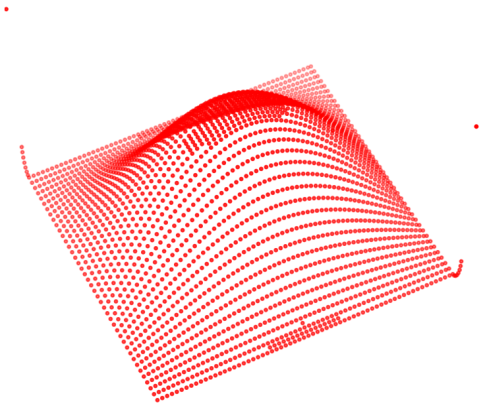
$$\begin{cases} H = r\mu + (1 - \mathrm{r})\sigma \\ \mu = \frac{\sum_{i=1}^{N} k_i}{N} \\ \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(k_i - \mu)^2} \end{cases}$$

其中，k_i为机器人加工第*i*个轨迹时的刚度评价指标，*r*为平均值μ和标准差σ的权重，可以根据加工需求进行调整。轨迹分割

对于给定加工曲面，其相邻的刀位点对应的理想最优姿态都比较相近，因为相邻的刀位点具有接近的位置和姿态，其对应的机器人加工姿态也是相似的。对于相对简单，变化平缓的曲面，可以使用相同的优化参数，而对于曲率大、法向变化频繁的曲面，可以对其进行分割，使用不同参数进行加工，提高加工性能的同时降低计算时间。

基于谱聚类的区域分割方法

首先使用UG软件生成加工轨迹，如图3-12所示。机器人的加工性能与铣削力和加工姿态相关，而铣削力和姿态又由加工轨迹决定。因此，相邻轨迹点的机器人姿态和所受铣削力是接近的，其所对应最优加工参数也应该是相近的。由此，可以使用聚类算法根据加工轨迹点对应的机器人刚度性能将曲面划分为不同的子区域，每个子区域内加工点使用同一组优化参数，提高轨迹优化的效率。



对于待分割的轨迹点，通过机器人运动学逆解，求解刀具轨迹点对应的机器人关节角，计算机器人当前姿态下刚度椭球的各长半轴长度λ_1、λ_2、λ_3，结合轨迹点的位置的刚度性能，定义一个轨迹点对应的六维向量为$V_i = (x_i, y_i, z_i, \lambda_1, \lambda_2, \lambda_3)$，并进行归一化处理。以加工轨迹点作为顶点，构造一个有权无向图$G = (V, E)$，E为边的集合，使用欧氏距离衡量两个顶点V_i之间的相似度，借助高斯核函数计算两个顶点之间的权重w_{ij}：

$$w_{i,j} = e^{\frac{-dist(i,j)}{2\sigma^2}}$$

其中，$dist_{ij}$表示两个顶点之间的距离函数，用于衡量两个相邻顶点之间的几何距离和刚度性能的相似度：

$$dist_{i,j} = |\boldsymbol{V}_i - \boldsymbol{V}_j|$$

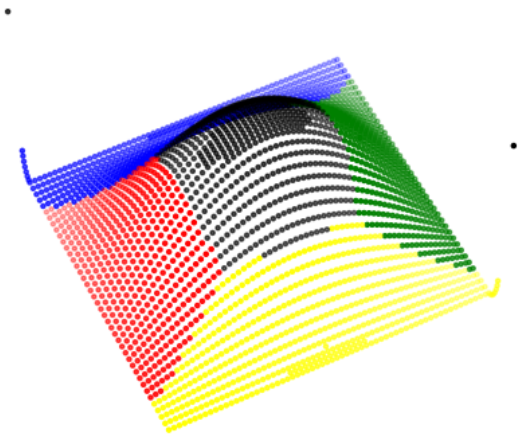
其中，σ是一个比例参数，如果σ太小了，可能会分离本应该属于同一子区域的点，如果σ太大了，可能会将不同子区域中的点错误的聚在一起。在本文中，根据以前的研究经验[65]，选择设置σ为刀具轨迹点相似度的平均值：

$$\sigma = \frac{1}{n^2}\sum_{(i\leqslant i,j\leqslant n)} dist_{i,j}$$

通过公式(3-21)构建一个的$n \times n$相似度矩阵$\boldsymbol{W}$，n为轨迹点数量：

$$\boldsymbol{W} = \begin{bmatrix} lw_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{bmatrix}$$

进一步地，使用谱聚类算法对所构建的相似度矩阵进行处理，选择将轨迹分割为5个子区域，得到最终的分类结果，轨迹聚类结果如图所示。



以下为基于刚度性能的加工轨迹分割：

```
#导入需要的包
import numpy as np
from sklearn.cluster import KMeans,spectral_clustering
import matplotlib.pyplot as plt
from kine import  PoeIkine
from kine import  PoeFkine
from CLSHander import clsTrans
import math

#距离计算
def dis(X1, X2):
    x1 = X1.copy()
    x2 = X2.copy()
    geoDist =  np.linalg.norm(x1[0:3] - x2[0:3])
    stiffnessDist  = np.linalg.norm(x1[3] - x2[3])  #总刚度
    return geoDist,stiffnessDist

#相似度矩阵计算
```

```
def affinity_matrix(X):
    disSum = np.zeros([len(X),len(X)])
    geoMatrix = np.zeros([len(X),len(X)])
    stiffMatrix = np.zeros([len(X),len(X)])
    #获取平均值
    for i in range(len(X) - 1):  # 长度为len(x) 但是从0开始
        for j in range(i + 1, len(X)):  # 从1开始，到len(x) 是方阵 为啥下角标取值的初始值不同？？？
            geoDist, stiffnessDist = dis(X[i], X[j])
            geoMatrix[i,j] = geoMatrix[j,i] = geoDist
            stiffMatrix[i,j] = stiffMatrix[j,i] = stiffnessDist
    geoMatrix = geoMatrix / np.mean(geoMatrix)
    angMatrix = stiffMatrix / np.mean(stiffMatrix)
    disSum = geoMatrix + stiffMatrix
    sigma = 0.7
    #相似度矩阵
    A = np.exp(-disSum  / (2 * sigma ** 2))
    return A

#计算拉普拉斯矩阵
def unnormalized_laplacian(adj_matrix):
    # 先求度矩阵
    R = np.sum(adj_matrix, axis=1)
    degreeMatrix = np.diag(R)
    return degreeMatrix - adj_matrix
# 对称归一化的laplacian矩阵
def normalized_laplacian(adj_matrix):
    R = np.sum(adj_matrix, axis=1)
    R_sqrt = 1/np.sqrt(R)
    D_sqrt = np.diag(R_sqrt)
    I = np.eye(adj_matrix.shape[0])
    return I - np.matmul(np.matmul(D_sqrt, adj_matrix), D_sqrt)

def get_eigen(L, num_clusters):#获取特征
    eigenvalues, eigenvectors = np.linalg.eigh(L)#获取特征值  特征向量
    best_eigenvalues = np.argsort(eigenvalues)[0:num_clusters]#argsort函数返回的是数组值从小到大的索引值
    U = np.zeros((L.shape[0], num_clusters))
    U = eigenvectors[:, best_eigenvalues]#将这些特征取出 构成新矩阵
    return U

#K-Means聚类
def cluster(data, num_clusters):
    data = np.array(data)
    W = affinity_matrix(data)
    L = normalized_laplacian(W)
    eigenvectors = get_eigen(L, num_clusters)
    clf = KMeans(n_clusters=num_clusters)
    s = clf.fit(eigenvectors)  # 聚类
    label = s.labels_
    return label

def plotRes(data, clusterResult, clusterNum):
    """
    结果可似化
    : data:  样本集
```

```
    : clusterResult: 聚类结果
    : clusterNum: 聚类个数
    :return:
    """
    fig1 = plt.figure()
    ax1 = fig1.add_subplot(projection='3d')
    ax1.axis("off")

    fig2 = plt.figure()
    ax2 = fig2.add_subplot(projection='3d')
    ax2.plot(data[:,0], data[:,1], data[:,2], c="red", marker='.')
    ax2.axis("off")

    n = len(data)
    scatterColors = ['black', 'blue', 'red', 'yellow', 'green', 'purple',
'orange']
    for i in range(clusterNum):
        color = scatterColors[i % len(scatterColors)]
        x1 = []
        y1 = []
        z1 = []
        for j in range(n):
            if clusterResult[j] == i:
                x1.append(data[j, 0])
                y1.append(data[j, 1])
                z1.append(data[j, 2])
        ax1.plot(x1, y1, z1, c=color, marker='.')
        plt.subplots_adjust(top=1, bottom=0, left=0, right=1, hspace=0, wspace=0)
def normalization(pos): #数据归一化
    data = np.copy(pos)
    length = len(data[0])
    for i in range(length):
    #最值归一化
        data[:, i] = (data[:, i] - np.min(data[:, i])) / ( np.max(data[:, i]) -
np.min(data[:, i])  ) #
    return data

#聚类主程序
def clusterMain(data,cluster_num):
    pos = data.copy()
    for i in range(len(pos)):#加上工件坐标系
        pos[i] = pos[i]
    vectorList = []#求解位置和变形向量
    initial = np.array([3.50 / 180 * math.pi, -70.43 / 180 * math.pi, 125.62 /
180 * math.pi,
                        6.41 / 180 * math.pi, -55.40 / 180 * math.pi, -2.67 / 180
* math.pi])
    for i in range(len(pos)):
        position = np.array(pos[i][0:3])
        Tmatrix = PoeIkine.RpToMatrix(pos[i])  # 加工任务转换为矩阵
        [theta,success] = PoeIkine.ikine(Tmatrix,initial)#逆运动学
        translation_vector,_ = PoeFkine.StiffnessMatrix(theta)  #总刚度
        vector = np.append(position,translation_vector)
        vectorList.append(vector)
        initial = theta
    data = np.array(vectorList)
```

```
    data = normalization(data)

    label = cluster(data, cluster_num)#进行聚类
    plotRes(np.array(vectorList), label, cluster_num)#绘图
    return label

def main():
    xpath = '../data/clsdata/曲面轨迹精密.cls' #cls文件路径
    workpieceFrame = np.array([1545.32,-49.85,730.99,0,0,0])#工件坐标系
    pos = clsTrans.cls_Main(xpath, workpieceFrame)#读取姿态列表
    label = clusterMain(pos,4)
    plt.show()
```