



CPIPC

中国研究生
创新实践系列大赛



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

学 校 西北工业大学

参赛队号 19106990050

1.王国玺

队员姓名 2.汪瑨昇

3.袁翔

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：

图像是人类获取信息的最形象，最丰富的来源，视频的本质也是一帧帧图像的合成。作为对真实世界的再现和浓缩，图像中蕴含了丰富的信息[1]，如何提取并运用图像与视频中的信息受到了摄影测量学，计算机视觉等领域研究人员的关注。当前移动机器人、无人驾驶、智慧工厂、无人机侦察等行业的快速发展也对图像与视频信息的分析手段提出了新的要求。为了解决“燃眉之急”，科研人员正在研究使用双目相机、多相机阵列、深度相机、多光谱融合等方法获取冗余信息，但在某些特定条件下，我们所能利用的只有普通的图像和视频。本文采用射影几何的思想搭建通用的量测模型，简化了问题的求解过程。本文的创新点在于，针对镜头畸变问题，使用随机采样一致性进行射影几何中直线的拟合，提升了算法的鲁棒性，使用最小二乘法提升算法的计算精度。视频中距离等信息的量测采用多次测量取平均的方法，降低信息的量测误差，在实时量测中使用深度学习算法中常用的指数加权移动平均法，降低因测量误差带来的数据波动。

问题一中，解决了单幅图片中距离，高度等信息的量测问题。通过假定图片中树坑、井盖等常见物体的尺寸已知，同时假设车辆等待测物体为质点，其连线与道路平行，将问题简化为已知平面内一条参考线段的长度以及所在平面的灭线，获得平面中与参考线段平行的直线上任意两点距离的问题。利用射影几何中的交比定理，得到具体的量测信息。

问题二中，解决了镜头畸变条件下视频中距离，速度等信息的量测。通过将后视镜的图像失真量化为镜头畸变，从而将问题简化为问题一在成像质量不可靠时如何保证精度的问题。针对视频中待测目标较小的问题，使用基于目标跟踪算法的图像裁剪，使得待测目标始终在图像中占据中心位置，在降低处理数据量大小的同时，保证了量测的精度，考虑到视频中信息的冗余，人工选取平稳拍摄的帧进行距离估计，降低不同尺度下估计带来的误差，提出基于单目视觉的车辆测距模型。针对逐帧手工处理工作量过大的问题，提出了使用 Canny 算法提取边缘，概率霍夫变换检测目标中的线段，K-means 聚类选择灭点的基于图像特征的灭点自动定位算法。基于射影几何和视频隐藏信息，提出了基于地理位置的车速分析以及辆车追逐模型分析。

问题三中，解决了平面视频中距离、速度等信息的量测问题。通过将视频分解为单张图片进行逐帧处理，将问题转化为多张图片量测信息的问题，通过多次测量取平均值的方法来弥补将视频分解成图片处理可能带来的信息损失。

问题四中，解决了立体视频中距离，高度，速度等信息的量测问题。通过将视频分解为单张图像进行逐帧处理，将问题简化为从多方位，多角度的图片中获取平面上物体的量测信息的问题，针对将视频分解为图片处理可能带来的信息损失，使用不同图片上多次测量取平均值的方法来降低偶然误差，提升精度。将无人机飞行高度和速度问题简化为数据由于量测误差造成波动时的处理问题，使用指数加权移动平均法降低偶然误差，实现视频中的实时数据稳定输出。

最后，本文对模型的优缺点作了相应的评价，并给出了镜头畸变等传统优化问题的解决方案。综上所述，借鉴射影几何的思想可以极大简化视觉情报的分析过程，该思路有广泛的应用价值。

关键词：随机采样一致性、指数加权移动平均法、最小二乘法、Canny 算法、概率霍夫变换、K-means 聚类

目录

一、 问题重述.....	4
1.1 引言	4
1.2 需要解决的问题.....	4
二、 模型的假设.....	4
三、 符号说明.....	5
四、 问题一.....	5
4.1 问题分析	5
4.2 模型建立	5
4.3 模型求解	10
4.4 结果分析	15
五、 问题二.....	15
5.1 问题分析	15
5.2 模型建立	15
5.3 模型求解	26
5.4 结果分析	26
六、 问题三.....	26
6.1 问题分析	26
6.2 模型建立	26
6.3 模型求解	27
6.4 结果分析	27
七、 问题四.....	27
7.1 问题分析	28
7.2 模型建立	28
7.3 模型求解	29
7.4 结果分析	33
八、 模型的评价与改进.....	33
九、 参考文献.....	34
十、 附录	35

一、问题重述

1.1 引言

随着科技的发展，图像采集设备逐渐覆盖了我们生活的方方面面，伴随而来的是图像数据与日俱增。图像作为对真实世界的再现和浓缩，其中不仅蕴含了丰富的光谱、纹理、几何形状等表层信息，而且还蕴含了物体的尺寸、深度变化等深层次的空间信息[1]。

而测量几乎是各个领域都必须面对的必不可少的问题之一。图像测量以其非接触性、普适性、实时性、可重复性等特点成为地物对象几何测量的重要手段之一[2]。当下图像采集设备呈现出多元化的趋势，由于无法得到成像设备的精准参数，因而我们不得不面临相机自标定的经典难题。研究人员也在不断探索使用双目相机、多相机阵列以及深度相机来获取更多的参考信息，从而达到更好的图像测量效果。但是在某些场景中并没有相应的条件，如历史场景，事故现场等等，此时基于单幅图像的空间信息提取就成了唯一的方法，在这种背景下，在单幅图像中实现几何量测就变得至关重要。

1.2 需要解决的问题

本赛题要求就以下四种情形对单目视觉系统采集到的图像和视频建立数学模型，并通过求解这些模型，进行几何量测。

问题 1：本题只考虑单幅图像的信息分析。作为图像几何量测的第一步，首先要建立数学模型，通过分析图像中隐含的信息，找出图像中量测信息与已知信息之间的关系，通过假定树坑、井盖等常见图形的尺寸，推算出所需的量测信息。

问题 2：本题考虑平面视频距离信息分析。本问题相对于问题一在图像中加入了时间尺度，且发生在镜头畸变的场景下。其核心在于通过时间尺度获得速度信息，并降低镜头畸变带来的误差。

问题 3：本题考虑平面视频距离信息分析。从侧方角度的视频中推算出量测信息以及相对速度。

问题 4：本题考虑立体视频距离信息分析。本题相对于问题三中的视频 360 度环绕，从而获得更多的隐藏信息。此时问题在于如何改进模型更好地利用这些隐藏信息从而获得更精准的量测。

二、模型的假设

- (1) 汽车等待测物体为质点，两质点之间的连线与马路平行。
- (2) 图片的主点位于图像中心。
- (3) 马路不考虑弯度。
- (4) 问题 1 图片 1 中 A 车红色福克斯车长 4.342m，其前后车轮距为 2.7m。
- (5) 问题 1 图片 2 中的井盖尺寸为 0.9m*0.9m。

- (6) 问题1图片2中的井盖上两个排水口连成的直线为井盖的直径。
- (7) 问题1图片4中塔体正面垂直于地面。
- (8) 不考虑问题2、3、4视觉采集系统抖动带来的误差。
- (9) 问题2中的汽车在较小的时间尺度上可以看作是匀速运动。
- (10) 问题4中庄园的形状为一个长方形和一个标准半圆。
- (11) 问题4中庄园屋檐的高度3ms。

三、 符号说明

CR	交比
$d(x, y)$	表示点 x 到点 y 之间的距离。
C	图像中的椭圆方程
大写字母表示现实空间中的点位，小写字母表示其在图像中的对应点。	

四、 问题一

4.1 问题分析

问题一需要在单幅图像中捕捉更多的视觉情报，具体的子问题包括单幅图像中任意两点的测距、拍摄者位置和高度的测定。由于图像并未给出传统方法需要的摄像头标定信息或深度信息，因此需要首先提取图片中的可用信息，如树坑、井盖、人行横道预告标线，地砖等，然后建立数学模型表示已知信息与量测信息之间的关系。

本题中我们使用了射影几何中的交比定理，通过提取图片中的信息找到平面的灭点与灭线，从而建立从可用信息到未知信息之间的数学关系模型。

本题的主要难点在于镜头畸变。镜头畸变是光学透镜因为透视原因造成的失真，它对光学成像的质量非常不利，且镜头畸变是所有透镜的固有特征，它只能改善而无法消除。在本题中我们将镜头畸变看作是一种误差因素，在模型中通过数学方法来降低这种误差。具体而言，使用随机采样一致性进行射影几何中直线的拟合，提升了算法的鲁棒性，在计算灭线时选用多个灭点通过最小二乘法提高灭线的计算精度。

4.2 模型建立

4.2.1 灭点与灭线

在射影几何中，射影变换是一种保线性变换。现实空间中的直线经过映射后在图像上还是一条直线，但是现实空间中平行的一组直线在映射后将不再平行，而是会汇聚在一点上，这个点就叫做这组平行线的灭点。在图像中所有原本平行的直线共享一个灭点，也就是说，现实空间中的一组平行线就可以确定唯一的灭点。对于现实空间中的一个平面，其

上所有方向的直线的灭点都位于一条直线上，这条直线被称为平面的灭线，也就是说一个平面只有一条灭线，这个平面上所有灭点都位于这条灭线上。

如图 4-1 所示，灭点与灭线可以由图像中的几何信息计算得出。在图像中选择两条或多条平行线即可得到一个灭点，而两组或多组不互相平行的平行线就可以确定平面的灭线。

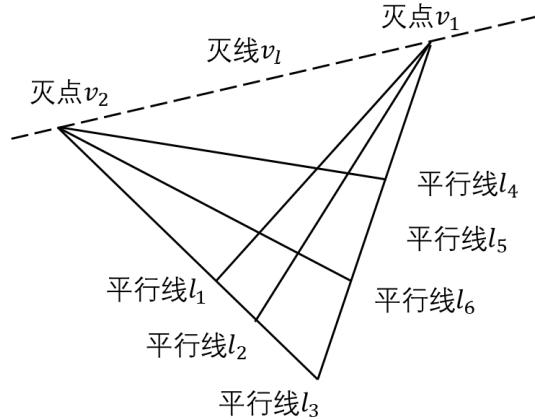


图 4-1 灭点与灭线

考虑到镜头畸变带来的误差，我们使用随机采样一致性来进行直线的拟合，在原始图像中选取多条平行线，从而确定多个灭点，使用最小二乘法来获得灭线。

4.2.2 交比定理

交比又称为非调和比，它的定义为共线四点组成的线段的比值。交比是一种重要的射影不变量，也是我们所建数学模型的基础。如图 4-2 所示，A 点与 B 点称为基点，C 点与 D 点称为分点。此处约定大写字母表示点在现实空间中的点位，小写字母表示其在图像中的对应点， $d(x, y)$ 表示两点之间的距离，CR (Cross Ratio) 为交比的英文缩写。交比在射影变换前后保持不变，共线四点的交比如公式(4-1)所示：

$$CR(AB, CD) = \frac{d_{AC}d_{BD}}{d_{AD}d_{BC}} = cr = \frac{d_{ac}d_{bd}}{d_{ad}d_{bc}} \quad (4-1)$$

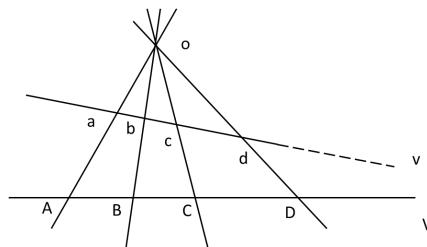


图 4-2 交比定理

在单幅图像的几何测量中，我们基于中心投影中交比保持不变的特性，结合图像中提取的可用信息，实现平面内任意两点的几何测量。具体而言，图 4-2 中同一直线上的四点 A、B、C、D，其交比值可以在图像空间内精确计算，若已知直线上某一线段长度的真实值，则可以推算得到另外一条线段的长度值。将这种一维空间上的射影不变量与其他几何条件相结合，就可以实现图像中任意两点的几何量测。

4.2.3 基于矩形的两点距离量测模型

交比本为一维空间上的射影不变量，可实现一维直线上的几何信息的提取，而通过与图像上的矩形信息（树坑）相融合，就可以实现二维平面上的几何信息提取和计算，也就能得到第一小问中红色车辆车头和白色车辆车头之间的距离以及拍照者居马路左侧边界的距离。

如图 4-3(a) 所示，若已知平面中的一个矩形 $S_1S_2S_3S_4$ ，其中 S_1S_2 所连成的边为 L_1 ，边长为 l_1 ， S_1S_4 所连成的边为 L_2 ，边长为 l_2 ，待求线段 A_1A_2 与 L_2 边平行，其射影变换后的图像如图 4-3(b) 所示， $s_i (i = 1, 2, 3, 4)$ 、 $a_i (i = 1, 2)$ 分别为对应的图像点（下同）， v_1 、 v_2 是通过矩形两条边算得的灭点， v_l 为该平面的灭线。由于每个平面只有一条灭线，任何直线必定与灭线交于一点，线段 a_1a_2 与灭线交于灭点 v_2 ，其长度可由以下算法获得：

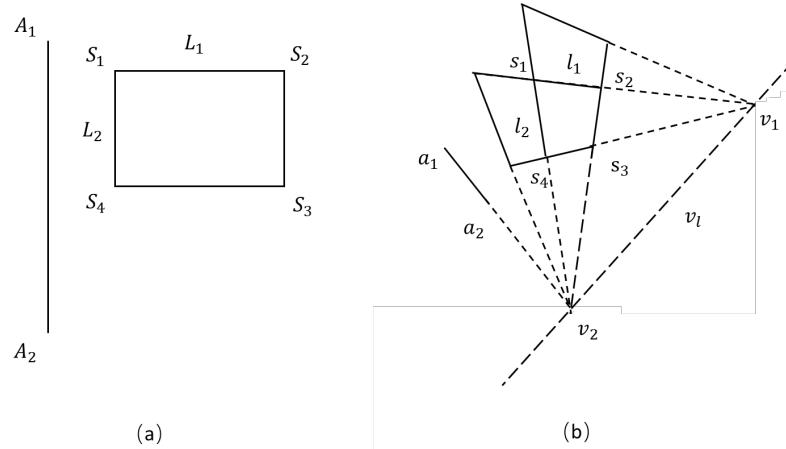


图 4-3 矩形的射影变换

为了简便，此时矩形 $S_1S_2S_3S_4$ 只关注与待求线段 A_1A_2 平行的边 S_1S_4 ，如图 4-4 所示。

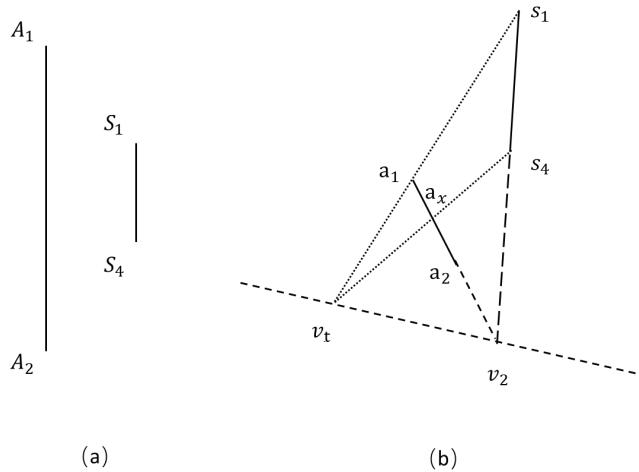


图 4-4 两平行线之间的关系

连接 s_1a_1 与灭线交于灭点 v_t ，连接 v_ts_4 与线段 a_1a_2 交于 a_x ，由于平行直线必定交于同一灭点，因此由 $s_1s_4a_xa_1$ 构成的四边形为平行四边形， a_1a_x 的实际长度必定等于 s_1s_4 所对应的实际长度，对共线四点 a_1 、 a_x 、 a_2 、 v_2 使用交比定理得：

$$CR(A_1, A_x, A_2, V_2) = CR(a_1, a_x, a_2, v_2) \quad (4-2)$$

用 $d(x, y)$ 表示点 x 到点 y 之间的距离得：

$$\lim_{V \rightarrow \infty} \frac{d(A_1, A_x) d(A_2, V_2)}{d(A_1, A_2) d(A_x, V_2)} = \frac{d(a_1, a_x) d(a_2, v_2)}{d(a_1, a_2) d(a_x, v_2)} \quad (4-3)$$

由于 V_2 为灭点，在直线的无穷远处，因此 $d(A_2, V_2)$ 为无穷大，式(4-3)左边通过变换有：

$$\begin{aligned} \lim_{V \rightarrow \infty} \frac{d(A_1, A_x) d(A_2, V_2)}{d(A_1, A_2) d(A_x, V_2)} &= \lim_{V \rightarrow \infty} \frac{d(A_1, A_x) \frac{d(A_2, V_2)}{d(A_2, V_2)}}{d(A_1, A_2) \frac{d(A_x, A_2) + d(A_2, V_2)}{d(A_2, V_2)}} \\ &= \lim_{V \rightarrow \infty} \frac{d(A_1, A_x)}{d(A_1, A_2) \left(1 + \frac{d(A_x, A_2)}{d(A_2, V_2)}\right)} = \lim_{V \rightarrow \infty} \frac{d(A_1, A_x)}{d(A_1, A_2)} \end{aligned} \quad (4-4)$$

等式(4-3)右边可以从图像中直接获取，设其值为 cr_1 ，而 $d(A_1, A_x) = d(S_1, S_4) = l_2$ ，则所求线段长度 $d(A_1, A_2) = l_2/cr_1$ 。

同时，考虑到镜头畸变带来的误差，使用随机采样一致性来进行直线的拟合，同时在原始图像中选择多个矩形，确定多个灭点，通过最小二乘法求得平面的灭线。

由以上分析，我们可以得到基于矩形的两点距离量测模型的具体算法：

算法 4.1 基于矩形的两点距离量测模型

目标：

给定平面上一个或多个矩形 $S_1S_2S_3S_4$ ，求平面上与已知长度为 l_2 的矩形一边平行的线段 A_1A_2 的长度。

算法：

- (1) 获取待测线段 A_1A_2 在图像中的对应线段 a_1a_2 。
 - (2) 通过矩形的边计算灭点 v_i ，其中 a_1a_2 对应的灭点为 v_2 。
 - (3) 在多个灭点间使用最小二乘法获得平面的灭线 v_l 。
 - (4) 连接 s_1a_1 与灭线交于灭点 v_t 。
 - (5) 连接 v_tv_t 与线段 a_1a_2 交于 a_x ，于是 a_1a_x 对应的线段的实际长度为 l_2 。
 - (6) 根据 $cr_1 = CR(a_1, a_x, a_2, v_2)$ ，计算 cr_1 。
 - (7) 计算线段 A_1A_2 的长度 $d(A_1, A_2) = l_2/cr_1$ 。
-

4.2.3 基于已知圆心圆形和两组平行线的距离量测模型

由灭点的性质可以得知，两组平行线就可以确定两个灭点，从而获得平面的灭线，此时，模型可以转化为已知平面的灭线以及一个带圆心圆形的量测模型，由于现实空间中的圆形经射影变换成为了图片中的椭圆，而椭圆方程不利于计算，因此选用该椭圆的外接矩形进行计算，从而模型转化为基于矩形的量测模型。

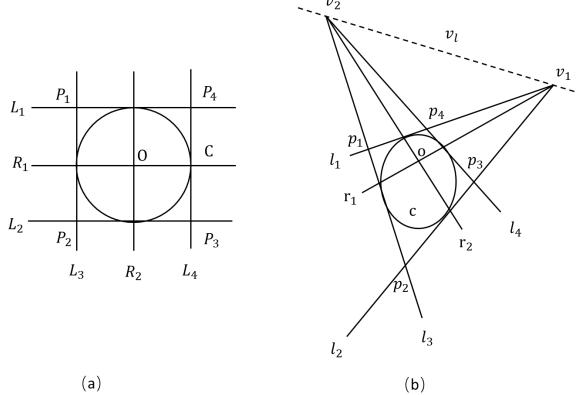


图 4-5 圆形与外接矩形

图 4-5(a)展示了圆形的外接矩形, 其中O表示圆心, $R_i(i=1,2)$ 为圆的两条直径, $L_i(i=1,2,3,4)$ 为与两条直径平行的切线, 其交点 $P_1P_2P_3P_4$ 构成了圆的外接矩形, 该矩形的长和宽为圆的直径。经射影变换后的图像如图 4-5 (b) 所示。当下的问题就是如何通过灭线和圆心找到这个外接矩形, 具体的方法如下所示:

在识别到椭圆C以及圆心o之后, 过圆心o取任意一条直径, 其所在的直线为 r_1 , 接着计算 r_1 与灭线 v_l 的交点 v_1 , 即对应于 r_1 的灭点:

$$v_1 = r_1 \times v_l$$

然后计算与直径 r_1 垂直的另一条直径 r_2 :

$$r_2 = C^{-1} \cdot v_1$$

得到另一条直径之后即可计算相对应的灭点 v_2 :

$$v_2 = r_2 \times v_l$$

已知两条直径就可以得到外接矩形和椭圆的交点, 分别连接这些交点与相对应的灭点, 形成两组平行的切线 l_1 、 l_2 以及 l_3 、 l_4 , 这四条切线所形成的四边形就是该圆的外接矩形 $p_1p_2p_3p_4$ 。

由以上分析, 我们可以得到基于已知圆心圆形和两组平行线的两点距离量测模型:

算法 4.2 基于已知圆心圆形和两组平行线的距离量测模型

目标: 给定包含圆形的平面, 已知圆形的直径 r_1 以及平面上两组平行线, 计算平面上任意两点之间的距离。

算法:

- (1) 识别椭圆C。
- (2) 根据两组平行线计算该平面的灭线 v_l 。
- (3) 计算与直径 r_1 垂直的另一条直径 r_2 。
- (4) 根据灭线 v_l 和直径 r_1 、 r_2 计算外接矩形对应的灭点 v_1 、 v_2 。
- (5) 计算直径 r_1 、 r_2 与椭圆的交点即外接矩形的切点。
- (6) 切点连接对应的灭点, 得到圆的外接矩形, 外接矩形的长和宽都等于直径。
- (7) 使用算法 4.1, 求已知矩形平面上的两点估计。

4.2.4 基于已知两个包含直径的圆形的距离量测模型

每一个包含直径的圆形都可以确定平面的一个灭点, 从而两个圆可以确定平面的灭线 v_l , 之后的算法与算法 4.2 基于已知圆心圆形和两组平行线的距离量测模型相同。

4.2.5 迭代式跨平面距离量测模型

算法 4.3 迭代式跨平面距离量测模型

目标: 已知平面上的线段 A_1A_2 的长度l, 求不同平面上与 A_1A_2 平行的线段 A_3A_4 的长度。

算法:

- (1) 选取当前平面到所求平面之间, 与当前平面相接的一个平面, 选择两平面交线上的一条线段 A_5A_6
 - (2) 根据平行线法等方法提取当前平面的灭线 v_l 。
 - (3) 根据交比定理建立已知线段 A_1A_2 长度到未知线段 A_5A_6 长度的数学关系, 从而将本平面的长度信息传递到下一个平面。
 - (4) 判断未知线段 A_5A_6 是否与待求线段 A_3A_4 处于同一平面, 若是, 则执行第(3)步计算待求线段 A_3A_4 的长度, 算法结束, 否则更新已知线段为 A_3A_4 , 执行第(1)步。
-

4.2.5 利用灭线性质估算摄像机位置

摄像机位置在摄像机内参已知的情况下可以准确求解。但是当只有单幅图片信息时, 还需要进一步利用灭线性质进行估算。

题目中要求求解摄像机在水平面(路面)上位置的距离以及高度信息。现有水平面h和两个互不平行且与h相垂直的平面 h_1 和 h_2 , 其成像面的灭线分别为 $line_h$, $line_{h1}$ 和 $line_{h2}$ 。从而我们可以得到如下结论:

- (1) 三个灭线的交点确立的三角形垂心为主点。
- (2) $line_{h1}$ 和 $line_{h2}$ 的交点即为相机在成像平面上的投影点。

通常在图片中难以确定两个垂直面, 但是当主点确定后, 可以利用水平面h的灭线得到其主线, $line_{h1}$ 或 $line_{h2}$ 与主线交点即为相机在成像平面上的投影点。

因此有如下算法:

算法 4.4 两个垂面确定相机成像位置

目标: 已知平面上有水平面h和两个互不平行且与h相垂直的平面 h_1 和 h_2 。

算法:

- (1) 利用图像信息求取两个垂直面的灭线 $line_{h1}$ 和 $line_{h2}$ 。
 - (2) 求取 $line_{h1}$ 和 $line_{h2}$ 的交点。
-

以及

算法 4.5 一个垂面和一个主点确定相机成像位置

目标: 已知平面上有水平面h和与h相垂直的平面 h_1 , 并知道主点位置。

算法:

- (1) 利用图像信息求取垂直面的灭线 $line_{h1}$ 。
 - (2) 利用主点求取主线位置。
 - (3) 求取 $line_{h1}$ 和主线的交点。
-

4.3 模型求解

4.3.1 第一题：

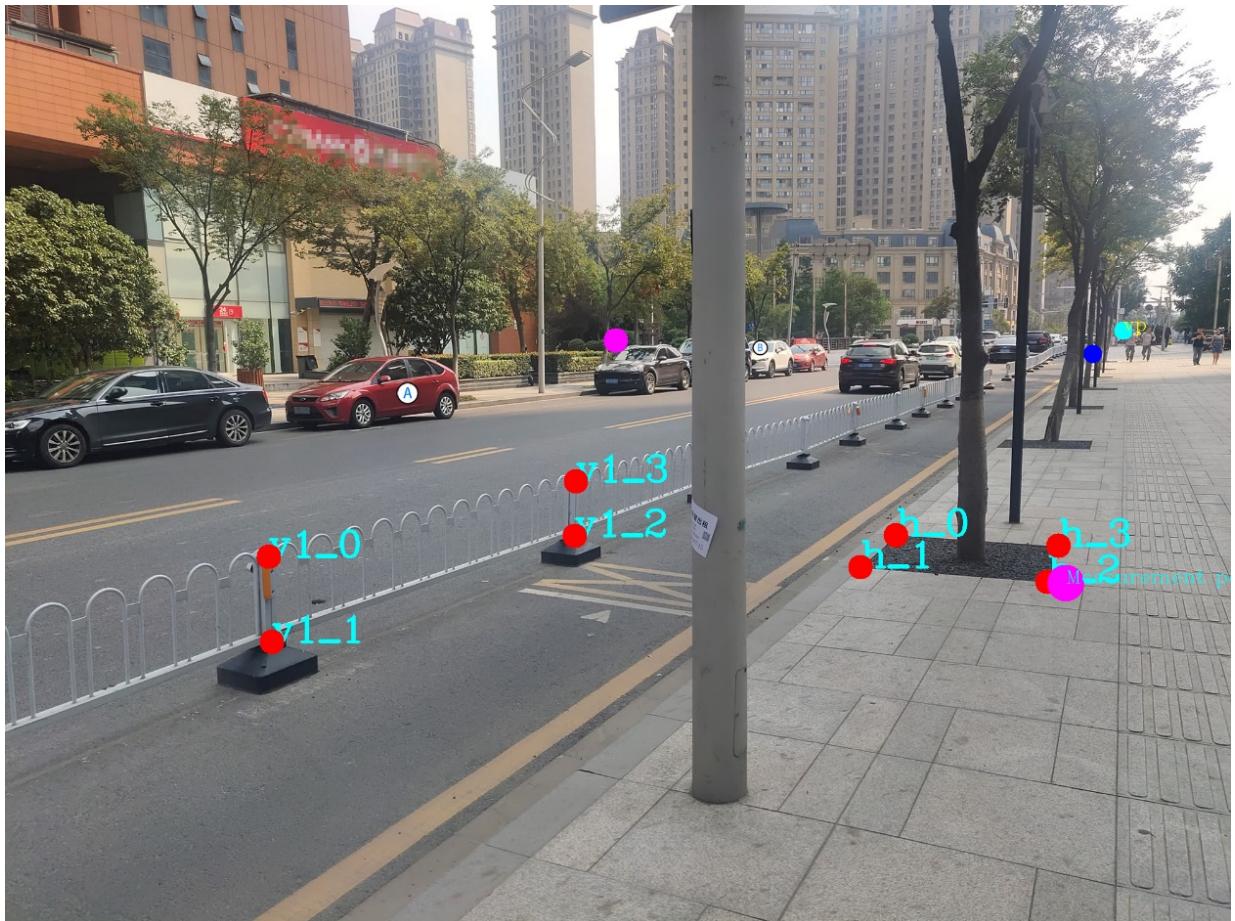


图 4-6 问题一选点以及灭点图

由图中的多个树坑我们可以获得多个矩形信息，使用最小二乘法我们可以获得水平面的灭线，通过算法 4.1 基于矩形的两点距离量测模型，结合 A 车红色福克斯前后轮的距离为 270cm，我们可以算得红色车辆 A 车头到白色车辆 B 车头之间的距离为：2554cm。

利用栏杆确定的垂直面，求取其灭线，在假设主点位于图片中心的假设下，由算法 4.5 一个垂面和一个主点确定相机成像位置可以得出拍照者在图中的位置信息，再通过算法 4.1 基于矩形的两点距离量测模型可以得出拍照者距离马路左侧的距离为：1580cm。

4.3.2 第二题

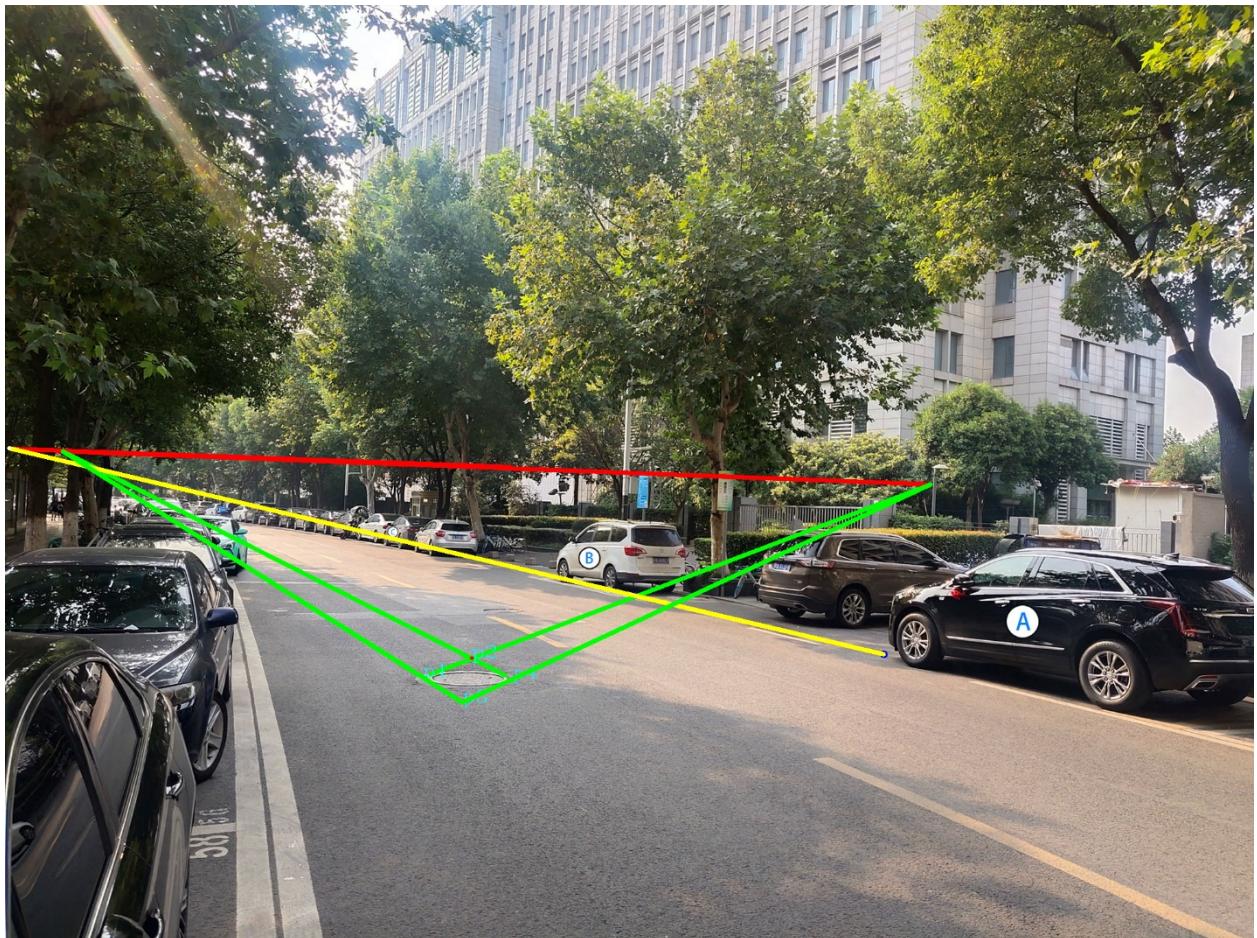


图 4-7 问题二选点以及灭线图

通过图中两个井盖的排水口我们可以获得两个带直径的圆形，根据上文中 4.2.4 基于已知两个包含直径的圆形的距离量测模型我们可以解出黑色车辆 A 车头与白色车辆 B 车尾之间的距离为：894cm。

通过图中楼房的表面与水平面垂直的条件，利用该平面上的平行线求解灭线，在假设主点位于图像中心的条件下，由算法 4.5 一个垂面和一个主点确定相机成像位置可以得出拍照者在图中的位置信息，再利用 4.2.4 基于已知两个包含直径的圆形的距离量测模型我们可以得出拍照者距离白色车辆 B 车头的距离为：1883cm。

4.3.3 第三题

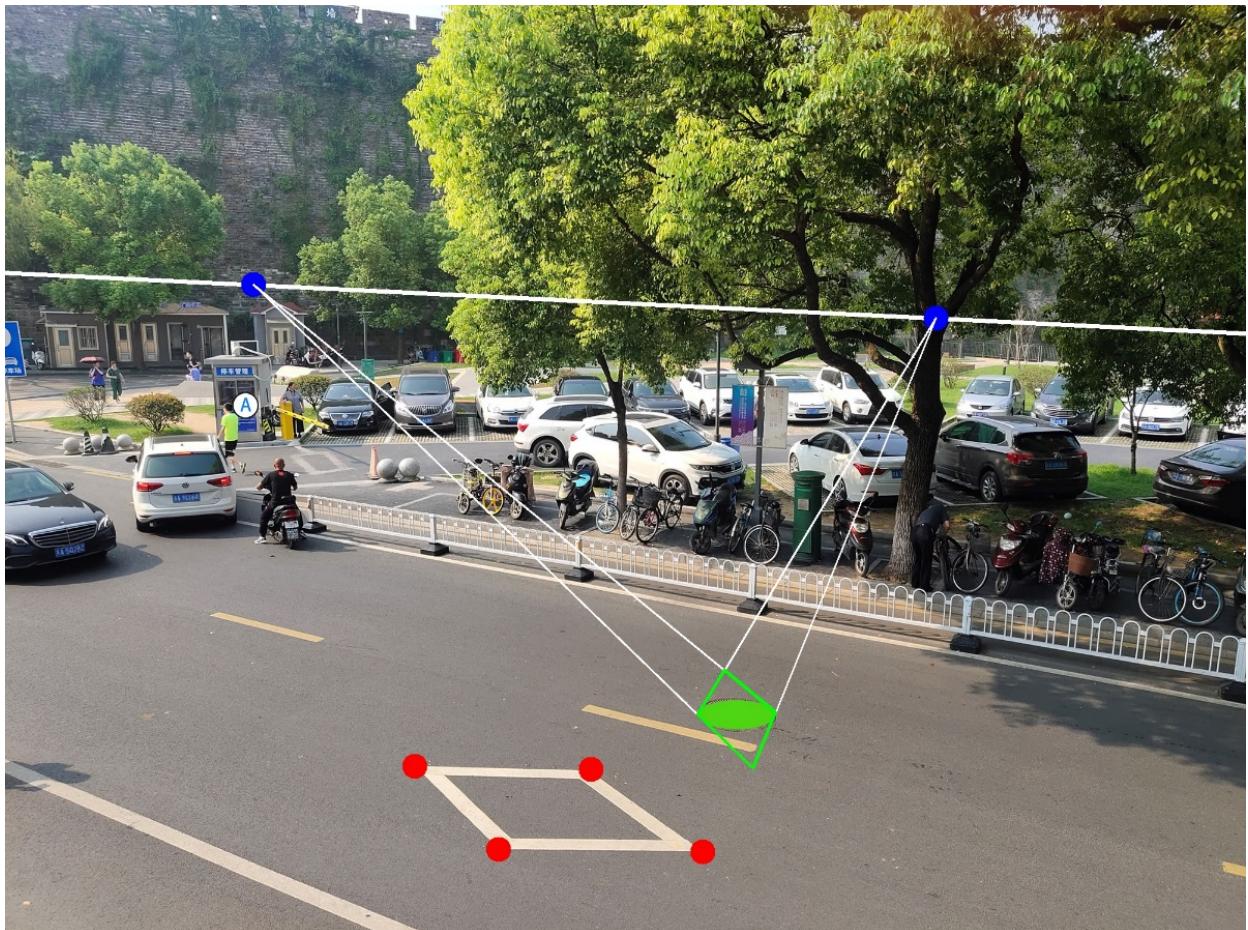


图 4-8 问题三选点以及灭线图

通过图中人行横道预告标线中的菱形我们可以得到平面上两组平行线，再利用井盖的圆心我们可以使用算法 4.2 基于已知圆心圆形和两组平行线的距离量测模型得到圆的外接矩形，从而获得水平面上任意两点的距离。

由图中的栏杆我们可以确定该垂直面的灭线，在假设主点位于图像中心的条件下，我们可以得到水平面的主线，主线与垂直面的交点即为相机在成像平面的位置，从而获得拍照者距岗亭 A 的距离：2529cm。

通过图中水平灭线与后排房屋的交点，垂直面与灭线重合的位置即为拍照者平视的方向，在假设后排房屋高 400cm 的条件下，求得拍照者距离：430cm。

4.3.4 第四题



图 4-9 问题四选点以及灭线图

假设门的两边是平行的。由地砖大小和传递性，可由基于矩形的两点距离量测得出 AB 距离。由于 AB, CD 是平行的，利用门的平行线即可得到 ABCD 面的灭点，使用与参考线段平行的两点距离量测方法计算得到 CD 的值，同样可以求得半圆直径。

注意到平面内由带直径的半圆，因而可以求得圆心 E，E 与半圆上任意一点连接都是圆半径，E 与门的两边是平行的灭点即可得到与 AB, CD 的垂线，按照与参考线段共线的两点距离测量算法可立即达到 AB 到 CD 的距离。

综上所述，AB 距离为：513.43cm, CD 距离为：325cm: AB 到 CD 距离为 627cm

4.4 结果分析

结果显示我们的数学模型可以很好地估计出待量测信息。

数学模型的误差主要分为系统误差和偶然误差，系统误差主要来源于图片中细节的模糊以及镜头畸变，我们使用随机采样一致性和最小二乘法来尽可能多得降低系统误差，同时，最小二乘法也有降低偶然误差的作用。在图片隐藏信息是圆形的情况下，使用外接矩形的方法可以避免椭圆各方向上单位像素表示实际距离的不同带来的误差，也使得求解的过程更具有鲁棒性。

五、问题二

5.1 问题分析

问题二需要根据一段视频来确定后方红色车辆与拍摄者所在车辆之间的距离，还需要确定所在车辆的速度以及超越白色车辆时两车的相对速度。由于拍摄者位于车辆副驾驶位置上，通过手持手机对车辆右侧后视镜录像得到视频，因而视频的清晰度和稳定性都存在很大问题，使用问题一中的测距方法已经不能得到较为准确的估计值，并且单帧图像很难能够保证估计模型的准确性，因此我们需要挖掘视频背后的信息。

对于第一个测距的子问题，我们首先将问题由相机拍摄后视镜模型简化为单目视觉车辆测距模型，从而在确保模型有效性的前提下，简化模型。对于测速的问题，我们利用视频中高速公路上的标志线等信息，并通过假设相邻的帧之间，车辆做匀速运动，进而获取运动模型，从而求解出拍摄者所在车辆速度和超越白车时的相对速度。

5.2 模型建立

5.2.1 简化后视镜模型

由于视频中待测距的红色车辆只占据视频像素中的很小一部分，并且后视镜有着很严重的畸变问题。如图 5-1 所示，为了简化模型，我们在不影响处理精度的前提下，通过目标跟踪方法获取红色车辆在每一个视频帧中的位置。我们使用目标检测中常用的矩形检测框 (x, y, w, h) 来表示红色车辆的位置信息，其中 (x, y) 表示目标检测框的中心像素坐标， (w, h) 表示检测框在水平和竖直方向上的长度。然后在每帧中截取一个图中 ABDC 的矩形区域，图中：

$$A = (x - 50, y - 75)$$

$$B = (x + 90, y - 75)$$

$$C = (x - 50, y + 105) \quad (5-1)$$

$$D = (x + 90, y + 105)$$

这样一来每一帧都会获得一个尺寸为 140*180 像素的后视镜内部区域，从而将原本由后视镜估计车距的问题转化为单目视觉车辆测距问题。对帧进行目标跟踪后通过跟踪边界框得到的裁剪区域如图 5-1 右侧所示：

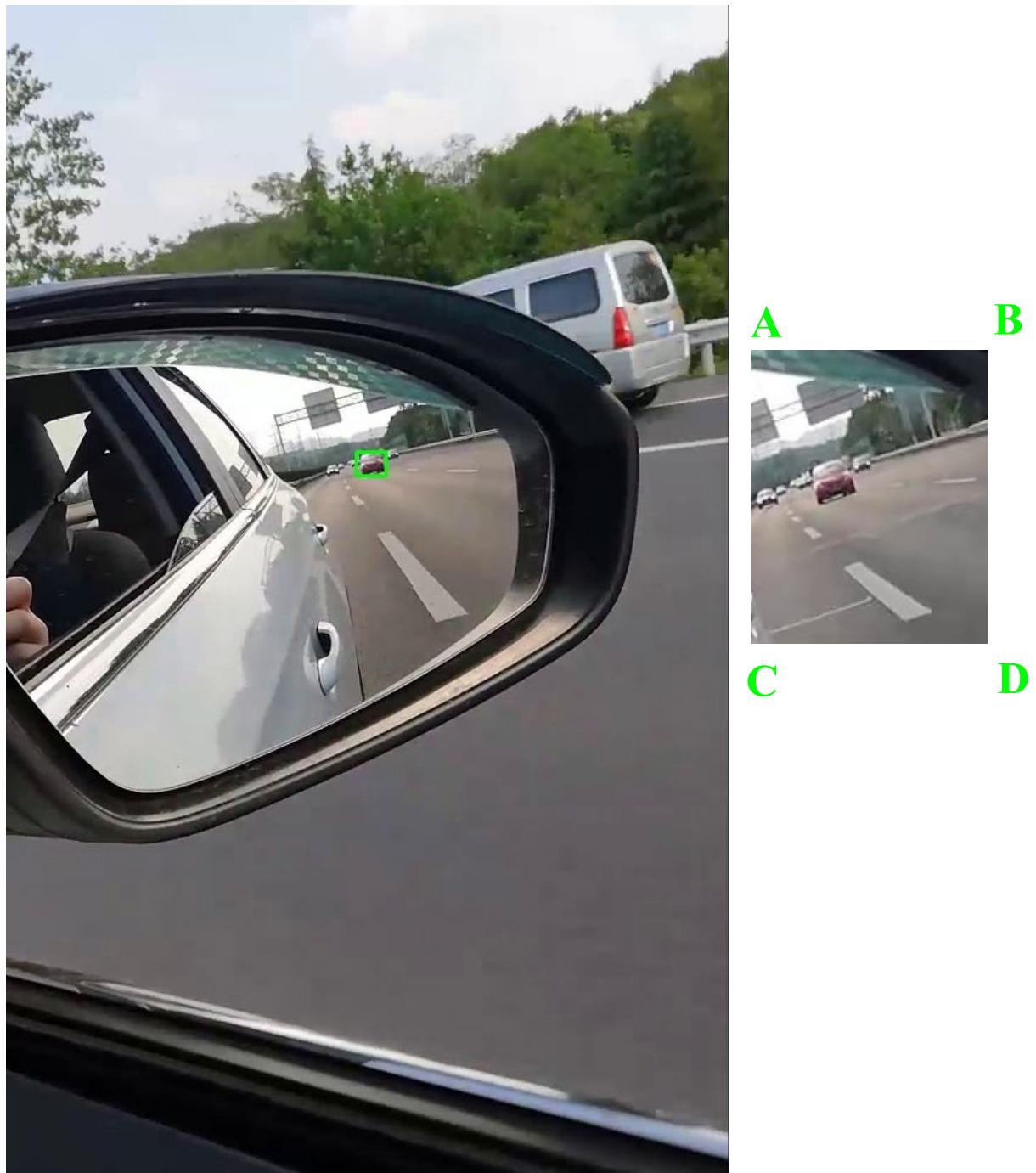


图 5-1 目标跟踪方法

我们可以看出，裁剪后的图不仅保存了完整的红色车辆信息和之后测距会用到的一些图像特征，如白色虚线，还减少了测距时其他无关区域造成的耗时。

5.2.2 单目视觉车辆测距模型

单目视觉测距模型多采用使用相机参数标定[6][9]，也有基于灭点进行测距的算法[7][8]。由于手机相机的参数不能完全知晓，因此我们采用根据灭点来确定车距的算法。在经过上述简化处理之后，车辆测距的模型可以简化为图 5-2 所示的过程。

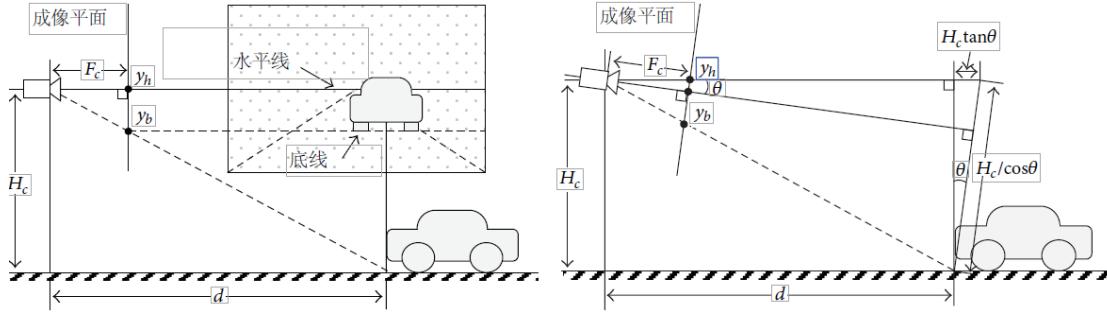


图 5-2 测距模型

其中 H_c 表示相机高度， F_c 为相机焦距，底线表示车辆底部与道路表面的交线，水平线表示由车道线确定的灭点的直线，相机水平放置时即相机俯仰角 θ 为 0 时，水平线会穿过图像的中心点， y_b 和 y_h 分别表示底线和水平线在成像平面上的位置， d 即为要求的车距。根据几何学的分析可推得：

$$d = \frac{1}{\cos^2 \theta} * \frac{F_c * H_c}{y_b - y_h} - H_c * \tan \theta \quad (5-2)$$

当相机的俯仰角 θ 很小时，上式又可简化为：

$$d = \frac{F_c * H_c}{y_b - y_h} \quad (5-3)$$

根据上述模型我么可以得出，要求解车距 d ，需要确定 F_c 、 H_c 、 y_b 、 y_h 和 θ 共五个参数。根据已知条件拍摄者所乘车辆为 2016 款别克英朗，其车高为 1798mm，后视镜距地面高度大约为 1200mm，即 $H_c \approx 1200$ mm， F_c 可以直接从视频信息中获得为 35mm，至此还有 y_b 、 y_h 和 θ 三个参数， θ 会影响 y_b 和 y_h 的成像位置，从而影响之后测距的精度。

通过分析视频可知，拍摄者手持手机的角度与行驶过程密切相关，由于路面不是绝对的平面，因此相机不可避免地会发生抖动，但是如果选取相邻视频帧进行分析，则抖动不大的情况可以看作相机是在平稳状态下以同一种姿态进行拍摄。由于在拍摄过程中，拍摄者在第 32 帧和第 63 帧进行了手动放大，因此视频中红色车辆位置发生跳变，也可以视为一种抖动，此外视频第 191 帧处也有明显抖动发生。为了减少模型由于拍摄抖动带来的误差，因此我们选取第 71 帧到第 170 帧共 100 帧图像作为估计红色车辆距离的图像。在这段可认为是平稳拍摄的视频区间中，拍摄者手机拍摄角度基本与所乘车辆的行驶方向平行，因此 θ 可认为处于一个平稳区间内，故取 $\theta = 10^\circ$ 。

由于之前进行的帧图像区域裁剪只关注红色车辆的周围区域，因此在一定程度上减少了相机抖动带来的红色车辆成像位置的跳变，为之后测距模型的精度提供了保障。

我们对截取出的 100 帧图像进行如下操作，其中 y_b 和 y_h 需要从视频帧中获取：

算法 5.1 多帧图像距离量测模型

目标: 从预处理后的多帧图像中准确两侧出距离信息。

算法:

- (1) 使用灭点定位算法获取每一帧图像中车道线沿行驶方向的灭点 $v_i(i = 1, 2, \dots, 100)$ 。
- (2) 根据跟踪算法生成的矩形检测框的下边沿即确定每一张图像中红色车辆的底线，记作 $b_i(i = 1, 2, \dots, 100)$ 。
- (3) 过灭点 v_i 作直线平行于 b_i ，便得到每幅图像的水平线 $h_i(i = 1, 2, \dots, 100)$ 。
- (4) 对于每帧图像：

$$y_i = y_b - y_h = b_i - h_i(i = 1, 2, \dots, 100) \quad (5-4)$$

- (5) 每张图中拍摄者所在车辆与红色车辆距离为：

$$d_i = \frac{1}{\cos^2 \theta} * \frac{F_c * H_c}{y_i} - H_c * \tan \theta \quad (5-5)$$

- (6) 最终取平均值降低误差：

$$d = \frac{1}{100} * \sum_{i=1}^{100} d_i \quad (5-6)$$

由上述算法我们可以得到两车车距的量测值。由于我们建立的模型是后视镜处的相机拍摄模型，因此实际使用时我们需要在其中减去后视镜至车尾的距离，得到最终要求的两车车距。

5.2.3 灭点自动定位算法

在单目视觉车辆测距模型中，测距时我们需要对每张图片都进行寻找灭点的处理，这样使得手动标注的工作量极大，因此我们采用了一种基于图像特征的灭点自动定位算法，先从目标图像中提取直线，根据线段角度确定交点候选点，再通过聚类算法得到最终的灭点，从而减少了人为选择基准点带来的测量误差，提高了之后的测距精度。

算法主要流程如下：

算法 5.2 灭点自动定位算法

目标: 实现基于图像特征的灭点自动定位算法，降低手工标注带来的测量误差。

算法:

- (1) 使用 Canny 算法[4]获取目标图像的边缘，Canny 算法是一个多阶段算法，可以有效提取出图像中物体的边缘，其算法流程图如图 5-3 所示：



图 5-3 Canny 算法模型

Canny 算法不易受噪声干扰，它使用两种不同的阈值来检测强边缘和弱边缘，因此可以提升最终的边缘检测效果。Canny 算法得到边缘检测图像如图 5-4 所示：



图 5-4 Canny 算法的效果

- (2) 使用概率霍夫变换（Progressive Probabilistic Hough Transform）[5]检测目标图像中的线段。首先通过(1)获取边缘图像，并将其前景点映射到极坐标系绘制曲线；当极坐标系里有交点达到最小投票数，则将对应的x-y坐标系中的直线L找出

来；随后搜索边缘图像上的前景点，将L上的点连成线段，然后这些点全部删除，并记录该线段参数。随后不断重复上述过程。线段检测的结果如图 5-5 所示：

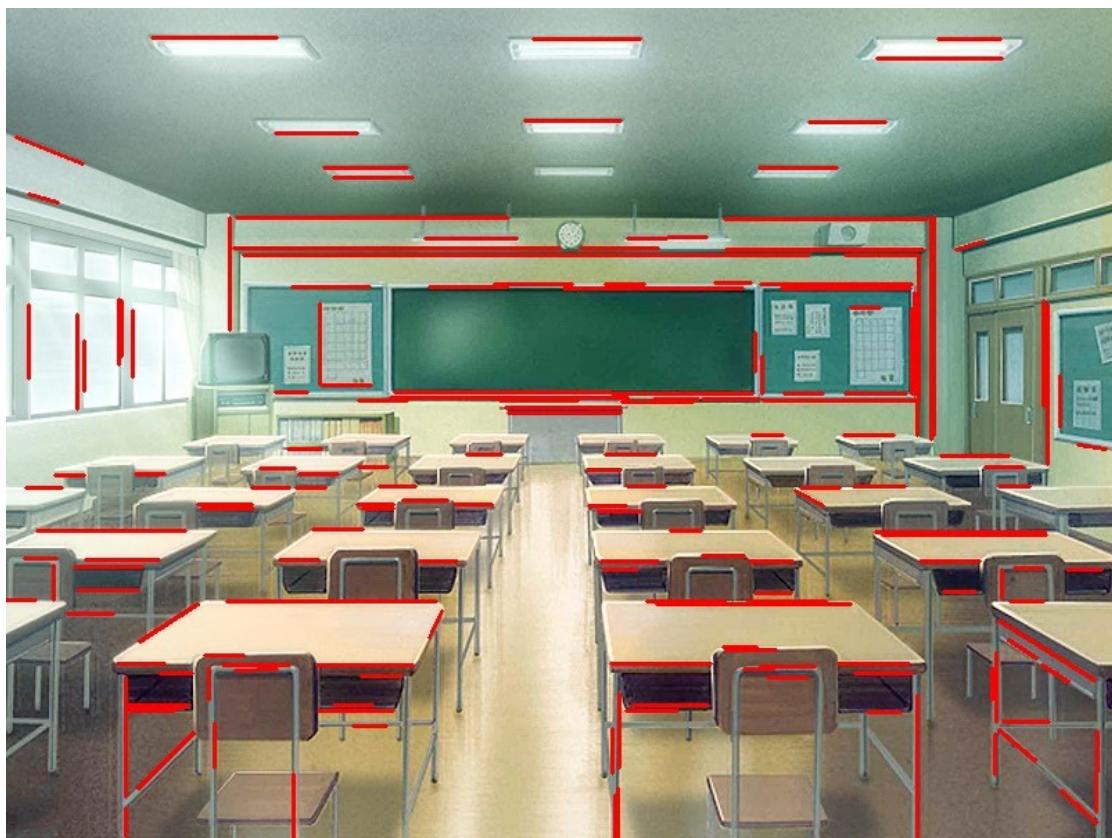


图 5-5 线段检测算法

- (3) 将（2）中得到的线段拓展并合并，规则是将同一方向上的直线片段进行扩展，并将位置十分接近的线段合并为一条线段，合并后的结果如图 5-6 所示：



图 5-6 线段合并效果

- (4) 将所有线段按照极坐标进行排序后，分别与相邻角度的线段求交点作为候选点，然后使用所有线段对所有候选点进行投票，具体操作为计算线段 I 中点到候选点 c 的直线和原线段的夹角 $\beta_{I,c}$ ，投票值为

$$Vote(I, c) = |I| * e^{\frac{\beta_{I,c}}{2u^2}} \quad (5-7)$$

其中 u 是一个鲁棒性参数。然后对投票之后的待选点做 K-Means 聚类，聚类的结束条件为候选点之间的最小距离大于 m 个像素点。

- (5) 对每个聚类中心计算票数加权中心，票数为聚类中所有待选点的票数之和，作为新的待选点。选择票数最高的聚类作为第一个输出点，并将所有线段中的点与该待选点的连线以及自身夹角小于 α 的线段删除。再转至第（4）步寻找下一个灭点。
 - (6) 若没有线段剩余，或者已经找到 V 个灭点，再或者已经没有待选点，则结束算法。
-

算法实际运行结果如图 5-7 所示，图中黄色圆点即为算法确定的灭点位置。

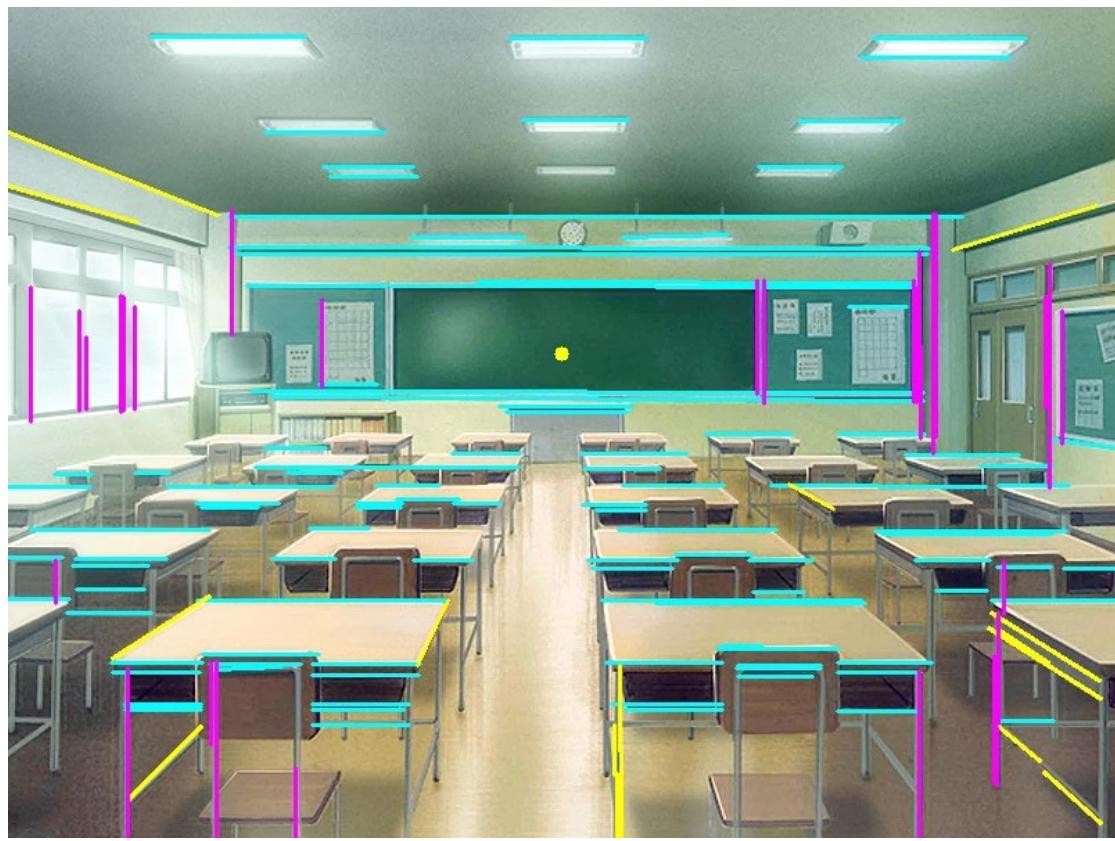


图 5-7 使用灭点自动定位算法得到的结果

算法的流程图如图 5-8 所示：

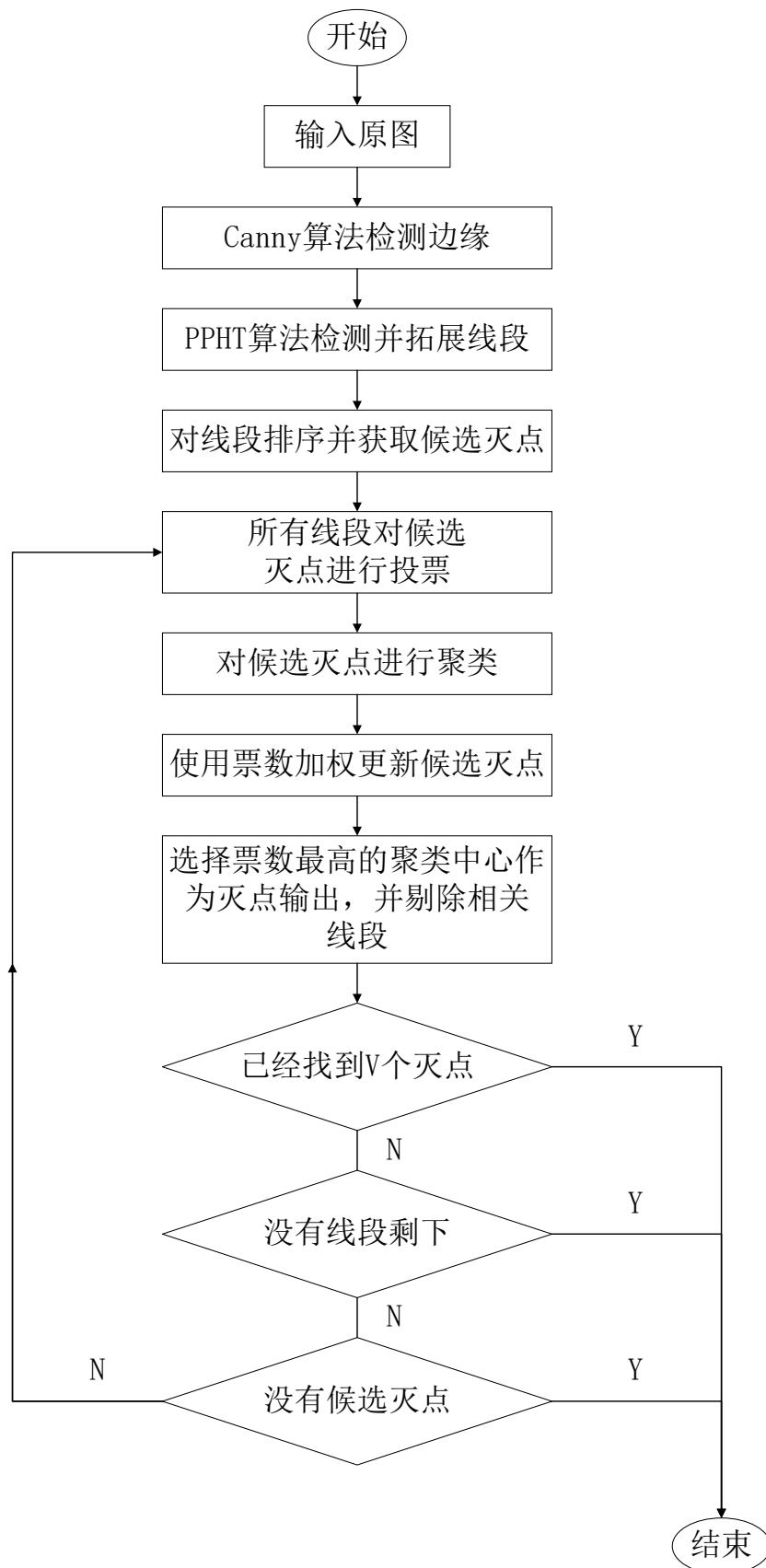


图 5-8 灭点自动定位算法流程图

5.2.4 车速估计模型

由视频中出现的高速公路标志牌可知，车辆行驶于 G42 高速公路上，其高速公路交通标志和标线符合国家标准（GB5768-1999），因此视频中的白色标线可作为车速估计的参照物。此外，我们假设汽车在较短的时间尺度内可以看作是匀速运动。

按照国家标准，我国国道高速公路中车行道分界线长度为 6m，每隔 9m 一个。为了得到准确的速度估计，我们对连续帧进行采样，采样规则是在采样开始帧和采样结束帧时，拍摄者所在车辆与道路标线的相对位置基本重合，则我们可以认为车辆在采样帧之间完成了一个相对于标线同一位置处的循环。具体操作中我们选取符合一组车行道分界线的初始帧和结束帧来估计车速。如图 5.9(a) 和 (b) 分别为第 38 帧和 56 帧时的图像，图中只截取了中间的一部分，足以判断是否符合初始帧和结束帧的条件。



图 5-9 车速估计模型的帧选择

类似地，我们共取了 10 组满足条件的初始帧和结束帧。由视频信息可知，其帧率为 30fps，因此我们可以由公式得到估计的车速(km/h)：

$$v = \frac{15}{\frac{c_{end} - c_{start}}{30}} * 3.6 \quad (5-8)$$

其中 v 表示估计的车速， c_{end} 和 c_{start} 分别表示结束帧和初始帧的帧序号。

我们对所有满足要求的初始帧和结束帧进行采样，估计得到的实时车速如表 5-1 所示：

初始帧 c_{start}	结束帧 c_{end}	所用时间/s	车速 km/h
9	27	0.6	90
38	56	0.6	90
74	92	0.6	90
100	117	0.567	95.294
128	145	0.567	95.294
163	181	0.6	90
199	216	0.567	95.294
233	251	0.6	90
286	304	0.6	90
294	311	0.567	95.294
平均		0.587	92.118

表 5-1 车速估计表

计算拍摄者所在车辆超过第一辆白色车辆(即驾驶座车门外印有“首汽共享汽车”字样的白色汽车)的距离时,由于我们已经算出拍摄者所在白色车辆的车速,假设两辆车在超越和被超越时都作匀速运动,则该问题转化为常见的两车追逐模型。

使用A表示拍摄者所在车辆,使用B表示白色车辆,则模型如图 5-10 所示:

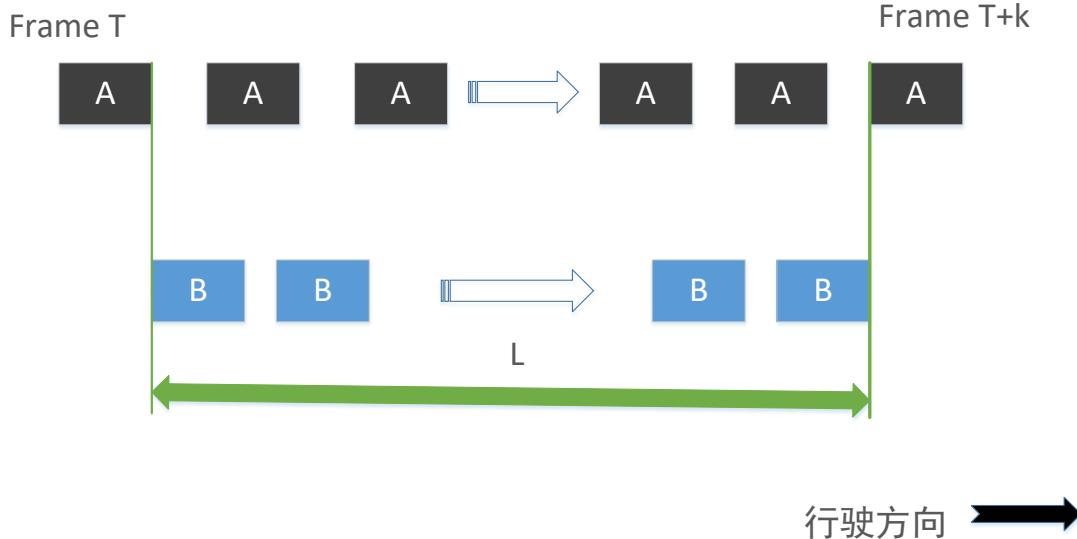


图 5-10 两车追逐模型

图中Frame T表示车A刚接近车B时所处的帧序号, Frame T + k表示车A恰好超过车B时的帧序号,若使用 $v_A(km/h)$ 表示车A的速度,使用 $v_B(km/h)$ 表示车B的速度,使用 l_A 表示车A的长度,则上述速度模型可以表示为:

$$\frac{v_A - v_B}{3.6} * \frac{k}{30} = l_A \quad (5-9)$$

5.3 模型求解

基于 5.2.2 中提出的单目视觉测距模型，我们可以得到拍摄者所在车辆与红色车辆的车距为：

$$d = 46.753 - 2.849 = 43.904m$$

基于 5.2.4 中提出的车速估计模型，我们可以得到拍摄者所乘车辆的速度为 92.118 km/h ，然后根据图 5-10 所示追逐模型，我们可以得到拍摄者所乘车辆在超越白色小车时两车的速度差异为： 23.328 km/h 。

5.4 结果分析

对于问题二，我们采用了简化成像模型的方法进行测距以及车速的估计，在不考虑后视镜本身的畸变以及忽略车辆行驶带来的成像抖动时，我们提出的模型能很好地对车距进行估计，并且鲁棒性较好；在车速估计问题上，我们充分挖掘视频中所蕴含的先验信息，并且考虑到视频的时间连续性以及连续帧间的运动可视为匀速运动模型后，我们提出的速度估计模型可以很好地估计出目标车辆的速度。

存在问题：我们所建立的测距模型中，没有考虑相机在水平位置上的偏角带来的误差，因此最终会导致测量误差，如果成像条件较好时，可以通过标定道路上的交通标线实现仿射变换，得到其单应性矩阵，从而将图像的水平方向畸变减小，再利用测距模型时，结果将会更加精确。

六、问题三

6.1 问题分析

问题三要求我们根据一段视频估计高铁行驶的速度，途径的河面宽度以及桥面距水面的高度。我们假设视频中高铁行驶在笔直的铁轨上，拍摄者的相机姿态保持不变，该问题的难点在于如何建立一个模型去估计高速行驶中的高铁速度，以及由于距离关系导致目标桥梁在视频中的成像质量较差带来的估计误差。

6.2 模型建立

6.2.1 多次测量取平均值

在我们的模型中，视频被认定为连续的图片，相当于给图片加入了冗余信息。考虑到量测的误差，系统误差不可避免，所以我们使用多次测量取平均值的方法来降低偶然误差。

视频中的每一帧都被认定为对物体进行的一次拍摄，我们的模型对每一次的拍摄进行一次量测，对于同一物体而言，我们将多次量测的算术平均值作为最终的测量结果，从而降低量测的偶然误差。

6.2.2 速度估计模型

这里我们采用 5.2.4 中提出的速度估计模型来估计车速，我们假设这一段视频内高铁处于匀速运动状态，注意到视频开始时经过的是一个科目二场地，其中车辆的长度基本处于 4.0m 至 4.5m 范围之内，因此我们选择截取相邻视频帧，估计下一帧时车辆较上一帧“移动”的距离，再利用 6.2.1 提到的多次测量取平均的方法得到高铁速度的近似估计。

6.2.3 长度估计模型

问题三中涉及的河面宽度以及桥梁距水面高度都可以看作是长度估计问题，我们采用算法 4.1 中提出的基于矩形的两点距离量测模型来对河面宽度和桥面高度进行估计，并且使用视频中出现的其他标志物如房屋等对估计模型做验证，同样采用 6.2.1 中的多次测量取均值的方法消除估计误差。在该估计过程中，由于河面并未整体出现在某一帧内，因此我们采用对多帧的标志点进行测量，选取标志点如桥墩与水面的交点，并利用 6.2.2 中的提出的速度估计模型对结果进行修正，从而在一段连续的视频帧内得出较准确的长度估计。

6.3 模型求解

基于 6.2.2 提出的速度估计模型，我们可以得到高铁的运行速度约为：307.853km/h。

基于 6.2.3 中提出的长度估计模型，我们得到桥面距离水面的高度为：11.226m，河面宽度为 235.622m

6.4 结果分析

我们首先根据高铁做匀速运动的假设再利用 6.2.2 中的速度估计模型，得到高铁运行速度的一个较为准确的估计值，并且经过分析可知该速度符合我国高铁运行的速度要求，因此可以作为之后长度估计问题的参考。然后我们利用问题一中提出的使用交比求长度的方法和估计得到的高铁速度，得到河面宽度和桥梁高度的估计值。

存在问题：误差主要来源于两方面，第一，在高铁速度估计模型中，我们选取的参考系是大地坐标系，但是由于高铁速度较快，参考物如车辆，距离拍摄者距离较远，会给速度估计带来误差，我们使用的多次测量取平均的方法可以在很大程度上提高测量的精度，但还是不能完全消除；第二，在长度估计模型中，我们需要确定灭点从而为使用交比求距离创造条件，但是由于成像质量的原因，平行线的选取比较困难，导致算法 5.2 中的灭点自动定位算法失效，不得不手动进行标记，这给后续的长度估计带来了误差。

七、 问题四

7.1 问题分析

问题四需要在航拍的立体视频中捕捉视觉情报，具体的子问题为估算环绕老宅道路的长度、宽度、各建筑物高度、后花园中树木的最大高度、估算老宅的占地面积、测算无人机的飞行高度和速度。由于图像清晰度较低，因此采用选取特定视频帧逐帧分析的方法得出量测信息，同时将视频信息看作是冗余的带时间序列的图片信息，分别使用多次测量取平均值、指数加权移动平均法恒定信息和实时信息，从而弥补逐帧处理带来的视频信息损失。

本题采用射影几何学的知识，挖掘视频中的隐藏信息，利用到了建筑物的影子，行人的速度来进行量测。

本题的主要难点在于高空拍摄，成像质量较差，而是用我们提出的基于图像特征的灭点自动定位算法可以大大降低人工操作带来的误差。

7.2 模型建立

7.2.1 指数加权移动平均法

由于本题要求测算无人机的飞行速度以及飞行高度，其量测数据会存在误差的干扰，而误差有正有负，因而会造成实时量测数据波动性较大，因此我们选用指数加权移动平均法对实时的量测进行平滑滤波处理。

指数加权移动平均法的算法如下：

算法 7.1 指数加权移动平均法

目标：对实时量测数据 θ 进行平滑滤波，降低数据输出的波动性。

算法：

- (1) 设置超参数 β ，其含义为当前数据想要平均之前 $1/(1-\beta)$ 个数据。
 - (2) 设置 $v_0 = 0$ 。
 - (3) 对于 $t(t \neq 0)$ 时刻， $v_t = \beta v_{t-1} + (1-\beta)\theta_t$ ，输出 v_t 。
-

指数加权移动平均法有一大问题在于，在平滑滤波初期值远远小于真实值，这是因为初期预测值从0开始增长，且增长速度很慢，因而会出现在一段时间内预测值一直落后于真实值的问题。在指数加权移动平均法中引入偏差修正可以很好地解决这个问题，其算法如下：

算法 7.2 待偏差修正的指数加权移动平均法

目标：对实时量测数据 θ 进行平滑滤波，降低数据输出的波动性。

算法：

- (1) 设置超参数 β ，其含义为当前数据想要平均之前 $1/(1-\beta)$ 个数据。
- (2) 设置 $v_0 = 0$ 。

(3) 对于 $t(t \neq 0)$ 时刻, $v_t = \beta v_{t-1} + (1 - \beta)\theta_t$, 输出 $v_t / (1 - \beta^t)$ 。

在引入了偏差修正之后, 指数加权移动平均法可以很好地对输入数据进行滤波, 在测试数据上的滤波效果如图 7-1 所示:

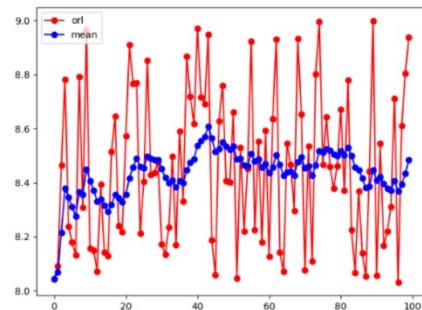


图 7-1 待偏差修正的指数加权移动平均法

7.3 模型求解

7.3.1 第一题



图 7-2 第一题求解示意 第 1759 帧

选取视频中第 1759 帧, 利用行人信息估算图片左小角的小花园信息, 估算长度为 12m*12m, 有了这个信息就可以根据第一大题中的算法估计所需量测信息:

物体	长度/m
正门围墙左半边	49.47
正门门宽	4.88
正面围墙全部	95.02
正面围墙（去除两个小花园）	71.19
侧面围墙	54.73

表 7-1 一次量测的数据

同时由于视角差异，本图推算得出大门并不在庄园正面正中间，与实际不符，因而使用大门正方向的视频帧第 90 帧进行修正，多次重复这个过程得到如下结果：



图 7-3 第一题求解示意 2 第 90 帧

物体	长度/m
正门门宽	5.27
正面围墙全部	101.48
正面围墙（去除两个小花园）	77.77
侧面围墙	55.01

表 7-2 多次测量取平均值的量测数据

计算高度时，以第 240 帧为例，假设庄园的房檐高度为 3m，多次测量取平均，结果如下：



图 7-4 第一题求解示意 3 第 240 帧

物体	高度/m
房屋	5.06
飞机	52.6

表 7-3 多次测量取平均值的量测数据

由于太阳光线可视为平行光，与光线平行的垂面都是相互平行的，利用太阳光影子，可以近似估算侧面物体高度，以第 504 帧和第 1050 帧为例，建立在屋檐高 3m 的假设上，我们可以得出：



图 7-5 第一题求解示意 4 第 504 帧



图 7-6 第一题求解示意 5 第 1050 帧

物体	高度/m
角楼	6.5
后花园中树木的最大高度	25

表 7-4 多次测量取平均值的量测数据

第一问最终的结果为：

物体	长度或高度/m
正门门宽	5.27
正面围墙全部	101.48
正面围墙（去除两个小花园）	77.77
侧面围墙	55.01
房屋	5.06
飞机高	52.6
角楼	6.5
后花园中树木的最大高度	25

表 7-5 第一问最终的量测数据

7.3.2 第二题

假设庄园由一个长方形和一个半圆形组成。

由上一问的结果可以推出长方形部分的面积为：

$$101.48 \times 55.01 = 5582.4148$$

半圆的面积为：

$$\pi \times (55.01 \div 2)^2 = 2376.69342$$

总面积为：

$$5582.4148 + 2376.69342 = 7959.10822$$

庄园的总面积为 7959.11 平方米。

7.3.3 第三题

利用第 170 帧的数据，测算出无人机的高度约为 50m，估计无人机在 65 秒内的运动距离为：

$$101.48 + 55.01 \times 2 + \pi * (55.01 \times 2 + 101.48) = 875.946835$$

于是无人机的速度为

$$875.946835 \div 65 = 13.4761052 (\text{m/s})$$

7.4 结果分析

本题中由于是无人机环绕采集的立体视频，拍摄距离远，对图像质量的影响较大，使用基于图像特征的灭点自动定位算法可以降低误差，但是如第 1759 帧所示，误差还是较大，而使用多次测量取平均值、指数加权移动平均法可以降低两侧误差，提升量测数据的精度。

八、模型的评价与改进

本文中提出的数学模型建立在射影几何的基础上，通过从图片中和视频中提取的先验信息提取灭点和灭线，利用交比定理得到所求信息与已知信息之间的数学关系，从而得到图片和视频中的量测信息。

模型的误差主要来自于镜头畸变以及参考信息细节不清晰造成的灭点提取不准，为了降低模型的误差，针对视频中待测目标较小的问题，使用基于目标跟踪算法的图像裁剪，使得待测目标始终在图像中占据中心位置，在降低处理数据量大小的同时，保证了量测的精度，考虑到视频信息中的冗余，人工选取平稳拍摄的帧进行距离估计，降低不同尺度下估计带来的误差，提出基于单目视觉的车辆测距模型。针对逐帧手工处理工作量过大的问题，提出了使用 Canny 算法提取边缘，概率霍夫变换检测目标中的线段，K-means 聚类选择灭点的基于图像特征的灭点自动定位算法。从而降低了人工提取信息带来的误差，并通过使用随机采样一致性算法提升特征提取的鲁棒性。针对灭点提取的误差在计算灭线时被放大，我们提出了计算多个灭点，用最小二乘法拟合生成灭线的方法，从而降低了误差。

模型对于带有时序的图像采取了逐帧处理的方法，将视频建模成对物体的多次测量。对于距离等信息的量测，采用多次测量取平均的方法，降低信息的量测误差，在实时量测中使用深度学习算法中常用的指数加权移动平均法，降低因测量误差带来的数据波动。

模型可以从两个方面进行改进，一方面是问题二中严重的镜头畸变带来的图像失真，如果在拍照前使用标定图片[3]获取成像设备的内参矩阵和外参矩阵，修正图像失真就可以得到更准确的量测。另一方面是对视频中时序信息的利用，将时间序列分析的方法引入模型，从而获取更多时间维度上的隐藏信息，将会提高量测的准确性。

由结果可以看出，借鉴射影几何思想可以极大简化视觉情报的分析过程，该思路有广泛的应用价值。

九、参考文献

- [1]. 隋铭明. 单幅近景图像空间信息提取及误差分析[D]. 南京师范大学, 2013.
- [2]. 王美珍. 单幅图像中地物目标几何量测研究[D]. 南京师范大学, 2011.
- [3]. Zhang Z . A Flexible New Technique for Camera Calibration[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000, 22(11):1330-1334.
- [4]. Canny J . A Computational Approach to Edge Detection [J]. 1986, 8(06)
- [5]. P.V.C. Hough, A method and means for recognizing complex patterns, US Patent: 3, 069, 654 (1962)
- [6]. J. Matas, C. Galambos, J. Kittler. Robust detection of lines using the progressive probabilistic hough transform. Comput. Vision Image Underst., 78 (1) (2000), pp. 119 – 137
- [7]. Park, K. and Hwang, S. Robust range estimation with a monocular camera for vision-based forward collision warning system. The Scientific World J., 2014, 1 – 9
- [8]. Han, J., Heo, O., Park, M. et al. Vehicle distance estimation using a mono-camera for FCW/AEB systems. Int. J Automot. Technol. (2016) 17: 483.
- [9]. 付晓宇. 基于单目视频的车辆测距算法研究[D]. 2016.
- [10]. 关闯, 魏朗, 乔洁, et al. 一种基于消隐点的单目视觉车辆测距方法[J]. 电子测量技术, 2018(11).
- [11]. 刘艳, 张伟, 厉健晖. 基于消隐点的摄像机标定关键技术研究现状及发展[J]. 传感器与微系统, 2018.

十、附录

1.demo 代码:

```
1. import numpy as np
2. import math
3. import cv2
4. from math import cos, pi, sin
5.
6. def tuple_toint(point):
7.     #绘图前需要整数
8.     return (int(point[0]), int(point[1]))
9.
10. def get_crosspoint(line1,line2):
11.     #求取相交点 line 输入样例格式(x,y)
12.     xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
13.     ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1]) #Typo was here
14.
15.     def det(a, b):
16.         return a[0] * b[1] - a[1] * b[0]
17.
18.     div = det(xdiff, ydiff)
19.     if div == 0:
20.         raise Exception('lines do not intersect')
21.
22.     d = (det(*line1), det(*line2))
23.     x = det(d, xdiff) / div
24.     y = det(d, ydiff) / div
25.     return (x, y)
26.
27.
28. def get_crossline(dian):
29.     #求灭线
30.     line1 = (dian[0],dian[1])
31.     line2 = (dian[2],dian[3])
32.     crosspoint1 = get_crosspoint(line1, line2)
33.
34.     line3 = (dian[0],dian[3])
35.     line4 = (dian[1],dian[2])
36.     crosspoint2 = get_crosspoint(line3, line4)
37.     return crosspoint1,crosspoint2
38.
39. def get_tuple_int(point):
```

```

40.     return ( int(point[0]), int(point[1]) )
41.
42. def compute_distance(point1, point2):
43.     (x1, y1) = point1
44.     (x2, y2) = point2
45.     d = np.sqrt(np.square(x1-x2) + np.square(y1-y2))
46.     return d
47.
48. def getAngle(point1, point2):
49.     #两点的 x、y 值
50.     (px1 ,py1) = point1
51.     (px2 ,py2) = point2
52.     x = px2-px1
53.     y = py2-py1
54.     hypotenuse = np.sqrt(np.square(x)+np.square(y))
55.     #斜边长度
56.     cos = x/hypotenuse
57.     radian = math.acos(cos)
58.     radian = math.acos(cos)
59.     if radian==0:
60.         return 0
61.     #求出弧度
62.     angle = 180/(np.pi/radian)
63.     #用弧度算出角度
64.     if (y<0) :
65.         angle = -angle
66.     elif (y == 0) and (x<0):
67.         angle = 180
68.     return angle
69.
70.
71. def get_point_inline(point1,point2,x):
72.     #计算 point1,point2 确定的直线上取 x 值时的点
73.     (x1, y1) = point1
74.     (x2, y2) = point2
75.
76.     if x1==x2:
77.         y = y1
78.         print('y = y1:', y)
79.     else:
80.         y = (y1 / (x - x1) -y2 / (x - x2))/(1 / (x - x1) -1 / (x - x2))
81.     return (x,y)
82.
83.

```

```

84. def get_pointer_rad(img,center):
85.     '''获取角度 求切线使用 粗定位'''
86.     c_y, c_x, depth =int(center[0]), int(center[1]), 1 # 142 ,124, 1#, shape[2]
87.     x1=max(img.shape[0], img.shape[1])
88.     src = np.zeros([img.shape[0],img.shape[1],3]) #img.copy()
89.     src = img
90.
91.     freq_list = []
92.     start_angle = 0
93.     debug = 0
94.     for i in range(start_angle,start_angle+359):
95.         x = x1 * cos(i * pi / 180) + c_x
96.         y = x1 * sin(i * pi / 180) + c_y
97.         temp = np.zeros(src.shape)#src.copy()
98.         cv2.line(temp, (c_x, c_y), (int(x), int(y)), (0, 0, 255), thickness=1)
99.         t1 = img.copy()
100.        if debug:
101.            cv2.imshow('t10', t1)
102.            t1[temp[:, :, 2] == 255] = 255
103.        if debug:
104.            cv2.imshow('t1', t1)
105.        c = img[temp[:, :, 2] ==255]
106.
107.        points = c[c == 255]
108.        if len(points)>0:
109.            freq_list.append((len(points), i))
110.        if debug:
111.            cv2.imshow('d', temp)
112.            cv2.imshow('d1', t1)
113.            cv2.waitKey(1)
114.    print('len(freq_list):',len(freq_list))
115.    return freq_list[0][1],freq_list[len(freq_list)-1][1]
116.
117.
118. def get_pointer_rad2(img,center,start_angle, end_angle):
119.     '''获取角度 求切线使用 精定位'''
120.     c_y, c_x, depth =int(center[0]), int(center[1]), 1
121.     x1=max(img.shape[0], img.shape[1])
122.     src = img
123.
124.     freq_list = []
125.     step = (end_angle-start_angle)/200
126.     debug = 0
127.     point_info = {}

```

```

128.     mask1 = cv2.inRange(src, np.array([125, 125, 125]), np.array([255, 255, 255]))
129.     for i in range(0,200):
130.         x = x1 * cos((start_angle + step * i) * pi / 180) + c_x
131.         y = x1 * sin((start_angle + step * i) * pi / 180) + c_y
132.         temp = np.zeros(src.shape,dtype=np.uint8)
133.
134.         temp = cv2.line(temp, (c_x, c_y), (int(x), int(y)), (0, 0, 255), thickness=1)

135.         img_masked = cv2.bitwise_and(temp, temp, mask=mask1)
136.         np_index = np.where(img_masked > 0)
137.         size = np_index[0].size
138.         if size>0:
139.             freq_list.append((size, start_angle + i*step,i))
140.             point_info[i] = np_index
141.         if debug:
142.             cv2.imshow('d', temp)
143.             cv2.waitKey(1)
144.     return freq_list[0][1],freq_list[len(freq_list)-1][1],point_info[freq_list[0][2]],point_info[freq_list[len(freq_list)-1][2]]
145. def add_tuple(point1, point2):
146.     return (point1[0] + point2[0], point1[1] + point2[1])
147.
148.

149. #直线的垂直向量
150. def get_vertical_vector(point1,point2):
151.     vector = (point1[0] - point2[0] , point1[1] - point2[1])
152.     if vector[0] == 0:
153.         vertical_vector = (1,0)
154.     else:
155.         vertical_vector = (-vector[1]/vector[0], 1)
156.     return vertical_vector
157.

158. #计算三角形垂心
159. def get_perpendicular_center(point1,point2,point3):
160.     vertical_vector1_3 = get_vertical_vector(point1, point3)
161.     line_v1_3 = (point2, add_tuple(point2,vertical_vector1_3))
162.
163.     vertical_vector2_3 = get_vertical_vector(point2, point3)
164.     line_v2_3 = (point1,add_tuple(point1,vertical_vector2_3))
165.     return get_crosspoint(line_v1_3, line_v2_3)
166.

167. #得到垂足点
168. def get_pedal_point(line,point):
169.     vertical_vector = get_vertical_vector(line[0], line[1])

```

```

170.     line_v = (point, add_tuple(point, vertical_vector))
171.     return get_crosspoint(line, line_v)
172.
173.
174.
175. def compute_distance(point1, point2):
176.     (x1, y1) = point1
177.     (x2, y2) = point2
178.     d = np.sqrt(np.square(x1-x2) + np.square(y1-y2))
179.     return d
180.
181. #计算交并比
182. def cross_ratio(c1, c2, c3, c4):
183.     cr = (compute_distance(c1, c3) * compute_distance(c2, c4)) / (compute_distance(c2,
184.         c3) * compute_distance(c1, c4))
185.     return cr
186. #计算交并比
187. def cross_ratio_v(x1, x2, x3, v):
188.     cr = (compute_distance(x1, x2) * compute_distance(x3, v)) / (compute_distance(x1,
189.         x3) * compute_distance(x2, v))
190.     return cr
191. #已经知道 x1,x2,求 x3,x4
192.     cr1 = cross_ratio_v(x1, x2, x3, v)
193.     x1_x2 = distance
194.     x1_x3 = x1_x2 / cr1
195.     cr2 = cross_ratio_v(x3, x4, x1, v)
196.     x3_x4 = x1_x3 / cr2
197.     return cr1,cr2,x3_x4
198.
199. def solve_height_demo():
200.     img1 = cv2.imread('./data/170.png')
201.
202.     point_1 = [[218, 434], [219, 464], [554, 516], [555, 481]]
203.     dian4_v1 = point_1
204.
205.     src1 = np.float32(dian4_v1).reshape(-1, 1, 2)
206.     for i in range(4):
207.         cv2.putText(img1, 'v1_{}'.format(i), (src1[i][0][0], src1[i][0][1]), cv2.FONT_
HERSHEY_COMPLEX, 1, (255, 255, 0), 1)
208.         img1 = cv2.circle(img1, (src1[i][0][0], src1[i][0][1]), 3, (0, 0, 255), -1)
209.
210.         line1 = (dian4_v1[0], dian4_v1[1])

```

```

211.     line2 = (dian4_v1[2], dian4_v1[3])
212.     crosspoint1_v1 = get_crosspoint(line1, line2)
213.     print('crosspoint1', crosspoint1_v1)
214.     crosspoint1_int_v1 = (int(crosspoint1_v1[0]), int(crosspoint1_v1[1]))
215.     img1 = cv2.circle(img1, (int(crosspoint1_v1[0]), int(crosspoint1_v1[1])), 20, (255
216. , 0, 0), -1)
217.     line3 = (dian4_v1[0], dian4_v1[3])
218.     line4 = (dian4_v1[1], dian4_v1[2])
219.     crosspoint2_v1 = get_crosspoint(line3, line4)
220.     crosspoint2_int_v1 = (int(crosspoint2_v1[0]), int(crosspoint2_v1[1]))
221.     print('crosspoint2', crosspoint2_v1)
222.     img1 = cv2.circle(img1, (int(crosspoint2_v1[0]), int(crosspoint2_v1[1])), 20, (255
223. , 0, 0), -1)
224.     dian4_h = [[82, 430], [18, 482], [732, 605], [728, 553]]
225.     src1 = np.float32(dian4_h).reshape(-1, 1, 2)
226.     for i in range(4):
227.         cv2.putText(img1, 'h_{}'.format(i), (src1[i][0][0], src1[i][0][1]), cv2.FONT_H
228. ERSHEY_COMPLEX, 1, (255, 255, 0),
229.                     1)
230.         img1 = cv2.circle(img1, (src1[i][0][0], src1[i][0][1]), 20, (0, 0, 255), -1)
231.         crosspoint1_h, crosspoint2_h = get_crossline(dian4_h)
232.         crosspoint1_h_int = (int(crosspoint1_h[0]), int(crosspoint1_h[1]))
233.         crosspoint2_h_int = (int(crosspoint2_h[0]), int(crosspoint2_h[1]))
234.         img1 = cv2.circle(img1, crosspoint1_h_int, 20, (255, 255, 0), -1)
235.         img1 = cv2.circle(img1, crosspoint2_h_int, 20, (255, 255, 0), -1)
236.         print('crosspoint_v2:', crosspoint1_h, crosspoint2_h)
237.
238.         line_cross_v1 = (crosspoint1_v1, crosspoint2_v1)
239.
240.         point_mid_img = (int(img1.shape[0] / 2), int(img1.shape[1] / 2))
241.         img1 = cv2.circle(img1, (point_mid_img[0], point_mid_img[1]), 40, (255, 0, 255), -
242. 1)
243.         line_cross_h = (crosspoint1_h, crosspoint2_h)
244.         point_mid_img_cross = get_pedal_point(line_cross_h, point_mid_img)
245.
246.         img1 = cv2.circle(img1, (int(point_mid_img_cross[0]), int(point_mid_img_cross[1]))
247. , 40, (255, 0, 255), -1)
248.         line_main = (point_mid_img_cross, point_mid_img)
249.         point_camera_projection = get_crosspoint(line_main, line_cross_v1)
249.         print('point_camera_projection:', point_camera_projection)

```

```

250.
251.
252.     line_hight = (dian4_v1[0], dian4_v1[1])
253.     height_crosspoint = get_crosspoint(line_cross_h, line_hight)
254.     vp_height = get_crosspoint(line_cross_v1, line_hight)
255.
256.     cr = cross_ratio_v(x1=dian4_v1[0], x2=dian4_v1[1], x3=height_crosspoint, v=vp_heig
ht)
257.     print('height_crosspoint:', height_crosspoint)
258.     print('cr:',cr)
259.     dis = 3
260.     hight_aircraft = dis / cr
261.     print('hight_aircraft:',hight_aircraft)
262.
263.
264.     cv2.namedWindow('img1', 0)
265.     cv2.imshow( "img1", img1 ) # 显示
266.     cv2.waitKey(0) # 按下任意键退出
267.     cv2.destroyAllWindows()
268.
269.
270. def solve_figure_demo():
271.
272.     img1 = cv2.imread('../img/img3.png')
273.
274.
275.     point_door = [[76, 518], [83, 583], [114, 584] ,[108, 519]]
276.     point_baluster = [[76, 518], [83, 583], [114, 584] ,[108, 519]]
277.
278.     #垂直面点
279.     dian4_v1 = point_door
280.     src1 = np.float32(dian4_v1).reshape(-1, 1, 2)
281.     for i in range(4):
282.         cv2.putText(img1, 'v1_{}'.format(i), (src1[i][0][0], src1[i][0][1]), cv2.FONT_
HERSHEY_COMPLEX, 1, (255, 255, 0),
283.                     1)
284.         img1 = cv2.circle(img1, (src1[i][0][0], src1[i][0][1]), 20, (0, 0, 255), -1)
285.
286.     line1 = (dian4_v1[0],dian4_v1[1])
287.     line2 = (dian4_v1[2],dian4_v1[3])
288.     crosspoint1_v1 = get_crosspoint(line1, line2)
289.     print('crosspoint1',crosspoint1_v1)
290.     crosspoint1_int_v1 = (int(crosspoint1_v1[0]),int(crosspoint1_v1[1]))

```

```

291.     img1 = cv2.circle(img1, (int(crosspoint1_v1[0]),int(crosspoint1_v1[1])), 20, (255,
292.     0, 0), -1)
293.     line3 = (dian4_v1[0],dian4_v1[3])
294.     line4 = (dian4_v1[1],dian4_v1[2])
295.     crosspoint2_v1 = get_crosspoint(line3, line4)
296.     crosspoint2_int_v1 = (int(crosspoint2_v1[0]), int(crosspoint2_v1[1]))
297.     print('crosspoint2',crosspoint2_v1)
298.     img1 = cv2.circle(img1, (int(crosspoint2_v1[0]), int(crosspoint2_v1[1])), 20, (255
299.     , 0, 0), -1)
300.     #垂直面点 2
301.     dian4_v2 = point_baluster
302.     src1 = np.float32(dian4_v2).reshape(-1, 1, 2)
303.     for i in range(4):
304.         cv2.putText(img1, 'h_{0}'.format(i), (src1[i][0][0], src1[i][0][1]), cv2.FONT_H
305.         ERSHEY_COMPLEX, 1, (255, 255, 0),
306.                     1)
307.         img1 = cv2.circle(img1, (src1[i][0][0], src1[i][0][1]), 20, (0, 0, 255), -1)
308.         crosspoint1_v2,crosspoint2_v2 = get_crossline(dian4_v2)
309.         crosspoint1_v2_int = (int(crosspoint1_v2[0]), int(crosspoint1_v2[1]))
310.         crosspoint2_v2_int = (int(crosspoint2_v2[0]), int(crosspoint2_v2[1]))
311.         img1 = cv2.circle(img1, crosspoint1_v2_int, 20, (255, 255, 0), -1)
312.         img1 = cv2.circle(img1, crosspoint2_v2_int, 20, (255, 255, 0), -1)
313.         print('crosspoint_v2:',crosspoint1_v2,crosspoint2_v2)
314.
315.     dian4_v2 = point_baluster
316.     src1 = np.float32(dian4_v2).reshape(-1, 1, 2)
317.     for i in range(4):
318.         img1 = cv2.circle(img1, (src1[i][0][0], src1[i][0][1]), 20, (0, 0, 255), -1)
319.
320.         crosspoint1_v2,crosspoint2_v2 = get_crossline(dian4_v2)
321.         crosspoint1_v2_int = (int(crosspoint1_v2[0]), int(crosspoint1_v2[1]))
322.         crosspoint2_v2_int = (int(crosspoint2_v2[0]), int(crosspoint2_v2[1]))
323.         img1 = cv2.circle(img1, crosspoint1_v2_int, 20, (255, 255, 0), -1)
324.         img1 = cv2.circle(img1, crosspoint2_v2_int, 20, (255, 255, 0), -1)
325.         print('crosspoint_v2:',crosspoint1_v2,crosspoint2_v2)
326.
327.     #水平面点
328.     dian4_h = [[660, 1226], [795, 1360], [1124, 1364], [944, 1231]]
329.     src1 = np.float32(dian4_h).reshape(-1, 1, 2)
330.     for i in range(4):
331.         img1 = cv2.circle(img1, (src1[i][0][0], src1[i][0][1]), 20, (0, 0, 255), -1)

```

```

332.
333.     crosspoint1_h,crosspoint2_h = get_crossline(dian4_h)
334.     crosspoint1_h_int = (int(crosspoint1_h[0]), int(crosspoint1_h[1]))
335.     crosspoint2_h_int = (int(crosspoint2_h[0]), int(crosspoint2_h[1]))
336.     img1 = cv2.circle(img1, crosspoint1_h_int, 20, (255, 255, 0), -1)
337.     img1 = cv2.circle(img1, crosspoint2_h_int, 20, (255, 255, 0), -1)
338.     print('crosspoint_v2:',crosspoint1_h,crosspoint2_h)
339.
340.     line_cross_v1 = (crosspoint1_v1, crosspoint2_v1)
341.     line_cross_v2 = (crosspoint1_v2,crosspoint2_v2)
342.
343.     point_mid_img = (int(img1.shape[1] / 2), int(img1.shape[0] / 2))
344.     print('point_mid_img:', point_mid_img)
345.     img1 = cv2.circle(img1, (point_mid_img[0], point_mid_img[1]), 40, (255, 0, 255), -
1)
346.
347.     line_cross_h = (crosspoint1_h,crosspoint2_h)
348.     point_mid_img_cross = get_pedal_point(line_cross_h, point_mid_img)
349.     img1 = cv2.circle(img1, (int(point_mid_img_cross[0]), int(point_mid_img_cross[1])) -
, 40, (255, 0, 255), -1)
350.     line_main = (point_mid_img_cross,point_mid_img)
351.     point_camera_projection = get_crosspoint(line_main, line_cross_v1)
352.     print('point_camera_projection:',point_camera_projection)
353.     point_camera_projection = get_crosspoint(line_main, line_cross_v2)
354.     print('point_camera_projection:',point_camera_projection)
355.
356.     #已知矩形，求取任意 2 点距离
357.     point_b = point_camera_projection
358.     point_a = (413,709)
359.     line_ab = (point_a,point_b)
360.
361.     point_rect1 = (1242, 1139)
362.     point_rect2 = (1159, 1072)
363.     point_rect3 = (1205, 1229)
364.     point_rect4 = (1116, 1144)
365.
366.     img1 = cv2.circle(img1, point_rect1, 20, (0, 0, 255), -1)
367.     img1 = cv2.circle(img1, point_rect2, 20, (0, 0, 255), -1)
368.     img1 = cv2.circle(img1, point_rect3, 20, (0, 0, 255), -1)
369.     img1 = cv2.circle(img1, point_rect4, 20, (0, 0, 255), -1)
370.
371.     cv2.putText(img1, 'p1', point_rect1 ,cv2.FONT_HERSHEY_COMPLEX, 3, (255, 255, 0),
5)

```

```

372.     cv2.putText(img1, 'p2', point_rect2, cv2.FONT_HERSHEY_COMPLEX, 3, (255, 255, 0), 5
    )
373.     cv2.putText(img1, 'p3', point_rect3, cv2.FONT_HERSHEY_COMPLEX, 3, (255, 255, 0), 5
    )
374.     cv2.putText(img1, 'p4', point_rect4, cv2.FONT_HERSHEY_COMPLEX, 3, (255, 255, 0), 5
    )
375.
376.     line24_rect = (point_rect2, point_rect4)
377.     vp24 = get_crosspoint(line24_rect, line_cross_h)
378.     line21_rect = (point_rect2, point_rect1)
379.     vp21 = get_crosspoint(line21_rect, line_cross_h)
380.
381.
382.     point_b1 = get_crosspoint(line_ab, line24_rect)
383.     point_b2 = get_crosspoint(line_ab, line21_rect)
384.     print('point_b1 point_b2:', point_b1, point_b2)
385.
386.     #在 line24 上线段 point_b1 point_rect2 point_rect4
387.     cr = cross_ratio_v(x1=point_rect2, x2=point_b1, x3=point_rect4, v=vp24)
388.     dis = 0.9
389.     distance_b1_2 = dis * cr
390.     #在 line34 上线段 point_b2 point_rect3 point_rect4
391.     cr = cross_ratio_v(x1=point_rect2, x2=point_b2, x3=point_rect1, v=vp21)
392.     cr2 = cross_ratio_v(x1=point_rect2, x2=point_rect1, x3=point_b2, v=vp21)
393.     print(cr, cr2, 1/cr)
394.     dis = 0.9
395.     distance_b2_2 = dis * cr
396.     disrance_b1_b2 = np.sqrt(np.square(distance_b1_2) + np.square(distance_b2_2))
397.
398.     dis = disrance_b1_b2
399.     print('disrance_b1_b2:', disrance_b1_b2)
400.     vp_ab = get_crosspoint(line_ab, line_cross_h)
401.     cr1, cr2, x3_x4 = cross_ratio_4point_v(x1=point_b1, x2=point_b2, x3=point_a, x4=po
        int_b, v=vp_ab, distance=dis)
402.     print('cr1*cr2', cr1*cr2, x3_x4)
403.     disrance_ab = dis/cr1/cr2
404.     print('disrance_ab:', disrance_ab, cr1, cr2)
405.
406.
407.
408.     cv2.namedWindow('img1', 0)
409.     cv2.imshow("img1", img1)
410.     cv2.waitKey(0)
411.     cv2.destroyAllWindows()

```

```
412.  
413.  
414. if __name__ == '__main__':  
415.  
416.     solve_figure_demo() #求解图三  
417.  
418.     #solve_height_demo() #求解飞机高度
```

2. 待偏差修正的指数加权移动平均法

```
1. import random  
2. import matplotlib.pyplot as plt  
3.  
4.  
5. def mean_average(data, num):  
6.     """  
7.         exponentially weighted averages with bias correction  
8.         :param data:  
9.             data: a list of data with noise  
10.            num: average how many data in the calculation  
11.            :return: new_data: a list of mean data  
12.        """  
13.    new_data = []  
14.    v = 0  
15.    length = len(data)  
16.    beta = 1 - (1 / num)  
17.    for i in range(length):  
18.        tmp = beta * v + (1 - beta) * data[i]  
19.        v = tmp  
20.        tmp = tmp / (1 - beta ** (i + 1))  
21.        new_data.append(tmp)  
22.  
23.    return new_data  
24.  
25. if __name__ == '__main__':  
26.     data = []  
27.     x = []  
28.     for i in range(100):  
29.         data.append(8 + random.random())  
30.         x.append(i)  
31.     new_data = mean_average(data, 10)  
32.     plt.plot(x, data, 'ro-', label='orl')  
33.     plt.plot(x, new_data, 'bo-', label='mean')  
34.     plt.legend()  
35.     plt.show()
```

3、求解图片 1 中两车距离和拍摄者距离左侧马路的距离：

```
1. import os
2. import numpy as np
3. import cv2
4.
5. def compute_distance(point1, point2):
6.     (x1, y1) = point1
7.     (x2, y2) = point2
8.     d = np.sqrt(np.square(x1-x2) + np.square(y1-y2))
9.     return d
10.
11.
12. def cross_ratio(c1, c2, c3, c4):
13.     cr = (compute_distance(c1, c3) * compute_distance(c2, c4)) / (compute_distance(c2,
14.         c3) * compute_distance(c1, c4))
15.     return cr
16.
17. def compute_fun(point1, point2):
18.     x1, y1 = point1
19.     x2, y2 = point2
20.     k = (y2-y1) / (x2-x1)
21.     b = y1 - k*x1
22.     return k, b
23.
24.
25. def predict_length(P1, P2, A1, A2, V1, V2):
26.     l1 = (P1, A1)
27.     v1 = (V1, V2)
28.     cross_point3 = get_crosspoint(l1, v1)
29.     l2 = (P2, cross_point3)
30.     la = (A1, A2)
31.     Ax = get_crosspoint(l2, la)
32.     cr = cross_ratio(A2, Ax, A1, V1)
33.     relative_length = known_length / cr
34.
35.     return relative_length
36.
37.
38. def predict_distance1(P1, P2, A1, A2, V1, V2, r1):
39.     l1 = (P1, A1)
40.     v1 = (V1, V2)
41.     cross_point3 = get_crosspoint(l1, v1)
```

```

42.     l2 = (P2, cross_point3)
43.     la = (A1, A2)
44.     Ax = get_crosspoint(l2, la)
45.     cr = cross_ratio(A2, Ax, A1, V1)
46.     require_distance = cr * rl
47.
48.     return require_distance
49.
50.
51. def predict_distance2(P1, P2, A1, A2, V1, V2, rl):
52.     # P1=P2, P2=P3
53.     l1 = (P1, A1)
54.     v1 = (V1, V2)
55.     cross_point3 = get_crosspoint(l1, v1)
56.     l2 = (P2, cross_point3)
57.     la = (A1, A2)
58.     Ax = get_crosspoint(l2, la)
59.     cr = cross_ratio(A2, Ax, A1, V2)
60.     require_distance = cr * rl
61.     return require_distance
62.
63.
64. def get_crosspoint(line1, line2):
65.     #求取相交点 line 输入样例格式(x,y)
66.     xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
67.     ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1]) #Typo was here
68.
69.     def det(a, b):
70.         return a[0] * b[1] - a[1] * b[0]
71.
72.     div = det(xdiff, ydiff)
73.     if div == 0:
74.         raise Exception('lines do not intersect')
75.
76.     d = (det(*line1), det(*line2))
77.     x = det(d, xdiff) / div
78.     y = det(d, ydiff) / div
79.     return (x, y)
80.
81. V1 = get_crosspoint((P2, P1), (P3, P4))
82. V2 = get_crosspoint((P2, P3), (P1, P4))
83. print('v1: {}, v2: {}'.format(V1, V2))
84.

```

```

85. relative_length = predict_length(P1=P1, P2=P2, A1=(720, 668), A2=(585, 684), V1=V1, V2=
   V2)
86. print(relative_length)
87.
88. distance2 = predict_distance1(P1=P1, P2=P2, A1=(1251, 599.5), A2=(585, 684), V1=V1, V2=
   V2, rl=relative_length)
89. print('distance of two cars: {} cm'.format(distance2))
90.
91. distance4 = predict_distance2(P1=P1, P2=P4, A1=(245, 828), A2=(1353, 915), V1=V1, V2=V2
   , rl=relative_length)
92. print('distance of half road: {} cm'.format(distance4))
93.
94. distance5 = predict_distance2(P1=P1, P2=P4, A1=(1352, 915), A2=(1818, 952), V1=V1, V2=V
   2, rl=relative_length)
95. print('distance of sidewalk: {} cm'.format(distance5))

```

4、求解图片 4 中 AB 和 CD 的长度

```

1. import os
2. import numpy as np
3. import cv2
4.
5. # 地砖长度为 80cm
6. known_length = 80
7.
8. def get_crosspoint(line1, line2):
9.     #求取相交点 line 输入样例格式(x,y)
10.    xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
11.    ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1]) #Typo was here
12.
13.    def det(a, b):
14.        return a[0] * b[1] - a[1] * b[0]
15.
16.    div = det(xdiff, ydiff)
17.    if div == 0:
18.        raise Exception('lines do not intersect')
19.
20.    d = (det(*line1), det(*line2))
21.    x = det(d, xdiff) / div
22.    y = det(d, ydiff) / div
23.    x = round(x)
24.    y = round(y)
25.    return (x, y)
26.
27.

```

```

28. def compute_distance(point1, point2):
29.     (x1, y1) = point1
30.     (x2, y2) = point2
31.     d = np.sqrt(np.square(x1-x2) + np.square(y1-y2))
32.     return d
33.
34.
35. def cross_ratio(c1, c2, c3, c4):
36.     cr = (compute_distance(c1, c3) * compute_distance(c2, c4)) / (compute_distance(c2,
37.         c3) * compute_distance(c1, c4))
38.     return cr
39.
40. # 根据一块地砖的四个角确定地砖平面的两个灭点位置
41. PP1 = (658, 1775)
42. PP2 = (1025, 1757)
43. PP3 = (656, 1934)
44. PP4 = (1139, 1907)
45. V = get_crosspoint((PP1, PP2), (PP3, PP4))
46.
47. # 已知一条直线上四点的坐标及其中一段线段的长度，并且已知该直线所在平面灭点的位置，可以求出这条
    直线任意线段的长度，从而求出阶梯长度
48. A = (492, 1638)
49. B = (658, 1633)
50. C = (924, 1621)
51. D = (1094, 1614)
52.
53. # 根据地砖长度得到台阶的近似长度
54. cr1 = cross_ratio(A, B, C, V)
55. AC = cr1 * known_length
56. cr2 = cross_ratio(C, D, A, V)
57. AD = AC / cr2
58. print('length of stair: {} cm'.format(AD))
59.
60. # 在台阶平面选取四个点确定台阶平面的两个灭点
61. P1 = (678, 1424)
62. P2 = (816, 1422)
63. P3 = (681, 1438)
64. P4 = (830, 1433)
65.
66. V1 = get_crosspoint((P1, P3), (P2, P4))
67. V2 = get_crosspoint((P1, P2), (P3, P4))
68. # print('v1 for stair plane: {}, v2 for stair plane: {}'.format(V1, V2))
69.

```

```

70. # 在图示 AB 线段上根据地砖长度及灭点求出 AB 长度
71. known_length1 = round(AD)
72. AA = (224, 1421)
73. BB = (561, 1411)
74. CC = (909, 1400)
75. DD = (1227, 1386)
76. VV = V2
77. ccrr = cross_ratio(AA, BB, CC, VV)
78. AACC = ccrr * known_length1
79. cr22 = cross_ratio(CC, DD, AA, VV)
80. AADD = AACC / cr22
81. AADD = 1007 / 355 * 181
82. print('length of AB: {}'.format(AADD))
83.
84. # 在垂直平面上选取四个点确定垂直平面的两个灭点
85. M1 = (500, 761)
86. M2 = (918, 754)
87. M3 = (497, 1385)
88. M4 = (966, 1370)
89.
90. m11 = (M1, M3)
91. m12 = (M2, M4)
92. VM = get_crosspoint(m11, m12)
93. # print('vanish-point of vertical plane: {}'.format(VM))
94.
95. # 根据图示 CD 点的位置及 VM 将其投影到图示 AB 线段上, 记为 MN
96. MC = (1026, 281)
97. MD = (355, 287)
98.
99. # 求得 MN 点的坐标
100. line1 = (VM, MC)
101. line2 = (VM, MD)
102. MN = (AA, DD)
103. MM = get_crosspoint(line1, MN)
104. NN = get_crosspoint(line2, MN)
105. temp = MM
106. MM = NN
107. NN = temp
108.
109. # 已知图示 AB 线段长度, 并且 ABMN 坐标及灭点位置均为已知, 则可求出 MN 长度
110. crm = cross_ratio(AA, MM, NN, VV)
111. crn = cross_ratio(AA, MM, DD, VV)
112.
113. MB = AADD / crn

```

```
114. AM = AADD - MB  
115. MN = AM / (crm-1)  
116. print('length of CD: {} cm'.format(MN))
```

5、求解距离红色车辆的车距

```
1. import os  
2. import numpy as np  
3. import cv2  
4.  
5.  
6. def compute_distance(point1, point2):  
7.     (x1, y1) = point1  
8.     (x2, y2) = point2  
9.     d = np.sqrt(np.square(x1-x2) + np.square(y1-y2))  
10.    return d  
11.  
12.  
13. def cross_ratio(c1, c2, c3, c4):  
14.     cr = (compute_distance(c1, c3) * compute_distance(c2, c4)) / (compute_distance(c2,  
c3) * compute_distance(c1, c4))  
15.    return cr  
16.  
17.  
18. def compute_fun(point1, point2):  
19.     x1, y1 = point1  
20.     x2, y2 = point2  
21.     k = (y2-y1) / (x2-x1)  
22.     b = y1 - k*x1  
23.    return k, b  
24.  
25.  
26. def predict_length(P1, P2, A1, A2, V1, V2):  
27.     l1 = (P1, A1)  
28.     v1 = (V1, V2)  
29.     cross_point3 = get_crosspoint(l1, v1)  
30.     l2 = (P2, cross_point3)  
31.     la = (A1, A2)  
32.     Ax = get_crosspoint(l2, la)  
33.     cr = cross_ratio(A2, Ax, A1, V1)  
34.     relative_length = known_length / cr  
35.  
36.    return relative_length  
37.  
38.
```

```

39. def predict_distance1(P1, P2, A1, A2, V1, V2, r1):
40.     l1 = (P1, A1)
41.     v1 = (V1, V2)
42.     cross_point3 = get_crosspoint(l1, v1)
43.     l2 = (P2, cross_point3)
44.     la = (A1, A2)
45.     Ax = get_crosspoint(l2, la)
46.     cr = cross_ratio(A2, Ax, A1, V1)
47.     require_distance = cr * r1
48.
49.     # la = (A1, A2)
50.     # lt = (P2, P1)
51.     # crosspoint_t = get_crosspoint(la, lt)
52.     # print(crosspoint_t, V1)
53.
54.     return require_distance
55.
56.
57. def predict_distance2(P1, P2, A1, A2, V1, V2, r1):
58.     # P1=P2, P2=P3
59.     l1 = (P1, A1)
60.     v1 = (V1, V2)
61.     cross_point3 = get_crosspoint(l1, v1)
62.     l2 = (P2, cross_point3)
63.     la = (A1, A2)
64.     Ax = get_crosspoint(l2, la)
65.     cr = cross_ratio(A2, Ax, A1, V2)
66.     require_distance = cr * r1
67.
68.     # # la = (A1, A2)
69.     # # lt = (P1, P2)
70.     # # crosspoint_t = get_crosspoint(la, lt)
71.     # # print(crosspoint_t, V2)
72.
73.     return require_distance
74.
75.
76. def get_crosspoint(line1, line2):
77.     #求取相交点 line 输入样例格式(x,y)
78.     xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
79.     ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1]) #Typo was here
80.
81.     def det(a, b):
82.         return a[0] * b[1] - a[1] * b[0]

```

```
83.  
84.     div = det(xdiff, ydiff)  
85.     if div == 0:  
86.         raise Exception('lines do not intersect')  
87.  
88.     d = (det(*line1), det(*line2))  
89.     x = det(d, xdiff) / div  
90.     y = det(d, ydiff) / div  
91.     return (x, y)  
92.  
93. l1 = (A, B)  
94. l2 = (C, D)  
95. l3 = (A, D)  
96. l4 = (B, C)  
97. V1 = get_crosspoint(l1, l2)  
98. V2 = get_crosspoint(l3, l4)  
99. print(V1)  
100.  
101. cr1 = cross_ratio(A, B, E, V1)  
102. AE = cr1 / (cr1-1) * known_length  
103. print(AE)  
104.  
105. cr2 = cross_ratio(A, B, F, V1)  
106. AF = cr2 / (cr2-1) * known_length  
107. print(AF)
```

6、灭点自动定位算法：

<https://github.com/SymenYang/Vanish-Point-Detection>