



中山大學  
SUN YAT-SEN UNIVERSITY

## 模式识别

### 全景图拼接

姓名 马岱

学号 22336180

学院 计算机学院

专业 计算机科学与技术

2025年4月15日

# 目录

<b>1 实验目的</b>	<b>3</b>
<b>2 实验内容</b>	<b>3</b>
<b>3 实验过程</b>	<b>3</b>
3.1 算法原理 . . . . .	3
3.1.1 Harris 角点检测器 . . . . .	3
3.1.2 SIFT 描述子 . . . . .	6
3.1.3 HOG 描述子 . . . . .	7
3.1.4 两种描述子的对比分析 . . . . .	8
3.1.5 RANSAC 抽样一致方法 . . . . .	9
3.2 实验步骤 & 代码 . . . . .	10
3.2.1 Harri 角点算法 . . . . .	10
3.2.2 关键点描述与匹配 . . . . .	12
3.2.3 全景图拼接 . . . . .	14
<b>4 实验结果</b>	<b>18</b>
4.1 Harri 角点算法 . . . . .	18
4.1.1 结果展示 . . . . .	18
4.1.2 实验分析 . . . . .	18
4.2 关键点描述与匹配 . . . . .	19
4.2.1 结果展示 . . . . .	19
4.2.2 实验分析 . . . . .	20
4.3 全景图拼接 . . . . .	20
4.3.1 二图拼接结果展示 . . . . .	20
4.3.2 二图拼接实验分析 . . . . .	21
4.3.3 多图拼接结果展示 . . . . .	22
4.3.4 多图拼接实验分析 . . . . .	22
<b>5 实验总结</b>	<b>22</b>

# 1 实验目的

- 熟悉 Harris 角点检测器的原理和基本使用
- 熟悉 RANSAC 抽样一致方法的使用场景
- 熟悉 HOG 描述子的基本原理

# 2 实验内容

- 使用 Harris 焦点检测器寻找关键点。
- 构建描述算子来描述图中的每个关键点，比较两幅图像的两组描述子，并进行匹配。
- 根据一组匹配关键点，使用 RANSAC 进行仿射变换矩阵的计算。
- 将第二幅图变换过来并覆盖在第一幅图上，拼接形成一个全景图像。
- 实现不同的描述子，并得到不同的拼接结果。

# 3 实验过程

## 3.1 算法原理

### 3.1.1 Harris 角点检测器

角点通常出现在图像中两条边界相交或者存在显著纹理变化的区域。传统的边缘检测虽然可以检测到轮廓，但边缘并不能提供足够的稳定性用于匹配和跟踪。简单来讲，Harris 角点检测就是使用一个规定好大小的“窗口”，在图片的各个地方采集灰度特征。如果在某个位置上，“窗口”的微小移动会导致灰度发生较大的变化，就认为该处是一个“角点”。

**1. 局部图像灰度变化模型** 设图像的灰度函数为  $I(x, y)$ ，考虑图像在局部邻域  $W$  内平移  $(u, v)$  后的灰度变化。定义误差函数为

$$E(u, v) = \sum_{(x,y) \in W} w(x, y) [I(x + u, y + v) - I(x, y)]^2,$$

其中

- $W$  表示窗口区域（如矩形或采用高斯加权的窗口），
- $w(x, y)$  为窗口内的权重函数（高斯权重为常用选择）。

**2. Taylor 展开与二次近似** 对于较小的平移  $(u, v)$ , 对  $I(x + u, y + v)$  在  $(0, 0)$  处进行一阶泰勒展开有:

$$I(x + u, y + v) \approx I(x, y) + I_x(x, y)u + I_y(x, y)v,$$

其中  $I_x$  与  $I_y$  分别为  $x$  与  $y$  方向的偏导数。代入误差函数可得:

$$\begin{aligned} E(u, v) &\approx \sum_{(x,y) \in W} w(x, y) [I_x(x, y)u + I_y(x, y)v]^2 \\ &= \sum_{(x,y) \in W} w(x, y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2(x, y) & I_x(x, y)I_y(x, y) \\ I_x(x, y)I_y(x, y) & I_y^2(x, y) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}. \end{aligned}$$

### 3. 结构张量的构建 构建结构张量

$$M = \sum_{(x,y) \in W} w(x, y) \begin{bmatrix} I_x^2(x, y) & I_x(x, y)I_y(x, y) \\ I_x(x, y)I_y(x, y) & I_y^2(x, y) \end{bmatrix},$$

则误差函数可以写为二次型:

$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}.$$

矩阵  $M$  中蕴含了窗口内的梯度信息, 是后续判定角点的重要依据。

**4. 角点、边缘与平坦区域的判定** 由于  $M$  为对称正半定矩阵, 其特征值  $\lambda_1$  和  $\lambda_2$  描述了局部灰度变化的特性:

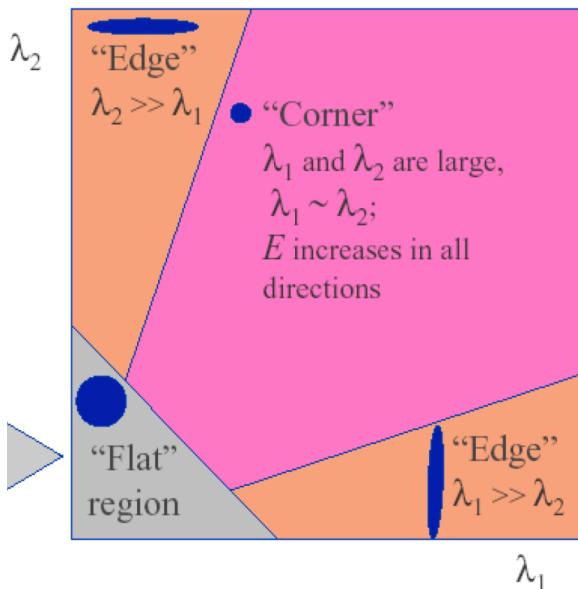


图 1: Classification via Eigenvalues

- **平坦区域**: 当  $\lambda_1 \approx 0$  且  $\lambda_2 \approx 0$  时, 区域内灰度变化不足;
- **边缘**: 当一特征值显著大于另一特征值 (如  $\lambda_1 \gg \lambda_2$  或反之) 时, 变化主要集中于一个方向;
- **角点**: 当  $\lambda_1$  与  $\lambda_2$  均较大时, 说明在两个正交方向上均存在显著灰度变化, 此时为角点。

**5. 角点响应函数** 由于直接利用特征值进行判断较为繁琐, Harris 角点检测器引入角点响应函数:

$$R = \det(M) - k [\text{trace}(M)]^2,$$

其中

- $\det(M) = \lambda_1 \lambda_2$  表示梯度乘积的量度;
- $\text{trace}(M) = \lambda_1 + \lambda_2$  代表整体灰度变化;
- $k$  为经验参数, 通常取 0.04 到 0.06 之间。

当  $R$  较大时, 表明该位置为角点; 若  $R$  较小或为负, 则为边缘或平坦区域。

**6. 非极大值抑制与阈值处理** 计算每个像素的响应值  $R$  后, 为避免在局部区域内重复检测到多个相邻点, 需进行:

- **阈值处理**: 仅保留  $R > T$  的像素,  $T$  为设定阈值;
- **非极大值抑制**: 在局部邻域内仅保留响应值最大的像素, 从而精确确定角点位置。

## 7. Harris 角点检测器整体流程如下

- 计算图像在  $x$  和  $y$  方向的梯度  $I_x$  和  $I_y$ ;
- 构造结构张量  $M$ ;
- 计算响应函数  $R = \det(M) - k [\text{trace}(M)]^2$ ;
- 对响应函数进行阈值处理和非极大值抑制, 最终确定角点位置。

由于算法通过梯度的平方和乘积构造矩阵  $M$ , 检测结果对图像旋转具有一定的不变性。相比于其他需要复杂特征提取的方法, Harris 算法计算相对简单且高效。在适当的参数设置和预处理下, 对噪声具有一定的鲁棒性。但是 Harris 角点检测器在面对图像尺度变化时效果不佳, 因为它未考虑多尺度特征检测。为了解决这一问题, 可以采用多尺度金字塔或使用诸如 SIFT 等算法。尽管高斯滤波能够在一定程度上降低噪声, 然而在噪声较严重的图像中仍可能出现误检。并且其  $k$  值和阈值的选择往往需要依据实验调整, 这可能限制其在某些应用中的自动化程度。

### 3.1.2 SIFT 描述子

SIFT (尺度不变特征变换) 描述子是计算机视觉领域中一种经典且非常有效的局部特征检测与描述方法。SIFT 的重要特点在于对尺度变化具有不变性。它通过构造尺度空间，在多种尺度下检测关键点，使得同一物体在不同分辨率或距离下均能提取到相似的特征。SIFT 关注图像中局部区域的信息，提取出的关键点及其描述子能够提供关于局部纹理、边缘和角点等详细信息，便于在不同图像或场景中匹配目标。

**1. 尺度空间极值检测** 为了检测出不同尺度下具有稳定性的关键点，引入尺度空间概念。给定图像  $I(x, y)$ ，构造尺度空间图像：

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

其中  $*$  表示卷积， $G(x, y, \sigma)$  为标准二维高斯核：

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right).$$

通过不同的  $\sigma$  值，可以获得图像在多尺度下的平滑版本。进一步计算相邻尺度之间的差分，得到差分高斯图像 (DoG)：

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma),$$

其中  $k$  为尺度间的比例因子。极值点在空间和尺度上满足局部极值条件，即当前点在  $3 \times 3 \times 3$  的邻域中为最大值或最小值，这些点候选为尺度不变的关键点。

**2. 关键点定位与精确化** 对候选关键点，通过拟合三维二次函数对其位置及尺度进行精细定位。采用泰勒展开近似 DoG 函数：

$$D(\mathbf{x}) \approx D + \frac{\partial D^T}{\partial \mathbf{x}} \mathbf{x} + \frac{1}{2} \mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2} \mathbf{x},$$

其中  $\mathbf{x} = (x, y, \sigma)^T$ 。通过求解偏导数为零的方程，进一步抑制低对比度点与边缘响应点。

**3. 方向赋值** 为了实现旋转不变性，针对每个关键点在其邻域内计算梯度幅值和方向：

$$m(x, y) = \sqrt{[L(x+1, y) - L(x-1, y)]^2 + [L(x, y+1) - L(x, y-1)]^2},$$

$$\theta(x, y) = \arctan \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}.$$

在关键点邻域内构建梯度方向直方图（通常分为 36 个 bin），并选择直方图中幅值最大的方向作为主要方向。若直方图中有其他峰值接近最大值，也可生成多个关键点，以增加匹配的稳健性。

**4. 描述子构建** 在以关键点为中心、按主要方向旋转对齐的局部邻域内，将区域划分成  $4 \times 4$  个子区域。对每个子区域计算梯度方向直方图（通常每个子区域使用 8 个方向的 bin），即：

$$\text{描述子维度} = 4 \times 4 \times 8 = 128.$$

在计算直方图时，每个梯度值按照幅值加权，并且乘以一个高斯窗口以减弱远离中心点的影响。最终，对得到的 128 维描述子进行归一化，抑制亮度变化带来的影响。

### 3.1.3 HOG 描述子

HOG (梯度方向直方图) 描述子是一种常用的计算机视觉特征提取方法，主要用于检测和识别图像中的局部形状和边缘信息。**HOG 描述子**的核心思想是在局部区域内统计像素梯度方向的分布信息。图像中物体的局部轮廓（例如边缘、角点等）往往具有特定的梯度分布，通过统计梯度方向，可以反映出物体的几何形状与结构特征。

**1. 梯度计算** 对图像  $I(x, y)$  采用微分算子（如 Sobel 算子）计算图像梯度：

$$I_x = I * G_x, \quad I_y = I * G_y,$$

其中  $G_x$  与  $G_y$  分别为水平方向和竖直方向的滤波器。例如，Sobel 算子的形式为：

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

梯度幅值和方向分别为：

$$m(x, y) = \sqrt{I_x(x, y)^2 + I_y(x, y)^2},$$

$$\theta(x, y) = \arctan \frac{I_y(x, y)}{I_x(x, y)}.$$

**2. 构建细胞 (Cells) 和直方图** 将图像划分为小的细胞 (cell)，每个 cell 的大小一般为  $8 \times 8$  像素。在每个 cell 内，依据每个像素的梯度方向，将幅值  $m(x, y)$  投票到预定的方向 bin 中。通常采用 9 个方向 bin，每个 bin 代表  $20^\circ$  的范围。

假设 cell 内的像素方向  $\theta$  分布在  $[0, 180^\circ]$  (或  $[0, 360^\circ]$ )，则直方图函数可表示为：

$$H_i = \sum_{(x,y) \in \text{cell}} m(x, y) \cdot \delta_i(\theta(x, y)),$$

其中  $\delta_i(\theta)$  为指示函数，如果  $\theta$  落在第  $i$  个方向的区间内，则其取值为 1，否则为 0。

**3. 块归一化** 为了提高对光照和对比度变化的鲁棒性，对相邻的 cell 进行组合形成块 (block)。一个块通常由  $2 \times 2$  个 cell 组成，对该块内所有 cell 的直方图向量进行归一化。设某块的所有特征向量为  $\mathbf{v}$ ，归一化过程可采用 L2 范数：

$$\mathbf{v}_{\text{norm}} = \frac{\mathbf{v}}{\sqrt{\|\mathbf{v}\|_2^2 + \epsilon}},$$

其中  $\epsilon$  为防止除零的小常数。

### 3.1.4 两种描述子的对比分析

在本实验中，我们的主要目标是实现全景图拼接。这一过程涉及关键点的检测、描述、匹配以及通过 RANSAC 求解仿射变换等步骤。其中，SIFT 和 HOG 分别用于构造**关键点描述子**。

#### 1. 关键点描述与匹配

- 在拼接图像之前，必须先提取图像中的局部特征。我们首先利用 Harris 角点检测器定位图像中的候选关键点。
- 接下来，为了描述每个关键点局部区域的外观信息，我们需要构造**特征描述子**。这一步骤能够将图像中局部区域的梯度和纹理信息转化为一个固定维度的向量，用于后续的特征匹配。

#### 2. SIFT 描述子的作用

- **尺度与旋转不变性**: SIFT 描述子通过在尺度空间中检测关键点，并对关键点邻域进行归一化处理，使得描述子对尺度变化和旋转具有较高的鲁棒性。在图像存在缩放或旋转的情况下，SIFT 依然能保持较高匹配率。
- **丰富的局部信息**: SIFT 在关键点局部区域构建梯度方向直方图，形成 128 维的高维特征向量，能够充分刻画局部图像的纹理和结构信息，从而使得匹配更加准确。
- 在本实验中，利用 SIFT 生成的描述子进行匹配，通常能获得较多正确的匹配点，使得后续 RANSAC 求解仿射变换矩阵时，内点数量较多，从而实现较为精确的全景图拼接。

#### 3. HOG 描述子的作用

- **梯度方向统计**: HOG 描述子通过统计图像中局部区域内的梯度方向分布，构成一个描述局部边缘和纹理特征的向量。尽管 HOG 描述子在尺度和旋转不变性上不如 SIFT 出色，但它依然可以很好地捕捉局部形状信息。

- **计算简单**: HOG 的计算相对简单，可以快速获得特征描述，对于处理较大图像或实时要求较高的场景具有优势。
- 在本实验中，我们使用 HOG 描述子作为另一种关键点描述方法，通过与 SIFT 的对比，可以评估不同描述子对匹配效果和全景图拼接效果的影响。例如，在匹配中可能存在误匹配情况，HOG 生成的描述子可能在纹理明确但旋转、尺度变化较小的区域表现较好。

### 3.1.5 RANSAC 抽样一致方法

在计算机视觉、机器人定位以及统计学中，数据往往受到噪声、异常值 (Outliers) 的干扰，直接应用最小二乘法等拟合方法可能会导致模型偏差较大。RANSAC (RANdom SAmple Consensus) 是一种鲁棒的参数估计方法，通过**迭代随机抽取最小样本集**来估计模型参数，并对数据中的**内点**进行筛选，从而得到较为可靠的模型。

假设我们有数据集  $\mathcal{D} = \{x_1, x_2, \dots, x_N\}$ ，其中存在一定比例的异常值。对于某一模型，其估计函数可以表示为  $f(x; \boldsymbol{\theta})$ ，而点  $x_i$  的残差  $r_i$  定义为

$$r_i = \|x_i - f(x_i; \boldsymbol{\theta})\|.$$

设定阈值  $T$ ，当  $r_i \leq T$  时认为  $x_i$  是内点。即内点集合定义为：

$$\mathcal{I} = \{x_i \in \mathcal{D} \mid r_i \leq T\}.$$

对于最小样本集  $\mathcal{S} \subset \mathcal{D}$ ，其包含的数据点个数为  $s$ ，使得模型参数  $\boldsymbol{\theta}$  能被唯一确定。比如，拟合直线模型时  $s = 2$ ；拟合二维仿射变换时可能需要 3 对匹配点。

在一次迭代中，我们计算模型参数  $\boldsymbol{\theta}$ ，然后内点数记为

$$n_{\text{inlier}} = |\mathcal{I}|.$$

最终我们希望获得一个最大内点集合  $\mathcal{I}_{\max}$ ，对应的模型参数记为  $\boldsymbol{\theta}^*$ 。

为了保证在迭代中找到正确模型的概率达到某一置信度  $p$ ，估计需要的迭代次数  $N$  可以通过下面的公式计算：

$$N = \frac{\log(1 - p)}{\log(1 - w^s)},$$

其中：

- $w$  为数据集中内点所占的比例；
- $s$  为估计模型所需的最小样本数。

该公式保证了在  $N$  次迭代中至少有一次抽样完全由内点组成，从而可以得到正确模型的概率达到  $p$ 。

**RANSAC 方法在实验中的应用** 在全景图拼接实验中，RANSAC 被用来根据匹配的关键点对求解仿射变换矩阵或单应矩阵。具体步骤如下：

- 给定匹配点对集合  $\{(x_i, x'_i)\}$ ，其中部分匹配可能由于误匹配而成为异常值；
- 通过随机选择最小样本集（例如 3 个匹配点对用于仿射变换）计算仿射变换矩阵  $M$ ；
- 利用  $M$  对所有匹配点对计算映射误差，并判断内点；
- 重复迭代，获取内点最多的一组匹配集合，进而利用这些内点重估  $M$ ；
- 得到最终变换矩阵后，将图像进行变换拼接，形成全景图。

通过 RANSAC，误匹配（异常值）的影响可以得到大幅度降低，使得拼接的仿射变换更为准确，从而增强全景图的整体一致性和视觉效果。

## 3.2 实验步骤 & 代码

根据上一小节的算法原理，我们在这一部分直接给出相关的代码和解释。

### 3.2.1 Harri 角点算法

首先通过两个  $3 \times 3$  的 Sobel 核 `sobel_x` 和 `sobel_y` 分别用于检测水平方向和垂直方向的图像梯度。然后调用 `cv2.filter2D` 将 Sobel 核与图像进行卷积，得到梯度图像 `Ix` 和 `Iy`。对梯度图像进行平方操作得到 `Ixx` 和 `Iyy`，以及计算梯度乘积 `Ixy`。这些值为后续构造局部结构矩阵提供依据。利用 `cv2.GaussianBlur` 对二次项进行平滑处理，得到了加权求和的结果，即 `Sxx`、`Syy` 和 `Sxy`，用于降低噪声的干扰。计算自相关矩阵的行列式  $\det M$  与迹  $\text{trace}(M)$ ，根据公式  $R = \det M - k \cdot (\text{trace}(M))^2$  得到角点响应图 `R`。

```

1 def harris_corner_detection(gray, block_size=3, ksize=3, k=0.04):
2     sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.float32)
3     sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=np.float32)
4     # 计算梯度
5     Ix = cv2.filter2D(gray, -1, sobel_x)
6     Iy = cv2.filter2D(gray, -1, sobel_y)
7     # 计算梯度的二次项
8     Ixx = Ix**2
9     Iyy = Iy**2
10    Ixy = Ix * Iy
11    # 利用高斯滤波器对二次项平滑
12    Sxx = cv2.GaussianBlur(Ixx, (block_size, block_size), sigmaX=1)
13    Syy = cv2.GaussianBlur(Iyy, (block_size, block_size), sigmaX=1)

```

```

14     Sxy = cv2.GaussianBlur(Ixy, (block_size, block_size), sigmaX=1)
15     # 计算角点响应  $R = \det(M) - k * \text{trace}(M)^2$ 
16     detM = Sxx * Syy - Sxy**2
17     traceM = Sxx + Syy
18     R = detM - k * (traceM**2)
19     return R

```

接着根据最大响应值与 `threshold_ratio` 来确定阈值，将低于该阈值的区域置零，得到 `R_threshold`。利用 `cv2.dilate` 对 `R_threshold` 进行膨胀操作，扩展局部区域的最大值。比较膨胀后的图像与原始阈值图像，只有在两者相等且大于零的位置，才认为是局部最大值，从而得到二值图像 `corners`。

```

1 def non_maximum_suppression(R, threshold_ratio=0.01, window_size=3):
2     threshold = threshold_ratio * R.max()
3     R_threshold = (R > threshold) * R
4     dilated = cv2.dilate(R_threshold, np.ones((window_size, window_size),
5                                         dtype=np.uint8))
5     corners = (R_threshold == dilated) & (R_threshold > 0)
6     return corners

```

最后我们调用 `harris_corner_detection` 计算图像的响应值矩阵 `R`。随后调用 `non_maximum_suppression` 对响应图做阈值处理和局部极大值抑制，得到二值图像 `corners`。利用 `np.nonzero` 提取出候选角点的  $(y, x)$  坐标，并将每个坐标转换为 `cv2.KeyPoint` 对象，关键点的尺度在此设为 6，最终返回这些关键点列表。

```

1 def extract_harris_keypoints(img_gray_float, threshold_ratio=0.01):
2     R = harris_corner_detection(img_gray_float)
3     corners = non_maximum_suppression(R, threshold_ratio=threshold_ratio)
4     y_coords, x_coords = np.nonzero(corners) # 返回 (y, x) 坐标
5     keypoints = [
6         cv2.KeyPoint(float(x), float(y), 6) for x, y in zip(x_coords, y_coords)
7     ]
8     return keypoints

```

为了排除假阳性（即误检测的角点） 可采用以下方法：

- 合适的阈值选择：**设置合适的 `threshold_ratio` 能使得响应较低的点被过滤，减少噪声干扰。
- 非极大值抑制：**本代码采用局部极大值判断来去除同一局部区域内多个重复角点。
- 后处理：**可以进一步对检测到的角点进行形态学处理或结合领域先验信息进行筛选，比如通过区域统计，剔除边缘检测而非角点的假阳性。

### 3.2.2 关键点描述与匹配

**SIFT** 这里直接调用 sift 的库函数来实现（由于实验文档中不要求自行实现这一部分），调用 `sift.compute(gray_uint8, keypoints)`。该函数依次处理传入的关键点列表，并根据图像中的局部信息计算每个关键点的特征描述子，生成一个描述子矩阵。最后将更新后的 `keypoints` 和描述子 `descriptors` 返回，用于后续的匹配、拼接等任务。

```

1 def compute_sift_descriptors(gray_uint8, keypoints):
2     sift = cv2.SIFT_create()
3     keypoints, descriptors = sift.compute(gray_uint8, keypoints)
4     return keypoints, descriptors

```

**HOG** 对于 hog，先对输入的图像片段（patch）计算 HOG 描述子。首先检查输入片段的尺寸是否为  $32 \times 32$  像素，如果不是，则使用 `cv2.resize` 将其调整到  $32 \times 32$  的大小。接着，创建一个 HOG 描述子对象，在创建 HOG 描述子对象后，调用 `hog.compute` 对调整后的图像片段进行计算，得到描述子的特征向量，最后通过 `flatten()` 方法将描述子由多维数组展平成一维数组并返回。

```

1 def compute_hog_descriptor_for_patch(patch):
2     """
3         对输入 patch (期望尺寸为 32x32) 计算 HOG 描述子
4     """
5     if patch.shape[0] != 32 or patch.shape[1] != 32:
6         patch = cv2.resize(patch, (32, 32))
7     hog = cv2.HOGDescriptor(
8         _winSize=(32, 32),
9         _blockSize=(16, 16),
10        _blockStride=(8, 8),
11        _cellSize=(8, 8),
12        _nbins=9,
13    )
14    descriptor = hog.compute(patch)
15    return descriptor.flatten()

```

接着在一幅灰度图像与 Harris 检测得到的关键点列表的基础上，为每个关键点提取其周围的图像区域（patch）并计算对应的 HOG 描述子。具体来说，先对输入灰度图像的尺寸进行获取，并计算出补丁尺寸的一半，然后对于传入的每个关键点，根据其坐标值判断对应的 patch 是否完全位于图像内部；如果超出边界，则直接跳过该关键点；如果满足条件，则通过数组切片提取关键点周围大小为 `patch_size × patch_size` 的区域，并调用前述的 `compute_hog_descriptor_for_patch` 函数获得该 patch 的 HOG 描

述子。计算出的描述子会被追加到描述子列表中，同时将对应的关键点（构造时设置关键点的尺度为 patch 尺寸）加入到有效关键点列表中。

```

1 def compute_hog_descriptors(gray_uint8, keypoints, patch_size=32):
2     """
3         对给定灰度图和 Harris 检测到的关键点列表,
4         提取每个关键点周围的 patch, 并计算 HOG 描述子;
5         返回有效关键点列表及描述子矩阵
6     """
7     descriptors_list = []
8     valid_kps = []
9     h, w = gray_uint8.shape
10    half = patch_size // 2
11    for kp in keypoints:
12        x, y = int(kp.pt[0]), int(kp.pt[1])
13        if x - half < 0 or x + half > w or y - half < 0 or y + half > h:
14            continue
15        patch = gray_uint8[y - half : y + half, x - half : x + half]
16        hog_desc = compute_hog_descriptor_for_patch(patch)
17        descriptors_list.append(hog_desc)
18        valid_kps.append(cv2.KeyPoint(float(x), float(y), patch_size))
19    if len(descriptors_list) == 0:
20        return valid_kps, None
21    descriptors = np.array(descriptors_list, dtype=np.float32)
22    return valid_kps, descriptors

```

**关键点匹配** 利用 BFMatcher 结合欧几里得距离对输入图像的描述子进行匹配。通过调用 cv2.BFMatcher 构造一个基于 L2 范数（欧几里得距离）且开启交叉检查（crossCheck=True）的匹配器，该匹配器在匹配时会只保留满足互相最近邻条件的匹配对。随后，利用该匹配器的 match 方法对两个描述子进行匹配，并按照匹配距离（distance）从小到大对匹配结果排序。然后，使用 cv2.drawMatches 函数将前 50 个匹配结果绘制到两幅彩色图像上，注意此处绘制方式设置了 flags=2 以便更清晰地显示匹配线条。

```

1 def match_and_draw(img1, kp1, des1, img2, kp2, des2, output_path,
2     ↵ desc_type="SIFT"):
3     if des1 is None or des2 is None:
4         print(f"{desc_type} 描述子为空, 无法匹配!")
5         return None
6     bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
7     matches = bf.match(des1, des2)

```

```

7     matches = sorted(matches, key=lambda x: x.distance)
8     img_matches = cv2.drawMatches(img1, kp1, img2, kp2, matches[:50], None,
9         → flags=2)
10    cv2.imwrite(output_path, img_matches)
11    print(f"{desc_type} 匹配结果保存至: {output_path}")
12    return matches

```

### 3.2.3 全景图拼接

**两幅图拼接** 根据两个图像之间匹配到的关键点，通过 RANSAC 方法求解仿射变换矩阵。我们从上一节匹配结果中提取关键点坐标，利用这些匹配点，调用 cv2.estimateAffine2D 采用 RANSAC 算法求解从 img2 到 img1 的仿射变换矩阵 M 并将 img2 的四个角点经过仿射变换后与 img1 的角点合并，计算出整个拼接图像的最小外接矩形区域。由此，构造了一个平移矩阵 T，用于将所有坐标平移到非负区域，并将仿射矩阵 M 扩展为 3x3 齐次矩阵后，组合出最终的变换矩阵 combined\_transform。然后，利用 cv2.warpAffine 对 img2 进行仿射变换，将其投影到新的输出尺寸的画布上。最后，函数将 img1 的像素复制到对应的位置上，实现两幅图像的拼接。

```

1 def stitch_images(img1, img2, kp1, kp2, matches, output_path,
2   → desc_type="SIFT"):
3
4     """
5       根据匹配关键点对利用 RANSAC 求仿射变换矩阵，
6       将 img2 进行仿射变换并拼接到 img1 上，保存拼接结果。
7     """
8
9     if matches is None or len(matches) < 3:
10        print(f"{desc_type} 匹配不足，无法拼接图像！")
11        return
12
13     # 提取匹配点对
14     pts1 = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 2)
15     pts2 = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 2)
16
17     # 利用 RANSAC 求解仿射变换矩阵
18     M, inliers = cv2.estimateAffine2D(
19         pts2, pts1, method=cv2.RANSAC, ransacReprojThreshold=5
20     )
21
22     if M is None:
23         print(f"{desc_type} RANSAC 无法求解仿射矩阵！")
24         return
25
26     h1, w1 = img1.shape[:2]
27     h2, w2 = img2.shape[:2]

```

```

22     # 将 img2 的角点映射到 img1 坐标系
23     pts_img2 = np.float32([[0, 0], [w2, 0], [w2, h2], [0, h2]]).reshape(-1, 1,
24                           ↵ 2)
24     pts_img2_trans = cv2.transform(pts_img2, M)
25     pts_img1 = np.float32([[0, 0], [w1, 0], [w1, h1], [0, h1]]).reshape(-1, 1,
26                           ↵ 2)
26     all_pts = np.concatenate((pts_img1, pts_img2_trans), axis=0)
27     [xmin, ymin] = np.int32(all_pts.min(axis=0).ravel() - 0.5)
28     [xmax, ymax] = np.int32(all_pts.max(axis=0).ravel() + 0.5)
29     # 构造平移矩阵 T (3x3)
30     T = np.array([[1, 0, -xmin], [0, 1, -ymin], [0, 0, 1]], dtype=np.float32)
31     # 将 M 扩展为 3x3 齐次矩阵
32     M_hom = np.vstack([M, [0, 0, 1]])
33     # 组合变换: 先仿射变换 M, 再平移 T
34     combined_transform = T.dot(M_hom) # 3x3 矩阵
35     output_size = (xmax - xmin, ymax - ymin)
36     warped_img2 = cv2.warpAffine(img2, combined_transform[:2, :], output_size)
37     # 将 img1 放入拼接图中
38     stitched_img = warped_img2.copy()
39     stitched_img[-ymin : h1 - ymin, -xmin : w1 - xmin] = img1
40
41     cv2.imwrite(output_path, stitched_img)
42     print(f"{'desc_type'} 拼接结果已保存至: {output_path}")

```

**多幅图拼接** 首先设定第一幅图为基准图，将其对应的累积单应性矩阵初始化为单位矩阵，并依次对后续图像进行处理：对于每对相邻图像，调用 Harris 角点检测来获得候选关键点，再利用 SIFT 计算各自的描述子；接着，利用 BFMatcher 进行描述子匹配，并将匹配结果按匹配距离进行排序，如果匹配点数量不足则停止处理。随后，提取匹配中对应的关键点坐标，并利用 cv2.findHomography 求解当前图像到前一图像之间的单应性矩阵，再通过矩阵相乘累积出从当前图像到第一幅基准图像的变换矩阵，并将该矩阵存入累积矩阵列表中，从而确定出所有图像覆盖区域的最小外接矩形，得到平移矩阵。利用该平移矩阵和各自的累积单应性矩阵，通过 cv2.warpPerspective 将每幅图像投影到全局画布上，得到各图变换后的图像，并利用简单的非零像素覆盖策略将这些投影图像叠加到一个预先初始化的黑色画布上，最终得到拼接后的全景图。

```

1 def stitch_multiple_images(image_paths, output_path):
2     # 依次读取各幅图像
3     images = []
4     for path in image_paths:
5         img = cv2.imread(path)

```

```

6     if img is None:
7         print(f" 无法加载图像: {path}")
8         return
9
10    images.append(img)
11
12    num_images = len(images)
13
14    # cum_H_list[i] 表示第 i 幅图 (i=0 为基准) 到基准图 (第 0 幅图) 的变换矩阵
15    cum_H_list = [np.eye(3)]
16
17    # 依次计算第 i 图 (i>=1) 到第 i-1 图之间的单应矩阵, 然后累积转换
18
19    for i in range(1, num_images):
20
21        img_prev = images[i - 1]
22
23        img_curr = images[i]
24
25        # 转换为灰度图, 供 Harris 与 SIFT 使用
26
27        gray_prev = cv2.cvtColor(img_prev, cv2.COLOR_BGR2GRAY)
28
29        gray_curr = cv2.cvtColor(img_curr, cv2.COLOR_BGR2GRAY)
30
31
32        # 利用 Harris 角点检测提取候选关键点
33
34        harris_kp_prev = harris.extract_harris_keypoints(
35            np.float32(gray_prev), threshold_ratio=0.01
36        )
37
38        harris_kp_curr = harris.extract_harris_keypoints(
39            np.float32(gray_curr), threshold_ratio=0.01
40        )
41
42        # 用 SIFT 对候选关键点计算描述子
43
44        kp_prev, des_prev = sift.compute_sift_descriptors(gray_prev,
45            ↪ harris_kp_prev)
46
47        kp_curr, des_curr = sift.compute_sift_descriptors(gray_curr,
48            ↪ harris_kp_curr)
49
50        if des_prev is None or des_curr is None:
51            print(" 描述子计算失败")
52            return
53
54        # 使用 BFMatcher (交叉验证) 进行匹配
55
56        bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
57
58        matches = bf.match(des_prev, des_curr)
59
60        matches = sorted(matches, key=lambda x: x.distance)
61
62        if len(matches) < 4:
63            print(" 匹配点不足, 无法计算单应性")
64            return
65
66        # 构造匹配点对; 注意 findHomography 的参数顺序为: 将当前图像点转换到上一图
67        ↪ 像坐标系

```

```

42     pts_prev = np.float32([kp_prev[m.queryIdx].pt for m in
43                           matches]).reshape(
44                               -1, 1, 2
45 )
46     pts_curr = np.float32([kp_curr[m.trainIdx].pt for m in
47                           matches]).reshape(
48                               -1, 1, 2
49 )
50
51     H, mask = cv2.findHomography(pts_curr, pts_prev, cv2.RANSAC, 5.0)
52
53     if H is None:
54         print(" 无法求解单应性矩阵")
55         return
56
57     # 累积变换: 从当前图到基准图 = (从上一图到基准图) * (从当前图到上一图)
58     cum_H = np.dot(cum_H_list[-1], H)
59     cum_H_list.append(cum_H)
60
61     print(f" 第 {i+1} 张图与前一图匹配完成")
62
63     # 计算所有图像在基准图坐标系下的角点位置, 确定全局画布范围
64     all_corners = []
65
66     for i, img in enumerate(images):
67
68         h, w = img.shape[:2]
69
70         pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
71
72         pts_trans = cv2.perspectiveTransform(pts, cum_H_list[i])
73
74         all_corners.append(pts_trans)
75
76     all_corners = np.concatenate(all_corners, axis=0)
77
78     [xmin, ymin] = np.int32(all_corners.min(axis=0).ravel() - 0.5)
79     [xmax, ymax] = np.int32(all_corners.max(axis=0).ravel() + 0.5)
80
81     # 构造平移矩阵: 将所有图像平移使其全部位于正坐标区域
82     translation = np.array([[1, 0, -xmin], [0, 1, -ymin], [0, 0, 1]],

83     dtype=np.float32)
84
85     canvas_width = xmax - xmin
86
87     canvas_height = ymax - ymin
88
89     # 建立黑色画布, 尺寸为全局范围
90
91     panorama = np.zeros((canvas_height, canvas_width, 3), dtype=np.uint8)
92
93     # 将各幅图像投影到全局画布中 (依次叠加)
94
95     for i, img in enumerate(images):
96
97         warped = cv2.warpPerspective(
98
99             img, translation.dot(cum_H_list[i]), (canvas_width, canvas_height)
100
101         )
102
103         # 简单叠加: 非黑像素覆盖
104
105         mask = (warped > 0).sum(axis=2) > 0  # 找到非黑区域

```

```

79     panorama[mask] = warped[mask]
80     # 自动裁剪掉依然存在的黑边
81     panorama = auto_crop(panorama)
82     os.makedirs(os.path.dirname(output_path), exist_ok=True)
83     cv2.imwrite(output_path, panorama)
84     print(f" 多图拼接完成, 结果已保存至 {output_path}")

```

## 4 实验结果

### 4.1 Harri 角点算法

#### 4.1.1 结果展示

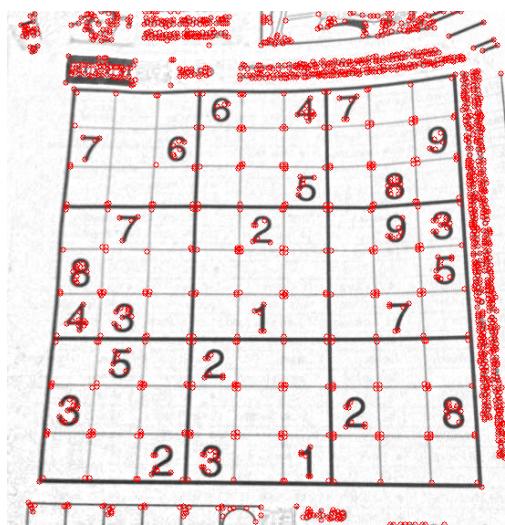


图 2: sudoku\_keypoints.png

#### 4.1.2 实验分析

检测过程中，我们利用 Sobel 算子计算图像的梯度信息，并进一步利用高斯滤波对梯度二次项进行平滑，从而降低噪声的干扰。随后，通过计算角点响应函数  $R = \det(M) - k \cdot (\text{trace}(M))^2$  得到每个像素点的响应值，再结合非极大值抑制对局部区域内的重复响应进行过滤，从而保留了局部极值作为候选角点。经过适当的后处理，最终检测结果成功地输出至 `results/1/sudoku_keypoints.png`。从实验结果可以看出，该方法能够较为准确地检测出图像中的明显角点（如交叉点和边界的拐角），同时去除了部分噪声和非角点的干扰。然而，在部分细微结构和纹理较弱的区域可能仍存在假阳性或者漏检情况。

## 4.2 关键点描述与匹配

### 4.2.1 结果展示



图 3: uttower1\_keypoints.jpg



图 4: uttower2\_keypoints.jpg



图 5: uttower\_match\_sift.png



图 6: uttower\_match\_hog.png

#### 4.2.2 实验分析

对于两幅图像（例如 `uttower1.jpg` 和 `uttower2.jpg`），我们采用基于 Harris 角点检测获得候选关键点，再分别用 SIFT 和 HOG 描述子进行特征计算。实验发现，使用 SIFT 描述子时，由于其 128 维的高维特征能够较全面地表达局部梯度变化信息，因此在不同尺度和旋转角度下的关键点也能获得较为稳定的描述；匹配过程中，相同关键点间的欧几里得距离表现出了较明显的差异性，从而有效过滤掉误匹配，得到了较多正确的匹配对。相比之下，HOG 描述子生成的特征维度较低，其主要依赖梯度方向直方图进行描述，对图像局部纹理的表达能力有限，导致在复杂场景下匹配对的相似度区分不够明显，从而使得误匹配较多。总体来说，关键点提取及匹配环节中，SIFT 能够提供更高的稳定性和鲁棒性，而 HOG 在计算效率上具有优势，但匹配稳定性和准确性相对较低。

### 4.3 全景图拼接

#### 4.3.1 二图拼接结果展示



图 7: `uttower_stitching_sift.png`



图 8: uttower\_stitching\_hog.png

#### 4.3.2 二图拼接实验分析

从拼接效果来看，基于 SIFT 的拼接结果 (uttower\_stitching\_sift.png) 具有较好的拼接精度和视觉一致性，其关键原因在于 SIFT 特征具有旋转、尺度不变性，在角点丰富、纹理明显的区域表现稳定，能提取出较为精确的关键点及其对应关系。

而基于 HOG 的拼接结果 (uttower\_stitching\_hog.png) 整体对齐效果略逊一筹。这是因为 HOG 特征本质上是对局部梯度方向进行统计编码，虽然能捕捉结构信息，但其缺乏对尺度、旋转变化的鲁棒性，在图像视角变化较大的情况下，匹配精度下降，容易出现匹配错误或漏匹配的情况。

### 4.3.3 多图拼接结果展示

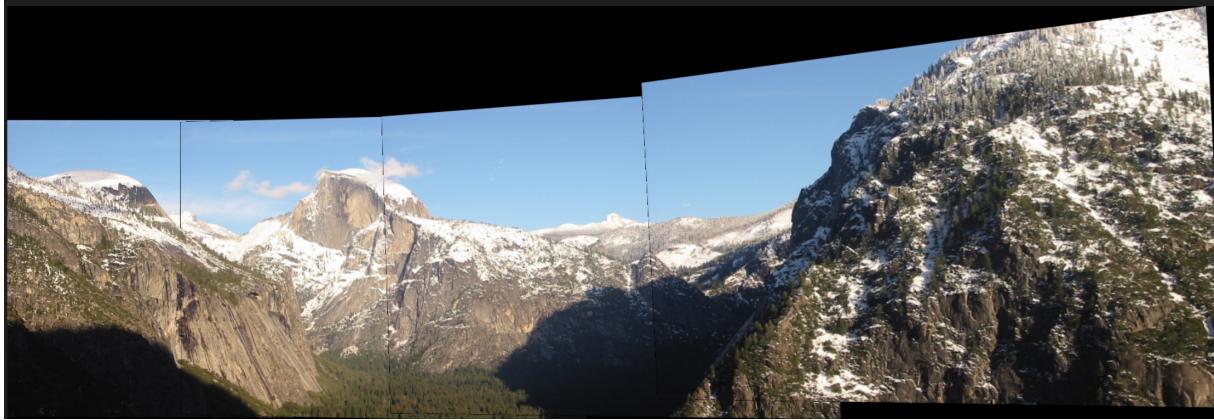


图 9: yosemite\_stitching.png

### 4.3.4 多图拼接实验分析

通过计算每两幅相邻图像之间的 SIFT 关键点匹配及 RANSAC 估计仿射变换，将各图像依次对齐并叠加至统一坐标系。从结果可见，多图拼接较好地还原了完整场景，图像之间接缝自然，对齐准确。这主要得益于 SIFT 特征在多图像序列中具有较强的一致性和稳定性，使得多幅图像间可建立可靠的几何约束关系。通过 RANSAC 滤除误匹配对，进一步保证了拼接精度。

## 5 实验总结

本次实验通过 Harris 角点检测、特征描述子提取（SIFT 与 HOG）、关键点匹配、RANSAC 变换估计及图像融合，涵盖了图像配准与拼接的完整流程。通过对两幅及多幅图像进行拼接实验，并结合不同特征方法的对比分析，我对各个步骤的原理和实现有了更深入的理解。在角点检测阶段，Harris 算法能较为稳定地提取出图像中灰度变化剧烈的区域，为后续特征描述子提取提供了较为可靠的关键点。在描述子提取方面，SIFT 特征因其具备尺度和旋转不变性，在面对视角变化和图像缩放时表现更为鲁棒，匹配精度较高；而 HOG 特征虽在目标检测领域表现良好，但用于关键点匹配时对图像几何变化的适应能力较弱，匹配效果不如 SIFT。在特征匹配过程中，利用 BFMatcher 和欧氏距离进行特征点匹配，再结合 RANSAC 方法对匹配对进行筛选，有效去除了错误匹配点，增强了仿射变换估计的准确性。在图像拼接过程中，我首先进行了两幅图像之间的拼接实验，结果表明基于 SIFT 特征的拼接在边缘对齐与图像连续性方面效果更佳；随后扩展至四幅图像的多图拼接，整体拼接结果较为自然，验证了 SIFT + RANSAC 方法在多图拼接任务中的有效性。通过本实验，我更深入理解了图像特征提取与匹配的基本方法，希望在后续的学习中更进一步。