中国研究生创新实践系列大赛

“华为杯”第十六届中国研究生

数学建模竞赛

题 目 视觉情报信息分析

摘 要：

本文从实际需求出发，针对单幅图像、平面视频和立体视频等几个场景进行距离信息以及相关速度等信息进行分析，建立了基于两灭点的相机标定模型，借助先验尺度信息完成单幅图像的目标定位和几何量测。综合运用逆透视变换、图像特征点匹配、运动结构恢复等模型进一步处理像平面与真实空间中的坐标转换，准确获取各位置点间距离以及运动物体速度等信息。

针对问题一，首先建立相机成像过程中的各相关坐标系转换关系，便于各位置点在不同坐标系下的变换；然后建立基于两灭点的相机标定模型，并借助环境中先验尺度信息完成单幅图像下的相机标定，获得内参及外参。基于相机参数以及坐标系间变换关系，建立相对深度求解模型，获得相机光心与图像坐标点真实空间中的相对距离。针对问题一中图像环境较为特殊的参照物与目标物不共面的情况，通过求解单应矩阵实现多个平面间的位置信息求解。最终结果图 1 中 AB 车头间距离为 23.504m，拍照者距离马路左侧边界的距离为 13.5m；图 2 中黑色车辆 A 车头和灰色车辆 C 车尾距离为 32.4m。拍照者距白色车辆 B 车头的距离 23.89m；图 3 中拍照者距岗亭 A 的距离 27.41m，拍照者距地面高度为 4.75m；图 4 中 AB = 5.2m、CD = 3.8m、AB 与 CD 距离为 8.4m。

针对问题二，因汽车后视镜为凸透镜故问题一中建立的标定模型不适合使用，因此提出基于逆透视变换的距离求解模型对两车距离进行求解，并且对视频中多时刻的距离信息均完成距离计算，更加准确把握两车的相对运动状态。对于题中汽车进行超越行为时速度差的取决提出基于特征提取、匹配并计算特征位移的模型在对图像帧连续计算，根据汽车尺寸作为先验尺度，求解出准确且连续的速度差数据。最终求解出平均车距为 51.716m，超越时平均速度差为 20.075km/h。

针对问题三，借助于前述基于两灭点的相机标定模型完成相机参数获取，同时利用运动结构恢复模型通过视频信息生成稠密的特征点云。此外，借助视频画面中符合国标的建筑信息完成尺度计算，从而将特征点的相对距离转换到真实空间中，从而易于求取各所需距离信息。最终，根据问题三要求解出桥距水面的距离 8.8m，桥距高铁轨道的距离为 485.3m，水面宽度为 195.6m，高铁行驶速度为 282.4km/h。

针对问题四，对于立体视频中距离的分析，结合环境中行人步伐等先验尺度信息，同时利用问题一以及问题三中建立的两灭点标定模型以及运动结构恢复模型将飞行器运动整个过程转化为环境稠密点云构建，通过 SFM 算法以及点标注两种方案对环境信息求解，得出最准确的数据。结果为老宅北侧长度 67.43m，老宅东侧长度 92.17m， 北侧道路宽度

2.26-2.37m，南侧道路长度 3.23-3.47m，东侧道路长度 4.43-6.53m，后花园短轴长度 32.86m，西面后花园处道路圆弧的长度 128.88m，最高树的高度 15.325m，建筑物的高度 11.492m，无人机飞行速度为 4.197m/s,飞行高度为 93.213m。建筑物部分占地面积 5971.55m^2，后花园占地面积 2358.187m^2，总占地面积 8329.742m^2。

关键词：两灭点标定、单应矩阵、逆透视变换、特征点匹配、SFM、对极约束

一、问题重述

视觉是我们平时获取各种信息的主要来源，同时图像和视频组成了承载视觉信息的主要载体。在视觉情报分析的相关工作中，主要就是从图像或者视频中获取的情报，比如其中一项重要内容就是从图像或视频中提取物体的大小、距离、速度等信息。

在历史中已有通过视觉情报分析获取机密信息的案例，如新中国的“照片泄密案”。当下，视觉信息在机器人、无人驾驶、军事侦察等方面发着越来越重要的角色。随着技术的发展，双目甚至多目视觉系统在视觉信息获取时有更优越的性能，但是在很多情况下我们只能利用普通的单目相机的图像或视频进行分析，需要我们综合考虑各种因素，通过合适的数学模型来提取。

本题从实际需求出发，选择单幅图像距离信息分析、平面视频距离信息分析和立体视频距离信息分析几个典型场景，研究以下几个问题：

（1）单幅图像距离信息分析：对给出的四张不同位置、不同角度、不同拍摄设备所拍出的普通图像，建立合适的数学模型求解出图中标记的不同点以及多处指定实际环境物之间的真实位置与距离，如图片中两辆汽车间的实际距离、拍摄高度以及塔体高度等在真实环境中的尺度数据。

（2）平面视频距离信息分析：从给定的视频中建立合适的数学模型分析运动的汽车之间的相对距离，以及发生相对运动的汽车的速度差等信息。

（3）平面视频距离信息分析：从给定的视频中建立合适的数学模型分析在运动的视角下的真实环境中的建筑物的尺度信息以及环境特征物间的距离信息和环境与拍摄者之间相对的拍摄者的运行速度等。

（4）立体视频距离信息：从给定的航拍视频中建立合适的数学模型，分析在航拍视角下建筑物、道路等真实的尺度信息以及计算建筑物面积，并估测航拍者（无人机）的运行状态，包括其速度和飞行高度。

二、模型假设

1、本赛题下所有任务中，拍摄设备均已完成校正，图像无桶形畸变、枕形畸变，无需再对图像或者视频材料进行去畸变的工作。
2、对于图像或者视频中出现的相关标志物，如汽车、道路标记线等均有国家标准或实际准确参数可提高参考；对出现的人物信息，可用中国公民平均身高等权威数据以供计算需求。
3、在任务 2 中视频中拍摄车超过第一辆白车时两车的相对位置是平行的。
4、图像的中心假设为摄像机在投影面上的投影中心。
5、对于任务 1 中图 4 的塔体底部看做平面，忽略地砖的粗糙问题。

三、符号说明

f，相机焦距；

R，旋转变换矩阵；
T，平移向量；
F_v，F_u，图像中灭点位置；
Zc, 相机光心到图像点的相对深度；

ε_d，灰度阈值；

minDis，最小汉明距离。

四、问题一分析与求解

4.1 问题一分析

问题一中四个子任务均是针对单幅图像进行建模分析，最终需要解出图片中的物体间或者指定位置在真实空间的距离信息。具体要求是测算图 1 中红色车辆 A 车头和白色车辆 B 车头之间的距离、拍照者距马路左侧边界的距离；图 2 中黑色车辆 A 车头和灰色车辆 C 车尾之间的距离以及拍照者距白色车辆 B 车头的距离；图 3 中拍照者距岗亭 A 的距离以及拍照者距离地面的高度；图 4 中塔体正面(图中四边形 ABCD)的尺寸，即 AB 和 CD 的长度以及 AB 和 CD 之间的距离 (已知地砖尺寸为 80cm×80cm)。

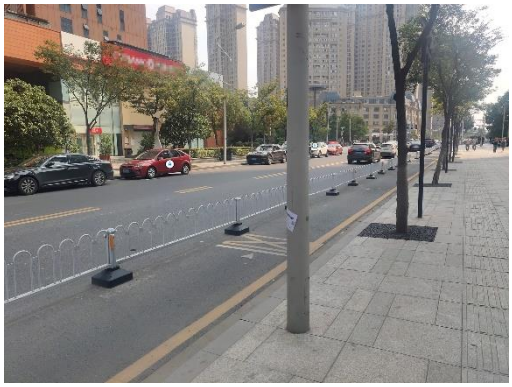


图 1

图 2

图 3

图 4

对题目中给出的条件以及题目要求进行分析，首先该题的一大特点是仅有普通单目相机拍摄出的 RGB 图像，可以获取得到图像的横纵像素数，但是拍摄的相机的具体参数均未知。此外，题目要求获得图像中两点或指定两处位置间的真实空间距离，从给出的相关信息来看，缺少准确的尺度信息，若直接进行建模求解两点位置也无法获得真实的尺度信息。因此，该题主要就是单目视觉测量的问题求解。

基于单幅图像的量测方法（单目视觉测量系统）是近几年发展起来的新型图像测量系统。与双（多）目测量系统相比，单目视觉量测无需图像匹配，也不需事先了解相机参数，从而使得测量过程灵活方便，已成为一个较为热门的研究方向。由于一幅图像无法完全重构出对应的三维模型，针对这一点，众多研究者利用图像中已知的几何信息对目标物体进行三维重建，实现目标在世界坐标系中的定位。

与目前在视觉信息研究与应用中常用的双像测量的立体模型不同，因为在成像过程中单幅图像缺失深度信息，所以如需进行环境中的几何量测，需要依赖于由相应物方空间提

供一些已知信息，包括足够的控制点或者空间对象的几何结构信息和先验度量信息。控制点是基于物方和像方的映射关系，并通过图像点坐标求取对应的物方坐标而实现几何量测，称之为间接法或基于单应的量测；几何结构信息和先验度量信息是通过中心投影的不变量的特性并结合已知的几何结构信息进行空间对象的量测，称之为直接解法或者基于几何关系的量测。

在本题中，经过对图片包含信息的分析，可以确定多处先验度量信息，如图 1 中红车 A 为福克斯两厢车型，轴距为 2648mm，该参数值准确且在图像的像素坐标系中便于定位；同理，在图 2 中的黑车 A 为凯迪拉克 XT5，轴距为 2857mm；在图 3 中人形横道预告标识线，长度为 3m;在图 4 中，已知条件地砖的尺寸为 800mm*800mm,并且通过对国家权威数据的查询，估测图 4 中出现的人物实际高度为 1600mm,以上信息均可作为问题求解前的先验信息。通过适当的数学模型的建立并结合已知的先验度量信息，方可实现单幅图像单目视觉的测距。

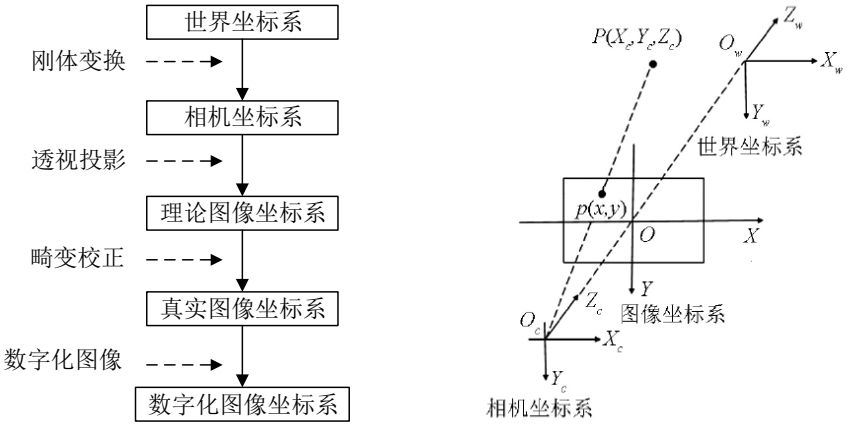
在先验信息的基础上，若想对物体进行三维重建以获得其在真实空间的位置进而求得两点间实际距离，需要借助已有信息完成对相机的标定，以此来获得相机的焦距即内参，旋转矩阵 R 与偏移向量 T 即外参。相机的标定也有多种解决方案，如传统标定法、基于主动视觉的标定法、自标定法以及基于灭点标定法。通过对以上各方法对场景等因素的要求的评估并结合问题中仅有单幅图像的实际情况，本题对图 1、2、3 的相关问题求解采用基于单幅图像中存在两个灭点的标定方法的数学模型，进行相机内外参数的求解。完成标定获得相应参数后，可建立世界坐标系、相机坐标系、图像坐标系以及像素坐标系之间的变换关系，从而借助几何数学模型完成最终深度方向上相对距离的求解，进而通过坐标点之间的运算求解问题中所要求的各个距离、高度等数据。

针对图 4，由于图像中灭点的确定以及验证灭点是否准确的工作较为复杂，为简化计算并保证准确度，采用计算多个世界平面与像平面之间的变换关系即单应矩阵，完成世界平面上的点与图像点的对应，通过四组匹配点求出单应矩阵 H 后确定图像坐标对应的世界点的欧氏坐标，从而计算出世界平面上两点距离。本题中由于指定点与参照物不共面，多次进行了世界平面的选择，从而求出最终解。

4.2 模型建立

4.2.1 位置变换模型建立

对于本题任务从单幅图像中分析距离信息来说，首先确定在相机成像的过程中，空间、平面间的位置变换关系，并在此后借助先验尺度信息恢复具体位置。相机成像过程中的变换关系如图 5 所示。



（a）成像过程　　（b）各坐标系空间示意

图 5 相机成像过程位置变换关系示意图

基于上述对相机成像原理的总结，可以得出图像坐标系下的成像点与其在世界坐标下的完整的变换关系如下式所示，这也是本题从图像中获得真实空间尺度信息的重要基础模型。

$$Z_C\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}\frac{1}{dx}&0&u_0\\0&\frac{1}{dy}&v_0\\0&0&1\end{bmatrix}\begin{bmatrix}f&0&0&0\\0&f&0&0\\0&0&1&0\end{bmatrix}\begin{bmatrix}R&T\\\vec{0}&1\end{bmatrix}\begin{bmatrix}X_W\\Y_W\\Z_W\\1\end{bmatrix}=\begin{bmatrix}f_x&0&u_0&0\\0&f_y&v_0&0\\0&0&1&0\end{bmatrix}\begin{bmatrix}R&T\\\vec{0}&1\end{bmatrix}\begin{bmatrix}X_W\\Y_W\\Z_W\\1\end{bmatrix} \quad (1)$$

上式中(u,v)为像素坐标系的坐标，(X_W,Y_W,Z_W)为世界坐标系中的坐标，$O\to O_c$即为焦距，f同样也是焦距，R,T为旋转矩阵和平移向量。在此模型下，建立了图像到真实空间的变换关系，从而可以进行从二维图像平面到三维空间的转换，对具体数值需继续建立模型对式（1）中相关参数进行求解。

4.2.2 基于两灭点的相机标定

根据相机成像中的透视原理，在世界空间中的两条平行线在成像平面的投影一定会在图像坐标系中交于一点，这个交点也称为灭点。假设三维空间中的一点W_0,方向矢量为**d**,D是空间中该方向上过点W_0的一条直线。W 是直线上任意一点，该直线可以用W_0和**d**表示为：

$$W = W_0 + l \times \boldsymbol{d} \quad (2)$$

上式中l 为W_0和W之间的距离。

因此，可以将上述三维空间中的点W和W_0投影到图像平面上从而得到投影点q_0和q。由于灭点所对应的空间位置为无穷远处，故l为无穷大，图像平面上直线 D 对应的灭点坐标$F_D(x_\infty,y_\infty,z_\infty)$为：

$$\begin{cases}x_\infty=\lim\limits_{l\to\infty}f\dfrac{a_x+l\cdot d_x}{a_z+l\cdot d_z}=f\dfrac{d_x}{d_z}\\y_\infty=\lim\limits_{l\to\infty}f\dfrac{a_y+l\cdot d_y}{a_z+l\cdot d_z}=f\dfrac{d_y}{d_z}\\z_\infty=f\end{cases} \quad (3)$$

上式中f为相机焦距，根据上式可以知道灭点的图像坐标只与对应空间直线的方向矢量有关，并且在三维空间中有且只有一个灭点F_D和直线 D 平行。对于本题中图 1、2、3 的场景，结合真实世界中的建筑规范，如直行车道、地砖、楼房窗户等规则图形的两边平行的特性，均可快速找到两组相互正交且与成像平面不平行的平行线。设从两组平行线中提取出两组对应的直线并在空间中构成空间矩形 ABCD,该矩形投影到图像平面的图形为 abcd,具体示意图如图 6 所示，可以直观的看到图像平面的直线 ad 与 bc 所形成的灭点F_v、F_u。

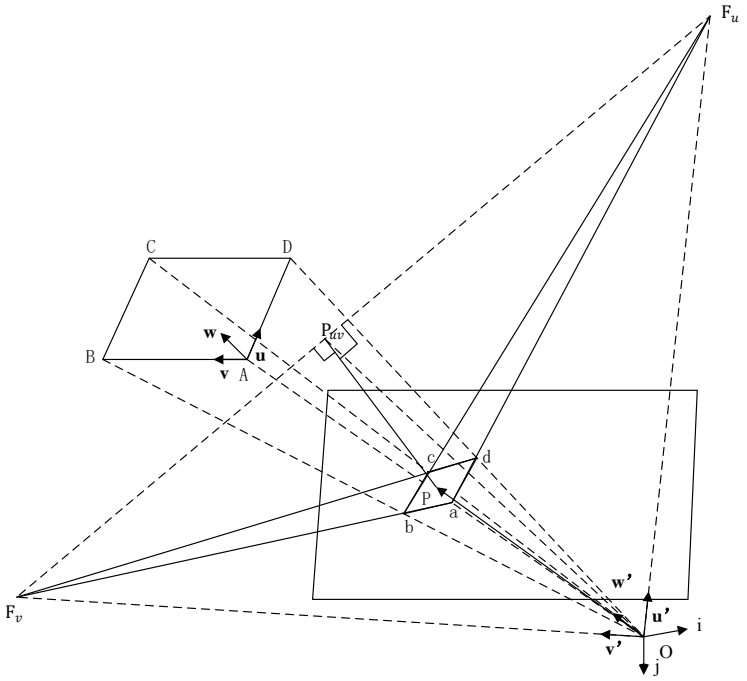


图 6 两灭点法标定示意图

根据上图所示，将Auvw记为世界坐标系R_0、$R_{s(ij)}$为图像坐标系、$R_{c(Oij)}$为相机坐标系并假设相机中心 O 在图像平面的投影点为 P 点。连接图像坐标系中的两个灭点，并过 P 点做垂直与灭点间连线的直线交点为P_{uv}，可以从图中得到$\Delta\mathrm{OPP}_{uv}$为直角三角形，因此根据三角形的勾股定理求得：

$$f = OP = \sqrt{O\mathrm{P}_{uv}{}^2 - \mathrm{PP}_{uv}{}^2} \qquad (4)$$

针对上式，由于P、F_v、F_u的坐标在图像坐标系中已知，利用$\mathrm{PP}_{uv} \perp \mathrm{F}_v\mathrm{F}_u$的条件求出$\mathrm{P}_{uv}$的坐标从而解算出$\mathrm{PP}_{uv}$的长度。通过图 6 中各连线的垂直关系，以及直角三角形$\Delta\mathrm{OF}_u\mathrm{P}_{uv}$、$\Delta\mathrm{OF}_v\mathrm{P}_{uv}$、$\Delta\mathrm{OF}_u\mathrm{F}_v$的相似关系可以求得：

$$O\mathrm{P}_{uv} = \sqrt{F_v\mathrm{P}_{uv} \times \mathrm{P}_{uv}F_u} \qquad (5)$$

将式（4）代入式（3）中可计算出相机的焦距为：

$$f = OP = \sqrt{F_v\mathrm{P}_{uv} \times \mathrm{P}_{uv}F_u - \mathrm{PP}_{uv}{}^2} \qquad (6)$$

计算出相机的内参f后，对外参进行求解，计算出从世界坐标系到相机坐标系的转换，包括旋转矩阵 **R** 和平移向量 **T**。由灭点的性质可知，$\mathrm{O}F_u \parallel AD \parallel BC$、$\mathrm{O}F_v \parallel AB \parallel DC$，因此世界坐标系与相机坐标系的变换矩阵等同于以**Ov′**、**Ou′**、**Ow′**为坐标轴的等效坐标系和相机坐标系间的变换矩阵，可以求得：

$$\begin{cases} u' = \dfrac{\overrightarrow{OF_u}}{\left\|\overrightarrow{OF_u}\right\|} = \left(\dfrac{F_{ui}}{\left\|\overrightarrow{OF_u}\right\|}, \dfrac{F_{uj}}{\left\|\overrightarrow{OF_u}\right\|}, \dfrac{f}{\left\|\overrightarrow{OF_u}\right\|}\right) \\ v' = \dfrac{\overrightarrow{OF_v}}{\left\|\overrightarrow{OF_v}\right\|} = \left(\dfrac{F_{vi}}{\left\|\overrightarrow{OF_v}\right\|}, \dfrac{F_{vj}}{\left\|\overrightarrow{OF_v}\right\|}, \dfrac{f}{\left\|\overrightarrow{OF_v}\right\|}\right) \\ w' = u' \times v' \end{cases} \tag{7}$$

已知世界坐标系下$\mathrm{u} = (1,0,0), \mathrm{w} = (0,0,1)$，则可求出旋转变换矩阵：

$$R = \begin{bmatrix} \dfrac{F_{ui}}{\sqrt{F_{ui}^2 + F_{uj}^2 + f^2}} & \dfrac{F_{vi}}{\sqrt{F_{vi}^2 + F_{vj}^2 + f^2}} & w_i' \\ \dfrac{F_{uj}}{\sqrt{F_{ui}^2 + F_{uj}^2 + f^2}} & \dfrac{F_{vj}}{\sqrt{F_{vi}^2 + F_{vj}^2 + f^2}} & w_j' \\ \dfrac{f}{\sqrt{F_{ui}^2 + F_{uj}^2 + f^2}} & \dfrac{f}{\sqrt{F_{vi}^2 + F_{vj}^2 + f^2}} & w_k' \end{bmatrix} \tag{8}$$

对于相机外参中的平移向量，仅凭图像坐标系的像素坐标等信息无法求出，需要借助图像中参照物的先验尺度信息建立从世界坐标系到相机坐标系的映射关系，并提供尺度标准。本题中根据图 1、2、3 分别选取可准确获得其真实尺度的参照物并以其中某一点作为世界坐标系的原点记为 A，同时真实中间的以 A 为起点且具体数值已知的线段记为 AD，投影关系如图 7 所示：

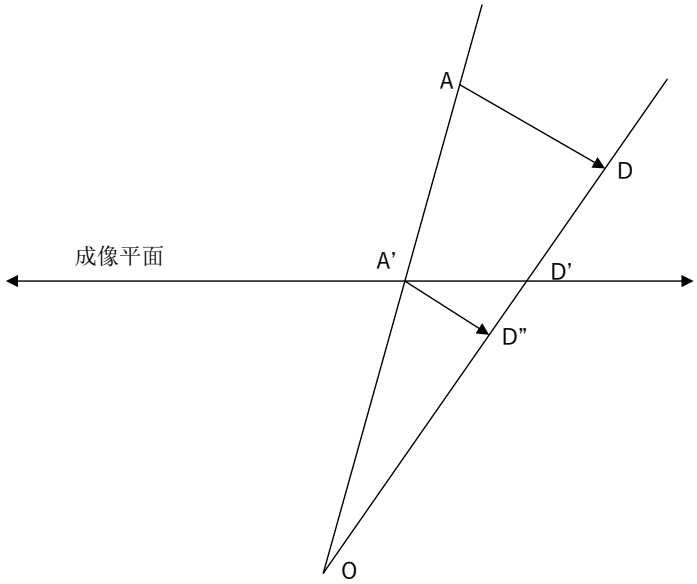


图 7 平移量计算示意图

上图中A'是A的投影，$\overrightarrow{AD'}$是$\overrightarrow{AD}$的投影。由于 AD 的真实距离已知，记作l，并记世界坐标系下有$\overrightarrow{AD}_{R_O}$、相机坐标系下有$\overrightarrow{AD}_{R_C}$，有$\overrightarrow{AD}_{R_C} = R \cdot \overrightarrow{AD}_{R_O}$。图中过$\mathrm{A}'$作一平行于$\overrightarrow{AD}$的直线交 OD 于点$D'$，借助$\Delta \mathrm{OA}D'$、$\Delta \mathrm{OA}D$的相似关系可以得出：

$$\frac{\left\|\overrightarrow{AD'_{RC}}\right\|}{\left\|\overrightarrow{AD_{RC}}\right\|}=\frac{\left\|\overrightarrow{OA'_{RC}}\right\|}{\left\|\overrightarrow{OA_{RC}}\right\|} \qquad (9)$$

因此

$$\left\|\overrightarrow{OA_{RC}}\right\|=\frac{\left\|\overrightarrow{OA'_{RC}}\right\|\bullet\left\|\overrightarrow{AD_{RC}}\right\|}{\left\|\overrightarrow{AD'_{RC}}\right\|} \qquad (10)$$

$$\overrightarrow{OA_{RC}}=\left\|\overrightarrow{OA_{RC}}\right\|\frac{\overrightarrow{OA'_{RC}}}{\left\|\overrightarrow{OA'_{RC}}\right\|} \qquad (11)$$

式（11）$\overrightarrow{OA_{RC}}$ 中记为所需要求解的平移向量。

4.2.3 相对深度计算

通过上述模型可借助图像中的先验尺度信息求解出相机的内参和外参，包括焦距以及旋转矩阵和平移向量，但对于式（1）中的Z_C，即相对深度信息的值还需要建立模型进行求解。根据式（1），给出旋转矩阵 R 和平移向量 T 的具体形式：

$$R=\begin{bmatrix}R_{11} & R_{12} & R_{13}\\ R_{21} & R_{22} & R_{23}\\ R_{31} & R_{32} & R_{33}\end{bmatrix},\quad T=\begin{bmatrix}x\\ y\\ z\end{bmatrix}$$

在大部分情况下，选择为地平面作为世界坐标系中 XOY 的平面，因此在该平面上的点$Z_W=0$，将R、T和Z_W带入式（1）可得：

$$Z_C\begin{bmatrix}u\\ v\\ 1\end{bmatrix}=\begin{bmatrix}f & 0 & u_0 & 0\\ 0 & f & v_0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}R_{11} & R_{12} & R_{13} & x\\ R_{21} & R_{22} & R_{23} & y\\ R_{31} & R_{32} & R_{33} & z\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}X_W\\ Y_W\\ 0\\ 1\end{bmatrix} \qquad (12)$$

化简可得：

$$Z_C\begin{bmatrix}u\\ v\\ 1\end{bmatrix}=\begin{bmatrix}aX_W+bY_W+c\\ dX_W+eY_W+f\\ R_{31}X_W+R_{32}Y_W+z\end{bmatrix} \qquad (13)$$

其中，a、b、c、d、e、f是对矩阵中参数的简化表示，各参数的具体为:

$a=fR_{11}+u_0R_{31}$，$b=fR_{12}+u_0R_{32}$，$c=fx+u_0z$，$d=fR_{21}+v_0R_{31}$，$e=fR_{22}+v_0R_{32}$，$f=fy+v_0z$。

式（13）中，X_W、Y_W、Z_C为未知数，整个式子构成三元一次方程组，可以解得:

$$Z_C=\frac{CD-AB}{A(\mathrm{u}_0-\frac{b}{R_{32}})-C(v_0-\frac{c}{R_{32}})} \qquad (14)$$

其中，由于用a、b、c、d、e、f最终结果过于繁琐，用A、B、C、D作为参数进行公式形式上的简化，各参数的具体含义分别为：

$A=d-\frac{R_{31}}{R_{32}}e$，$B=-c+\frac{z}{R_{32}}b$，$C=a-\frac{R_{31}}{R_{32}}b$，$D=-f+\frac{z}{R_{32}}e$。

由此，因为焦距以及旋转变换矩阵和平移向量的各参数已通过先前模型求解得出，同时一副图像的像素大小已知，故(u_0,v_0)已知，对于问题一中图 1、2、3 的题目要求，可根据变换关系计算出在世界坐标系下中 *XOY* 平面上的点，进而求真实空间中的相对距离。

4.2.4 多平面转换模型建立

在问题一中，给出的图 4 为一座小型塔体照片，要求借助地面地砖的具体尺寸信息测算出关于塔身的相关尺寸数据。图 4 中图像的拍摄场景以及角度与问题一图 1、2、3 中的明显区别在于参照物与所需要求解距离的图中物体没有共同接触的平面，并且根据对塔体的研究，塔身自身又存在一个倾斜的平面，而非垂直与地面，因此如果依然借助之前的模型进行求解，过于繁琐且无法忽视倾斜平面带来的误差问题，因此建立针对问题给出的图 4 的具体模型进行求解。

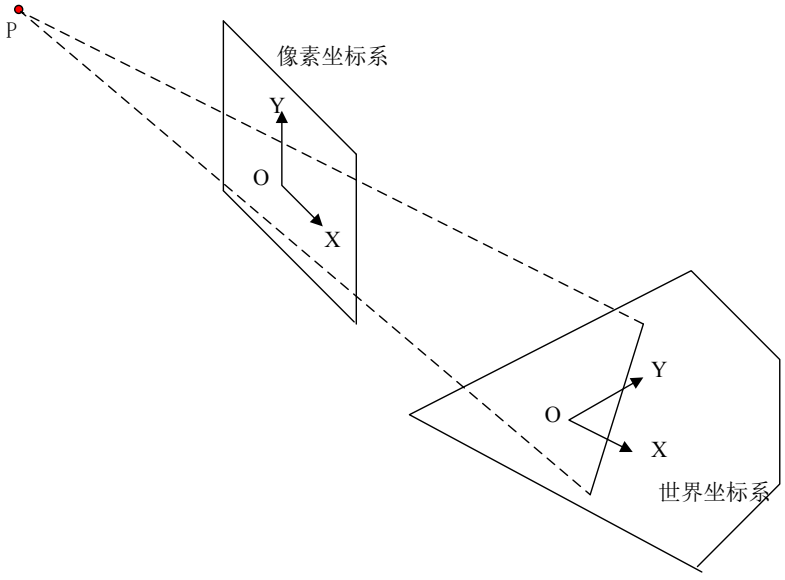


图 8 平面摄像机模型

世界平面上的点，设为$X=(X_W,Y_W,Z_W)^T$与它在图像平面上的点$x=(x,y,1)^T$的对应关系可以用一个3×3齐次矩阵 H 来描述：

$$X=Hx \qquad (15)$$

H 也称为单应矩阵，自由度为 8。同理，由点到线，世界平面上的一条直线与在图像平面上对应的直线l的对应关系同样可以用该单应矩阵描述。

设单应矩阵的具体形式为：$H=\begin{pmatrix}h_{11} & h_{12} & h_{13}\\ h_{21} & h_{22} & h_{23}\\ h_{31} & h_{32} & h_{33}\end{pmatrix}$，通过代入式（15），可以求解出世界坐标系下的点利用单应矩阵参数以及像素平面上点表示的形式：

$$\begin{cases} X = \dfrac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \\ Y = \dfrac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \end{cases} \tag{16}$$

显然，平面摄像机模型可以由单应矩阵 H 所确定。单应矩阵!可直接从对应点或者对应线来计算。一旦确定了单应矩阵，就可以对图像上由公式（16）确定它对应的世界点的欧氏坐标，于是可以根据下述公式（17）计算出世界平面上的两点间距离：

$$d(X_1, X_2) = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \tag{17}$$

对于单应矩阵的求解主要依赖于先验信息进行图像平面与世界平面的匹配点约束，一组匹配点可以构造出三项约束，但因为需要线性相关故只取前两个，因此单应矩阵可以通过 4 对匹配特征点算出，并且进行匹配的特征点不能有三点共线的情况出现，假设有 4 对符合条件的匹配点 $(u_1^1, v_1^1), (u_1^2, v_1^2), (u_1^3, v_1^3), (u_1^4, v_1^4)$ 对应于 $(u_2^1, v_2^1), (u_2^2, v_2^2), (u_2^3, v_2^3), (u_2^4, v_2^4)$，具体的求解如下所示：

$$\begin{pmatrix} u_1^1 & v_1^1 & 1 & 0 & 0 & 0 & -u_1^1 u_2^1 & -v_1^1 u_2^1 \\ 0 & 0 & 0 & u_1^1 & v_1^1 & 1 & -u_1^1 v_2^1 & -v_1^1 v_2^1 \\ u_1^2 & v_1^2 & 1 & 0 & 0 & 0 & -u_1^2 u_2^2 & -v_1^2 u_2^2 \\ 0 & 0 & 0 & u_1^2 & v_1^2 & 1 & -u_1^2 v_2^2 & -v_1^2 v_2^2 \\ u_1^3 & v_1^3 & 1 & 0 & 0 & 0 & -u_1^3 u_2^3 & -v_1^3 u_2^3 \\ 0 & 0 & 0 & u_1^3 & v_1^3 & 1 & -u_1^3 v_2^3 & -v_1^3 v_2^3 \\ u_1^4 & v_1^4 & 1 & 0 & 0 & 0 & -u_1^4 u_2^4 & -v_1^4 u_2^4 \\ 0 & 0 & 0 & u_1^4 & v_1^4 & 1 & -u_1^4 v_2^4 & -v_1^4 v_2^4 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{pmatrix} = \begin{pmatrix} u_2^1 \\ v_2^1 \\ u_2^2 \\ v_2^2 \\ u_2^3 \\ v_2^3 \\ u_2^4 \\ v_2^4 \end{pmatrix}$$

通过直接线性变换可以解出单应性矩阵的具体值，从而针对图像中的相应点求解在世界平面下的欧式坐标。在本题的图 4 中，题目给出的参考物地砖与所需要求的塔体的相关位置在真实空间中不在同一平面，且塔体正面向内倾斜，无法简单的利用投影将塔体顶部位置垂直投影到底部，因此借助单应性矩阵可联系图像平面与世界平面的特性，将问题分解为多个平面间的位置坐标转换，如首先以参考地砖所在平面，以地砖的四个角作为匹配点，从而求出单应矩阵后计算地砖平面中台阶的底部两端距离，并将该距离等效于台阶顶部的两端距离，再以塔体底部的 A、B 两点以及台阶顶部两端点所在平面为世界平面，借助先前求出的尺度信息求出该平面下单应矩阵，进而算出 A、B 两点的欧式坐标，计算出 AB 的真实距离。最后，由于 AD、BC 组成的平面不垂直与底面，再以该平面作为世界平面，借助之前求出的 AB 距离以及人物身高作为尺度信息，求解单应矩阵并计算出 CD 间真实距离以及 AB、CD 间的距离。

4.3 模型求解

4.3.1 问题一图 1 结果

图 1 中的环境简易示意图如图 9 所示，测算红色车辆 A 车头和白色车辆 B 车头之间的距离、拍照者距马路左侧边界的距离：

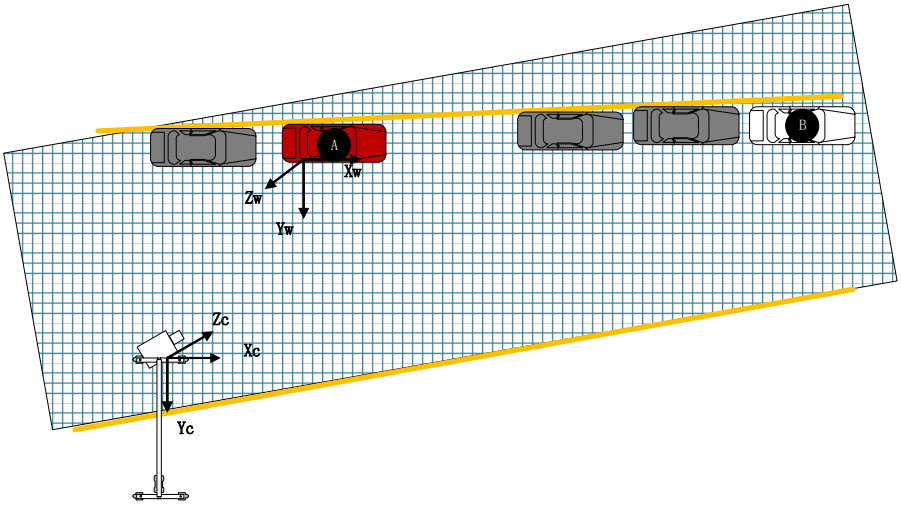


图 9 问题一图 1 简易图

图中红色车位福克斯两厢车型，故通过相关资料可以获取其轴距数据为 2648mm，因此选择以红车前轮与地面接触点为世界坐标系的原点，通过 4.2 节所建立的模型，首先求解出该图像中的两个灭点F_u、F_v，像素坐标系中为（10745,9708），（7069,9819），根据求出的灭点位置在图中进行标注如图 10 所示。

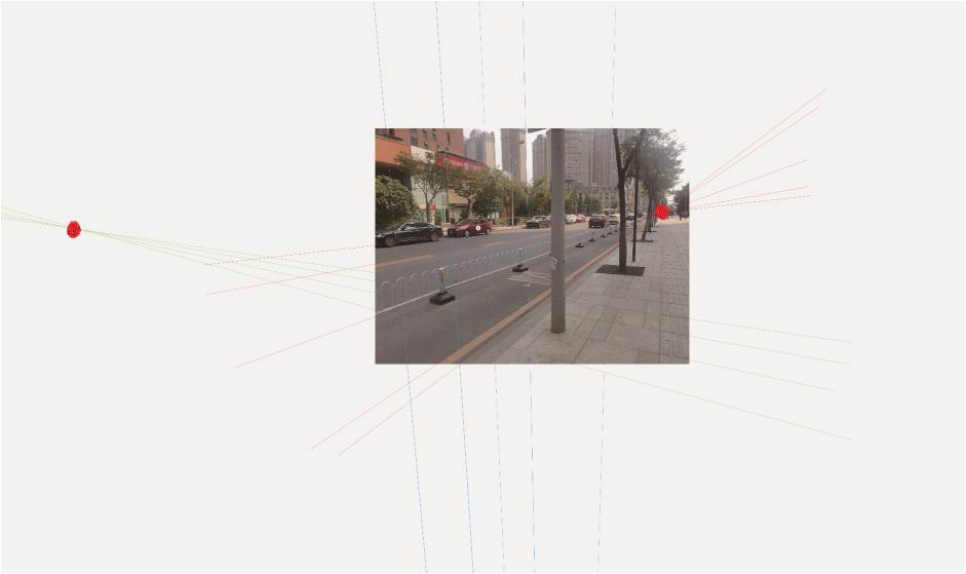


图 10 灭点示意图

通过灭点进行相机标定，得出旋转矩阵为 $R=\begin{bmatrix} 0.4716 & -0.8812 & -0.0299 \\ -0.1219 & -0.0315 & -0.99 \\ 0.8733 & 0.47 & -0.122 \end{bmatrix}$，平移向量为 $t=\begin{bmatrix} -5.07 & -0.72 & 18.6 \end{bmatrix}$。通过霍夫变换以及加入边缘检测算子确定 A 车车头与 B 车车头在图像中的位置坐标，根据求出的相机外参，可以求得 A、B 车头在世界坐标系下的位置坐标为（0,0,0）、（23.4,-0.468，0），从而求出 AB 车头间距离为 23.504m。拍照者距离马路左侧边界的距离为-13.5m。

4.3.2 问题一图 2 结果

问题一图 2 中的环境简易示意图如图 11 所示，求黑色车辆 A 车头和灰色车辆 C 车尾之间的距离以及拍照者距白色车辆 B 车头的距离。

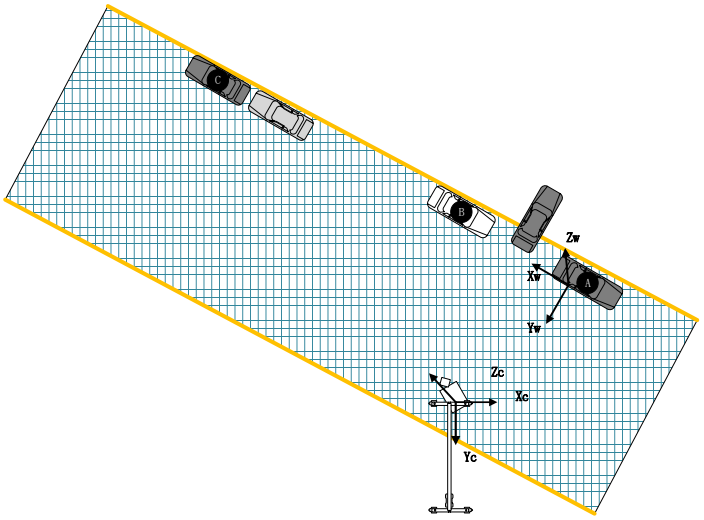


图 11 问题一图 2 简易图

通过灭点进行相机标定，得出旋转矩阵为 $R=\begin{bmatrix} -0.3238 & 0.84832 & -0.4188 \\ 0.03468 & 0.453 & 0.8908 \\ 0.945 & 0.2739 & -0.1761 \end{bmatrix}$，平移向量为 $t=\begin{bmatrix} 3.929 & 1.9341 & 9.509 \end{bmatrix}$。通过霍夫变换以及加入边缘检测算子确定黑色车辆 A 车头和灰色车辆 C 车尾在图像中的位置坐标，根据求出的相机外参，可以求得黑色车辆 A 车头和灰色车辆 C 车尾在世界坐标系下的位置坐标为（3.34,0,0）、（35,7，-0.74,0），从而求出黑色车辆 A 车头和灰色车辆 C 车尾距离为 32.4m。拍照者距白色车辆 B 车头的距离 23.89m。

4.3.3 问题一图 3 结果

问题一图 3 中的环境简易示意图如图 12 所示，求拍照者距岗亭 A 的距离以及拍照者距离地面的高度。

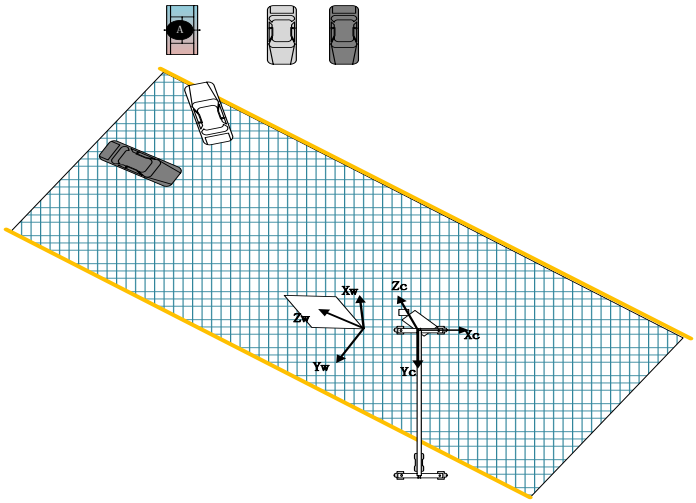


图 12 图 3 环境简易示意图

通过灭点进行相机标定，得出旋转矩阵为$R=\begin{bmatrix}0.4856 & -0.87417 & -0.005\\ -0.1924 & -0.1017 & -0.9758\\ 0.8525 & 0.4748 & -0.2185\end{bmatrix}$，平移向量为$t=\begin{bmatrix}0.598 & 3.070 & 8.0209\end{bmatrix}$。通过霍夫变换以及加入边缘检测算子确定人形横道预告标识线在图像中的位置坐标，可以求出相机外参，进而拍照者距岗亭 A 的距离 27.41m，拍照者距地面高度为 4.75m。

4.3.4 问题一图 4 结果

问题一图 4 中的求解示意图为图 13，计算塔体正面(图中四边形 ABCD)的尺寸，即 AB 和 CD 的长度以及 AB 和 CD 之间的距离。

图 13 图 4 求解过程

利用之前建立的多平面转换模型，以及计算公式，首先以地砖平面为第一世界平面，且以地砖一个顶点作为世界坐标系原点，计算得到单应矩阵后根据 P0 和 P1 的像素坐标获得 P0P1 的真实距离；根据图中等效关系，P0P1=P2P3，进而以 P2、P3 和 A、B 两点所在平面为第二世界平面，计算图像面到其单应矩阵，并根据 A、B 的像素坐标获得 AB 的真实距离；最后以 A、B、C、D 所在面为第三世界平面，计算单应矩阵，根据在地平面已知的地砖尺寸（80cm×80cm），可以将尺度信息转换到塔体正面的平面，进而求得 AB＝5.2m、CD＝3.8m、AB 与 CD 距离为 8.4m。

4.4 模型评价

位置变换模型准确的表示出单幅图像中相关坐标系间的转换关系，便于具体问题处理时的位置求解与变换。基于两灭点的相机标定模型通过借助图像中的先验尺度信息，完成对相机内参、外参的标定，解决了单目相机单幅图像下无法获知相机参数的问题。相对深度计算模型则在确定相机参数的基础上，通过代数运算，求解出图像点在世界坐标系下与相机的相对距离，完成了从图像到真实空间的转换。多平面转换模型则解决了先验尺度参照物与待解位置不共面且无法直接投影计算的问题，通过多个平面的单应矩阵变换获得各位置点在设定的不同世界平面中的欧式坐标。

五 、问题二分析与求解

5.1 问题二分析

根据问题二所给出的材料，首先需要对平面视频中距离进行分析，并求解出视频中指定拍摄者所在车辆与后视镜中红色车辆的距离，如图 14 为一帧图像中从后视镜看到的后方红车以及两车的相对位置，并且由于两车都在移动状态，故整个视频中的两车间距离是动态变化的。

图 14 视频中两车相对距离示意图

其次，问题二要求求出拍摄者所在车辆在超越视频中第一辆白色车辆时两车的速度差异，在两车相遇并完成超越的整个过程中可将原本在公路环境下的两车运动简化为两车间的相对运动，而无需考虑其具体车速，将重点求解问题放在速度之差上。

针对第一个任务而言，若采用问题一中对单幅图像距离信息求解的模型与方法，理论上通过两灭点标定并借助先验尺度信息可以计算出单帧图像下与红车的距离。但是，通过对汽车组成部件的调查，视频中手机通过拍摄后视镜才获取到红车的图像，后视镜为凸透镜故在两灭点标定的模型中相机光心以及灭点的确定都存在很大误差，故不适用于本任务。

通过对视频后视镜中画面信息分析，利用汽车相对运动模型可借助地面固定间隔的白色道路标记线计算红车与拍摄车距离。

此外，还可根据视频中拍摄者所在车英朗的前后车门把手的固定尺度信息，可通过透视变换转换视角，从而根据变换后图像中拍摄车与红车的像素差以及相关先验尺度信息获得两车距离。

在整个视频拍摄过程中，拍摄车与红车之间的相对距离并非固定不变的，因此通过对多帧图像分别进行变换求解相对距离，从而更准确把握视频中两车的位置状态。

针对第二个任务，可以建立两个不同的模型进行求解并对比验证结果，根据对视频信息的分析，视频中所超越的白车具体型号为凯翼 E3，其车身为 4450mm，从而可以看作拍摄者所在车从与白车相遇并完成超越的整个过程的相对位移，再通过对视频的时间信息进行分析得出超越所用时间进行速度差的计算。

另一种方案是对视频进行分割，只保留从两车相遇时到完成超越过程中排除后视镜画面后的内容，对该部分视频内容建立模型进行特征点提取，并对连续的图像帧之间进行特征匹配，根据相同特征点在不同帧间的位置差计算不同帧时刻下的速度差，最终拟合出超越的整段时间内的速度差异变化曲线，得出平均的速度差。由于相机拍摄角度倾斜问题，容易造成匹配失败以及测量精度不高，因此在利用该模型求解的过程中，需要针对视频中画面进行透视变换调整，并且对过程中误匹配问题也需要建立筛选模型进行剔除。

5.2 模型建立

5.2.1 图像逆透视变换模型

对于图像中因拍摄角度等问题，当摄像机和目标物有一个倾斜角的时候，采集到的图像会有一个透视畸变，原本真实环境中的平面在图像或视频中出现变形倾斜的问题十分常见。若需以该平面为主进行图像中相关信息解析、尺度测算时，显然无法满足要求。因此需要对透视畸变进行处理，选取正确的四个顶点进行逆透视变换。本质上，透视变换与逆透视变换是一样的，只是因模型结果不同而有所区分。

透视变换通用的公式为：

$$[x',y',w']=[u,v,w]\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{bmatrix} \tag{19}$$

其中变换矩阵 $\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{bmatrix}$ 可以看作四部分组成，$\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix}$ 为线性变换，$\begin{bmatrix}a_{31} & a_{32}\end{bmatrix}$ 表示平移，$\begin{bmatrix}a_{13} & a_{23}\end{bmatrix}^T$ 用于产生透视变换。对式（20）继续求解可以得到：

$$\begin{cases} x=\dfrac{x'}{w'}=\dfrac{a_{11}u+a_{21}v+a_{31}}{a_{13}u+a_{23}v+a_{33}} \\ y=\dfrac{y'}{w'}=\dfrac{a_{12}u+a_{22}v+a_{32}}{a_{13}u+a_{23}v+a_{33}} \end{cases} \tag{20}$$

通过选取四组对应的点可以求解变换矩阵，从而对图像进行处理，得到以目标平面为基础的正确图像，便于后续处理。经过逆透视变换后，视角均垂直于某一平面，从而图像中每一个像素所代表的真实环境中的距离可由图像中先验尺度信息解得，故可以直接从逆透视变换后的图像借助像素差获得两点间距离对应于真实环境中两物体距离。

5.2.2 相对运动求解模型

根据视频信息结合汽车运动模型，并依据先验的尺度信息可以对视频中拍摄车与红车间距离进行估算，位置关系如图 15 所示。由环境特征确定拍摄环境高速公路，从而根据道路标线的安全标准以及白线与两条白线间隔的比例关系确定道路标线的先验尺度信息，从而利用标线及其间隔数量近似得出两车间距离。

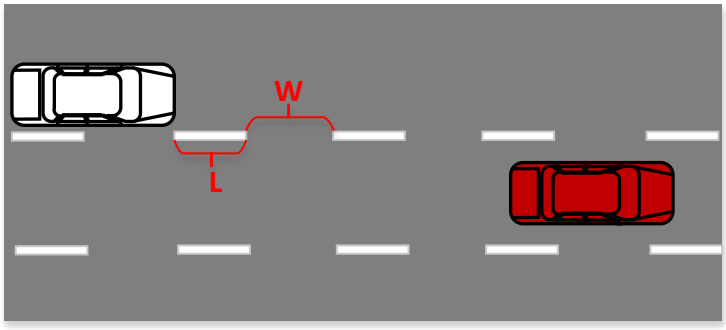


图 15 两车位置关系示意图

根据运动学原理，若两运动汽车有追赶与超越的行为，并且完成超越的整个过程用时较短，则可近似认为在超越的过程中两车各自的运行速度恒定，整个过程均以同一速度差在总超越时间内完成运动。并且，针对汽车超越的行为特点，可将测量参照物选取为此两车中的一个，不考虑两车相对于真实环境的运动。以被超越汽车为参照物，整个超越过程的运动模型如图 16 所示，为快车车头从慢车尾部开始到快车车尾超过慢车车头的过程，整个过程位移为慢车的车身长度，记为 L。通过对视频中画面内容以及对应帧时间记录，设完成超越的整个过程为 t，则速度差Δv为：

$$\Delta v = \frac{L}{t} \quad (21)$$

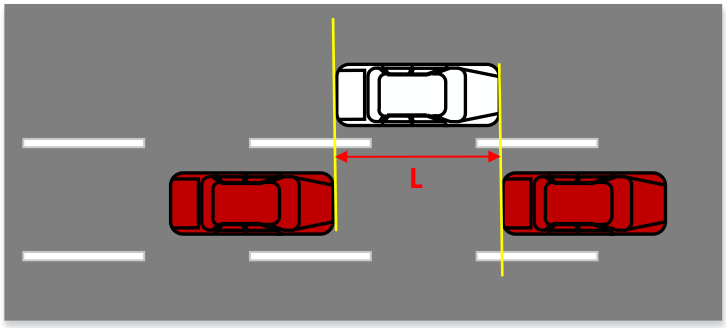


图 16 相对运动简易示意图

5.2.3 图像特征提取与匹配模型

对于视频信息的处理，首先进行图像特征点的提取，从而更准确把握视频中各帧或不同图像间的联系以及区别之处。图像的特征点可以简单的理解为图像中比较显著显著的点，如轮廓点，较暗区域中的亮点，较亮区域中的暗点等。通过建立图像特征提取模型，基于图像中特征点周围图像灰度值变化差别明显的特点，从图像中选取一点如 P 点，以该点为圆心画一个半径为 3 个像素的圆，计算圆周上像素点的灰度值如图 17 所示：

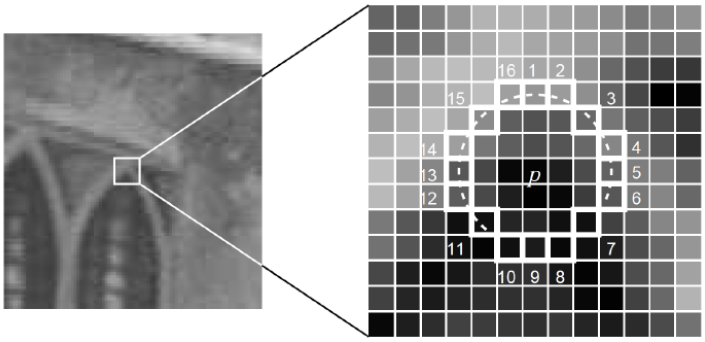


图 17 图像特征点确定示意图

若计算出的圆周上像素点与 P 点的灰度值的差别满足特征点候选模型如下式（22），其中 ε_d 为灰度差的阈值，$I(p)$ 为圆心的灰度，则认为该点为特征点。

$$N = \sum_{x \forall (circle(p))} \left| I(x) - I(p) \right| > \varepsilon_d \quad (22)$$

完成特征的确定后，为满足后续的图像帧之间特征匹配，需要对特征点加上属性描述。首先计算出图像中的矩，如下式（23 ），其中 $I(x, y)$ 为图像灰度表达式：

$$m_{pq} = \sum_{x,y \in r} x^p y^q I(x,y) \quad (23)$$

通过使用矩来计算图像中特征点以 r 为半径范围内的质心如下式（24），并计算出从特征点坐标指向质心的向量，并取向量的角度即为特征点的方向，如下式（25）。

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \quad (24)$$

$$\theta = \arctan\left(\frac{\frac{m_{01}}{m_{00}}}{\frac{m_{10}}{m_{00}}}\right) = \arctan\left(\frac{m_{01}}{m_{10}}\right) \quad (25)$$

根据建立的特征提取模型的结果，对一幅图像中的特征点，通过建立匹配模型计算此图中特征点与另一幅图像中的特征点之间的汉明距离 d，进行特征点间的匹配。匹配模型包括对误匹配点的剔除，首先确定最小距离即为 minDis，若两特征点间的汉明距离大于最小距离的 2 倍且小于最小距离的 4 倍则认为是正确的匹配，具体表达式如式（26）：

$$1.6 \times d < \min Dis < 4 \times d \quad (26)$$

5.2.4 特征位置变化求解模型

按照上述模型提取特征并完成特征匹配后，就可以得到连续帧间的特征关联与对比，从而确定同一特征在两张图像中位置的变化，并根据两帧间特征像素位移结合车身的实际尺寸获得在两帧中汽车的位置差，间隔时间Δt即为两帧图像的时间间隔。关于两帧图像的时间间隔的计算，取决于视频的帧率，设为F_R，时间间隔如下式（27）：

$$\Delta t = \frac{1}{F_R}(s) \quad (27)$$

对于两帧图像的同一特征点在图像上的像素位移记为D_p，通过前述模型中对图像进行逆透视变换获得汽车侧面的正视图，从而可建立两点间像素差D_p与实际距离D_R的对应关系：

$$\frac{D_p}{D_R} = \frac{l}{L} \quad (28)$$

其中 l 为特征点真实移动距离、L 为车身全长。

从而可以得到在两帧过程中快车与慢车的速度差Δv如下：

$$\Delta v = \frac{l}{\Delta t} \quad (29)$$

通过此方法可以根据连续多帧图像求出相对应的速度差，从而可以根据多个计算结果拟合有关速度差值的曲线，或通过平均值的方式掌握总体状态。

5.3 模型求解

5.3.1 相对运动模型求解距离

对问题二中视频信息进行分析，借助上文 5.2.2 中求解模型，由视频中白色道路标线以及其间隔距离的比例关系以及相关国家标准，确定了白色标线长为 6 米，两条白线间隔 9

米。基于此，在多帧图像中对红车与拍摄车之间的道路标线及其间隔进行统计并记录，最终生成估测的距离信息如下图 18 所示：

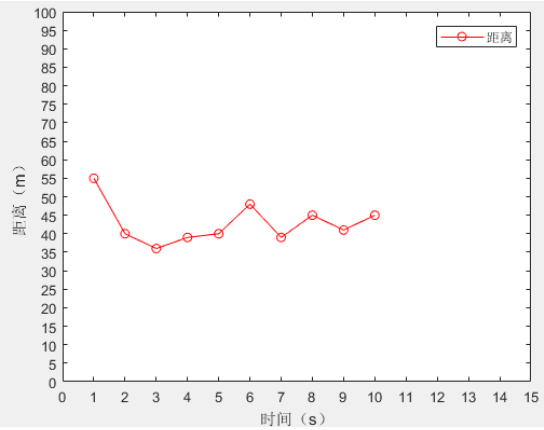


图 18 相对运动模型求解距离结果

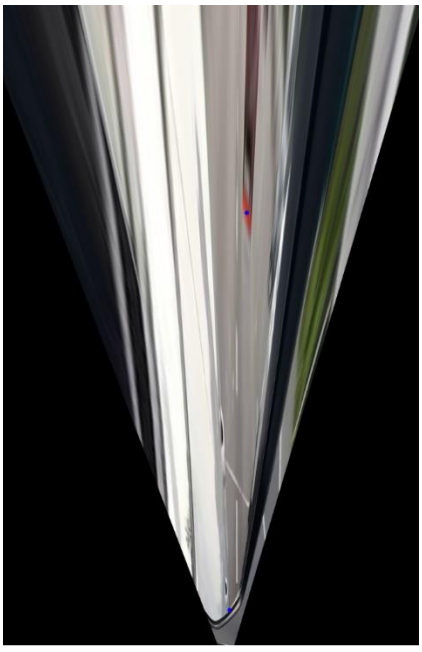
5.3.2 基于逆透视变换求距离

对于本题中视频画面进行分析，借助于 5.2.1 节中的逆透视变换模型，以道路地面为目标平面，借助英朗汽车的相关尺寸如图 19 所示，完成现有多帧图像的逆透视变换。

图 19 英朗汽车尺寸标注图

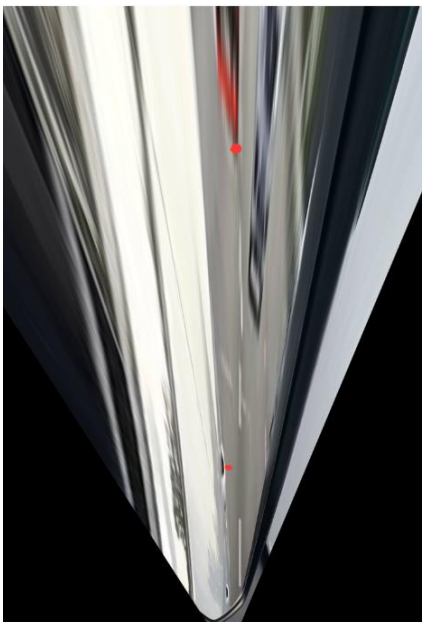
选取几帧下的逆透视变换前后对比如图 20 所示：

（a） 5.7s 原始图像



（b）5.7s 逆透视变换图像

（c） 10.4s 原始图像



（d）10.4s 逆透视变换图像

图 20　原始图像与逆透视变换示意图

再利用汽车的尺寸信息、道路标线尺寸信息以及变换后红车车头与拍摄车尾的像素坐标差解出不同帧时刻下的两车距离，由于本题视频的参数为每秒 30 帧，故通过帧数除帧率算得每一帧对应的时间，具体数据如表 1 所示。

表 1 逆透视变换下解得两车距离

时刻(s)	距离(m)
0.3	53.57
1.63	52.21
2.76	48.75
3.63	54.12
4.63	54.86
5.73	38.01
6.9	47.03
8.07	37.73
8.63	45.56
9.8	40.67
10.4	44.65

,5.3.3 相对运动模型求解速度差

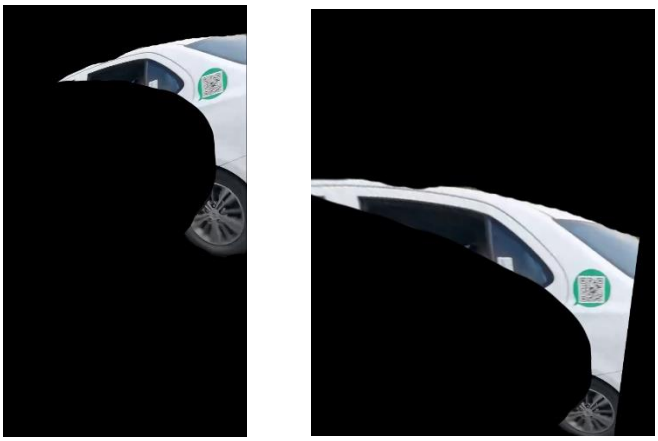
通过对视频中拍摄车与第一辆白车的相对位置分析，并且记录下两车相遇的时刻以及拍摄车车尾离开白车的时刻，求得时间差为 0.9s。此外，结合已有的先验尺度信息即白车长为 4450mm 以及 5.2.2 中提出的计算速度差的模型信息，计算得到速度差为 17.8km/h。

5.3.4 图像特征位移求解速度差

首先对图像进行分割，剔除画面中对速度差的求解起到干扰的内容，仅保留两车相遇时出现白车的一段时间内的内容，通过分割可以为后期的图像特征提取减轻计算复杂度并提高准确率以及计算精度，完成分割后为验证分割效果，验证还原的汽车图像如图 21 所示：

图 21　图像分割后还原验证图

通过前述模型，由于两车相遇时并非平行，且相机与白车侧面的整个平面形有一定的倾斜角度，因此首先对视频中白色车辆第一次出现在画面时，根据车上的二维码的四个顶点作为对应点，进行逆透视变换，如图 22 所示：



（a）原始图像　　（b）逆透视变换后图像

图 22　逆透视变换前后对比图

完成逆透视变换后，进行连续多帧图像的特征点提取，此处以某两个连续帧为例，效果如图 23 所示：

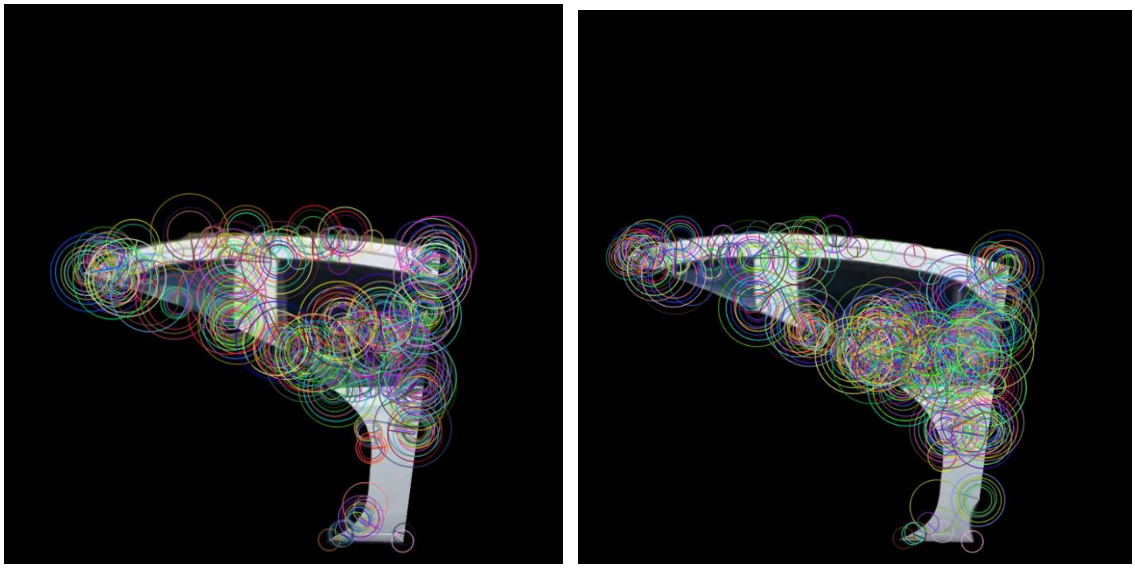
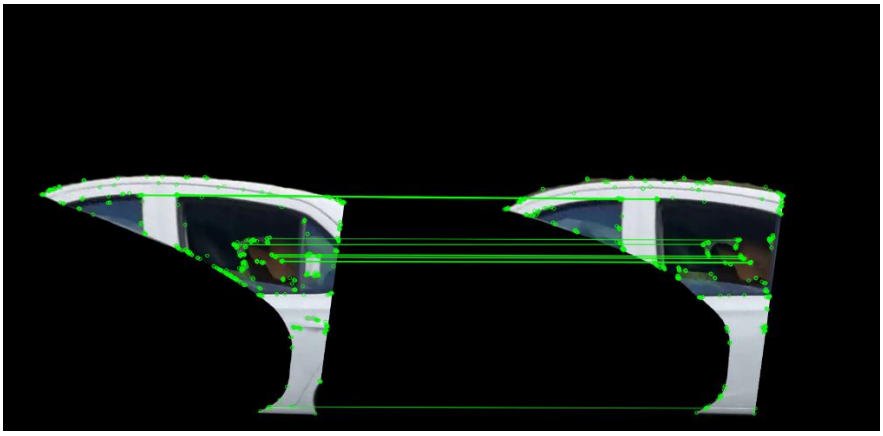
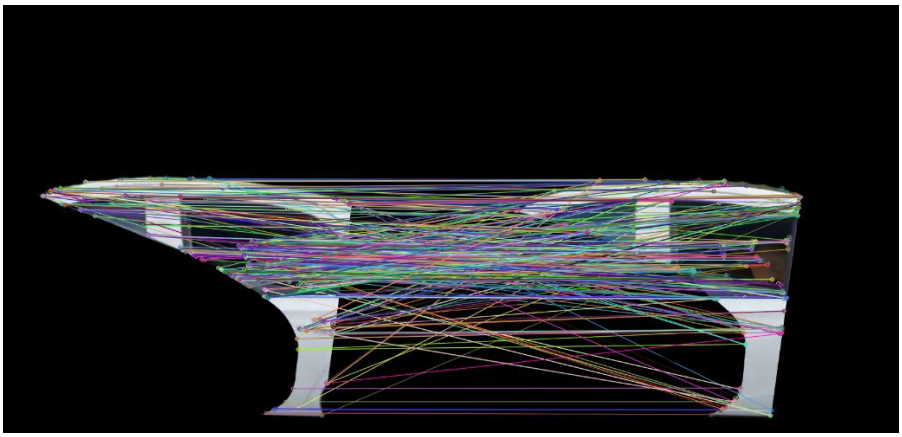


图 23　两帧图像中特征点对比图

完成特征点提取后，进行连续帧间特征点的匹配，此处仍以某两个连续帧为例，成功匹配以及匹配失败的情况分别如图 24（a）、（b）所示：



（a）两帧图像特征点正确匹配



（b）两帧图像特征点错误匹配

图 24 两帧图像匹配情况示意图

通过对正确匹配的两帧图像计算同一特征在像素平面的位移并进行记录，完成后根据像素与白车的车长这一先验尺度信息，获得在一帧时间内的汽车实际位移，从而可以求得该时刻的速度差，对多帧图像都进行速度差的计算，从而获得整体的变化情况数据如下表 2 所示，并算出平均速度差。

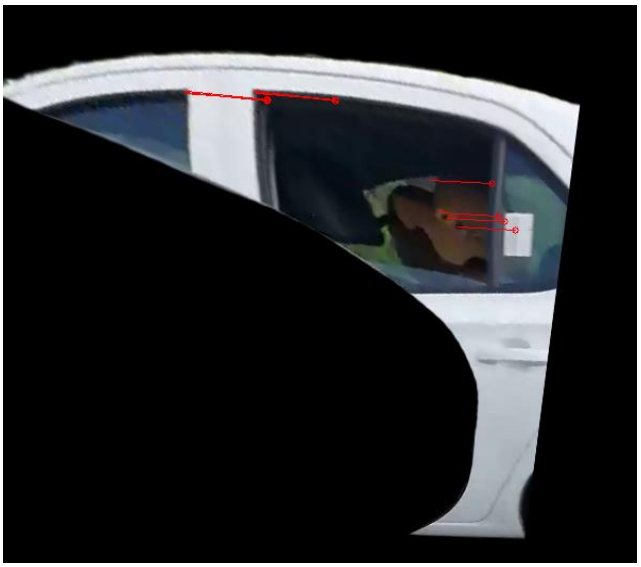


图 25　同一特征像素平面位移图

表 2 两车速度差及均值

速度差（组号）	差值（km/h）
1	19.58
2	19.05
3	21.37
4	20.30
平均速度差	20.075

5.4 模型评价

基于汽车相对运动的模型在求解本题任务中两车距离以及两车速度差时，计算简单同时也便于理解，但是很多情况下取决于人工确定相关位置并判断时间等信息，误差较大，偶然性太大。基于逆透视变换的距离求解模型在结果准确性上更高，并且具有足够的理论依据，但是由于本题视频拍摄角度问题，逆透视后的画面不够直观，并由部分情况下因真实距离太远无法获得另一辆车的位置。基于特征提取、匹配并计算特征位移的模型在计算速度差时因逆透视后画面效果较好，相对位移较为清晰，便于计算，并且可对视频中的图像帧连续计算，可获取更多速度差数据，但若出现误匹配且未在模型中剔除的情况容易引入误差且难以消除。

六 、问题三分析与求解

6.1 问题三分析

通过分析问题三中的视频可以看出，火车在运动中相机的位置近似固定，故相对于以水平面为世界平面的相机位姿可看作固定的。同时，根据对视频内容的分析，可以借助问题一中基于两灭点的相机标定模型，通过寻找图像中两处灭点，并结合具有已知尺度的先验参照物进行相机标定，求解相机参数。通过对真实空间以及像素坐标系下各自旋转矩阵与平移向量的求解，建立真实世界和像素尺度下的比例关系，通过该比例完成真实运动轨迹记录以及图像中特征点真实距离的计算。

6.2 模型建立

SFM 是将一系列从不同角度获取的拍摄的图像重建为 3D 结构的过程，其算法输出为一系列的相机位姿估值，及用点云表示的对应的重建出的场景结构，可以使用相机位姿估值进行相机运动的估计，使用构建的场景结构进行场景距离、高度估计。算法第一阶段进行特征的提取和匹配，进而进行几何验证；第二阶段利用第一阶段输出的图像匹配信息，选择最好的初始化图像，然后重复增加新图像、使用 Bundle Adjust(BA)对模型进行调整，其基本流程如图 26 所示:

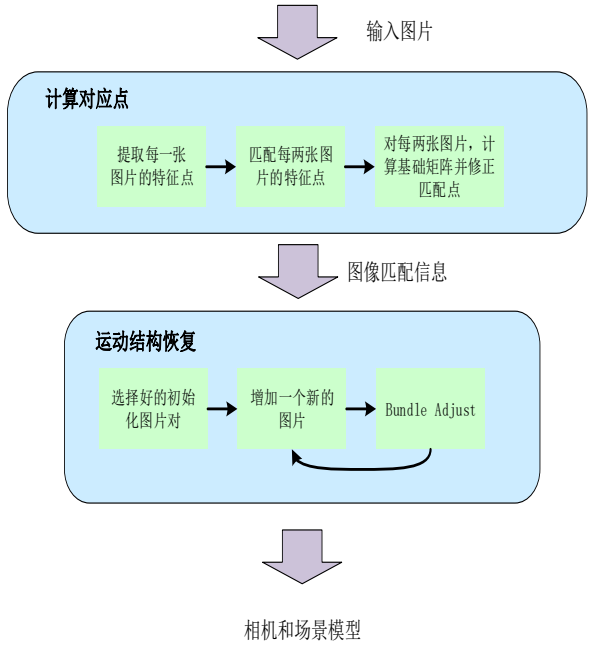


图 26 SFM 算法流程

6.2.1 对应点计算模型

计算对应点阶段完成的任务是从输入图片集中，识别具有重合区域的图像之间的对应点，输出一系列经过几何验证的对应点以及每个点在图像上投影的关系图。分为特征提取、特征匹配和几何验证三个阶段。

（1）特征提取

特征提取阶段使用的是具有尺度和旋转不变性的 SIFT 描述子,其鲁棒性较强，适合用来提取各种尺度变换和旋转角度的图片特征点信息，其准确性强，在离线不需要考虑时间成本的情况下也较有优势。SIFT 算法通过不同尺寸的高斯滤波器(DOG)计算得到特征点的位置信息(x,y),同时还提供一个描述子 descriptor 信息，在一个特征点周围 4*4 的方格直方

图中，每一个直方图包含 8 个 bin 的梯度方向，即得到一个 4*4*8=128 维的特征向量。

（2）特征匹配

一旦每个图片的特征点被提出来以后，就需要进行图片两两之间的特征点匹配，一般有两种方法：一是粗暴匹配,对所有特征点都穷举计算距离，这在大规模的图像集中是不可取的。二是邻近搜索,建立 KD 树,缩小搜索范围,能提高效率,但也有可能不是最优,所以邻域取值是关键,越大越准确,越大计算量越大。

匹配程度一般采用欧式距离，用 I_i 表示图像 I 在 i 像素的特征向量，用 J_j 表示图像 J 在 j 像素的特征向量，$i \in \{0,1,....N_I\}$，$j \in \{0,1,....N_J\}$，N_I、N_J 指图像 I、J 总像素数，则两特征向量之间的匹配程度为：

$$d_{I_i J_j} = \| I_i - J_j \|_2 \quad (30)$$

（3）几何验证

第三步就是对图像匹配对进行几何约束校验。SFM 进行匹配验证的方法是通过估计两幅图像间的变换关系，即投影几何关系。对于世界坐标系下的某一点，使用同一相机在不同角度进行拍照，可以得到多个图像与实际点的几何关系，如图 27 所示。

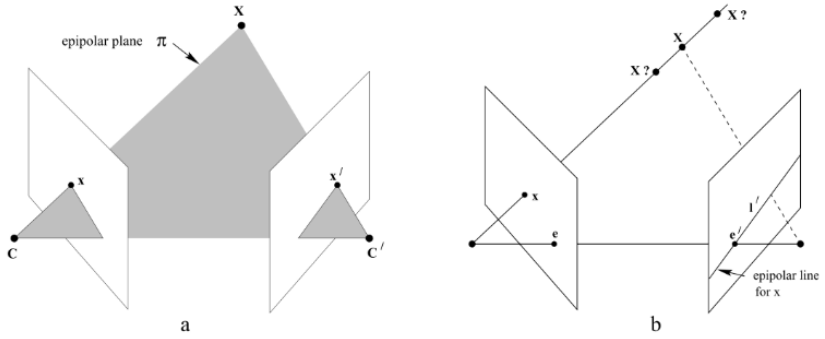


图 27 对极几何示意图

摄像机由相机中心 C,C’以及各自的成像平面表示，对于任意一个空间中的点 X,在两个像平面上的点分别为 x、x’，第一幅图像上的点 x 反向投影成空间三维的一条射线，它由摄像机中心和 x 确定，这条射线向第二个图像平面上投影得到一条直线 l’， x 的投影 x’必然在 l’上。

两视图的对极几何可以理解为图像平面与以基线为轴的平面束相交的几何关系，其中主要有下述几种概念：

1）基线:两个相机中心的连线 CC'称为基线。

2）对极点:ee'是对极点，是基线与两个成像平面的交点，也就是两个相机在另一个成像平面上的像点。

3）对极平面:过基线的平面都称之为对极平面，其中两个相机的中心 C 和 C',三维点 X，以及三维点在两个相机成像点 xx'这五点必定在同一对极平面上，当三维点 X 变化时，对极平面绕着基线旋转，形成对极平面束。

4）对极线：是对极平面和成像平面的交线，所有的对极线都相交于极点。

那么由对极几何的基本性质引出了对极约束的概念，对极约束是指在平面 2 上的 p 点在平面 1 上的对应点一定在基线 I'上，这说明了对极约束是一个点到直线的射影映射关系。如图 28 所示：

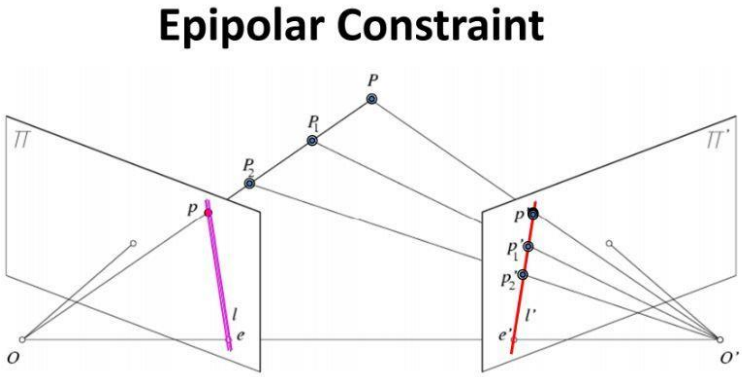


图 28 点到线的映射

根据对极约束可以引出本质矩阵和基础矩阵。在已知相机标定的情况下，假设有一个三维坐标点 P（X,Y,Z）在两个视图上的点分别为 p1,p2，由于第一个相机的中心作为世界坐标系的原点，也就是说第一个相机没有旋转 R 和平移 t，通过小孔相机模型有：

$$p_1 = KP \quad (31)$$

$$p_2 = K(\mathrm{RP}+\mathrm{t}) \quad (32)$$

上式合并可得:

$$p_2 = K(\mathrm{R}\,K^{-1}p_1 + \mathrm{t}) \quad (33)$$

同乘 K^{-1}。并设 $x_1 = K^{-1}p_1, x_1 = K^{-1}p_2$ 带入化简可得

$$x_2^T F x_1 = 0 \quad (34)$$

即对每一个符合的匹配对像素坐标都需要满足：

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} F \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \quad (35)$$

对极几何通过基础矩阵 F(未校正)描述了一个运动相机，F 矩阵可以把两张图片之间的像素坐标联系起来，并包含相机的内参信息。只有满足上式的特征点对才是正确的特征点对。

6.2.2 运动结构恢复

这一步的输入是场景图结构，其输出是一系列的相机位姿估值，及用点云表示的对应的重建出的场景结构，可以使用相机位姿估值进行相机运动的估计，使用构建的场景结构，建模过程如图 29 所示。

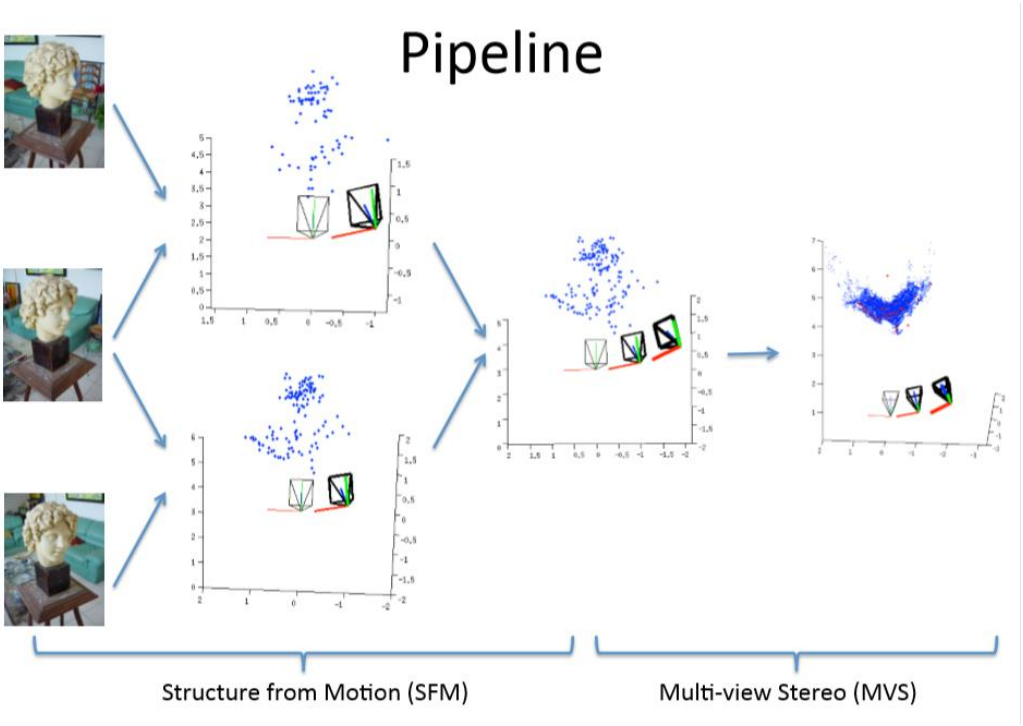


图 29 运动结构恢复过程示意图

SFM 初始化很谨慎，因为差的初始化可能导致重建不能完成，所以选择合适的初始化图像对至关重要。通常选择在 image graph 与其他帧重叠较多的作为初始化帧，因为这样就可以提供更多冗余，由此保证更加鲁棒和精准的重建。

新配准的图像必须能够观测到现有的场景点，不然无法确定新帧的位姿等参数。每当有新的图片加入进来时，就可以产生新的三角化的场景点，三角化在 SFM 中非常关键，因为该步骤增加了新的场景点，从而增加了现有模型的冗余度。

Bundle Adjust(BA)通过最小化重投影误差优化相机参数，进行微调，使得 SFM 不会漂移到一个无法恢复的状态，具体如图 30 所示。

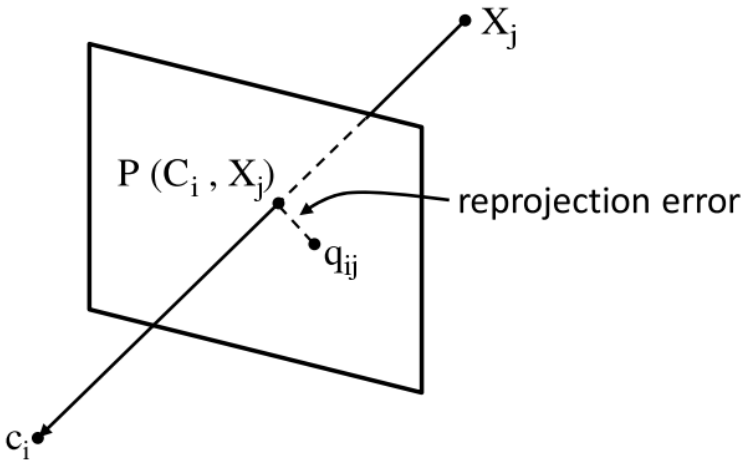
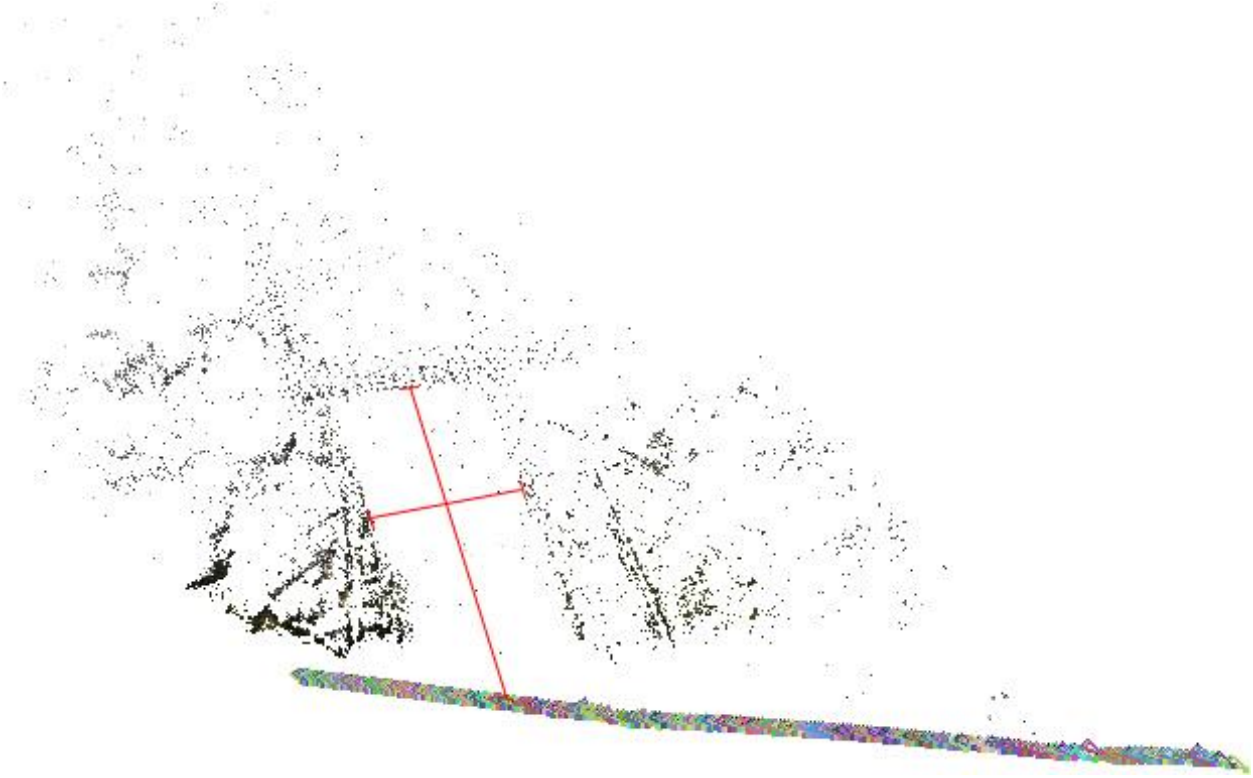


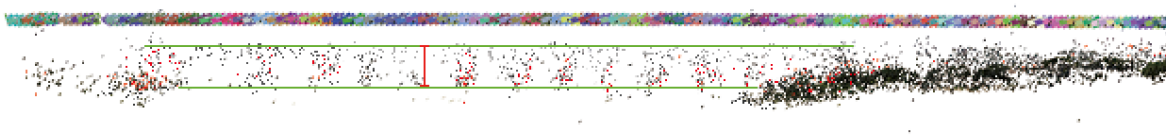
图 30 集束调整（BA）优化算法示意图

6.3 模型求解

将问题三“水面.mp4”视频逐帧输入到 SFM 模型中，可以得到火车在运动过程中的轨迹以及相机所获取图像 SIFT 特征点所叠加的稠密环境 3D 特征点云，如图 31 所示。此运动轨迹及所叠加的 3D 点云均通过相机以像素为尺度单位进行计算所得到，只能表示相机的帧间运动关系和点云中的特征点的相对位置关系，并不能表示真实的尺度信息。



(a) 3D 点云(红线标注河道宽以及桥到地铁距离)



（b） 水平方向 3D 点云（红线标注桥到水面距离）

图 31　视频重建出的点云图

为了获取真实尺度到像素尺度的在个维度上的比例系数，采用解决本文问题一中所建立的基于两灭点的相机标定模型，联合在场景第 24 帧中出现的驾校训练场地的“S 型弯”场地的真实尺度信息，如图 32 所示，并设定“S 型弯”的起点左线端点为零点。

图 32 两灭点标定位置示意图

能够求出该帧相对与原点和地面的旋转矩阵 R0 和平移矩阵 t0。将 R0 和 t0 与像素尺度下的 Rp 和 tp 进行比较，求出相机在 row、pitch、yaw、x、y、z 下的真实值和像素值的尺度之比，使用该尺度之比遍历相机的所有轨迹中的旋转矩阵、平移矩阵和点云中的所有特征点的坐标，即可求出相机图像求出的相机真实运动轨迹和特征点之间的真实距离。

将特征点的真实坐标和相机的轨迹值画入统一图像，通过将点云和生成该帧点云的相机图像之间进行对比，求得对应特征点之间的距离，如下表 3，可以求得任务三的所要求的距离信息。

表 3 问题 3 所要求距离信息

桥距水面的距离	8.8m
桥距高铁轨道的距离	485.3m
水面宽度	195.6m
高铁行驶速度	282.4km/h

6.4 模型评价

本题模型借助于前述基于两灭点的相机标定模型完成相机参数获取，同时利用运动结构恢复模型通过视频信息生成稠密的特征点云。此外，借助视频画面中符合国标的建筑信息完成尺度计算，从而将特征点的相对距离转换到真实空间中，从而易于求取各所需距离信息。不足之处在于点云生成的计算量过大，模型求解效率受限。

七 、问题四分析与求解

7.1 问题四分析

根据对问题 4 视频分析，结合对问题 3 的求解，利用 SFM 方法对场景进行三维重构，获得当前场景的三维结构图，通过对图中目标物体的尺度估计，可以为三维结构增加尺度信息从而获得所需要的建筑物周边道路的长度、宽度、各建筑物高度以及后花园中树木的最大高度。同时，在问题 1 所建立的模型基础上利用老宅的各个边长可以对面积进行求解。

使用任务一中问题 1 所建立的基于两灭点的相机标定模型，先求出相机参数，再根据古宅实际尺寸求单应性矩阵，进而求解古宅与相机的位置关系，从而求算飞机的飞行高度，并利用多个图像之间的时间关系解算无人机的飞行速度。

7.2 模型建立

针对问题 4 所提供的视频材料，可以建立如图 33 所示的简易环境表示图。

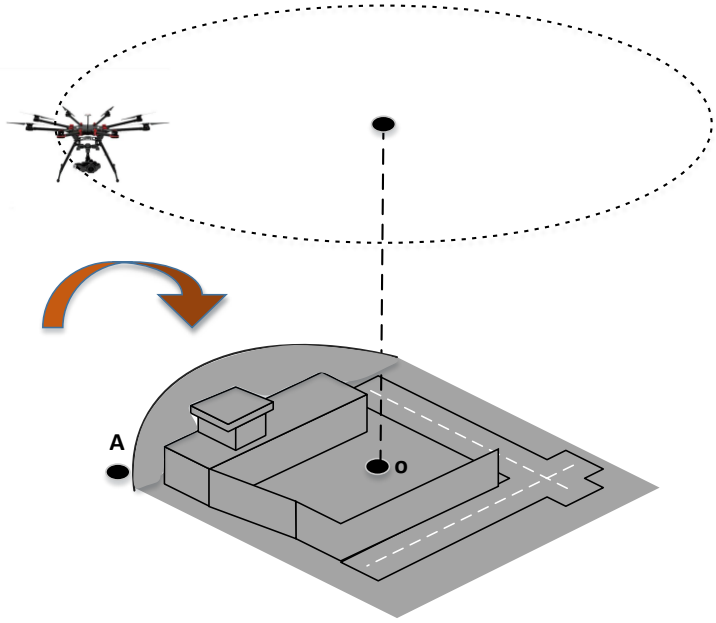


图 33　立体环境简易示意图

本问题所用的模型与问题 3 以及问题 1 中建立的模型一致，求解思路基本一致，区别在于视频侧重于对立体视频的距离分析与求解，具体工作在下一节的模型求解中有详细的利用先前所述模型求解的过程。

7.3 模型求解

选择视频中的一帧图像，计算出两个灭点，如图 34 所示，使用问题 1 中建立的基于两灭点的相机标定模型，估算环绕老宅道路的长度、宽度以及老宅占地面积；使用问题 3 中的 SFM 模型进行无人机轨迹、速度和树和建筑物高度的确定。

为求解图中物体的真实距离，我们需要确定图片中在地面所在水平面上的一真实场景的线段距离。通过对给定视频的观测，如图 2 中有一个人所走的步伐大小趋于稳定，行驶路径十分贴近于平行于该人旁边草坪的平行线。根据数据计数所得，该人行走通过该段草坪一共走过了 27 步，按照数据统计，正常成年男子的一个步伐的长度为 0.7m，按此式计

算可以得到，该段草坪 AB 长约 18.9m。

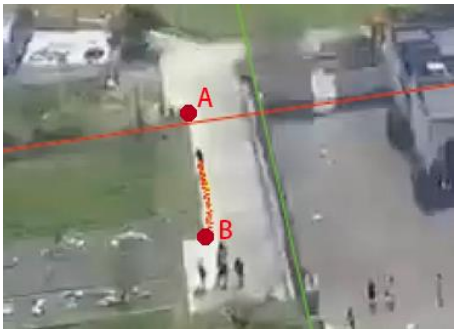


图 34 行走轨迹

在得到视频中的真实距离数据后，即可使用 SFM 算法中的每一个轨迹和特征点进行真实化，求得飞行器的真实运动轨迹，和真实点云地图。

如图 35 所示由飞行器的飞行轨迹可以得到点云地图，A 为房屋高度，B 是最高的树的高度。无人机飞行速度在 4.0m/s-4.3m/s 之间，飞行高度在 85-95m，对该无人机的飞行高度和飞行速度求平均即可求得无人机飞行速度为 4.197m/s，飞行高度为：93.213m。

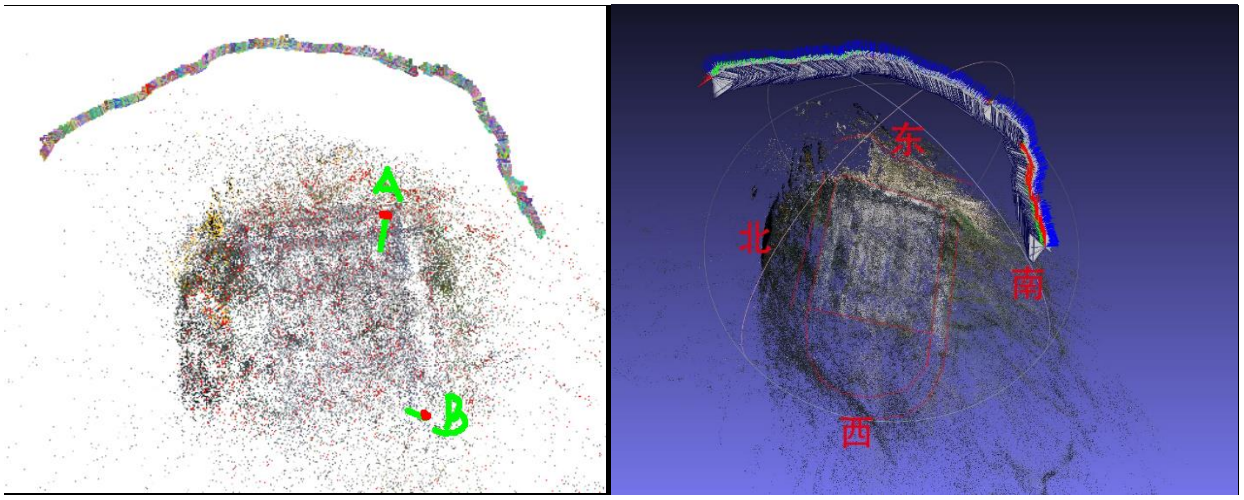


图 35 真实运动轨迹和点云地图

再之，通过对 SFM 算法获取的特征点点云图上两特征点之间的距离求 L 二范数即可求出特征点之间的距离。为了方便答案的给出，我们根据视频中的信息定义东南西北如图所示，通过选取任务四中，需要求解距离所对应特征点之间的距离，即可求出目标距离的大小，给出答案如表 4 所示，图 36 为方位辅助标记：

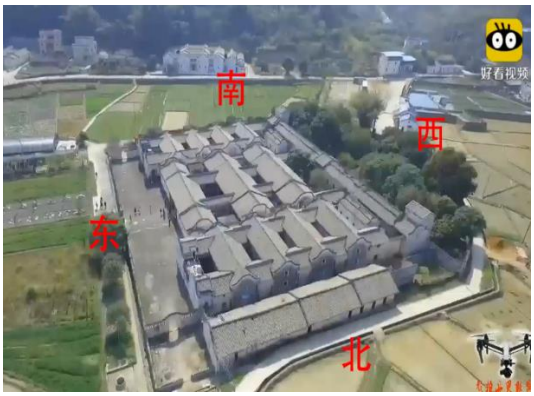


图 36 老宅方位辅助标记图

表 4 老宅参数

名称	实际参数
老宅北侧长度	67.43 m
老宅东侧长度	92.17 m
北侧道路宽度	2.26-2.37 m
南侧道路长度	3.23-3.47 m
东侧道路长度	4.43-6.53 m
后花园短轴长度	32.86 m
西面后花园处道路圆弧的长度	128.88 m
建筑物部分占地面积	5971.55 m^2
后花园占地面积	2358.187 m^2
总占地面积	8329.742 m^2
建筑物高度	11.492m
最高树的高度	15.325m

7.3 模型验证

为了验证该方案的准确性，我们提出了另一种计算方案，通过对图像进行处理，寻找到在世界坐标系下两两正交直线形成的灭点，如图 37 所示，同时选择水平面为世界坐标系下 XOY 平面，利用任务一的模型，进行距离求解，依此计算老宅周围道路的长宽。

图 37　两灭点位置示意图

为了求解后花园的长短轴，另选一帧图像如图 38，采用相同的模型，选取合适的实际标度，重新得到模型参数，将用于老宅后花园的参数的计算。

图 38 古宅换角度视图

在此认为后花园形状为半椭圆，通过再次利用灭点模型，即可求得椭圆的长短轴，按此结构，求得庄园的俯视图如图 39 所示，白色为道路，绿色为花园地区，灰色为建筑物地

区，与上文 SFM 方法求出的庄园面积相比相差不大，可以认为本文算法所求得数据较为准确。

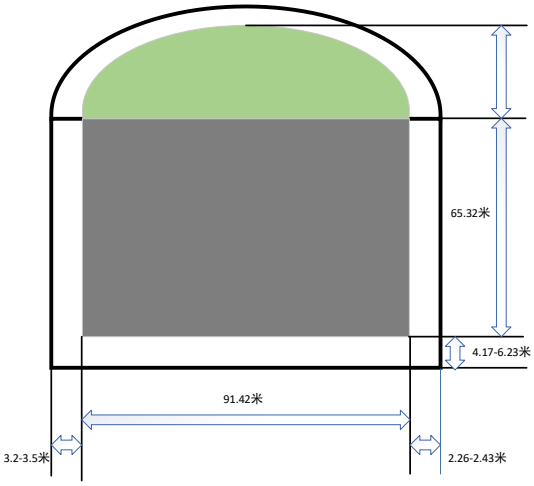


图 39 古宅俯视示意图

参考文献

[1] 刘震, 汪家悦, 陈丽娟. 基于灭点优化的单幅图像三维重建[J]. 浙江工业大学学报, 2019, 47(02):66-71.

[2] 余烨, 刘晓平. 基于灭点的三维物体重建[J]. 系统仿真学报, 2008, 20(15):4069-4072.

[3] 邓刚, 张志强, 孙济洲. 用未定标相机构造三维模型[J]. 系统仿真学报, 2001(S2):42-44.

[4] Guizilini V, Ambrus R, Pillai S, et al. PackNet-SfM: 3D Packing for Self-Supervised Monocular Depth Estimation[J]. 2019.

[5] Schönberger J L, Frahm J M. Structure-from-Motion Revisited[C]// IEEE Conference on Computer Vision & Pattern Recognition. 2016.

附录

附录一 单应矩阵

```
import cv2
import numpy as np
import matplotlib.pylab    as plt

img = cv2.imread('/home/wfcola/tu1.png')
rows,cols,ch = img.shape

x=5000

pts1 = np.float32([[282,431],[295,427],[323,500],[348,491]])
pts2 = np.float32([[0+x,0+x],[15+x,0+x],[0+x,600+x],[15+x,600+x]])

M = cv2.getPerspectiveTransform(pts1,pts2)

dst = cv2.warpPerspective(img,M,(10000,10000))

cv2.imshow( 'dst' , dst )
cv2.imwrite('/home/wfcola/Downloads/tu1.jpg', dst)

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')

cv2.waitKey(0)
```

附录二 两灭点标定相机

```
def distanceinpicture(task,x1,y1,x2,y2,left_top_x,left_top_y,\
    vanish_x1,vanish_y1,vanish_x2,vanish_y2,\
linein_pictrue1,linein_pictrue2,linein_world=2.6,center_x=1000,center_y=750,):

        if x1>2000:
            x1 = x1-left_top_x
            y1 = y1-left_top_y
        if x2>2000:
            x2 = x2-left_top_x
            y2 = y2-left_top_y

        v1_x = vanish_x1-left_top_x
        v1_y = vanish_y1-left_top_y

        v2_x = vanish_x2-left_top_x
```

```
        v2_y = vanish_y2-left_top_y

        k,b = line(v1_x,v1_y,v2_x,v2_y)

        distance_ = line_distance(center_x,center_y,k,b)

        v1i = ((point_distance(center_x,center_y,v1_x,v1_y)**2-distance_**2))**0.5
        v2i = ((point_distance(center_x,center_y,v2_x,v2_y)**2-distance_**2))**0.5
        f = (v1i*v2i-distance_**2)**0.5

        print("f:",f)

        #相机坐标系

        v1_ca = np.array([v1_x-center_x,v1_y-center_y,f])
        v2_ca = np.array([v2_x-center_x,v2_y-center_y,f])
        #print(v1_ca,v2_ca)

        v1_ca_nor = v1_ca/np.linalg.norm(v1_ca)
        v2_ca_nor = v2_ca/np.linalg.norm(v2_ca)
        #print(v1_ca_nor,v2_ca_nor)

        v3_ca_nor = np.cross(v1_ca_nor,v2_ca_nor)
        #print(v3_ca_nor)

        R= np.array([v1_ca_nor[0],v2_ca_nor[0],v3_ca_nor[0],v1_ca_nor[1],\
    v2_ca_nor[1],v3_ca_nor[1],v1_ca_nor[2],v2_ca_nor[2],v3_ca_nor[2]]).reshape(3,3)
        print("R:\n",R)

        l_world = linein_world

        l_imgae_x1_ = linein_pictrue1[0]
        l_imgae_y1_ = linein_pictrue1[1]
        l_imgae_x2_ = linein_pictrue2[0]
        l_imgae_y2_ = linein_pictrue2[1]

        l_imgae_x1 = l_imgae_x1_-left_top_x
        l_imgae_y1 = l_imgae_y1_-left_top_y
        #l_imgae_z1 = f
        l_imgae_x2 = l_imgae_x2_-left_top_x
        l_imgae_y2 = l_imgae_y2_-left_top_y

        l_imgae = np.linalg.norm(np.array([l_imgae_x1-l_imgae_x2,l_imgae_y1-l_imgae_y2]))    # 1 在图
像中长度
```

```
        l_=np.linalg.norm(np.array([v1_x-l_imgae_x2,v1_y-l_imgae_y2]))                    # l 上顶
点到消失点长度
        l=np.linalg.norm(np.array([v1_x-center_x,v1_y-center_y,f]))                       # 消失点
到相机坐标系原点距离
        AB_ = l_imgae*l/l_

        t = l_world*np.linalg.norm(np.array([l_imgae_x1-center_x,\
        l_imgae_y1-center_y,f]))/AB_

        t=t*np.array([l_imgae_x1-center_x,l_imgae_y1-center_y,f])/np.linalg.norm(np.array([l_imgae_x1-
        center_x,l_imgae_y1-center_y,f]))
        print("t:",t)
        xiangjiyuandian = np.matmul(np.linalg.inv(R),-t)
        print("相机原点:",xiangjiyuandian)

        neican = np.array([f,0,center_x,0,0,f,center_y,0,0,0,1,0]).reshape(3,4)
        waican = np.array([R[0][0],R[0][1],R[0][2],t[0],R[1][0],\
    R[1][1],R[1][2],t[1],R[2][0],R[2][1],R[2][2],t[2],0,0,0,1]).reshape(4,4)
        canshu = np.matmul(neican,waican)
        #print("canshu:\n",canshu)
        def find_xyw(x1,y1,w1,u1,v1,x2,y2,w2,u2,v2,x3,y3,w3,u3,v3,u,v,canshu=canshu):

            z1 = np.matmul(canshu,np.array([x1,y1,w1,1]).reshape(4,1))[2]
            z2 = np.matmul(canshu,np.array([x2,y2,w2,1]).reshape(4,1))[2]
            z3 = np.matmul(canshu,np.array([x3,y3,w3,1]).reshape(4,1))[2]
            z_xishu = np.array([[u1,v1,1],[u2,v2,1],[u3,v3,1]])
            z_zhi = np.array([z1,z2,z3])
            so = solve(z_xishu,z_zhi)
            #print(so)
            z_ = so[0]*u+so[1]*v+so[2]
            #print(z1,z2,z3)
            k = (z1-z2)/(v1-v2)
            b = z1-k*v1

            z = k*v+b
            print("z:",z,z_)
            xishu = np.array([canshu[0][:3],canshu[1][:3],canshu[2][:3]])
            zhi = np.array([z*u-canshu[0][3],z*v-canshu[1][3],z-canshu[2][3]])
            #print(np.linalg.norm(np.array(solve(xishu,zhi))))
            return solve(xishu,zhi)

        #print("distance:",np.linalg.norm(np.array([zuobiao1-zuobiao2])))
        a = f*R[0][0] + center_x*R[2][0]
        b = f*R[0][1] + center_x*R[2][1]
```

```
c = f*t[0] +center_x*t[2]
d = f*R[1][0] + center_y*R[2][0]
e = f*R[1][1] + center_y*R[2][1]
f = f*t[1] + center_y*t[2]

A = d-R[2][0]/R[2][1]*e
B = b/R[2][1]*(t[2])-c
C = a - R[2][0]/R[2][1]*b
D = e/R[2][1]*(t[2])-f

def z__(u,v):
    return -(A*B-C*D)/(A*(u-b/R[2][1])-C*(v-e/R[2][1]))
def solve_world(z,u,v):
    xishu = np.array([canshu[0][:3],canshu[1][:3],canshu[2][:3]])
    zhi = np.array([z*u-canshu[0][3],z*v-canshu[1][3],z-canshu[2][3]])

    return solve(xishu,zhi)
z1 = z__(x1,y1)
print("z:",z1)
world1 = solve_world(z1,x1,y1)
print(world1)
z2 = z__(x2,y2)
world2 = solve_world(z2,x2,y2)
print(world2)
#print([world1-world2])
print("distance:",np.linalg.norm(np.array([world1-world2])))

if task == "1":
    print("距离左侧马路：",xiangjiyuandian[0])
if task == "2":
    x3 = 9804-left_top_x
    y3 = 10093-left_top_y
    z3 = z__(x3,y3)
    world3 = solve_world(z3,x3,y3)
    print(world3)
    print("distance:",np.linalg.norm(np.array([xiangjiyuandian-world3])))
if task == "3":
    x3 = 9263-left_top_x
    y3 = 9875-left_top_y
    z3 = z__(x3,y3)
    world3 = solve_world(z3,x3,y3)
    print(world3)
    print("distance:",np.linalg.norm(np.array([xiangjiyuandian-world3])))
```

附录三 特征点选取与匹配

```
#include<iostream>
using namespace std;

// OpenCV 特征检测模块
#include <opencv2/features2d/features2d.hpp>
// #include <opencv2/nonfree/nonfree.hpp> // use this if you want to use SIFT or SURF
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/video/tracking.hpp>
#include <vector>
#include <opencv2/xfeatures2d.hpp>
#include <opencv2/highgui.hpp>

void KeyPointsToPoints(vector<cv::KeyPoint> kpts, vector<cv::Point2f> &pts);

bool refineMatchesWithHomography(
        const std::vector<cv::KeyPoint>& queryKeypoints,
        const std::vector<cv::KeyPoint>& trainKeypoints,
        float reprojectionThreshold, std::vector<cv::DMatch>& matches,
        cv::Mat& homography);

using namespace cv::xfeatures2d;

int main( int argc, char** argv )
{
    // 声明并从 data 文件夹里读取两个 rgb 与深度图
    cv::Mat rgb1 = cv::imread( "/home/wfcola/catkin_gu/photo/9.jpg");
    cv::Mat rgb2 = cv::imread( "/home/wfcola/catkin_gu/photo/10.jpg");

    // 声明特征提取器与描述子提取器
    cv::Ptr<cv::FeatureDetector> detector;
    cv::Ptr<cv::DescriptorExtractor> descriptor;
    cv::Ptr<cv::FeatureDetector> sift ;

    // 构建提取器，默认两者都为 ORB

    // 如果使用 sift, surf ，之前要初始化 nonfree 模块
    // cv::initModule_nonfree();
    // _detector = cv::FeatureDetector::create( "SIFT" );
    // _descriptor = cv::DescriptorExtractor::create( "SIFT" );
    sift = cv::xfeatures2d::SIFT::create();
    detector = cv::ORB::create();
    descriptor = cv::ORB::create();

    vector< cv::KeyPoint > kp1, kp2; //关键点
```

```
        detector->detect( rgb1, kp1 );   //提取关键点
        detector->detect( rgb2, kp2 );

        cout<<"Key points of two images: "<<kp1.size()<<", "<<kp2.size()<<endl;

        // 可视化， 显示关键点
        cv::Mat imgShow;
        cv::drawKeypoints( rgb1, kp1, imgShow, cv::Scalar::all(-1),
cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS );
        cv::imshow( "keypoints", imgShow );
        cv::imwrite( "/home/wfcola/slam_vo/keypoints.png", imgShow );


        // 可视化， 显示关键点
        cv::Mat imgShow2;
        cv::drawKeypoints( rgb2, kp2, imgShow2, cv::Scalar::all(-1),
cv::DrawMatchesFlags::DRAW_RICH_KEYPOINTS );
        cv::imshow( "keypoints2", imgShow2 );
        cv::imwrite( "/home/wfcola/slam_vo/keypoints2.png", imgShow2 );

        cv::waitKey(0); //暂停等待一个按键

        // 计算描述子
        cv::Mat desp1, desp2;
        descriptor->compute( rgb1, kp1, desp1 );
        descriptor->compute( rgb2, kp2, desp2 );

        // 匹配描述子
        vector< cv::DMatch > matches;
        cv::BFMatcher matcher;
        matcher.match( desp1, desp2, matches );
        cout<<"Find total "<<matches.size()<<" matches."<<endl;

        // 可视化：显示匹配的特征
        cv::Mat imgMatches;
        cv::drawMatches( rgb1, kp1, rgb2, kp2, matches, imgMatches );
        cv::imshow( "matches", imgMatches );
        cv::imwrite( "/home/wfcola/slam_vo/matches.png", imgMatches );
        cv::waitKey( 0 );

        cv::Mat matHomo;
        refineMatchesWithHomography(kp1, kp2, 3, matches, matHomo);
        cout << "[Info] Homography T : " << matHomo << endl;
        cout << "[Info] # of matches : " << matches.size() << endl;
```

```
cv::Mat imResult;
drawMatches(rgb1, kp1, rgb2, kp2, matches, imResult,
            CV_RGB(0,255,0), CV_RGB(0,255,0));
cv::imshow( "matches", imResult );
cv::imwrite( "/home/wfcola/slam_vo/imResult.png", imResult );
cv::waitKey( 0 );

// 筛选匹配，把距离太大的去掉
// 这里使用的准则是去掉大于四倍最小距离的匹配
vector< cv::DMatch > goodMatches;
double minDis = 9999;
for ( size_t i=0; i<matches.size(); i++ )
{
      if ( matches[i].distance < minDis && matches[i].distance != 0)
            minDis = matches[i].distance;
}
cout<<"min dis = "<<minDis<<endl;

for ( size_t i=0; i<matches.size(); i++ )
{
      if (matches[i].distance < 4.0*minDis && matches[i].distance > 1.6*minDis)
            goodMatches.push_back( matches[i] );
}

// 显示  good matches
cout<<"good matches="<<goodMatches.size()<<endl;
cv::drawMatches( rgb1, kp1, rgb2, kp2, goodMatches, imgMatches );
cv::imshow( "good matches", imgMatches );
cv::imwrite( "/home/wfcola/slam_vo/good_matches.png", imgMatches );
cv::waitKey(0);

      // 计算光流
vector<uchar> vstatus;
vector<float> verrs;

cv::Mat imOFKL = rgb1.clone();
vector<cv::Point2f> points1;
vector<cv::Point2f> points2;

 for ( size_t i=0; i<goodMatches.size(); i++ )
{

      points1.push_back (kp1[goodMatches[i].queryIdx].pt );
```

```
            points2.push_back (kp2[goodMatches[i].trainIdx].pt );

            cv::line(imOFKL, kp1[goodMatches[i].queryIdx].pt, kp2[goodMatches[i].trainIdx].pt,
CV_RGB(255,0,0), 1, 8, 0);
            cv::circle(imOFKL, kp2[goodMatches[i].trainIdx].pt, 3, CV_RGB(255,0,0), 1, 8, 0);

        }
        vector<float> speed;
        int a = 0;
        for ( size_t i=0; i<goodMatches.size(); i++ )
        {
        a = sqrt( (points1[i].x - points2[i].x)*(points1[i].x - points2[i].x) + (points1[i].y -
points2[i].y)*(points1[i].y - points2[i].y ) ) *0.35732342;

            speed.push_back(sqrt( (points1[i].x - points2[i].x)*(points1[i].x - points2[i].x) +
(points1[i].y - points2[i].y)*(points1[i].y - points2[i].y ) ) *0.35732342 ) ;

            cout<<"speed ="<< sqrt( (points1[i].x - points2[i].x)*(points1[i].x - points2[i].x) +
(points1[i].y - points2[i].y)*(points1[i].y - points2[i].y ) ) *0.35732342 <<endl;

        }
        /*
        KeyPointsToPoints(kp1, points1);

        cv::calcOpticalFlowPyrLK(rgb1, rgb2, points1, points2, vstatus, verrs);


        for (int i = 0; i < vstatus.size(); i++) {
            if (vstatus[i] && verrs[i] < 15) {
                cv::line(imOFKL, points1[i], points2[i], CV_RGB(255,255,255), 1, 8, 0);
                cv::circle(imOFKL, points2[i], 3, CV_RGB(255,255,255), 1, 8, 0);
            }
    }
     */
        cv::imshow( "OFKL : ", imOFKL );
        cv::imwrite( "/home/wfcola/slam_vo/imOFKL.png", imOFKL );

        cv::waitKey(0);


        return 0;
    }

    bool refineMatchesWithHomography(
```

```
        const std::vector<cv::KeyPoint>& queryKeypoints,
        const std::vector<cv::KeyPoint>& trainKeypoints,
        float reprojectionThreshold, std::vector<cv::DMatch>& matches,
        cv::Mat& homography)
{
    const int minNumberMatchesAllowed = 8;

    if (matches.size() < minNumberMatchesAllowed)
        return false;

    // Prepare data for cv::findHomography
    std::vector<cv::Point2f> srcPoints(matches.size());
    std::vector<cv::Point2f> dstPoints(matches.size());

    for (size_t i = 0; i < matches.size(); i++) {
        srcPoints[i] = trainKeypoints[matches[i].trainIdx].pt;
        dstPoints[i] = queryKeypoints[matches[i].queryIdx].pt;
    }

    // Find homography matrix and get inliers mask
    std::vector<unsigned char> inliersMask(srcPoints.size());
    homography = cv::findHomography(srcPoints, dstPoints, CV_FM_RANSAC,
            reprojectionThreshold, inliersMask);

    std::vector<cv::DMatch> inliers;
    for (size_t i = 0; i < inliersMask.size(); i++) {
        if (inliersMask[i])
            inliers.push_back(matches[i]);
    }

    matches.swap(inliers);
    return matches.size() > minNumberMatchesAllowed;
}

void KeyPointsToPoints(vector<cv::KeyPoint> kpts, vector<cv::Point2f> &pts)
    {
    for (int i = 0; i < kpts.size(); i++) {
        pts.push_back(kpts[i].pt);
    }

    return;
}
```