Programming Assignment 3
3D重建

说明

1. **诚信与协作：**我们鼓励学生以小组形式工作，但每个学生必须提交自己的作品。如果你以小组形式工作，请在你的报告中写明合作者的名字。代码不应该被分享或复制。除非允许，否则请不要使用外部代码。抄袭是被强烈禁止的，并可能导致本课程的不及格。

2. **尽早开始!** 特别是不熟悉python的同学。

3. **问题：**如果你有任何问题，请先在Piazza上查看。其他学生可能也遇到过同样的问题，而且可能已经解决了。如果没有，请在讨论区发表你的问题。教学人员会尽快回复。

4. **写作:** 每个问题中都提到了需要写的部分。请注意，我们不接受你在本作业中的手写扫描件。请将你的理论问题的答案和实验的讨论打成电子版

5. **讲义:** 讲义zip文件包含3个项目。assgn3。pdf是作业讲义。data包含来自Middlebury MVS temple data集的2个寺庙图像文件，以及一些npz文件。Python包含了你将在作业中用到的脚本。

6. **提交:**创建一个压缩文件，`<andrew-id>.zip`，由你的文章、你的Python实现（包括辅助函数）和你的实现、结果组成。请确保删除data/文件夹、loadVid.py、helper.py和任何其他你生成的临时文件。你最终上传的文件应该按照这个布局排列：

- `<AndrewID>.zip`
  - `<AndrewID>.pdf`
  - `python`
    - `submission.py` (provided)
    - `helper.py` (provided)
    - `project_cad.py` (provided)
    - `test_depth.py` (provided)
    - `test_params.py` (provided)
    - `test_pose.py` (provided)
    - `test_rectify.py` (provided)
    - `test_temple_coords.py` (provided)
  - `data`
    - `im1.png` (provided)
    - `im2.png` (provided)
    - `intrinsics.npz` (provided)
    - `some_corresp.npz` (provided)
    - `temple_coords.npz` (provided)
    - `pnp.npz` (provided)
    - `extrinsics.npz` (generate)
    - `rectify.npz` (generate)

在上传zip文件到Canvas之前，请确保遵循上面提到的提交规则。违反此提交规则的作业将被处以总分数10%的处罚。

1 概述

计算机视觉的一个主要领域是三维重建。给定一些环境的2D图像，我们能否恢复环境的3D结构，以及相机/机器人的位置?这在机器人和自主系统中有很多用途，因为理解环境的3D结构对导航至关重要。你肯定不希望你的机器人经常撞到墙上，或者撞到人身上!

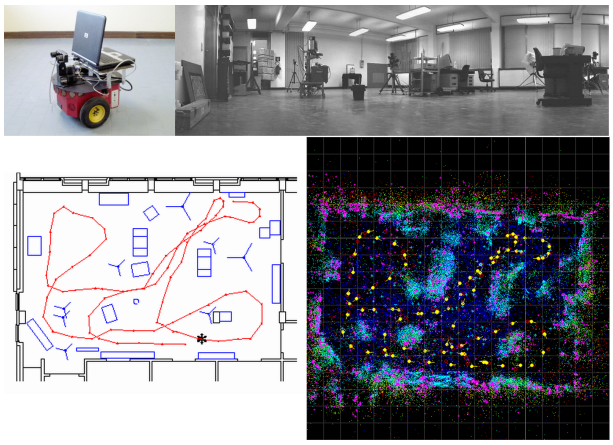

图1:机器人使用SLAM的例子，三维重建和定位算法

在这个作业中有两个编程部分:稀疏重建和密集重建。稀疏重构通常包含许多点，但仍然设法描述有问题的对象。密集重构是详细的和细粒度的。在3D建模和图形学等领域，当生成真实世界的物体和场景的3D模型时，极其精确的密集重建是非常具有价值的。

在第1部分中，你将编写一组函数，为我们提供给您的一些测试图像生成稀疏点云。测试图像是一个寺庙从两个不同角度的渲染。我们还为你提供了一个npz文件，其中包含两个图像之间良好的点对应关系。你将首先编写一个函数来计算两个图像之间的基本矩阵。然后编写一个函数，使用极线约束在两个图像之间寻找更多的点匹配。最后，你将编写一个函数，该函数将对每对2D点对应的3D点进行三角测量。

图2:我们提供给你的两张寺庙图片

我们为您提供了一些npz文件。文件 data/some_corresp.npz包含良好的点对应关系。你将用它来计算基本矩阵。文件 data/intrinsics.npz包含摄像机内部参数矩阵，你需要用他们来计算完整的投影矩阵。最后，文件 data/temple_coords.npz包含第一个图像上的一些点，应该很容易在第二个图像中定位。

在第二部分中，我们利用第一部分计算的外部参数进一步实现了该寺庙的致密三维重建。您需要计算校正参数。 我们为您提供了 test_rectify.py (和一些辅助函数)，你可以使用矫正函数对立体像对进行矫正。然后，您可以选择使用立体像对来计算视差图，最后计算密深度图。

在这两种情况下，都需要多个图像，因为如果没有两个大部分重叠的图像，这个问题在数学上是不明确的。基于同样的原因，生物学家认为人类和其他掠食性动物，如鹰和狗，有两个正面的眼睛。猎人在追逐猎物时需要能够辨别深度。另一方面，像鹿和松鼠这样的食草动物，它们的眼睛长在头部的两侧，牺牲了大部分的深度知觉来获得更大的视野。整个三维重建问题的灵感来自于人类和许多其他动物在导航和与环境互动时依赖圆顶深度感知的事实。给自主系统提供这些信息是非常有用的。

2 稀疏重建

在本节中，您将编写一组函数来计算寺庙的两个样本图像的稀疏重建。您将首先估计基本矩阵，计算点对应关系，然后在3D中绘制结果。

现在阅读第2.5节可能会有帮助。在第2.5节中，我们要求您编写一个运行整个流程的测试脚本。当你一个接一个地完成每一个问题时，把它们添加进去就容易多了。

2.1 实现八点算法 (10 points)

在这个问题中，你们将使用八点算法这在课上已经讲过了来估计基本矩阵。请使用 data/some_corresp.npz中提供的点对应关系; 你可以按如下方式加载和查看一个.npz文件
的内容:

```
data = np.load("../data/some_corresp.npz")
print(data.files)
```

写一个带有以下输入的函数:

```
F = eight_point(pts1, pts2, M)
```

其中 pts1 和 pts2 是 $N \times 2$ 的矩阵，分别对应第一幅和第二幅图像中N个点 (x,y)坐标，M为尺度参数。

- 归一化点和非归一化F:你应该通过使用变换矩阵T将每个坐标除以M(图像的宽度和高度的最大值)来缩放数据。在计算F之后，你将不得不“ 缩小 ”基本矩阵。如果 $x_{norm} = Tx$, 那么 $F_{unnorm} = T^T FT$。(注意:这个公式不应该与作业2中单应性归一化的方法相混淆。)
- 你必须在F上执行秩2约束。回想一下，一个有效的基本矩阵F会有所有的极线相交于某一点，这意味着F存在一个非平凡零空间。一般来说，对于实数点，F的8点解不会满足这个条件。为了加强秩2约束，用SVD分解F，得到三个矩阵$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ ，使 $\mathbf{F} = \mathbf{U\Sigma V}^T$.然后通过将 $\mathbf{\Sigma}$中的最小奇异值设置为零，使矩阵的秩为2，从而得到一个新的 $\mathbf{\Sigma}'$。现在用 $\mathbf{F}' = \mathbf{U\Sigma' V}^T$计算适当的基本矩阵。
- 你可能会发现使用局部极小化来改进解决方案很有帮助。这可能不会修复一个完全破碎的解决方案，但可能通过局部最小化几何代价函数使一个好的解决方案变得更好。为此，我们在helper.py中提供了一个函数refineF，它通过输入 F和两组点来修复F，你可以使用八点法时在缩放F之前使用它。
- 记住，图像中一个点的x坐标是它的列分量，y坐标是行分量。还要注意的是，八点法只是名字，只是意味着至少使用八个点，你的算法应该是一个过确定的系统(N>八个点)
- 为了可视化估计F的正确性，使用 python/helper.py中的displayEpipolarF函数，它输入的是F和两张图像。 这个GUI允许您在一个图像中选择一个点，并在另一个图像中可视化相应的极线 (图3)。

在你的文章中:请包括你恢复的F和一些极线的可视化(类似于图3)。

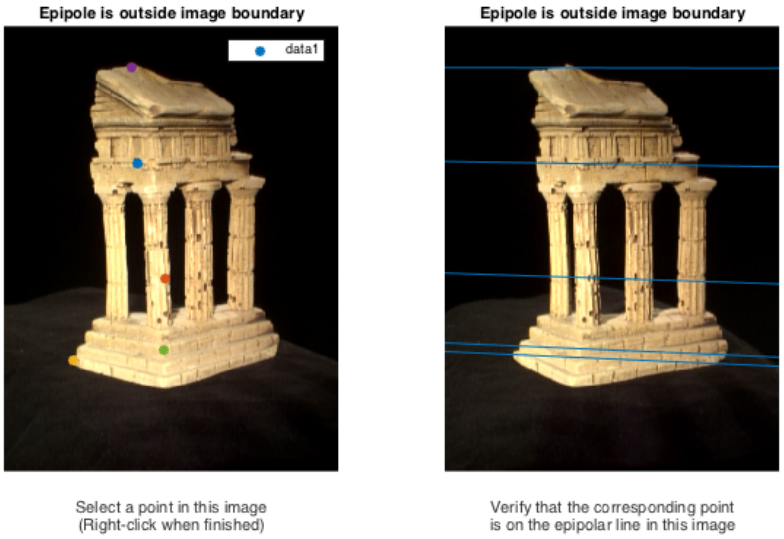

图3:displayEpipolarF的极线可视化

2.2 寻找极线对应关系 (20 points)

为了用一对立体图像重建三维场景，我们需要找到许多点对。点对是每个图像中对应同一个3D场景点的两个点。有了足够多的这些对，当我们绘制出最终的3D点时，我们将得到3D对象的粗略轮廓。在之前的家庭作业中，我们使用特征检测器和特征描述符找到了点对，并用另一个图像中的每一个点测试一个图像中的一个点。但是这里我们可以用基本矩阵来极大地简化这个搜索。

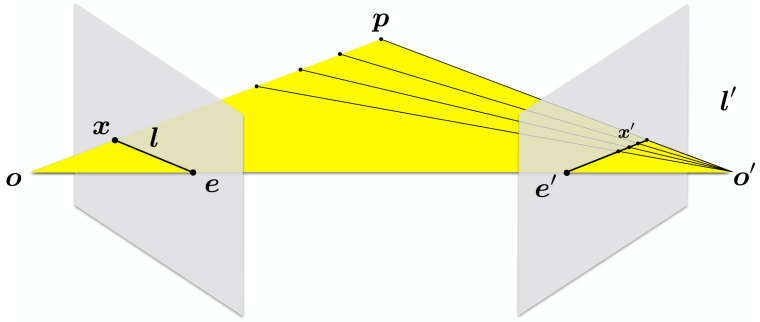

图4:极线几何(来源维基百科)

回顾一下课文，在一幅图像中给定一个点x（图4中的左视图）。它对应的三维场景点p可能位于从摄像机中心o到点x的任何地方。这条线与第二幅图像的摄像机中心o（图4中的右视图）形成一个平面。这个平面与第二台摄像机的图像平面相交，在第二幅图

像中形成一条线l，描述了x在第二幅图像中可能出现的所有位置。线条l是上极线，我们只需要沿着这条线搜索，就能找到第一幅图像中发现的x点的匹配。请写一个具有以下特征的函数：

```
pts2 = epipolar_correspondences(im1, im2, F, pts1)
```

其中 `im1` 和 `im2` 是双目图像对中的两个图像， `F`是使用八点法函数为这两个图像计算的基本矩阵， `pts1` 是一个$N \times 2$ 的矩阵，包含第一个图像中的 (x, y)点。函数应该返回 `pts2`, 一个包含第二个图像中的对应点的 $N \times 2$矩阵。

- 为了匹配图像1中的一个点 x，使用基本矩阵估计对应的极线 l'，并在第二幅图像中生成一组候选点。
- 对于每个候选点 x'，计算 x 和 x'之间的相似度分数。候选点中得分最高的点被视为极线对应点。
- 有很多方法来定义两点之间的相似性。你可以随意使用任何你想要的，并在你的文章中描述它。一种可能的解决方案是在点x周围选择一个大小为w的小窗口，然后将这个目标窗口与第二幅图像中候选点的窗口进行比较。对于我们给你的图像，简单的欧氏距离或曼哈顿距离应该足够了。
- 记住要注意数据类型和索引范围。

你可以在python/helper.py中使用函数epipolarMatchGUI来可视化地测试你的函数。你的函数不需要完美，但它应该得到最容易的点，如角，点等。

在你的文章中：请包含一个epipolarMatchGUI的截图，该截图与你的极线对应实现一起运行(类似于图5)。请提及您决定使用的相似性度量。还要评论一下你的匹配算法总是失败的情况，以及你认为为什么会这样。

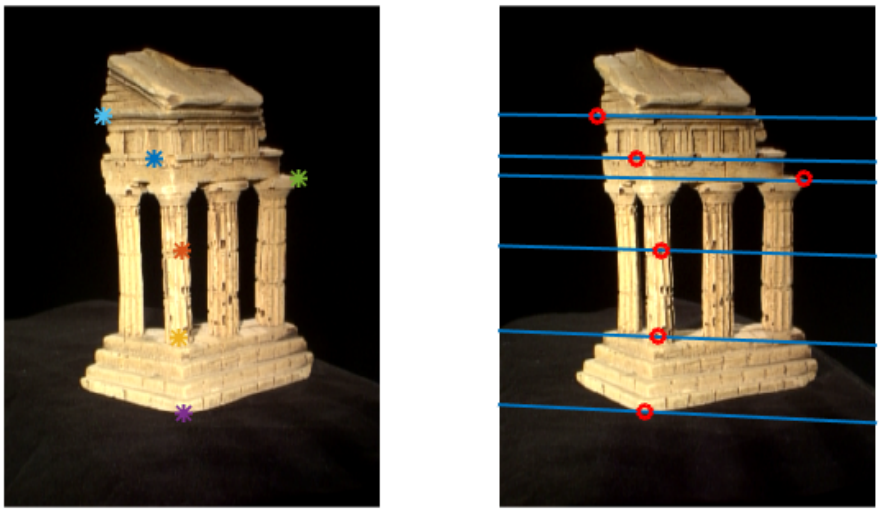

图5:极线匹配可视化。有一些错误是可以的，但它应该正确处理最简单的点(角、点等)

2.3 实现一个计算基本矩阵的函数 (10 points)

为了得到完整的摄像机投影矩阵，我们需要计算基本矩阵。到目前为止，我们只使用基本矩阵。写一个带有以下特征的函数:

```
E = essential_matrix(F, K1, K2)
```

其中 `F`为两幅图像之间计算的基本矩阵，`K1` 和 `K2`分别为第一幅和第二幅图像的相机内参矩阵(包含在`data/intrinsics.npz`中)， `E` 为计算的本质矩阵。相机的固有参数通常是通过相机标定获得的。有关本质矩阵和基本矩阵之间的关系，请参阅课堂幻灯片。

在你的文章中:请包括你估计的寺庙图像对的本质矩阵E。

2.4 实现三角化 (20 points)

编写一个函数，将图像中的2D点对三角化为一组3D点:

```
pts3d = triangulate(P1, pts1, P2, pts2)
```

其中pts1和pts2是带有2D图像坐标的N × 2矩阵，P1和P2是3 × 4相机投影矩阵，而pts3d是带有相应3D点的N × 3矩阵(在所有情况下，每行有一个点)。记住，您将需要将给定的内参矩阵与外部摄像机矩阵的解相乘，以获得最终的摄像机投影矩阵。对于P1，你可以假设没有旋转或平移，所以外部矩阵就是[I | 0]。对于P2，将基本矩阵传递给python/helper.py中提供的函数camera2，以获得四个可能的外部矩阵。您需要确定使用哪一个是正确的(参见2.5节中的提示)。有关一种可能的三角测量算法，请参阅课堂幻灯片。实现之后，通过查看重投影错误来检查性能。为了计算重投影误差，将估计的3D点投影回图像1，并计算投影的2D点与给定的pts1之间的平均欧氏误差。

在你的文章中:描述你如何确定哪个外部矩阵是正确的。注意，仅仅重新措辞提示是不够的。用给定的data/some_corresp.npz中的pts1和pts2报告你的重投影误差。如果实现正确，重投影误差应该小于2像素。

2.5 编写一个使用 data/temple_coords.npz的测试脚本 (10 points)

现在，您已经拥有了生成完整3D重建所需的所有部件。编写一个测试脚本

python/test_templecoords.py,执行以下操作:

1. 加载两个图像和从data/some_corresp.npz读取的对应点
2. 运行八点法计算基本矩阵 F
3. 从 data/temple_coords.npz加载在图像1上的点，之后运行你编写的 epipolar_correspondences函数，获得在图像2上的对应点。
4. 从 data/intrinsics.npz读取内参矩阵，计算本质矩阵 E。
5. 计算第一个摄像机投影矩阵 P_1，使用 camera2 函数计算 P_2的四个候选点。
6. 使用四组相机矩阵候选、data/temple_coords.npz中的点和他们计算出来的对应点运行你的 triangulate 函数。
7. 求出正确的 P_2 和对应的3D点。提示:你将从基本矩阵中得到camera2的4个投影矩阵候选矩阵。正确的配置是大多数3D点都在两个相机的前面(正深度)。
8. 使用matplotlib的 scatter 函数来绘制这些重建点。

9. 将计算出来的外部参数 (R1,R2,t1,t2)保存到 data/extrinsics.npz.
   这些外部参数将在下一节中使用。

我们将使用你的测试脚本来运行您的代码，因此要确保它运行平稳。特别是，使用相对路径加载文件，而不是绝对路径。

在你的文章中: 包含3张不同角度下对data/temple_coords.npz中的点进行重建后的重建点图片，如图6所示

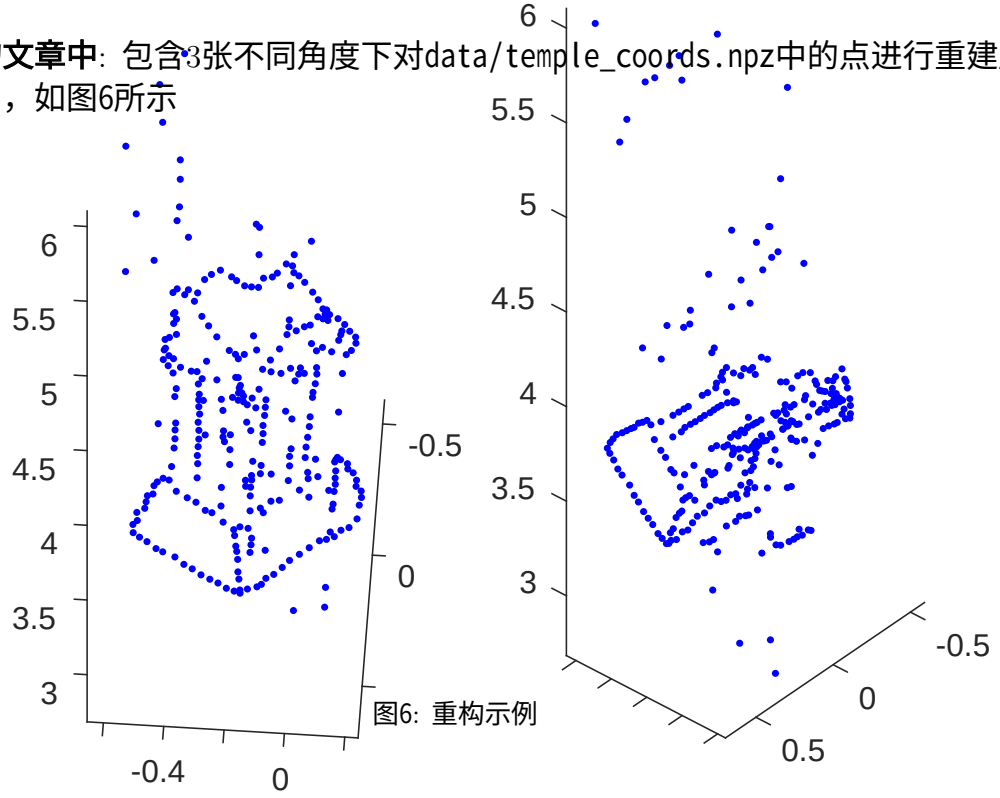

图6: 重构示例

3 稠密重建

在3D建模、3D打印、AR/VR等应用中，稀疏模型是不够的。当用户查看重构时，处理密集的重构会更令人愉快。要做到这一点，调整图像使匹配更容易是很有帮助的。

在本节中，您将编写一组函数来对我们的寺庙示例进行密集重建。给定所提供的内在参数和计算的外在参数，您将需要编写一个函数来计算两个图像的校正参数。经过校正的图像使得极线是水平的，因此寻找对应点就变成了简单的线性。对每个点都是这样。最后，可以计算视差和深度图。

3.1 图像校正 (10 points)

写一个程序来计算矫正矩阵。

```
M1,M2,K1p,K2p,R1p,R2p,t1p,t2p = rectify_pair(K1,K2,R1,R2,t1,t2)
```

这个函数接受左相机参数(K1,R1,t1)和右相机参数(K2,R2,t2)，并返回左(M1)和右(M2)校正矩阵和更新后的相机参数(K1p,R1p,t1p,K2p,R2p,t2p)。你可以使用提供的脚本python/test_correct.py来测试你的函数。根据我们在课堂上学到的知识，纠正配对函数应该连续执行以下步骤:

1. 计算每个相机的光学中心c1和c2: $\mathbf{c}_i = -(\mathbf{K}_i\mathbf{R}_i)^{-1}(\mathbf{K}_i\mathbf{t}_i)$
2. 计算新的旋转矩阵 $\widetilde{\mathbf{R}} = \begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3\end{bmatrix}^T$ 其中 $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3 \in \mathbb{R}^{3\times 1}$是分别表示相机参考系的 x、y 和 z 轴的正交向量。

   (a) 新的x轴(r1)平行于基线: $\mathbf{r}_1 = (\mathbf{c}_1 - \mathbf{c}_2)/\|\mathbf{c}_1 - \mathbf{c}_2\|$

   (b) 新的y轴(r2)与x轴和任意单位向量正交，我们将其设置为旧左矩阵的 z 单位向量: $\mathbf{r}_2$ = $\mathbf{R}_1(3,:)^T$ 和 $\mathbf{r}_1$ 的叉乘

   (c) 新的z轴(r3)正交于x轴和y轴: $\mathbf{r}_3$ = $\mathbf{r}_2$ 和 $\mathbf{r}_1$的叉乘

   设置新的旋转矩阵为 $\mathbf{R}_1' = \mathbf{R}_2' = \widetilde{\mathbf{R}}$
3. 计算新的相机内参矩阵为$\mathbf{K}_1' = \mathbf{K}_2' = \mathbf{K}_2$
4. 计算新的平移向量为 $\mathbf{t}_1 = -\widetilde{\mathbf{R}}\mathbf{c}_1$ 和 $\mathbf{t}_2 = -\widetilde{\mathbf{R}}\mathbf{c}_2$
5. 最后，可得到摄像机的矫正矩阵 $(\mathbf{M}_1, \mathbf{M}_2)$

$$_i'$$

完成之后，运行 python/test_rectify.py 。 这个脚本将使用提供的内部参数和计算的外部参数在寺庙图像上测试您的校正代码。它还将估计的矫正矩阵和更新的相机参数保存在 data/rectify.npz这将会在后面的脚本 python/test_depth.py中被使用。

在你的文章中:包括 python/test_rectify.py在寺庙图像上的结果截图。结果应该显示一些极线是完全水平的，两个图像中的对应点位于同一条线上。

3.2 密集窗口匹配确定每个像素的差距 (20points)

编写一个程序，从一对校正后的图像创建视差图。

```
dispM = get_disparity(im1,im2,max_disp,win_size)
```

其中 max_disp是最大的视差， win_size是窗口大小。输出dispM 的尺寸与 im1 和 im2相同。由于修正了 im1 和 im2，计算对应关系简化为一维搜索问题。 dispM(y, x)的值由下面公式获得：

$$\texttt{dispM}(y,x) = \underset{0 \le d \le \texttt{max\_disp}}{\operatorname{argmin}} \ \texttt{dist}(\texttt{im1}(y,x), \texttt{im2}(y,x-d)) \tag{1}$$

其中，

$$\texttt{dist}(\texttt{im1}(y,x), \texttt{im2}(y,x-d)) = \sum_{i=-w}^{w} \sum_{j=-w}^{w} (\texttt{im1}(y+i,x+j) - \texttt{im2}(y+i,x+j-d))^2 \tag{2}$$

$w = (\texttt{win\_size} - 1)/2$.通过使用 scipy.signal.convolve2d函数(即用一个1的掩码进行卷积)，可以很容易地计算出对窗口的这种求和。请注意，这并不是实现的唯一方法。

3.3 深度图 (10 points)

编写一个函数，从视差图创建深度图

```
depthM = get_depth(dispM,K1,K2,R1,R2,t1,t2)
```

我们可以利用

$$\texttt{depthM}(y,x) = b \times f / \texttt{dispM}(y,x) \tag{3}$$

其中b是基线， f相机的焦距。为了简单起见，假设 $b = \|c_1 - c_2\|$（即光学中心之间的距离），$f = \texttt{K1}(1,1)$。最后，当dispM$(y,x) = 0$时，就让depthM$(y,x) = 0$ 以避免除以0。现在可以使用 python/test_depth.py测试视差和深度图函数。确保矫正矩阵已保存(通过运行 python/test_rectify.py)。此函数将校正图像，然后计算视差图和深度图。

在你的文章中: 请包括视差和深度图的图像。

4 姿态估计

在本节中，你将实现你在课堂上所学到的知识，以估计相机的内在和外在参数，给定图像上的二维点x及其对应的三维点X。 换句话说，给定一组匹配的点{Xi, xi}和相机模型

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{1}\end{bmatrix} = f(\mathbf{X};\mathbf{p}) = \mathbf{P}\begin{bmatrix}\mathbf{X}\\ \mathbf{1}\end{bmatrix} \tag{4}$$

我们想要求解相机矩阵 $\mathbf{P} \in \mathbb{R}^{3\times4}$的估计值，以及内在参数矩阵 $\mathbf{K} \in \mathbb{R}^{3\times3}$、相机旋转矩阵 $\mathbf{R} \in \mathbb{R}^{3\times3}$ 和相机平移向量 $\mathbf{t} \in \mathbb{R}^3$，比如：

$$\mathbf{P} = \mathbf{K}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} \tag{5}$$

4.1 估计摄像矩阵 P (10 points)

写一个函数，在给定2D和3D点 $\mathbf{x}, \mathbf{X}$的情况下估计相机矩阵P。

```
P = estimate_pose(x, X)
```

其中 x 是 $2 \times N$ 矩阵，表示图像平面上 N 个点的 (x, y) 坐标， X 是 $3 \times N$矩阵，表示 3D 世界中对应点的 (x, y, z) 坐标。 回想一下，可以使用与直接线性变换 (DLT) 的单应性估计相同的策略来计算此相机矩阵。你可以运行提供的脚本python/test_pose.py来测试您的实现。

在你的文章中: 请包含脚本 python/test_pose.py的输出。

4.2 估计内部/外部参数 (20 points)

编写一个函数，从相机矩阵中估计内部参数和外部参数。

```
K, R, t = estimate_params(P)
```

从我们在课堂上学到的， estimate_params 函数应该运行以下步骤:

1. 通过使用SVD计算摄像机中心c。提示：c是最小的特征值所对应的特征向量。
2. 通过使用QR分解计算内部参数K和旋转矩阵R。K是一个右上角三角形矩阵，而R是一个正交矩阵。
3. 通过t=-Rc计算平移矩阵。

一旦完成了以上步骤，就可以运行提供的脚本 python/test_params.py。

在你的文章中:请包含脚本 python/test_params.py的输出。

4.3 将CAD模型投影到图像上 (20 points)

现在，您将利用已经实现的内容从一个真实的图像(如图7(左)所示)中估计摄像机矩阵，并将3D对象(CAD模型)(如图7(右)所示)投影到图像平面上。编写一个脚本 `python/project_cad.py` 完成以下工作:

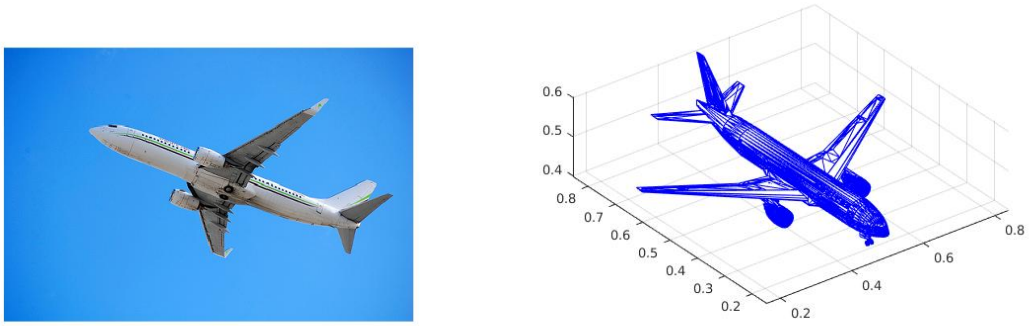

图7:提供的图像和3D CAD模型

1. 从数据文件 `data/pnp.npz`中加载一幅图像image、一个CAD模型cad、2D点x和3D点X。
2. 运行 `estimate_pose` 和 `estimate_params`以估计相机矩阵 `P`、内部参数矩阵 `K`、旋转矩阵 `R`和平移向量 `t`。
3. 使用你估计的相机矩阵 `P`，将给定的三维点X投射到图像上。
4. 绘制给定的二维点x和投影的三维点。图8（左）显示了一个例子。
5. 画出由你估计的旋转R旋转的CAD模型。例子见图8（中）。
6. 将CAD的所有顶点投影到图像上，并将投影的CAD模型与二维图像重叠。例子见图8（右）。

在你的文章中：请包含与图8类似的三张图片。您必须使用与图8不同的颜色。例如，绿圈为给定的2D点，黑点为投影的3D点，蓝色CAD模型，红色投影CAD模型在图像上重叠。如果你用同样的颜色，你将得不到分数。

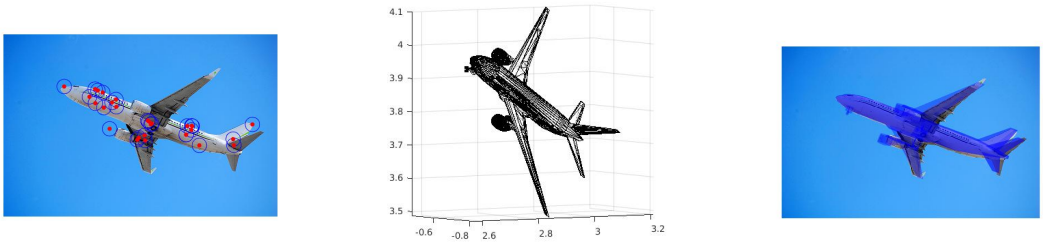

图8:将CAD模型投影回图像上。左:图像标注了给定的2D点(蓝圈)和投影的3D点(红圈);中:通过估计R旋转的CAD模型;右:与投影CAD模型重叠的图像。