论文复现实验报告

PB21010479　王曹励文

2023 年 10 月 13 日

1　问题重述

本文的研究对象是考察图像的变形方法，基于移动最小二乘方法，在给定部分点之间的映射后，基于该信息对其余点给出三种变换，包括仿射变换，相似变换，刚体变换. 以上类别变换均非常真实，在现实生活中应用广泛.

2　提出算法

算法的大体框架如下:

1. 给出个别已经确定变换后的像的点，将该点列存储在数组 p 中，将该点列的像存储在数组 q 中；
2. 根据已知的 p,q 信息基于移动最小二乘法计算具体的变换函数，输入为原坐标，输出为变换后的坐标；
3. 根据变换后的坐标进行图像的绘制，笔者在实现时进行了方法的改进，会在后文详细说明.

2.1　前言：变换函数满足的三个原则

本文作者提及在该工作前的工作中，有学者给出了变换函数应该满足的三个原则:

1. Interpolation: p 点列应该被映射至 q 点列；
2. Smoothness: f 应当是光滑的映射；
3. Identity: p 与 q 相同时应确定 f 为 Id.

原则 1 与原则 2 认定函数的光滑性，也确实与现实所需要的匹配；原则 3 与线性精度相关.

2.2　准备工作

将已知的变形原点记作 p，他们的像记作 q，分别存储在 n×2 矩阵 p 与 n×2 矩阵 q 中，这里的 n 表示固定点的个数. 根据给定的 p,q，求解任何一个点 v(1×2 向量) 变换后的坐标. 则定义

$$l_v(x) = xM + T. \tag{2.1}$$

该变换需要使得如下能量函数

$$E = \sum_i \omega_i |l_v(p_i) - q_i|^2. \tag{2.2}$$

取得最小值，其中 ω_i 表示权重向量，为 1×n 向量，其定义为

$$\omega_i = \frac{1}{|p_i - v|^{2\alpha}}. \tag{2.3}$$

从这里可以看出该权重向量取决于 v 与 p，因此笔者写了一个函数 Precompute_w 来先计算该向量. 将l_v的假设式 (2.1) 带入能量函数 (2.2)，可以得到

$$E = \sum_i w_i \left| p_i M + T - q_i \right|^2. \tag{2.4}$$

对式 (2.4) 两边对 T 求导，由于能量函数取得极值，有

$$\frac{\partial E}{\partial T} = 2 \sum_i \omega_i \left| p_i M + T - q_i \right| = 0. \tag{2.5}$$

由 (2.5) 可以解得

$$T = q^* - p^* M. \tag{2.6}$$

其中 q^* 与 p^* 定义如下：

$$\begin{aligned} p_* &= \frac{\sum_i w_i p_i}{\sum_i w_i}. \\ q_* &= \frac{\sum_i w_i q_i}{\sum_i w_i}. \end{aligned} \tag{2.7}$$

可以发现，p^* 与 q^* 的定义取决于 ω 权重向量与 p、q 向量，因此笔者写了 Precompute_pstar 函数来计算该操作，该函数需要首先调用已经写好的用来计算权重向量 ω 的 Precompute_w 函数. 将已经求得的 T 带入 $l_v(x)$ 的定义，即有

$$l_v(x) = (x - p^*)M + q^*. \tag{2.8}$$

此时可以将能量函数改写为

$$E = \sum_i w_i \left| \hat{p}_i M - \hat{q}_i \right|^2. \tag{2.9}$$

其中 $\hat{p}_i$ 与 $\hat{q}_i$ 具有如下的定义：

$$\begin{aligned} \hat{p}_i &= p_i - p^*. \\ \hat{q}_i &= q_i - q^*. \end{aligned} \tag{2.10}$$

因此我们的核心任务转化为求解矩阵 M，对于不同的变换矩阵 M 的计算有些许的不同. 之后对于函数 $l_v(x)$，令

$$f(v) = l_v(v). \tag{2.11}$$

该结果即为所求的 v 的像，可以注意到其随着 v 的变化而变化.

2.3 仿射变换

将 E 的表达式 (2.9) 做如下变形：

$$E = \sum_i w_i \left(\hat{p}_i M - \hat{q}_i \right) \left(\hat{p}_i M - \hat{q}_i \right)^\top = \sum_i w_i \left(\hat{p}_i M M^\top \hat{p}_i{}^\top - \hat{q}_i M^\top \hat{p}_i{}^\top - \hat{p}_i M \hat{q}_i{}^\top + \hat{q} \hat{q}_i{}^\top \right). \tag{2.12}$$

两边对 M 求导，即有

$$\frac{\partial E}{\partial M} = 2 \sum_i \omega_i (\hat{p}_i{}^\top \hat{p}_i M - \hat{p}_i{}^\top \hat{q}_i) = 0. \tag{2.13}$$

这里应用了对于矩阵的求导，则可以求得

$$M = (\sum_i \hat{p_i}^\top \omega_i \hat{p_i})^{-1} \cdot (\sum_j \omega_j \hat{p_j}^\top \hat{q_j}). \tag{2.14}$$

再带入 (2.8) 和 (2.11) 式可以得到

$$f_a(v) = (v - p^*)(\sum_i \hat{p_i}^\top \omega_i \hat{p_i})^{-1} \cdot (\sum_j \omega_j \hat{p_j}^\top \hat{q_j}) + q^*. \tag{2.15}$$

该表达式中 $f_a(v)$ 即表示仿射变换，式 (2.15) 给出了仿射变换的表达式. 注意到如果需要实现交互性，即使用者可以通过改变 q 的位置实现实时动画效果，为了方便计算，可以对 (2.15) 做如下变形：

$$f_a(v) = \sum_j A_j \hat{q_j} + q^*. \tag{2.16}$$

A_j 的定义为

$$A_j = (v - p^*)(\sum_i \hat{p_i}^\top \omega_i \hat{p_i})^{-1} \cdot (\omega_j \hat{p_j}^\top). \tag{2.17}$$

注意到 A_j 的定义与 q 的取值无关，因此只需计算一次. 根据以上原理笔者编写了函数 Compute_Affine，该函数需要参数 p,q,v，输入点的坐标，结果为仿射变化后点的坐标，均为 1×2 的向量.

2.4 相似变换

然而仿射变换可能出现分布不均匀或者存在断裂的情形，现实生活中的物体通常不执行此类简单的变换. 而相似变换仅包含着平移、旋转与均匀的范围分布. 我们因此考察相似变换下的映射函数，核心仍为求出矩阵 A.

相似变换要求矩阵拥有 $M^\top M = \lambda^2 I$ 的性质，若设

$$M = (M_1 \quad M_2).$$

则有 $M_1^\top M_1 = M_2^\top M_2 = \lambda^2$ 且 $M_1^\top M_2 = 0$. 表明

$$M_2 = M_1^\perp. \tag{2.18}$$

则能量函数 E 根据 (2.9) 式可改写为

$$E = \sum_i \omega_i \left| \begin{pmatrix} \hat{p}_i \\ -\hat{p_i}^\perp \end{pmatrix} M_1 - \hat{q_i}^\top \right|^2. \tag{2.19}$$

对 M_1 求导，有

$$\frac{\partial E}{\partial M_1} = 2\sum_i \omega_i \begin{pmatrix} \hat{p}_i \\ -\hat{p_i}^\perp \end{pmatrix} (\begin{pmatrix} \hat{p_i}^\top & -(\hat{p_i}^\perp)^\top \end{pmatrix} M_1 - \hat{q_i}^\top) = 0. \tag{2.20}$$

可解得

$$M_1 = \frac{1}{\mu_s} \sum_i \omega_i \begin{pmatrix} \hat{p}_i \\ -\hat{p_i}^\perp \end{pmatrix} \hat{q_i}^\top. \tag{2.21}$$

其中

$$\mu_s = \sum_i \omega_i \hat{p}_i \hat{p_i}^\top. \tag{2.22}$$

结合 (2.18) 即知

$$M = \frac{1}{\mu_s}\sum_i \omega_i \begin{pmatrix} \hat{p}_i \\ -\hat{p}_i{}^{\perp} \end{pmatrix} \begin{pmatrix} \hat{q}_i{}^{\top} & -(\hat{q}_i{}^{\perp})^{\top} \end{pmatrix} \tag{2.23}$$

同样为了使得用户的交互性更完善，结合 (2.8) 与 (2.11) 对表达式做如下改善：

$$f_s(v) = \sum_i \hat{q}_i(\frac{1}{\mu_s}A_i) + q^*. \tag{2.24}$$

其中

$$A_i = \omega_i \begin{pmatrix} \hat{p}_i \\ -\hat{p}_i{}^{\perp} \end{pmatrix} \begin{pmatrix} v-p^* \\ -(v-p^*)^{\perp} \end{pmatrix}^{\top}. \tag{2.25}$$

$f_s(v)$ 即为相似变换映射，且能看出矩阵 A_i 仅与 p,v 有关.

实现相似变换的代码见函数 Compute_Similar，输入输出与仿射变换相同.

相似变换有良好的保角性质，因此得到的结果相比于仿射变换更加理想，但是对于距离变换点较远的部分容易出现范围的扩大化.

2.5 刚体变换

在部分之前的工作中，发现更加逼真的变形更需要尽可能地接近刚体变换. 作者提出了引理 2.1，并且给出了如下简洁的表达形式：

$$\boldsymbol{f_r(v)} = \sum_i \hat{q}_i A_i. \tag{2.26}$$

$$f_r(v) = |v-p^*|\frac{\boldsymbol{f_r(v)}}{|\boldsymbol{f_r(v)}|} + q^*. \tag{2.27}$$

$f_r(v)$ 为所求的刚体变换函数. 相比于相似变换，刚体变换利用了归一化的手段，因此计算量也偏大. 实现刚体变换的代码见函数 Compute_Rigid，输入输出与其余变换相同.

2.6 图像处理

笔者第一次采用了如下的做法：由于 MATLAB 存储图像的方式为三维数组，前两位为行列坐标，第三个元素含有 RGB 的三维数组，考虑将变换前点的颜色赋值给变换后点的颜色即可，之后作出的图像含有部分的黑点，并没有良好的复现.

因此笔者后来使用了 imwarp 内置函数，该函数输入两个参数 imwarp(A,D)，A 为输入的图像，D 为 rows×cols×2 数组，前两个指标表示原点，第三个指标存储的二维数组表示位移，注意为初始位置-终点位置，该内置函数可以较好的实现作图. 该段代码见 hw_12.1.

3 复现结果

所用的代码存储在文件夹论文复现中. 代码只需要在 hw_12.1 中调用所需要的函数即可，注意调用时应注释其余函数. 使用仿射变换、相似变换、刚体变换的结果分别如图 1(a)(b)(c) 所示.

可以从图中看出，仿射变换的变形过于剧烈，确实不适合显示物体的运动；相似变换右胳膊处相比于刚体变换偏高，由于距离变换点的几何中心较远；而刚体变换确实是最符合现实的，效果也是极佳的.

(a) 仿射变换 (b) 相似变换 (c) 刚体变换

图 1: 论文复现结果

4 总结与改善

本文中的代码实际可以进行交互式处理，可以实现更好的动画效果. 本文已经实现了二维平面内的刚体变换，因此无法再做到最好. 推广的方向可以是对于三维物体的变换研究，以及作者提到的可能出现的"foldback" 现象.