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	南京大学
参赛队号	19102840053
队员姓名	1.胡鑫雯
	2.王浩楠
	3.郑　睿

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　　C　视觉情报信息分析

摘　　要：

本文主要探讨图像与视频中的信息，下面针对每个问题给出方法与结果。

任务一：在该任务中主要建立了透视变换模型、Canny 边缘检测模型、景深图测距模型、平面等高转移模型对各个问题进行求解。针对测算图 1 中红车 A 车头和白车 B 车头之间的距离的问题：首先要先对原图进行了边缘检测处理，然后检测出的物体边缘进行平行查找，拟合出真实世界中的两组平行线。根据四个畸变坐标进行透视变换，最后参考汽车轴距 2.64 米，便可求得图像内 A 车头和 B 车头两点的距离为 21.826 米。

针对测算图 1 拍照者距马路左侧边界的距离：该距离等价于拍照者距马路右侧边界距离与马路宽的总和。问题的难点在于无法确定拍照者的准确位置，因此可将题意转化为：拍照者所站地的马路平行线与马路左侧边界的垂直距离。为此可以建立景深图模型，**本文创新性地将景深图最亮点近似为拍照者正前方的位置，并在模型中进行了正确性验证**。最终求得拍照者距离右边缘 2.097 米，马路总宽 12.6 米，所以问题中待求距离为 14.697 米。

针对测算图 2 中黑车 A 车头和灰车 C 车尾之间的距离：因为道路存在弯曲，所以对 A 车到 C 车的空间位置截取为三段，化曲为直。第一段为 A 车前轮与 B 车后轮，第二段为 B 车后轮到银车后轮，第三段为银车后轮到 C 车后轮，然后对每一段进行透视变换求解真实长度，最终求得长度为 26.712 米。针对测算拍照者距白色车辆 B 车头的距离：对于该问题，首先求得拍照者距离右侧停车线的垂直距离为 7.006 米、拍照者距离 B 车车头的平行方向距离 15.145 米，之后利用勾股定理便可求出答案 16.697 米。

针对测算图 3 中拍照者距离地面的高度：首先通过边缘检测模型，可以取得由点 A、B、C、D 构成的平面 ABCD。然后**通过本文创新性提出的平面等高转移模型，将拍照者距离地面的高度等价于该平面内的某段高度，该模型在文中也进行了正确性验证。**最后通过透视变换模型，选取自行车高度 1.2 米为参考高度，便可求得拍照者距离地面的真实高度 5.624 米。针对测算拍照者距岗亭 A 的距离：由于该马路无任何偏转，因此拍照者距离岗亭的距离可由两点之间水平距离 23.528 米和垂直距离 15.576 米通过勾股定理求得 28.217 米，其斜边值即为拍照者距离岗亭的真实距离，考虑到拍照者距离地面的高度为 5.624 米，因此最终结果为 28.772 米。

针对测算图 4 中塔底 AB 和塔顶 CD 长度：分析图像可知，该图存在一定的空间距离，地砖与塔底并不在同一水平面，因此直接进行透视变换误差较大。因此，我们的解题思路为：采用传递测量法化立体为平面。即通过地砖长度 0.8 米求台阶长度 1.824 米，通过台阶

长度求单个塔前小砖长度 0.601 米，通过塔前砖长求塔底 AB 长度为 5.38 米，通过塔底 AB 长度求塔顶 CD 长度为 4.308 米。针对测算塔底 AB 和塔顶 CD 之间的高度：假设四边形 ABCD 为等腰梯形，通过透视变换可求得等腰梯形的下底角为 85.7 度，因此通过简单的数学方法便可求得塔底与塔顶之间的高度为 7.128 米。

任务二：在该任务中主要采用了透视变换模型和边缘检测模型。针对测算视频 2 中该车和后方红色车辆之间的距离：通过视频分析，获取红车和白车并排行驶时的特殊时刻视频帧第 231 帧。提前求得两车间相对速度为 5.64m/s，在红车与该车之间的距离保持不变的假设前提下，此时在道路径向方向上，该车与红车的距离应等价于该车与白车水平方向的距离 44.18 米。针对测算视频 2 中该车超越第一辆白色车辆时两车的速度差异：首先对视频进行帧处理提取出 2 个关键的视频帧：白车即将出现 101 帧和白车即将消失 126 帧，又因为车长已知 4.7 米，所以可求解得相对速度的值 5.64m/s。

任务三：该任务主要采用了光流测速模型。针对测算高铁行驶方向左侧第一座桥桥面距水面的高度：首先选取两个特殊的视频帧 245 帧和 246 帧，保证桥墩所在平面与镜头所在平面平行。取桥墩左边进行光流检测，提前求得高铁速度为 304.56km/h，便可知桥墩的宽，由其宽高比得到桥高，即桥面距水面的高度 11.844 米。针对测算桥距高铁轨道的距离：假设桥与两岸互相垂直，通过视频帧数与提前求得的高铁速度可求在该平面内水面斜边的值为 327.12 米，提前求解下问可知水面宽度为 219.96 米。通过勾股定理，求得桥距离高铁轨道 242.13 米。针对测算水面宽度：选取两个特殊的视频帧，左下角点刚好到达水面左边界的 157 视频帧和左下角的点刚好到达水面右边界的 235 视频帧，该运动中恰好行驶了水面的整个宽度，当知道高铁速度时，便可求得水面宽度 219.96 米。针对测算拍摄时高铁的行驶速度：以连续两帧 30、31 作为光流测速模型的输入，根据两帧中同一参照点的相对位移求得高铁的移动速度 304.56km/h。

任务四：该任务采用了光流测速模型、透视变换模型和平面等高转移模型。针对测算其中环绕老宅道路的长度、宽度的测量：在已求得无人机飞行速度 14.535m/s 的前提下，可采用光流测速模型，选取人的相对位移作为参照物，便可求得老宅外围直线路长为 379.491 米，弧线路长为 596.103 米，路宽为 17.892 米。针对老宅建筑高度和后花园最高树的高度：主要通过透视转换模型进行求解，假设门高固定为 2 米，可求解各个建筑的高度，根据建筑高度的透视变换，也可求得后花园最高树约 16.668 米。针对老宅的面积：其等价于一个正方形面积加上一个半圆面积，该半圆直径为整个四合院外围道路长度，求得老宅总面积为 200567.357m^2。针对无人机的飞行高度：高度计算方向可通过本文提到的平面等高转移模型可进行求解，其中参照物为老宅双层楼房的高度 10 米，最后求得的高度为 30.869 米。针对无人机飞行速度：采用光流测速模型，选取大巴车宽度 2.5 米作为参照物，便可求得无人机飞行速度为 14.525m/s。

关键字：透视变换模型；边缘检测模型；景深图；光流测速

目　录

一、问题重述

1.1 问题背景

研究表明，一般人所获取的信息大约有 80%来自视觉。视觉信息的主要载体是图像和视频，视觉情报指的是通过图像或者视频获取的情报。

从图像或视频中提取物体的大小、距离、速度等信息是视觉情报分析工作的重要内容之一。在当前很热门的移动机器人、无人驾驶、计算机视觉、无人机侦察等领域，更是存在着大量的应用需求。为了获得视频或图像中的信息需要综合考虑各种因素，通过合适的数学模型来提取。

1.2 问题的提出

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。

任务一：对单幅图像进行信息处理，（1）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（该距离不考虑拍照者高度）；（2）测算图 2 中黑色车辆 A 车头和灰色车辆 C 车尾之间的距离、拍照者距白色车辆 B 车头的距离（该不考虑拍照者高度）；（3）测量图 3 中拍照者距离地面的高度、拍照者距岗亭 A 的距离（考虑拍照者离地面的高度）；（4）在已知地砖尺寸为 80cm×80cm 前提下，测量图 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。

图 1

图 2

图 3

图 4

任务二：通过对“车辆.mp4”视频（别克英朗 2016 款车上乘客通过后视镜拍摄的视频）进行分析，（1）测算该车和后方红色车辆之间的距离；（2）测算该车超越第一辆白色车辆时两车的速度差异；

任务三：通过对“水面.mp4”视频（高铁乘客拍摄的一块水面的视频）进行处理，（1）测算高铁行驶方向左侧第一座桥桥面距水面的高度；（2）测算桥距高铁轨道的距离以及水面宽度；（3）测算拍摄时高铁的行驶速度。

任务四：通过对“无人机拍庄园.mp4”视频进行分析，（1）测算其中环绕老宅道路的长度、宽度、各建筑物的高度、后花园中树木的最大高度；（2）测算该老宅的占地面积；（3）测算无人机的飞行高度和速度。

二、问题分析及建模准备

2.1 任务一的分析

（1）针对测算图 1 中红车 A 车头和白车 B 车头之间的距离的问题：首先要先对原图进行边缘检测处理，然后检测出的物体边缘进行平行查找，拟合出真实世界中平行的四条线。根据四个畸变坐标进行透视变换，最后参考汽车轴距，便可求得图像内 A 车头和 B 车头两点的距离。针对测算拍照者距马路左侧边界的距离：该距离等价于拍照者距马路右侧边界距离与马路宽的总和。问题的难点在于无法确定拍照者的准确位置，因此可将题意转化为：拍照者所站地的马路平行线与马路左侧边界的垂直距离。为此可以建立景深图模型，创新性地将深景图最亮点近似为拍照者正前方的位置。

（2）针对测算图 2 中黑车 A 车头和灰车 C 车尾之间的距离：图 2 可知，该道路存在弯曲，因此要对 A 车到 C 车的空间位置截取为三段，化曲为直。第一段为 A 车前轮与 B 车后轮，第二段为 B 车后轮到银车后轮，第三段为银车后轮到 C 车后轮，然后对每一段进行透视变换求解真实长度。针对测算拍照者距白色车辆 B 车头的距离：对于该问题，首先要求得拍照者距离右侧停车线的垂直距离、拍照者距离 B 车车头的平行方向距离，之后利用勾股定理便可求出答案。

（3）针对测算图 3 中拍照者距离地面的高度：首先通过边缘检测模型，可以取得由点 A、B、C、D 构成的平面 ABCD。然后通过本文创新性提出的平面等高转移模型，将拍照者距离地面的高度等价于该平面内的某段高度。最后通过透视变换模型，选取自行车高度为参考高度，便可求得拍照者距离地面的真实高度。针对测算拍照者距岗亭 A 的距离：由于该马路无任何偏转，因此，拍照者距离岗亭的距离可由两点之间平行距离和垂直距离通过勾股定理求得，其斜边值即为拍照者距离岗亭的真实距离。

（4）针对测算图 4 中塔底 AB 和塔顶 CD 长度：分析图像可知，该图存在一定的空间距离，地砖与塔底并不在同一水平面，因此直接进行透视变换误差较大。因此，我们的解题思路为：采用传递测量法化立体为平面，即通过地砖长度求台阶长度，通过台阶长度求单个塔前小砖长度，通过塔前砖长求塔底 AB 长度，通过塔底 AB 长度求塔顶 CD 长度。针对测算塔底 AB 和塔顶 CD 之间的高度：假设四边形 ABCD 为等腰梯形，通过透视变换可求等腰梯形的下底角，因此通过简单的数学方法便可求得塔底与塔顶之间的高度。

2.2 任务二的分析

（1）针对测算视频 2 中该车和后方红色车辆之间的距离：通过视频分析，获取红车和白车并排行驶时的特殊时刻视频帧。在红车与该车之间的距离保持不变的假设前提下，此时在道路径向方向上，该车与红车的距离应等价于该车与白车水平方向的距离。

（2）针对测算视频 2 中该车超越第一辆白色车辆时两车的速度差异：首先对视频进行帧处理。针对该问题提取出 2 个关键的视频帧：白车即将出现和白车即将消失的视频帧，通过帧率获得时间。又因为车长已知，所以可求解得相对速度的值。

2.3 任务三的分析

（1）针对测算高铁行驶方向左侧第一座桥桥面距水面的高度：首先选取两个特殊的视频帧，保证桥墩所在平面与镜头所在平面平行。取桥墩左边进行光流检测模型，当获得高铁速度时，便可知桥墩的高，即桥面距水面的高度。

（2）针对测算桥距高铁轨道的距离：提前求解下问得高铁速度为 304.56km/h，选取两个特殊视频帧（水面刚出现在视频中左下角的帧和桥端完全消失在视频中的帧）。根据帧数可得时间，于是便知道了水面的斜边，通过勾股定理可求解的桥距离高铁轨道的距离。针对测算水面宽度：选取两个特殊的视频帧，左下角点刚好到达水面左边界的视频帧和左下角的点刚好到达水面右边界的视频帧，即运动中恰好行驶了水面的整个宽度。当知道高铁速度时，便可求得水面宽度。

（3）针对测算拍摄时高铁的行驶速度：帧率为 30 帧/秒，光流测速模型以连续两帧作为输入，根据两帧中同一参照点的相对位移求得高铁的移动速度。

2.4 任务四的分析

（1）针对测算其中环绕老宅道路的长度、宽度、各建筑物的高度、后花园中树木的最大高度；对于长度的测量，在已求得拍摄速度的前提下，可采用光流测速模型，通过两个特殊连续视频帧的输入，选取相应参照物，便可求得图像中任意两点的距离。

（2）针对测算该老宅的占地面积：由于四合院的长度已知，所以老宅的面积等价于一个正方形面积加上一个半圆面积，该半圆直径为整个四合院外围道路长度。

（3）针对测算无人机的飞行高度：高度计算方向，可通过本文提到的平面等高转移模型可进行求解，其中参照物为双层楼房的高度。针对无人机飞行速度：帧率为 30，对于无人机飞行速度的测量，可采用光流测速模型，通过两个特殊连续视频帧的输入，选取相应参照物，便可求得飞行速度。

三、基本假设

（1）假设城市交通中黄白线的宽度为标准宽度 0.15m。

（2）假设题目中所有马路严格标准对称。

（3）假设题中所有汽车（参照福特车）总长为 4.7m，前后车轮轴距为 2.64m，车头为 1.2m，车尾 0.86m。

（4）假设题中所有自行车高度为 1.2 米。

（5）假设任务一图 2 中，短距离长度内可忽略道路偏转角度。

（6）假设任务一图 4 中，四边形 ABCD 为标准的等腰梯形。

（7）假设任务二中，视频中该车与红车之间的距离始终保持不变。

（8）假设任务三中，视频中汽车、房屋等均是静止不动。

（9）假设任务三中，视频拍摄的相机相对高铁静止。

（10）假设任务四中，人的步行速度一般为 1.275m/s。

（11）假设任务四中，老宅外围直线道路同长，所有道路同宽，后花园为半圆状。

（12）假设任务四中，大巴车的真实宽度为 2.5 米。

（13）假设任务四中，老宅的门高为 2 米。

四、符号说明

符号	说明
l_{refer}	参照物的真实长度。
l_{refer_pixel}	参照物的像素长度
l_{pixel}	图像中待求两点的像素长度

五、任务一求解

5.1 模型介绍

5.1.1 透视变换模型

透视变换目标是将图像投影到一个新的视平面。利用透视中心、像点、目标点三点共线的条件，根据透视旋转定律使透视面绕透视轴旋转某一角度。破坏原有投影光线束，仍保持透视面上投影几何图形不变的变换[1]。针对任务一中测量图像真实距离的问题，我们使用一种平行透视变换方法，用于畸变结构光图像的自校正。当然，我们会使用精准的边缘检测模型，使得透视变换矩阵准确，从而得到精确的结果。

假如点 S 为视点，经验可知视线与矩形 ABCD 所在的平面存在一定的夹角，使得投影面上的图像发生透视畸变，从而会形成新的四边形 $A'B'C'D'$，如图 5 所示。

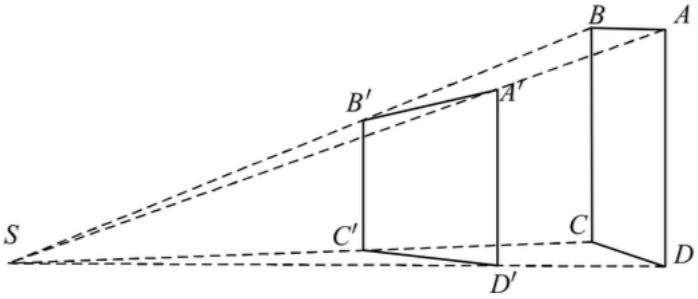

图 5:透视变换原理图

以图中的四个角点为例，经过透视变换后，A 点对应 A′点，B 点对应 B′点，C 点对应 C′点，D 点对应 D′点，原四边形的形状发生改变。透视投影是矩形 ABCD 平面上的每一个点在视角的作用下投影到 A′B′C′D′平面的过程。若矩形 ABCD 为无畸变正视图，则透视变换是畸变图形 A′B′C′D′上的每一个像素点对应到正视图上对应的像素点的过程[2]。而实现透视畸变的矫正，就是要找到 A′B′C′D′平面上的点与正视图上的点的一一对应关系。

通用的变换公式为：

$$[x', y', w'] = [u, v, w]\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{1}$$

$$x = \frac{x'}{w'}, \quad y = \frac{y'}{w'} \tag{2}$$

其中，u 和 v 是原始图像的坐标，经过透视变换后的坐标变为 x 和 y 。$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ 是透视变换矩阵，可拆为四部分，其中 $\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ 是线性变换部分，$\begin{bmatrix} a_{31} & a_{32} \end{bmatrix}$ 的作用是平移，$\begin{bmatrix} a_{13} & a_{23} \end{bmatrix}$ 用于产生透视变换。所以(1)式和(2)式联立可以整理为：

$$x = \frac{x'}{w'} = \frac{a_{11}u + a_{21}v + a_{31}}{a_{13}u + a_{23}v + a_{33}} \tag{3}$$

$$y = \frac{y'}{w'} = \frac{a_{12}u + a_{22}v + a_{32}}{a_{13}u + a_{23}v + a_{33}} \tag{4}$$

综上可得，(x,y) 是正视图的像素坐标，(u,v) 是畸变图像的像素坐标。因此可以推出，只要得到四个畸变图像与正式图像相对应的坐标就可以解出透视变换参数。因此，我们根据透视变换之后的图像，寻找已知像素距离的参照物长度，便可以求解图像中两点间的真实距离。

$$l_{pixel} = (x_b - x_a) \tag{5}$$

$$l = \frac{l_{refer}}{l_{refer\_pixel}} \times l_{pixel} \tag{6}$$

其中，l_{refer} 是参照物的真实长度，l_{refer_pixel} 是参照物的像素长度，l_{pixel} 是图像中待求两点的像素长度。

5.1.2Canny 边缘检测算法

四点的选择对透视变换矩阵的精度有着很大的影响，因此本文采用 Open CV 的改进版 Canny 边缘检测和轮廓检测算法，从而确定四条边缘线的交点。

1986 年，Canny 定义了边缘检测的 3 个检测准则，并在此基础上提出了 Canny 边缘检测算子。主要分为图像平滑、梯度幅值大小和方向的计算、非极大值抑制和检测连接四部分。

(1)图像平滑。

用高斯滤波器对图像进行去噪声和平滑处理，目的在于减少噪声对梯度计算的影响，高斯滤波函数公式为：

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2+y^2}{2\sigma^2}) \tag{7}$$

(2)计算梯度幅值大小和方向

Canny 算法的基本思想是寻找一幅图像中灰度值变化最强的位置，平滑后的图像可以由 Sobel 算子来获得梯度的幅值 M 和梯度的方向θ。

$$M=\sqrt{G_x^2+G_y^2+G_{45}^2+G_{135}^2} \tag{8}$$

$$\theta=\arctan(G_y/G_x) \tag{9}$$

检测图像x和y方向的 Sobel 算子分为：

$$S_x=\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix}\quad S_y=\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}\quad S_{45^\circ}=\begin{bmatrix}-2&-1&0\\-1&0&1\\0&1&2\end{bmatrix}\quad S_{135^\circ}=\begin{bmatrix}0&1&2\\-1&0&1\\-2&-1&0\end{bmatrix}$$

（3）非极大值抑制

此步骤得到梯度幅值内最大的像素点，将此点确定为边缘点。采用非极大值抑制技术来消除边缘误差，可以将模糊的边界变得清楚。

（4）双阈值检测和边缘连接

对(x,y)的梯度幅值进行边缘点检测。采用双阈值技术对非极大值抑制后的图像做进一步处理，然后用滞后技术跟踪边缘，进行边缘连接。

5.1.3 利用景深图模型确定正前方位置（创新模型）

（1）模型介绍

在 3D 计算机图形中，Depth Map（深度图）是包含与视点的场景对象的表面的距离有关的信息的图像或图像通道。其中，Depth Map 类似于灰度图像，只是它的每个像素值是传感器距离物体的实际距离。通常 RGB 图像和 Depth 图像是配准的，因而像素点之间具有一对一的对应关系。所以我们采用景深图，来近似估计拍照者正前方位置。

（2）模型验证

为了验证本文提出的通过景深图确定拍照者正前方的位置，我们选取场景进行拍照测验。通过图 8 可知，蓝色点为拍照者正前方位置。通过对场景图 6 进行景深图转换得到图 7，并求得场景中最亮点为点 B。

已知 A 点坐标（2649,3941），B 点坐标（3008,3976），所以通过勾股定理，可求得 AB 长度为 360.702 个像素。因为 360.702 个像素长度在图片像素长宽为 3009*4010 的尺度下，距离较短，可以忽略不计，从而证得：可以将景深图模型中最亮的点近似为拍照者正前方位置。

图 6：深景图结果

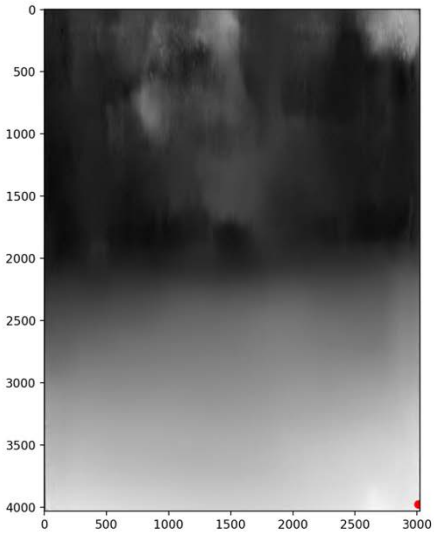

图 7：深景图

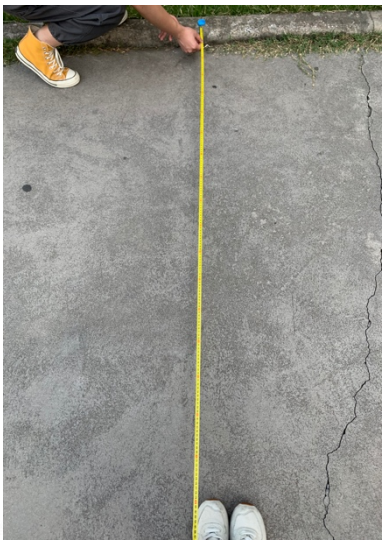

图 8：深景图验证图

5.1.4 平面等高转移模型（创新模型）

（1）模型介绍

考虑到图像中拍照者距离地面的高度无法直接得到，因此本文创新性提出平面等高转移模型进行测距。如图 9 所示，通过边缘检测模型，可以取得由点 A、B、C、D 构成的平面 ABCD，交叉线的交点与一边长的中点可构成该平面等高投影线，由于给定定点 G，根据等腰三角形知识，便可确定点 H 的位置，因此线段 HG 为等高转移后的距离。

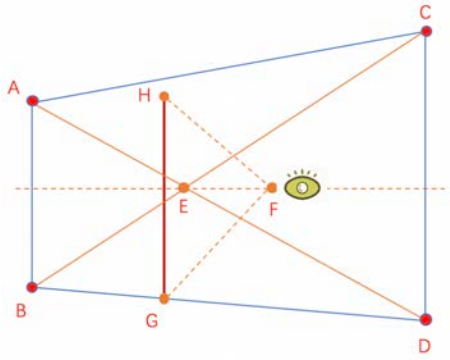

图 9：平面等高转移模型原理图

如果已知该平面内参照物的高度，通过透视变换，便可将拍照者距离地面的高度等价于平面 ABCD 内线段 HG 的真实高度。

（2）模型验证

由图 10 和图 11 可得灌木丛高 1.18 米，测试员眼睛高度为 1.52 米。为了验证本文提出的平面等高转移模型的准确性，我们选取场景进行拍照测验，通过对场景建立平面等高转移模型，求解拍照镜头的高度。

如图 12，我们对该场景建立平面等高转移模型，将拍照的高度转移为 DG 的高度。通过对图 12 进行投影变换可得图 13，根据图 13 可得，灌木丛垂直方向上像素差 270，线段 DG 之间像素差为 349。因此，可得预测的拍照镜头高度为 1.525 米。

$$h = l_{DG} = \frac{l_{HI}}{l_{HI\_pixel}} \times l_{DG\_pixel} = \frac{1.18m}{270} \times 349 = 1.525m$$

预测的拍照镜头高度为 1.525 米，真实的镜头拍照高度为 1.52 米，误差为 0.005m 即 5 毫米可忽略不计。因此，可证明本文提出的平面等高转移模型的正确性。

图 10：灌木丛测量图

图 11：拍照者人眼高度

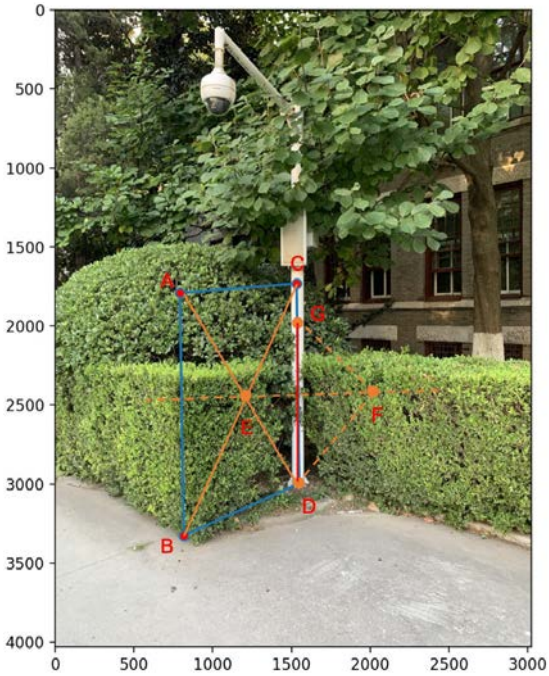

图 12：测试场景图

图 13：测试图的投影变换图

5.2 模型的求解

5.2.1 针对任务一图 1 问题的求解

（1）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

分析：首先要先对原图进行了边缘检测处理，然后检测出的物体边缘进行平行查找，拟合出真实世界中平行的四条线。根据四个畸变坐标进行透视变换，最后参考汽车轴距，便可求得图像内 A 车头和 B 车头两点的距离。

求解：通过边缘检测处理和平行查找程序，拟合出四条平行的四条线，最后得到图像中四个畸变坐标，其中右上、右下、左上、左下坐标分别为（1288,544）、（1290,603）、（449,579）、（451,708）。

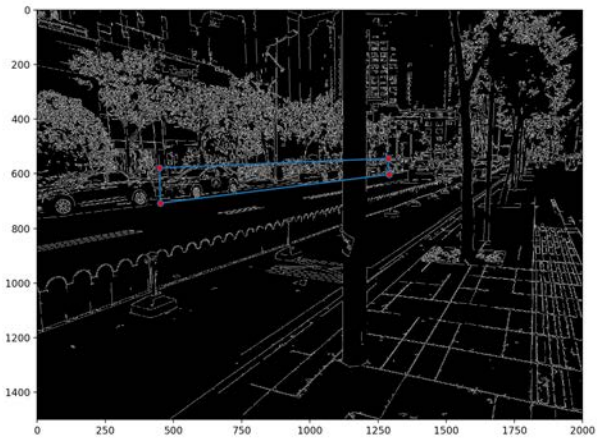

图 14：小轿车边缘检测图

图 15：小轿车效果图

通过透视变换模型，我们得到的平行图像如下图所示:

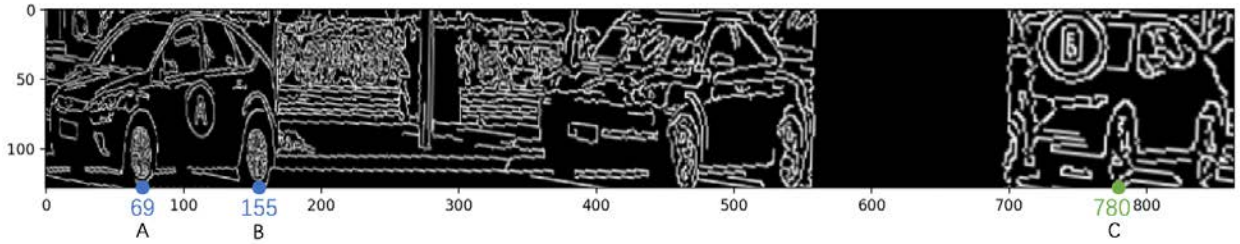

图 16:边缘检测图的透视变换图

图 17：彩色图的透视变换图

已知点 A 与点 B 之间的车轮轴距为 2.64 米，相距 86 个像素点，点 A 与点 C 之间相距 711 个像素点。通过求解公式（5）和公式（6），可求得红色 A 车车头与白色 B 车车头的距离为 21.826 米。

$$l_{AC}=\frac{l_{AB}}{l_{AB\_pixel}}\times l_{AC\_pixel}=\frac{2.64m}{86}\times 711=21.826m$$

（2）测算拍照者距马路左侧边界的距离

分析：该距离等价于拍照者距马路右侧边界距离与马路宽的总和。问题的难点在于无法确定拍照者的准确位置，因此可将题意转化为：拍照者所站地的马路平行线与马路左侧

边界的垂直距离。因此可将题意转化为：拍照者所站地的马路平行线与马路左侧边界的垂直距离。为此可以建立景深图模型，创新性地将深景图最亮点近似为拍照者正前方的位置。

求解：效果如下图所示，拍照者正前方位置近似坐标为（1497,1959）。

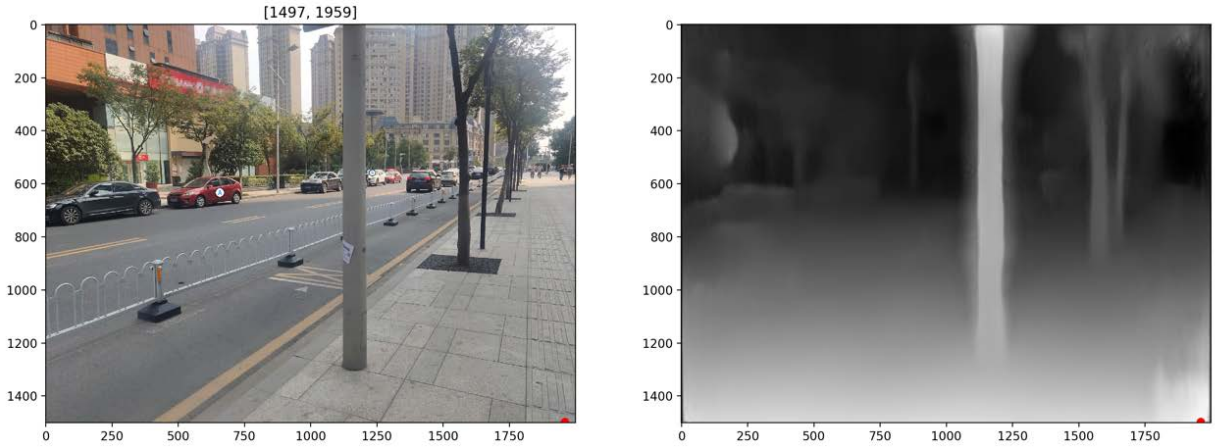

图 18：针对图 1 的深度亮点图

因此：拍照者距离马路右侧边界的垂直距离，等价于计算拍照者正前方位置与马路右侧边界的垂直距离。根据拍照者的位置，通过边缘检测处理和平行查找，最后得到图像中四个畸变坐标，其中右上、右下、左上、左下坐标分别为（1882,954）、（1960,1496）、（1268,905）、（770,1238）。

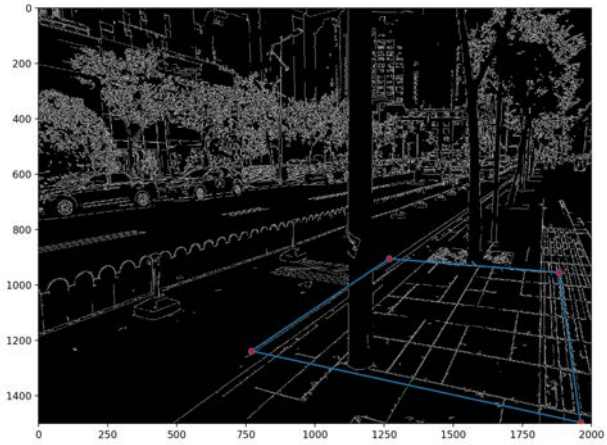

图 19：马路边边缘图

图 20：马路边效果图

通过透视变换模型，我们得到的平行图像如下图所示:

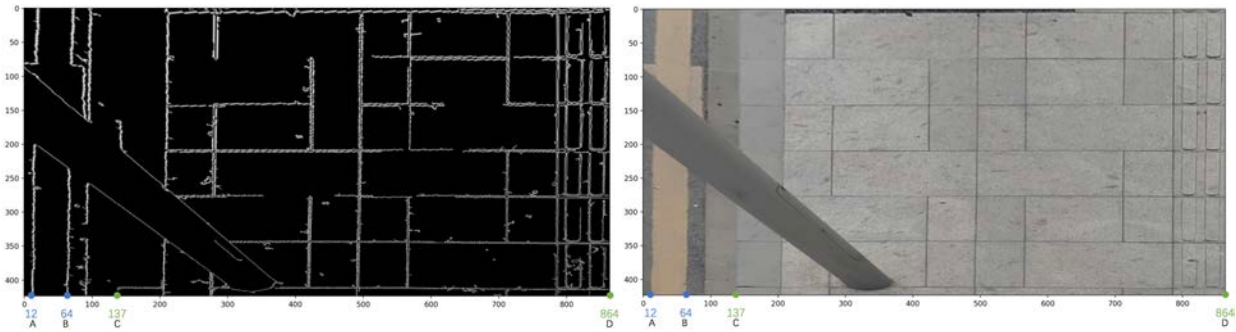

图 21：透视变换图

参照国家标准黄线宽度 0.15 米，即点 A 与点 B 之间相距 0.15 米，由图可得点 A 与点 B 间相距 52 个像素点，点 C 与点 D 之间相距 727 个像素点。通过求解公式（5）和公式（6），可求得马路右侧边缘线点 C 与拍照者正前方位置点 D 相距 2.097 米。

$$l_{CD} = \frac{l_{AB}}{l_{AB\_pixel}} \times l_{CD\_pixel} = \frac{0.15m}{52} \times 727 = 2.097m$$

因为上述计算出拍照者距离马路右侧的距离为 2.097 米，所以可将整个问题聚焦于计算马路宽的长度。采用同样的模型方法，首先马路一半的长度可以通过边缘检测处理和平行查找，最后得到图像中四个畸变坐标，其中右上、右下、左上、左下坐标分别为（1454,865）、（1358,950）、（467,792）、（207,840）。

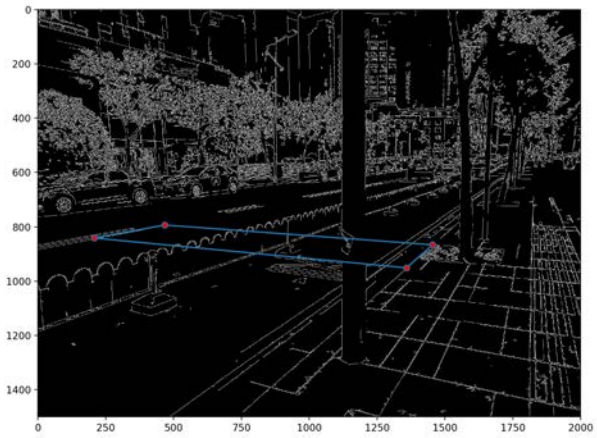

图 22：边缘图

图 23：效果图

通过透视变换模型，我们得到的平行图像如下图所示:

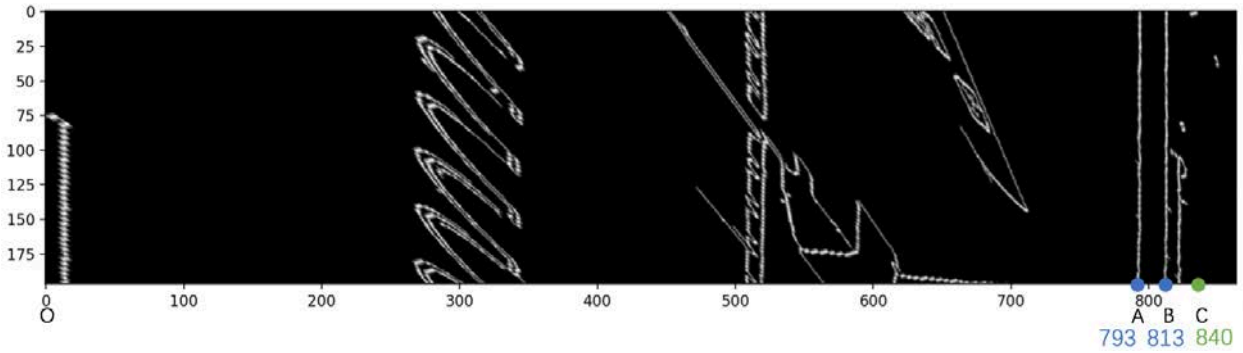

图 24：马路透视边缘图

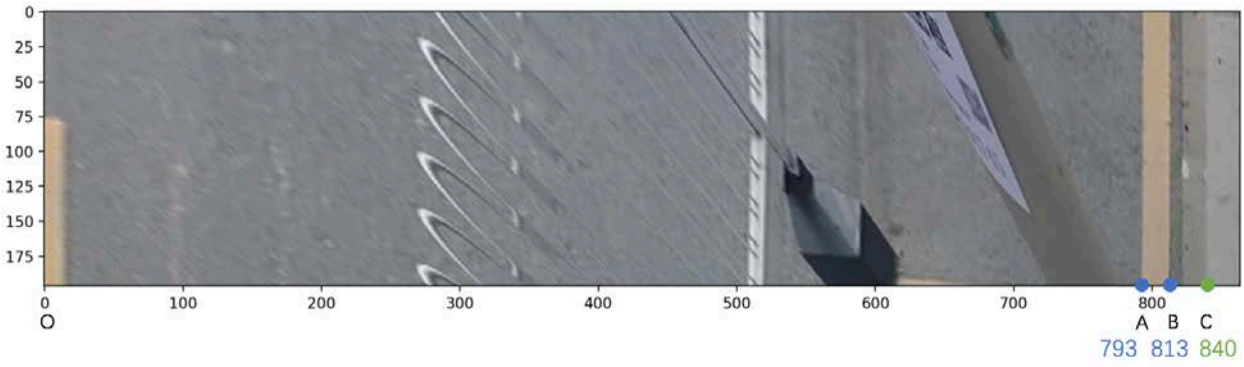

图 25：马路透视效果图

同样参照国家标准黄线宽度 0.15 米，即点 A 与点 B 之间相距 0.15 米，由图可知点 A 与点 B 间相距 20 个像素点，点 O 与点 C 之间相距 840 个像素点。通过求解公式（5）和公式（6），可该马路一半的长度为 6.3 米。

$$l_{OC}=\frac{l_{AB}}{l_{AB\_pixel}}\times l_{OC\_pixel}=\frac{0.15m}{20}\times 840=6.3m$$

因为拍照者距马路左侧边界的距离等价于拍照者距马路右侧边界垂直距离与马路宽之和，马路总宽等于 2 倍的马路一半长。所以，可求得拍照者距马路左侧边界的距离为 14.697 米。

$$l_{all}=l_{CD}+2\times l_{OC}=2.097m+2\times 6.3m=14.697m$$

5.2.2 针对任务一图 2 问题的求解

（1）测算黑色车辆 A 车头和灰色车辆 C 车尾之间的距离

分析：图 2 可知，该道路存在弯曲，因此要对 A 车到 C 车的空间位置截取为三段，化曲为直。第一段为 A 车前轮与 B 车后轮，第二段为 B 车后轮到银车后轮，第三段为银车后轮到 C 车后轮，然后对每一段进行透视变换求解真实长度。

求解：对于该问题同样采用透视变换进行测距，对于第一段其中右上、右下、左上、左下坐标分别为（1494,837）、（1493,1096）、（883,823）、（882,924）。

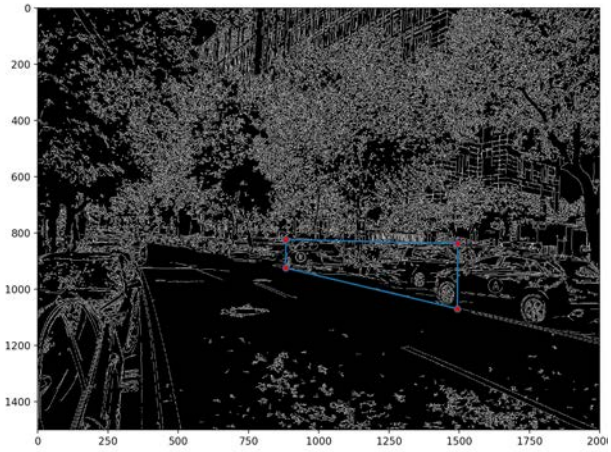

图 26

图 27

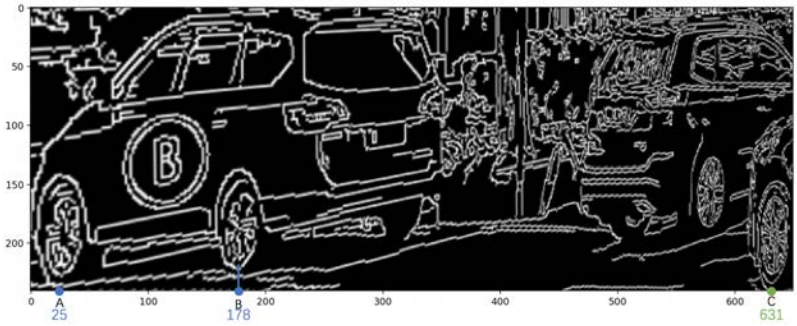

图 28

图 29

已知点 A 与点 B 之间的车轮轴距为 2.64 米，相距 153 个像素点，点 A 与点 B 之间相距 453 个像素点。通过求解公式（5）和公式（6），可求得黑色车 A 前轮与白色 B 车前轮的距离为 7.816 米。

$$l_{BC}=\frac{l_{AB}}{l_{AB\_pixel}}\times l_{BC\_pixel}=\frac{2.64m}{153}\times 453=7.816m$$

同理，对于第二段图 30 可得 B 车后轮到银车后轮距离为 15.048 米。

$$l_{BC}=\frac{l_{AB}}{l_{AB\_pixel}}\times l_{BC\_pixel}=\frac{2.64m}{50}\times 285=15.048m$$

对于第三段图 31 可得银车后轮到 C 车后轮求得的距离为 5.908 米。

$$l_{AC}=\frac{l_{BC}}{l_{BC\_pixel}}\times l_{AC\_pixel}=\frac{2.64m}{21}\times 47=5.908m$$

本题在于测算黑色车辆 A 车头和灰色车辆 C 车尾之间的距离，在三段测量中，我们测量的是 A 车前车轮距离 C 车后车轮的距离为 28.772 米。与题意相比，多算了一个车头和车尾的距离，由于假设标准车长为 4.7 米，轴距为 2.64 米，所以车头与车尾长度之和为 2.06 米。因此，黑色车辆 A 车头和灰色车辆 C 车尾之间的距离为 26.712 米。

$$l_{all}=(7.816+15.048+5.908)-2.06=26.712m$$

图 30

图 31

（2）测算拍照者距白色车辆 B 车头的距离。

分析：对于该问题，首先要求得拍照者距离右侧停车线的垂直距离、拍照者距离 B 车车头的平行方向距离，之后利用勾股定理便可求出答案。

求解：同样采用透视变换进行测距进行马路宽度测距，其中马路一半的右上、右下、左上、左下坐标分别为（1307,1204）、（1562,1315）、（370,1322）、（370,1498）。

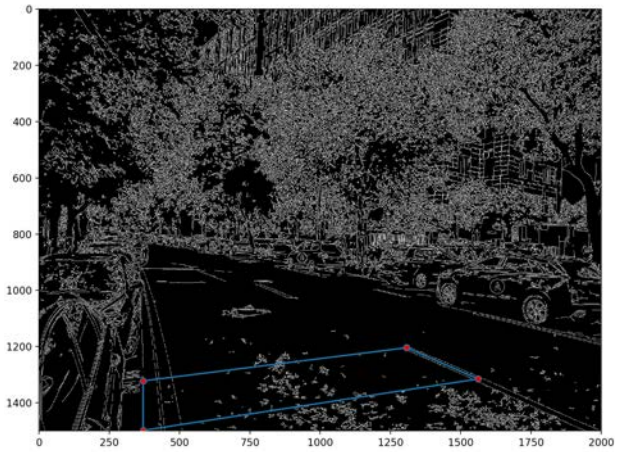

图 32

图 33

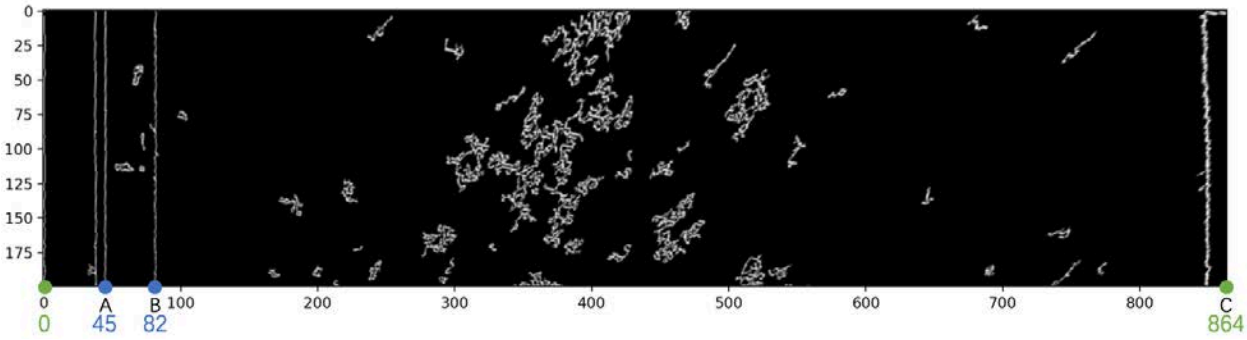

图 34

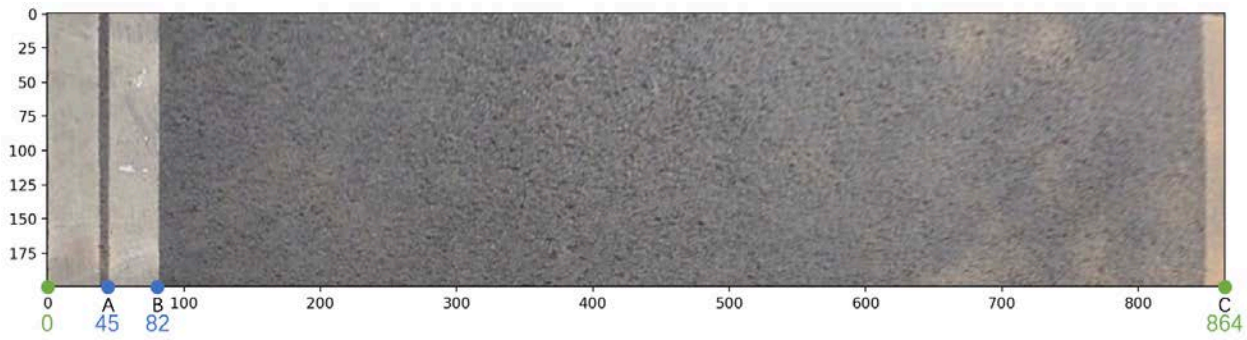

图 35

参照国家标准黄白线宽度 0.15 米，即点 A 与点 B 之间相距 0.15 米，由图可得点 A 与点 B 间相距 37 个像素点，点 C 与点 O 之间相距 864 个像素点。通过求解公式（5）和公式（6），可求得马路一半宽度为 3.503 米，因此马路整个宽度为 7.006 米。

$$l_{OC} = \frac{l_{BC}}{l_{BC\_pixel}} \times l_{OC\_pixel} = \frac{0.15m}{37} \times 864 = 3.503m$$

拍照者距离 B 车车头的水平距离可近似为 A 车后轮与 B 车前轮的距离、B 车车头、A 车车尾的距离之和，所以水平方向距离为 15.145 米，通过勾股定理，可求得拍照者距白色车辆 B 车头的距离为 16.697 米。

$$l_{horizon} = 7.816 + 2.64 \times 2 + 2.06 = 15.156m$$

$$d=\sqrt{15.156^2+7.006^2}=16.697m$$

5.2.3 针对任务一图 3 问题的求解

（1）测算拍照者距离地面的高度

分析：首先通过边缘检测模型，可以取得由点 A、B、C、D 构成的平面 ABCD。然后通过本文**创新的**平面等高转移模型，如图 36，将拍照者距离地面的高度等价于该平面内的 HG 的高度。最后通过透视变换模型，选取自行车高度为参考高度，便可求得拍照者距离地面的真实高度。

求解：通过边缘检测得到畸变平面 ABCD 的右上、右下、左上、左下坐标分别为（1919,286）、（1920,1012）、（1226,357）、（1215,899）。透视变换后的图像如图 37 所示，由于自行车高速为 1.2 米，可求得 HG 的真实高度为 5.624 米，即拍照者距离地面的高度 5.624 米。

$$h_{HG}=\frac{l_{IJ}}{l_{IJ\_pixel}}\times l_{HG\_pixel}=\frac{1.2}{131}\times 614=5.624m$$

图 36

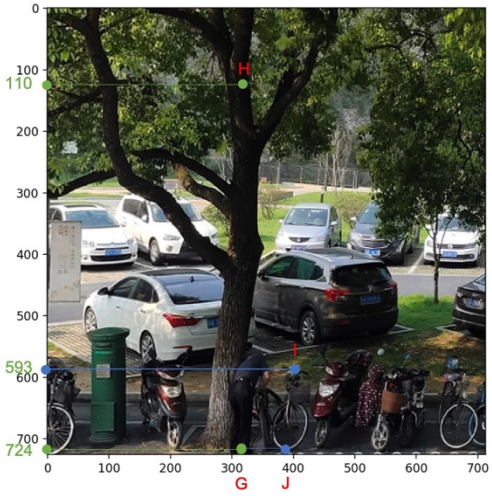

图 37

（2）拍照者距岗亭 A 的距离

分析：由于该马路无任何偏转，因此，拍照者距离岗亭的距离可由两点之间平行距离和垂直距离通过勾股定理求得，其斜边值即为拍照者距离岗亭的真实距离。

求解：同样采用边缘检测和透视变换模型，参照黄白线宽度为 0.15 米，可求得整条马路的宽度为 15.576 米。

$$l_1=\frac{0.15}{21}\times 536=3.828m\ l_2=\frac{0.15}{10}\times 264=3.96m\ l_{all}=2\times(l_1+l_2)=15.576m$$

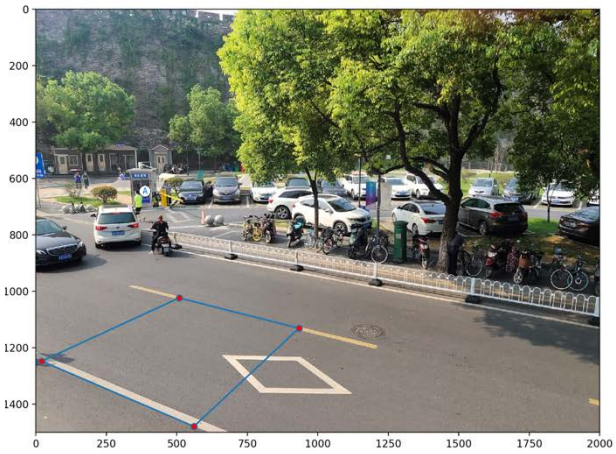

图 38

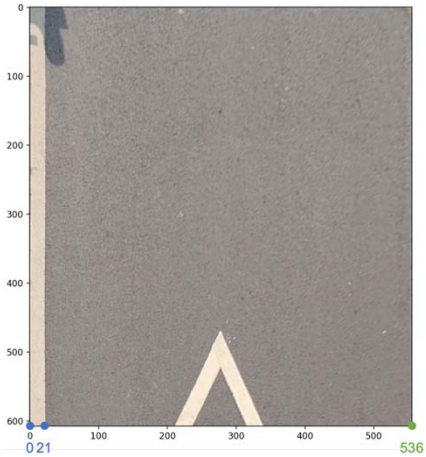

图 39

图 40

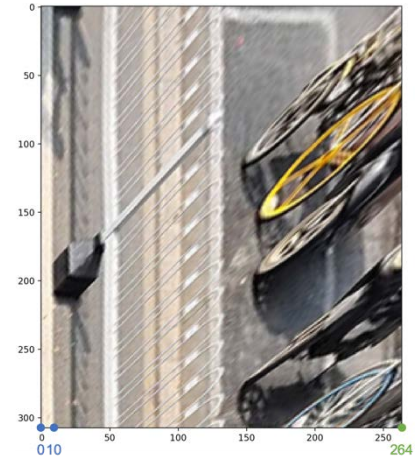

图 41

同理，对于平行于马路的方向，采用边缘检测和透视变换模型，首先测得单节栏栅的长度为 2.941 米，假设平行马路方向，拍照者与亭 A 相距 8 节栏栅，因此，可得平行马路的方向相距 23.528 米。

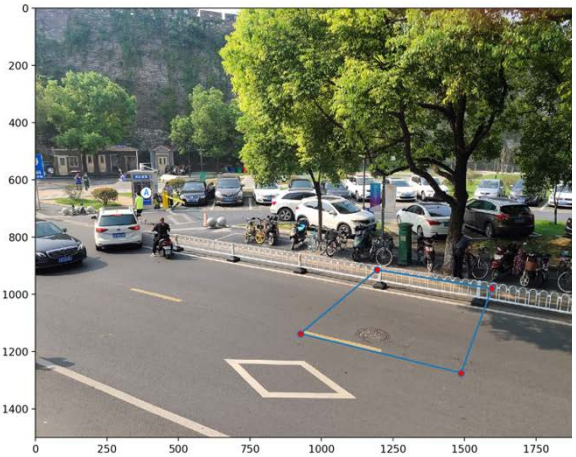

图 42

图 43

如下图 44 所示，通过勾股定理可求得亭 A 与拍照者映射到地面 C 点的距离为 28.217 米。由于拍照者 B 与地面 C 的垂直距离已求得为 5.624 米，因此再次通过勾股定理可求得拍照者与亭 A 的真实距离为 28.772 米。

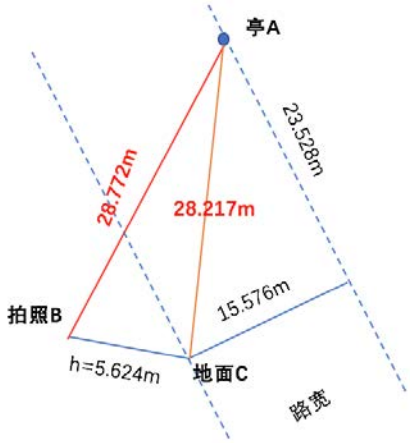

图 44：亭 A 与拍照者 B 之间的位置关系

5.2.4 针对任务一图 4 问题的求解

（1）测算塔底 AB 和塔顶 CD 长度

分析：由于该图存在一定的空间距离，地砖与塔底并不在同一水平面，因此直接进行透视变换误差较大。我们的解题思路为：采用**传递测量法化立体为平面**，即通过地砖长度求台阶长度，通过台阶长度求单个塔前小砖长度，通过塔前砖长求塔底 AB 长度，通过塔底 AB 长度求塔顶 CD 长度。

求解：对于该问题同样采用透视变换进行测距，对于地砖长度求台阶长度，得到的右上、右下、左上、左下坐标分别为（1095,1543）、（1367,1731）、（443,1569）、（280,1797）。

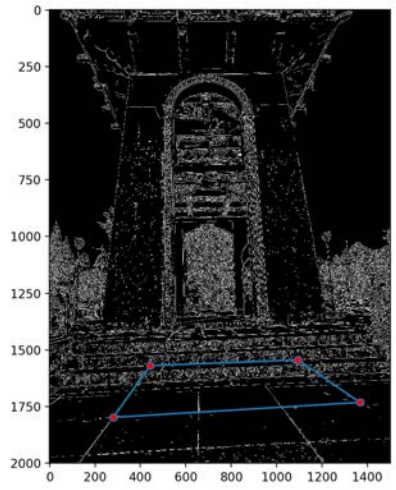

图 45

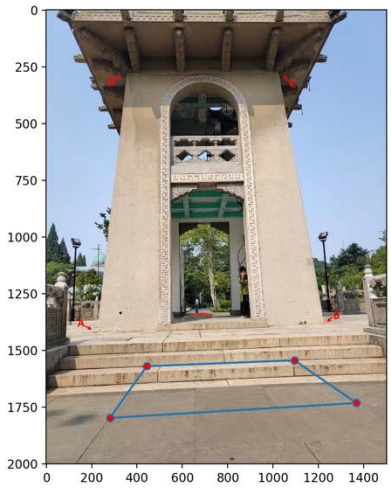

图 46

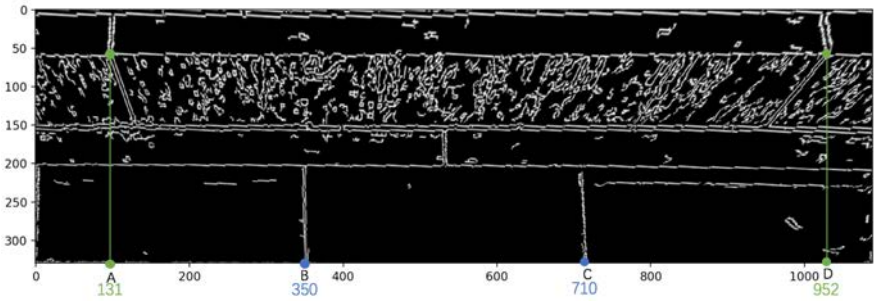

图 47

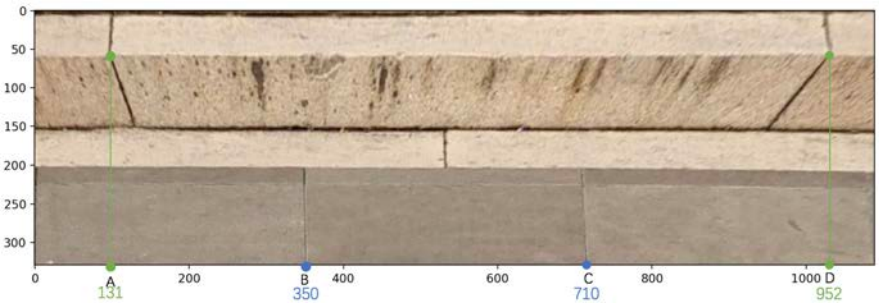

图 48

参照已给的地砖宽度为 0.8 米，即点 B 与点 C 之间相距 0.8 米，由图可得点 B 与点 C 间相距 360 个像素点，台阶端点 A 与点 D 之间相距 821 个像素点。通过求解公式（5）和公式（6），可求得一个台阶长度为 1.824 米。

$$l_{step} = l_{AD} = \frac{l_{BC}}{l_{BC\_pixel}} \times l_{AD\_pixel} = \frac{0.8m}{360} \times 821 = 1.824m$$

同理可得：根据图 51，参照上述已得的台阶长度 1.824 米，可求得塔前小砖的宽度为 0.601 米。

$$l_{brick} = l_{AC} = \frac{l_{BC}}{l_{BC\_pixel}} \times l_{AC\_pixel} = \frac{1.824m}{510} \times 168 = 0.601m$$

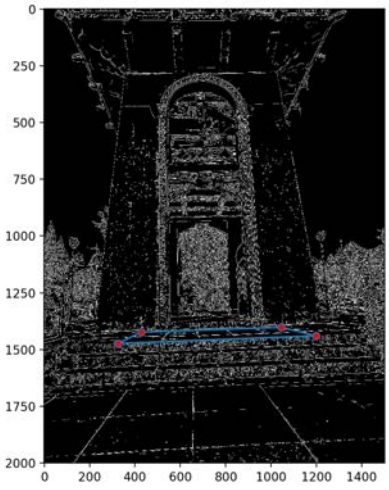

图 49

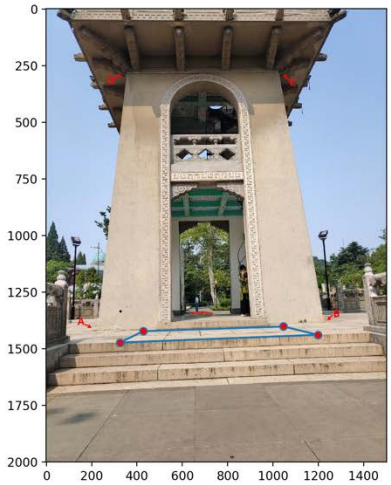

图 50

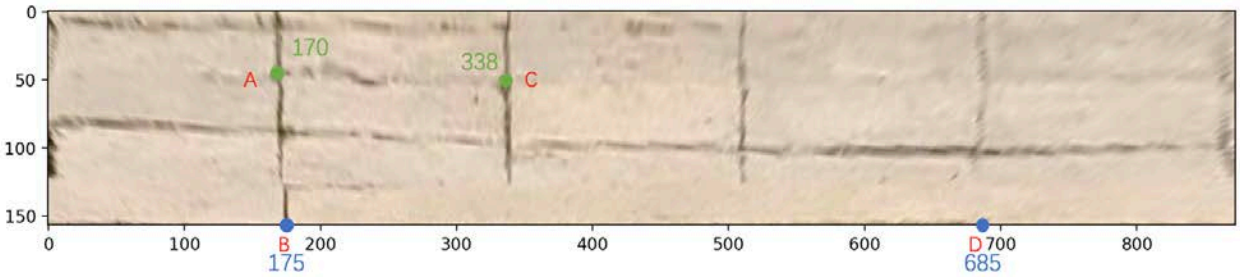

图 51

同理可得：根据图 54，参照上述已得的塔前小砖长度 0.601 米，可求得塔底 AB 长度为 5.38 米。

$$AB_{tower} = l_{AD} = \frac{l_{BC}}{l_{BC\_pixel}} \times l_{AD\_pixel} = \frac{0.601m}{125} \times 1119 = 5.38m$$

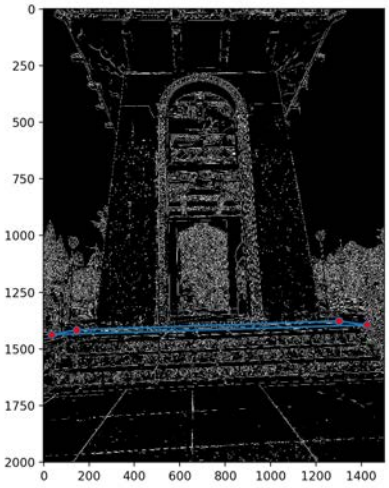

图 52

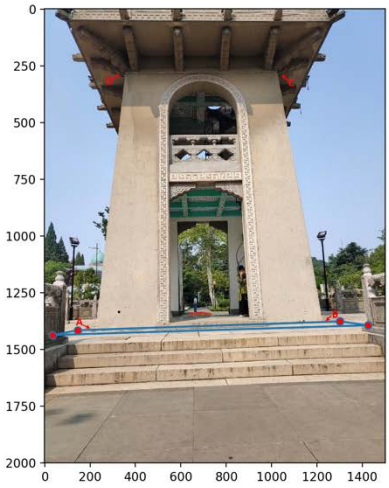

图 53

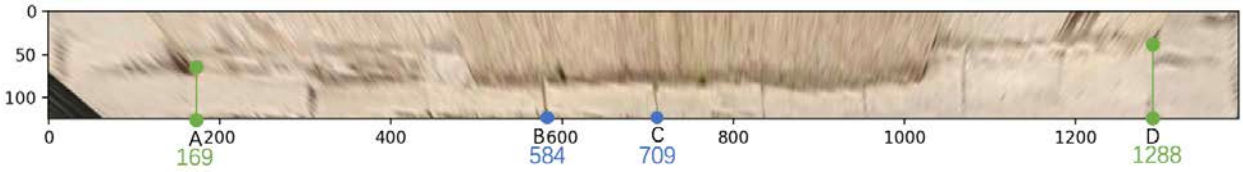

图 54

同理可得：根据图 57，参照上述已得的塔底 AB 长度为 5.38 米，求得塔顶 CD 长度为 4.308 米。

$$CD_{tower} = l_{BC} = \frac{l_{AD}}{l_{AD\_pixel}} \times l_{BC\_pixel} = \frac{5.38m}{1019} \times 816 = 4.308m$$

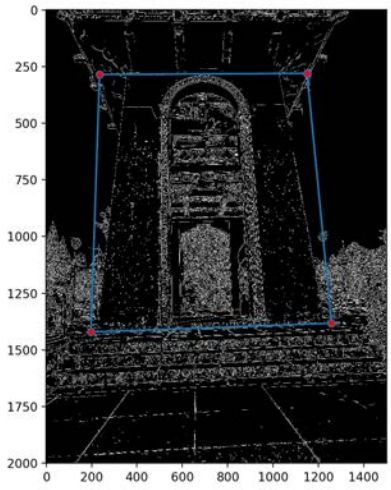

图 55

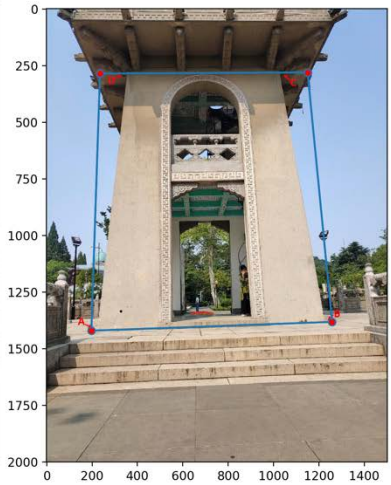

图 56

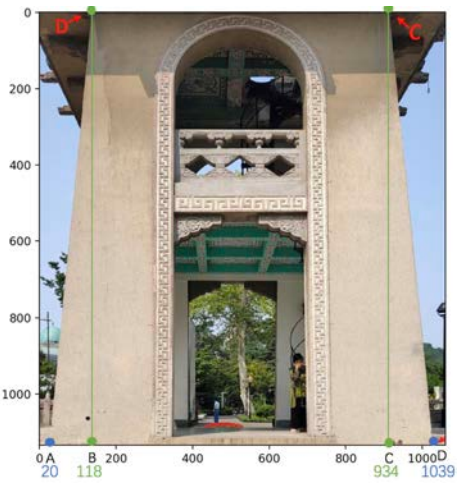

图 57

（2）测算塔底 AB 和塔顶 CD 之间的高度

分析：假设四边形 ABCD 为等腰梯形，通过透视变换可求等腰梯形的下底角，因此通过简单的数学方法便可求得塔底与塔顶之间的高度。

求解：通过上述求解可得塔上底为 4.308m，下底为 5.380m，过 D 点向 AB 边做高 DE，可抽象出如下图 58，由透视变换后的图 57 可测出，$\angle\alpha = \angle DAE = 85.7^{\circ}$，因此可求得塔底 AB 和塔顶 CD 之间的高度为 7.128 米。

$$h = \frac{(AB - CD)}{2} \times \tan 85.7^{\circ} = 7.128m$$

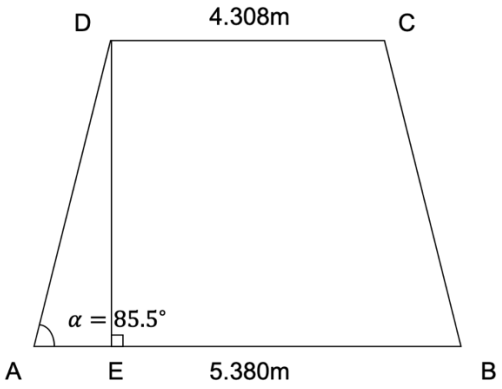

图 58

5.3 结果总结

针对任务一中的图 1-图 4 的各个问题，采用了透视变换模型、Canny 边缘检测模型、深景图测距模型、平面等高转移模型，求得的所有结果如下表 1 所示：

表 1. 任务一求解结果

图片编号	结果
图片 1	（1）红色 A 车车头与白色 B 车车头的距离为 21.826 米。 （2）拍照者距马路左侧边界的距离为：14.697 米。
图片 2	（1）黑色车辆 A 车头和灰色车辆 C 车尾之间的距离为 26.712 米。 （2）可求得拍照者距白色车辆 B 车头的距离为 16.697 米。
图片 3	（1）拍照者距离地面的高度 5.624 米。 （2）拍照者与亭 A 的真实距离为 28.772 米。
图片 4	（1）塔底 AB 长度为 5.38 米，求得塔顶 CD 长度为 4.308 米。 （2）塔底 AB 和塔顶 CD 之间的高度为 7.128 米。

六、任务二求解

6.1 模型求解

（1）测算视频中该车和后方红色车辆之间的距离

分析：通过视频分析，获取红车和白车并排行驶时的特殊时刻视频帧。在红车与该车之间的距离保持不变的假设前提下，此时在道路径向方向上，该车与红车的距离应等价于该车与白车水平方向的距离。

求解：通过视频分析，在视频第 361 帧时，如图 59 和图 60，红车和白车并排行驶。因此，通过先行求解下一问“该车超越第一辆白色车辆问题”可得相对速度差为 5.64m/s，所以该车与红车间的距离为 44.18 米。

$$l = v \times t = 5.64_{m/s} \times \frac{361-128}{30} = 44.18m$$

证明：白车和红车并排，即图 61 中 ABCD 点四点共线：A(120.4,233.3), B(148.9,235.0), C(193.4,237.4), D(223.1,239.3)。通过 AB 点的坐标求解得方程如下所示，将点 C、点 D 的 x 坐标值带入方程，可求解得其相应 y 值，在误差允许的范围内，证得 ABCD 四点共线。

$$f_{AB}: 1.7x - 28.5y + 6444.37 = 0$$

$$x_C = 193.4 \quad y_C = 237.6 \quad x_D = 223.1 \quad y_D = 239.4$$

图 59

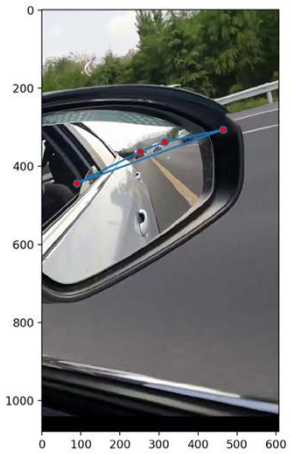

图 60

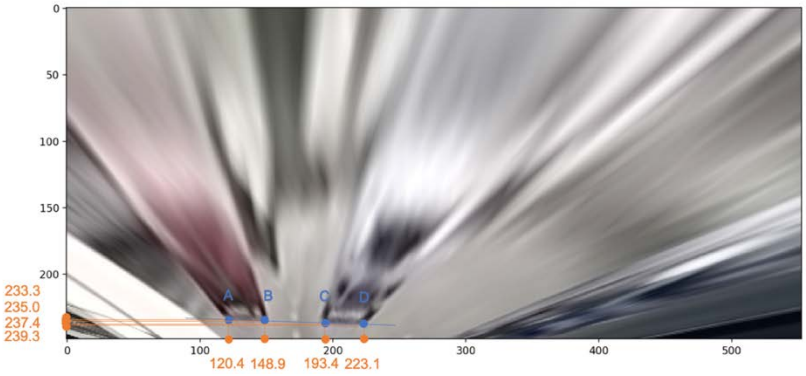

图 61

（2）测算视频中该车超越第一辆白色车辆时两车的速度差异

分析：首先对视频进行帧处理，帧率已知 30。针对该问题提取出 2 个关键的视频帧：白车即将出现和白车即将消失的视频帧，通过帧率获得时间。又因为车长已知，所以可求解得相对速度的值。

求解：总帧数为 395 帧，帧率为 30 帧/秒。提取出关键的视频帧：第 101 帧与第 126 帧，分别是白车即将出现和白车即将消失的视频帧，历时 25 帧，时长为 0.833 秒。

$$t = \frac{n_{frame}}{speed_{frame}} = \frac{25}{30} = 0.833$$

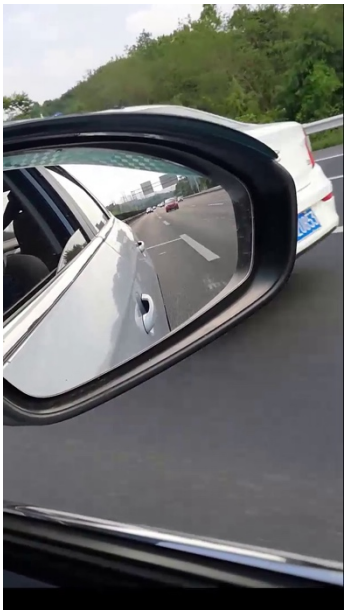

图 62

图 63

假设本文中所有的汽车车身长为 4.7 米，则超速时的相对速度差为 5.64m/s，即 20.3km/h。

$$v = \frac{l}{t} = \frac{4.7m}{0.833s} = 5.64(m/s)$$

6.2 结果总结

针对任务二中的问题，采用了透视变换模型、Canny 边缘检测模型，求得的所有结果如下表 2 所示：

表 2. 任务二求解结果

问题编号	结果
问题 1	该车与后方红车车辆距离为 44.18 米。
问题 2	视频中该车超越第一辆白色车辆时两车速度差异 5.64m/s，即 20.3km/h。

七、任务三求解

7.1 模型介绍

针对任务三的问题，建立光流测速模型。光流是用于描述连续两帧图像中物体的显著运动模式，它通常通过一个 2D 的向量来表示图像中的点从前一帧的某个位置移动到下一帧的某个位置。

假如在前一帧中有一个像素 $I(x,y,t)$，它在下一帧中经历时间 dt，在 x，y 方向上分别移动了 dx,dy，因为前后帧的两像素是同一像素，因此可以得到两像素的关系为：

$$I(x,y,t)=I(x+dx,y+dy,t+dt) \tag{10}$$

将式 10 右式进行泰勒展开，可以得到光流方程：

$$f_x u+f_y v+f_t=0 \tag{11}$$

其中，$f_x=\frac{\partial f}{\partial x}$；$f_y=\frac{\partial f}{\partial y}$；$u=\frac{dx}{dt}$；$v=\frac{dy}{dt}$，$f_x$ 和 f_y 是图像的梯度，f_t 是时间的梯度，但是 u 和 v 在这里是未知量，本题我们通过 Lucas-Kanade 方法利用最小二乘法反解出 u，v。

$$\begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}\sum_i f_{x_i}^2 & \sum_i f_{x_i}f_{y_i}\\ \sum_i f_{x_i}f_{y_i} & \sum_i f_{y_i}^2\end{bmatrix}^{-1}\begin{bmatrix}-\sum_i f_{x_i}f_{t_i}\\ -\sum_i f_{x_i}f_{t_i}\end{bmatrix} \tag{12}$$

从而在视频帧中实时绘制出了光流图像。由于一张帧中的像素过多，在本题中人工选定关键帧和关键点，标注关键点的目标物体需要和镜头画面所在平面平行，接着对前后两帧进行光流追踪，得到关键点在 x 方向上的位移 dx 个像素，假设目标物体的长度已知为 L，像素个数为 N，视频帧率为 F，由此可求得镜头的运动速度 v 为：

$$v=\frac{L*dx}{N*F} \tag{13}$$

7.2 模型求解

7.2.1 测算高铁行驶方向左侧第一座桥桥面距水面的高度

分析：选取两个特殊的连续视频帧，保证桥墩所在平面与镜头所在平面平行。取桥墩左边通过光流检测，当知道高铁速度时，便可求得桥墩的高，即桥面距水面的高度。

求解：取第 245、246 帧作为关键帧，取桥墩左边坐标 A（14,654）进行光流检测，可知两帧之间，运动的距离为 10 像素。通过透视变换，由坐标 C（4,654）、D（4,696）可知桥墩的高度为 42 像素，如图 65 所示。

图 64

图 65

通过先行求解下一问“高铁行驶的速度”可得高铁速度为 304.56km/h=84.6m/s，所以两帧之间 A 点运行的实际距离为 84.6/30=2.82m，因此，该坐标系下，1 像素的尺度为 2.82/10=0.282m/pix，因此桥墩的宽 BC 为 11.844m，即高度约为 11.844 米。

$$h = l_{BC} = \frac{l_{AB}}{l_{AB\_pixel}} \times l_{BC\_pixel} = \frac{2.82}{10} \times (696 - 654) = 11.844m$$

7.2.2 测算桥距高铁轨道的距离以及水面宽度

（1）测算桥距高铁轨道的距离

分析：提前求解下问得高铁速度为 304.56km/h，选取两个特殊视频帧：水面刚出现在视频中左下角的帧和桥端完全消失在视频中的帧。根据帧数可得时间，于是便知道了水面的斜边，通过勾股定理可求解的桥距离高铁轨道的距离。

求解：如图 66，设大桥两端为 A，B，沿着河左岸做直线 AC，由大桥与河岸的几何关系知，角 BAC=90º。

如图 67 和图 68，取左侧河岸完全消失的关键帧第 157 帧，取 B 点完全消失的关键帧第 272 帧，由于帧率为 30 帧/秒，高铁的速度为 84.6 米/秒，可求得水面斜边的值为 327.12 米。提前求解下问可知水面宽度为 219.96 米，通过勾股定理，求得桥距离高铁轨道 242.13 米。

$$l = v \times t = 84.6_{m/s} \times \frac{272-156}{30}_{s} = 327.12m$$

$$d = \sqrt{327.12^2 - 219.96^2} = 242.13m$$

图 66

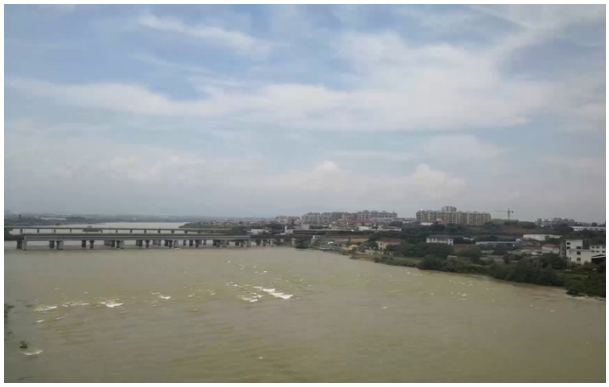

图 67

图 68

（2）测算水面宽度

分析：选取两个特殊的视频帧，左下角点刚好到达水面左边界的视频帧和左下角的点刚好到达水面右边界的视频帧，即运动中恰好行驶了水面的整个宽度。当知道高铁速度时，便可求得水面宽度。

求解：取第 157 帧和 235 帧，如图 69 和图 70。通过先行求解下一问“高铁行驶的速度”可得高铁速度为 304.56km/h=84.6m/s，所以便求得整个水面宽度为 219.96m。

$$l_{width} = v \times t = 84.6_{m/s} \times \frac{235-157}{30}_{s} = 219.96m$$

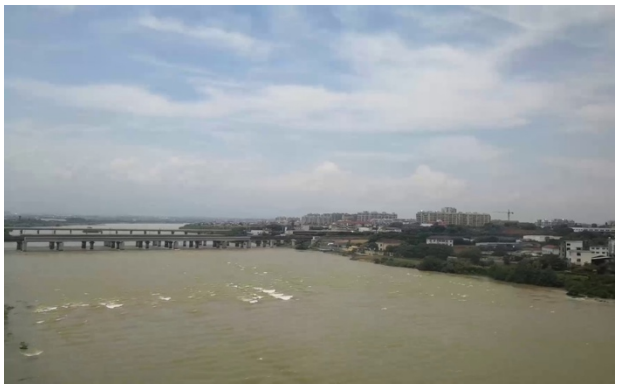

图 69

图 70

7.2.3 测算拍摄时高铁的行驶速度

分析：帧率为 30 帧/秒，光流测速模型以连续两帧作为输入，根据两帧中同一参照点的相对位移求得高铁的移动速度。

求解：选取视频第 30、31 连续两帧图像输入，由于本文假设所有汽车长度为 4.7m, 观察可知 N 车与高铁相对平行，所以以 N 车车头点为光流测速对象，用其相对速度等效高铁的速度较为准确。

如图 77 所示，汽车 N 的相对像素位移为 33，所以高铁相对位移 2.82 米，又因为两帧之间间隔时间为 1/30 秒，所以可求得高铁的速度为 84.6m/s，即 304.56km/h。

$$l_{railway} = \frac{l_{AC}}{l_{AC\_pixel}} \times l_{AB\_pixel} = \frac{4.7}{55} \times 33 = 2.82m$$

$$v = \frac{l_{railway}}{t} = \frac{2.82m}{1/30s} = 84.6(m/s) = 304.56(km/h)$$

图 71

7.3 结果总结

针对任务三中的问题，采用了光流测速模型、透视变换模型、Canny 边缘检测模型，求得的所有结果如下表 3 所示：

表 3. 任务三的求解结果

问题编号	结果
问题 1	高铁行驶方向左侧第一座桥桥面距水面的高度为 11.844 米。
问题 2	（1）桥距高铁轨道的距离为 242.13 米。 （2）水面宽度为 219.96m。
问题 3	高铁的速度为 84.6m/s，即 304.56km/h。

八、任务四求解

8.1 模型求解

8.1.1 测算环绕老宅道路的长度、宽度、各建筑物的高度、后花园中树木的最大高度

分析：对于长度的测量，在已知拍摄速度的前提下，可采用光流测速模型，通过两个特殊连续视频帧的输入，选取相应参照物，便可求得图像中任意两点的距离。

（1）测算环绕老宅道路的长度、宽度

求解：通过先行求解下问中“无人机飞行速度”可得飞行速度为 14.535m/s。对第 1921 至 1941 帧这 20 帧图像中红框内的人进行光流测速，结果如图 73 所示。

图 72 将四条路命名为甲乙丙丁，在图 73 中，BA 向量表示该人像素位移的长度，点 C 表示甲路长边左侧（乙路宽边左侧），点 D 表示乙路宽边右侧，点 E 表示甲路长边中间位置。其坐标分表为：A(771.5,742)、B(812.5,740.6)、C(296.7,764)、D(366.3,764)、E(1034.8,764)，所以 AB 间相隔 41 个像素值，CD 间相隔 69.6 个像素值，CE 间隔 738.1 个像素值。需要注意的是视频中该人的步行方向与无人机相反，假设人的步行速度一般为 1.275m/s，所以无人机相对于人的移动速度为 15.81m/s，因此可求得 AB 实际距离为 10.54 米。

假设甲乙丙道路同长，甲乙丙丁道路同宽，丁道路为标准半圆弧，所以可求解得甲乙丙路长为 379.491 米，丁路长为 596.103 米，路宽为 17.892 米。

$$l_{AB} = v \times t = (14.535 + 1.275) \times \frac{20}{30} = 10.54m$$

$$l_1 = \frac{l_{AB}}{l_{AB\_pixel}} \times l_{CE\_pixel} \times 2 = \frac{10.54m}{41} \times 738.1 \times 2 = 379.491m$$

$$l_2 = \frac{l_1}{2} \times PI = 596.103m$$

$$w = \frac{l_{AB}}{l_{AB\_pixel}} \times l_{CD\_pixel} = \frac{10.54m}{41} \times 69.6 = 17.89m$$

图 72：第 1914 帧

图 73：第 1914 帧的光流图像

（2）测算老宅各建筑高度和后花园最高树的高

分析：如图 74，当无人机运动到如图关键帧位置时，蓝色框内的二层房子坐在平面与

视频画面平行，对其进行透视变换，得到楼房的平面图，该楼房的门脚与门楣分别经过点 A，点 B，屋顶经过点 C。

图 74

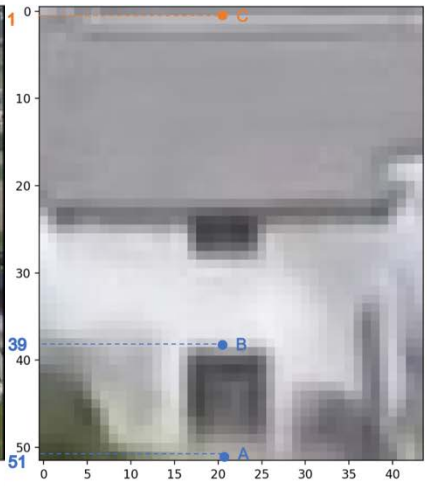

图 75

求解：假设门高 2 米，可得二层楼房的高度为 8.333 米。为简化计算，取一层楼的高度为 4.167 米。

$$AC = \frac{l_{AC\_pixed} \times l_{AB}}{l_{AB\_pixed}} = \frac{(51-1)\times 2}{51-39} = 8.333m$$

$$h = \frac{AC}{2} = 4.167m$$

同理可得，老宅各建筑高度如表 4 所示：

图 76

图 76 中用红色圆圈标注的树为后花园中最高的树木，根据老宅楼房可估算大约有 4 层楼的高度为 16.668 米。

$$h_{tree} = 4 \times h = 16.668m$$

表 4. 老宅各个楼房的层数与高度

房屋图片	房屋楼层	估算高度
	3	12.501 米
	2	8.333 米
	2	8.333 米
	2	8.333 米
	1	4.167 米

8.1.2 测算该老宅的占地面积

分析：由于四合院的外墙道路长度已知，所以老宅的面积等价于一个正方形面积加上一个半圆面积，该半圆直径为整个四合院外围道路长度。

求解：通过上述求解可得路长为 379.391 米，半圆的半径等价于该路长的一半，所以面积可求得为 200567.357 ㎡

$$S = l_1^2 + \frac{l_1^2}{8} = 200567.357 m^2$$

8.1.3 测算无人机的飞行高度和速度

（1）无人机飞行高度

分析：该题计算的属于高度，通过上文提到的平面等高转移模型可进行求解，其中参照物为双层楼房的高度。

求解：如图 77 取第 1470 帧，道路与画面下边缘平行的时候，进行透视变换，结果如图 78 所示。在图 78 中，AB 为甲路长度，即 379.494m，所 AC=688/541*AB=482.610m。建立相似三角形：O 为 AB 中点，CO=AC-AB/2=292.863，以 DE 处双层楼房的高度 10m 作为参考，OD 可求得为 1/4 的 AB=379.491m/4=94.873m，所以可得无人机高度 FC=DE*CO/DO=30.869m。

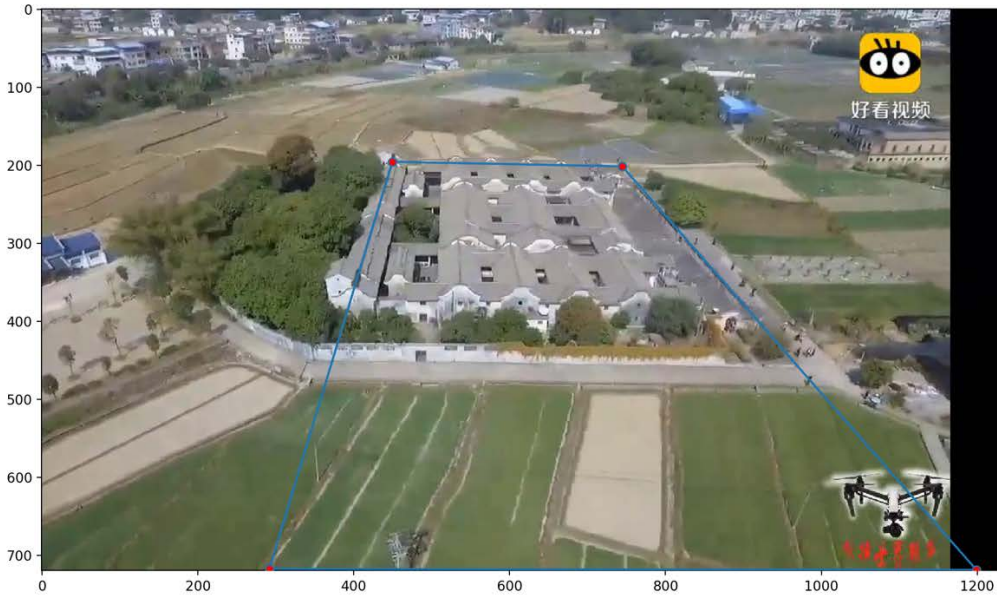

图 77

图 78

图 79

（2）无人机飞行速度

分析：对于无人机飞行速度的测量，可采用光流测速模型，效果如图 80，通过两个特殊连续视频帧的输入，选取相应参照物，便可求得飞行速度。

求解：如图 81 和图 82，取第 471、479 帧图像中的大巴车头左顶点为光流测速对象，其特殊于在该帧中车的车宽可估计，且与无人机相对平行，所以用其相对速度等效无人机的速度较为准确。

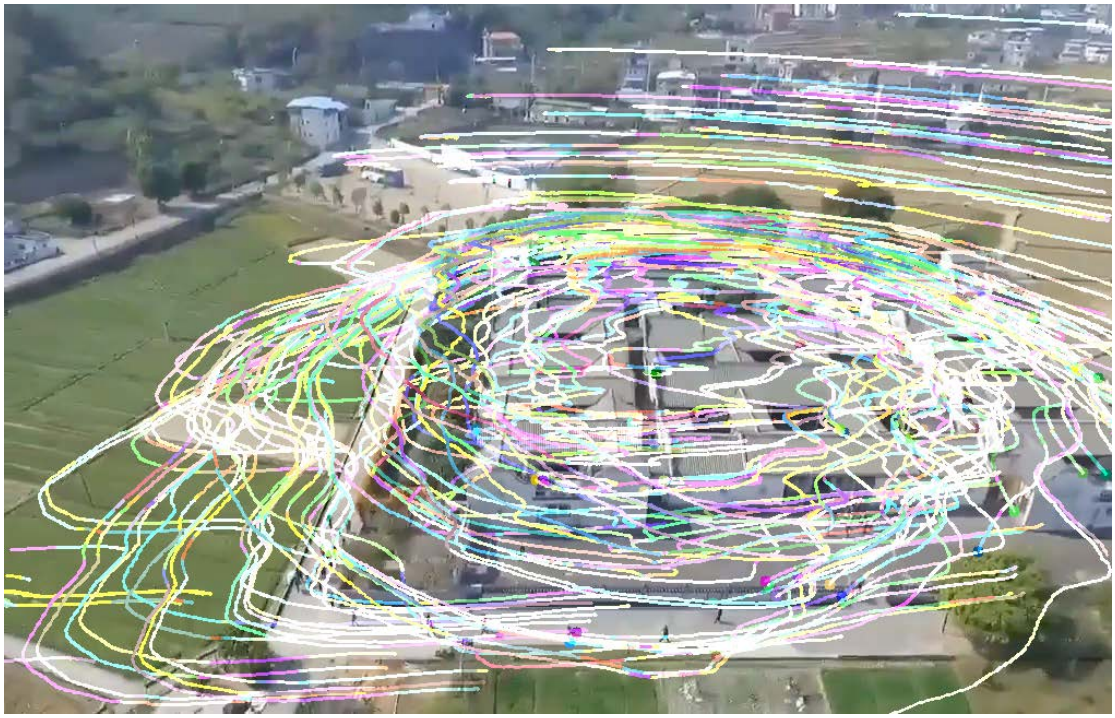

图 80

图 81: 第 471 帧图

图 82: 第 479 帧图

图 83: 第 479 帧光流图

图 84: 第 479 帧光流图局部版

如图 X 所示，大巴车 AB 的相对像素位移为 837.5-820.6=16.9，大巴车 AB 的像素宽度为 845.5-834.6=10.9，假设大巴车真实宽度为 2.5 米，所以大巴的实际相对位移为 3.876 米，即无人机的实际位移为 3.8976 米。由于帧率为 30，所以无人机运动时间为 8/30 秒，所以可求得无人机的飞行速度为 14.535m/s，即 52.326km/h。

$$l_{\Delta} = \frac{l_{AB}}{l_{AB\_pixel}} \times l_{\Delta\_pixel} = \frac{2.5m}{10.9} \times 16.9 = 3.876m$$

$$v = \frac{l_{\Delta}}{t_{\Delta}} = \frac{3.876m}{8/30s} = 14.535(m/s)$$

8.2 结果总结

针对任务四中的问题，采用了光流测速模型、透视转换模型、平面等高转换模型，求得的所有结果如下表 5 所示：

表 5. 任务四的求解结果

问题编号	结果
问题 1	（1） 测得直线路长 379.491 米，弧形路长 596.103 米，路宽 17.892 米。 （2） 老宅各个建筑高度参照表，老宅最高的树为 16.668 米。
问题 2	老宅的占地面积为 200567.357 m^2。
问题 3	（1）无人机的飞行高度为 30.869 米。 （2）无人机的飞行速度为 14.535m/s，即 52.326km/h。

参 考 文 献

[1]李江晨，徐小维，韩君佩，等.基于同步脉冲光源的抗环境红外 FTIR 多点触摸算法[J]. 红外与激光工程，2013.(06):1415-1419.
[2] 黄湛, 张淼, 程攀,等. 基于光流算法的粒子图像测速技术研究和验证[J]. 实验力学, 2016, 31(5):673-682.
[3] 来佳伟，何玉青，李霄鹏 等：基于单目视觉的机械臂目标定位系统设计[J]，《光学技术》，2019.01
[4]刘军, 后士浩, 张凯,晏晓娟: 基于单目视觉车辆姿态角估计和逆透视变换的车距测量[J], 《农业工程学报》, Jul. 2018(pp70-76)
[5]刘学军，王美珍，甄艳等：单幅图像几何量测研究进展[J]，《武汉大学学报》（信息科学版）,36(8) : pp941 - 947.

附　　件

1、透视变换模型代码

```
# import the necessary packages
from scipy.spatial import distance as dist
import numpy as np
import cv2

def order_points(pts):
    # sort the points based on their x-coordinates
    xSorted = pts[np.argsort(pts[:, 0]), :]

    # grab the left-most and right-most points from the sorted
    # x-roodinate points
    leftMost = xSorted[:2, :]
    rightMost = xSorted[2:, :]

    # now, sort the left-most coordinates according to their
    # y-coordinates so we can grab the top-left and bottom-left
    # points, respectively
    leftMost = leftMost[np.argsort(leftMost[:, 1]), :]
    (tl, bl) = leftMost

    # now that we have the top-left coordinate, use it as an
    # anchor to calculate the Euclidean distance between the
    # top-left and right-most points; by the Pythagorean
    # theorem, the point with the largest distance will be
    # our bottom-right point
    D = dist.cdist(tl[np.newaxis], rightMost, "euclidean")[0]
    (br, tr) = rightMost[np.argsort(D)[::-1], :]

    # return the coordinates in top-left, top-right,
    # bottom-right, and bottom-left order
    return np.array([tl, tr, br, bl], dtype="float32")

def four_point_transform(image, pts):
    # obtain a consistent order of the points and unpack them
    # individually
    # rect = order_points(pts)
    rect = np.array([pts[3], pts[0], pts[1], pts[2]], dtype=np.float32)
    (tl, tr, br, bl) = rect
```

```
    # compute the width of the new image, which will be the
    # maximum distance between bottom-right and bottom-left
    # x-coordiates or the top-right and top-left x-coordinates
    widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
    widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
    maxWidth = max(int(widthA), int(widthB))

    # compute the height of the new image, which will be the
    # maximum distance between the top-right and bottom-right
    # y-coordinates or the top-left and bottom-left y-coordinates
    heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
    heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
    maxHeight = max(int(heightA), int(heightB))

    # now that we have the dimensions of the new image, construct
    # the set of destination points to obtain a "birds eye view",
    # (i.e. top-down view) of the image, again specifying points
    # in the top-left, top-right, bottom-right, and bottom-left
    # order
    dst = np.array([
        [0, 0],
        [maxWidth - 1, 0],
        [maxWidth - 1, maxHeight - 1],
        [0, maxHeight - 1]], dtype="float32")

    # compute the perspective transform matrix and then apply it
    M = cv2.getPerspectiveTransform(rect, dst)
    warped = cv2.warpPerspective(image, M, (maxWidth, maxHeight))

    # return the warped image
    return warped

# USAGE:
# python scan.py (--images <IMG_DIR> | --image <IMG_PATH>) [-i]
# For example, to scan a single image with interactive mode:
# python scan.py --image sample_images/desk.JPG -i
# To scan all images in a directory automatically:
# python scan.py --images sample_images

# Scanned images will be output to directory named 'output'

from pyimagesearch import transform
from pyimagesearch import imutils
from scipy.spatial import distance as dist
```

```
from matplotlib.patches import Polygon
import polygon_interacter as poly_i
import numpy as np
import matplotlib.pyplot as plt
import itertools
import math
import cv2

import argparse
import os

class DocScanner(object):
    """An image scanner"""

    def __init__(self, interactive=False, MIN_QUAD_AREA_RATIO=0.25, MAX_QUAD_ANGLE_RANGE=
40):
        """
        Args:
            interactive (boolean): If True, user can adjust screen contour before
                transformation occurs in interactive pyplot window.
            MIN_QUAD_AREA_RATIO (float): A contour will be rejected if its corners
                do not form a quadrilateral that covers at least MIN_QUAD_AREA_RATIO
                of the original image. Defaults to 0.25.
            MAX_QUAD_ANGLE_RANGE (int):  A contour will also be rejected if the range
                of its interior angles exceeds MAX_QUAD_ANGLE_RANGE. Defaults to 40.
        """
        self.interactive = interactive
        self.MIN_QUAD_AREA_RATIO = MIN_QUAD_AREA_RATIO
        self.MAX_QUAD_ANGLE_RANGE = MAX_QUAD_ANGLE_RANGE

    def filter_corners(self, corners, min_dist=20):
        """Filters corners that are within min_dist of others"""
        def predicate(representatives, corner):
            return all(dist.euclidean(representative, corner) >= min_dist
                    for representative in representatives)

        filtered_corners = []
        for c in corners:
            if predicate(filtered_corners, c):
                filtered_corners.append(c)
        return filtered_corners

    def angle_between_vectors_degrees(self, u, v):
        """Returns the angle between two vectors in degrees"""
```

```
    return np.degrees(
        math.acos(np.dot(u, v) / (np.linalg.norm(u) * np.linalg.norm(v))))

def get_angle(self, p1, p2, p3):
    """
    Returns the angle between the line segment from p2 to p1
    and the line segment from p2 to p3 in degrees
    """
    a = np.radians(np.array(p1))
    b = np.radians(np.array(p2))
    c = np.radians(np.array(p3))

    avec = a - b
    cvec = c - b

    return self.angle_between_vectors_degrees(avec, cvec)

def angle_range(self, quad):
    """
    Returns the range between max and min interior angles of quadrilateral.
    The input quadrilateral must be a numpy array with vertices ordered clockwise
    starting with the top left vertex.
    """
    tl, tr, br, bl = quad
    ura = self.get_angle(tl[0], tr[0], br[0])
    ula = self.get_angle(bl[0], tl[0], tr[0])
    lra = self.get_angle(tr[0], br[0], bl[0])
    lla = self.get_angle(br[0], bl[0], tl[0])

    angles = [ura, ula, lra, lla]
    return np.ptp(angles)

def get_corners(self, img):
    """
    Returns a list of corners ((x, y) tuples) found in the input image. With proper
    pre-processing and filtering, it should output at most 10 potential corners.
    This is a utility function used by get_contours. The input image is expected
    to be rescaled and Canny filtered prior to be passed in.
    """
    lsd = cv2.createLineSegmentDetector()
    lines = lsd.detect(img)[0]

    # massages the output from LSD
    # LSD operates on edges. One "line" has 2 edges, and so we need to combine the edges back into lines
```

```
    # 1. separate out the lines into horizontal and vertical lines.
    # 2. Draw the horizontal lines back onto a canvas, but slightly thicker and longer.
    # 3. Run connected-components on the new canvas
    # 4. Get the bounding box for each component, and the bounding box is final line.
    # 5. The ends of each line is a corner
    # 6. Repeat for vertical lines
    # 7. Draw all the final lines onto another canvas. Where the lines overlap are also corners

    corners = []
    if lines is not None:
        # separate out the horizontal and vertical lines, and draw them back onto separate canvases
        lines = lines.squeeze().astype(np.int32).tolist()
        horizontal_lines_canvas = np.zeros(img.shape, dtype=np.uint8)
        vertical_lines_canvas = np.zeros(img.shape, dtype=np.uint8)
        for line in lines:
            x1, y1, x2, y2 = line
            if abs(x2 - x1) > abs(y2 - y1):
                (x1, y1), (x2, y2) = sorted(((x1, y1), (x2, y2)), key=lambda pt: pt[0])
                cv2.line(horizontal_lines_canvas, (max(x1 - 5, 0), y1), (min(x2 + 5, img.shape[1] - 1), y2), 255, 2)
            else:
                (x1, y1), (x2, y2) = sorted(((x1, y1), (x2, y2)), key=lambda pt: pt[1])
                cv2.line(vertical_lines_canvas, (x1, max(y1 - 5, 0)), (x2, min(y2 + 5, img.shape[0] - 1)), 255, 2)

        lines = []

        # find the horizontal lines (connected-components -> bounding boxes -> final lines)
        contours = cv2.findContours(horizontal_lines_canvas, cv2.RETR_EXTERNAL, cv2.CHAIN_APPRO
X_NONE)
        contours = contours[1]
        contours = sorted(contours, key=lambda c: cv2.arcLength(c, True), reverse=True)[:2]
        horizontal_lines_canvas = np.zeros(img.shape, dtype=np.uint8)
        for contour in contours:
            contour = contour.reshape((contour.shape[0], contour.shape[2]))
            min_x = np.amin(contour[:, 0], axis=0) + 2
            max_x = np.amax(contour[:, 0], axis=0) - 2
            left_y = int(np.average(contour[contour[:, 0] == min_x][:, 1]))
            right_y = int(np.average(contour[contour[:, 0] == max_x][:, 1]))
            lines.append((min_x, left_y, max_x, right_y))
            cv2.line(horizontal_lines_canvas, (min_x, left_y), (max_x, right_y), 1, 1)
            corners.append((min_x, left_y))
            corners.append((max_x, right_y))

        # find the vertical lines (connected-components -> bounding boxes -> final lines)
        contours = cv2.findContours(vertical_lines_canvas, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
```

```
NONE)
            contours = contours[1]
            contours = sorted(contours, key=lambda c: cv2.arcLength(c, True), reverse=True)[:2]
            vertical_lines_canvas = np.zeros(img.shape, dtype=np.uint8)
            for contour in contours:
                contour = contour.reshape((contour.shape[0], contour.shape[2]))
                min_y = np.amin(contour[:, 1], axis=0) + 2
                max_y = np.amax(contour[:, 1], axis=0) - 2
                top_x = int(np.average(contour[contour[:, 1] == min_y][:, 0]))
                bottom_x = int(np.average(contour[contour[:, 1] == max_y][:, 0]))
                lines.append((top_x, min_y, bottom_x, max_y))
                cv2.line(vertical_lines_canvas, (top_x, min_y), (bottom_x, max_y), 1, 1)
                corners.append((top_x, min_y))
                corners.append((bottom_x, max_y))

            # find the corners
            corners_y, corners_x = np.where(horizontal_lines_canvas + vertical_lines_canvas == 2)
            corners += zip(corners_x, corners_y)

        # remove corners in close proximity
        corners = self.filter_corners(corners)
        return corners

    def is_valid_contour(self, cnt, IM_WIDTH, IM_HEIGHT):
        """Returns True if the contour satisfies all requirements set at instantitation"""

        return (len(cnt) == 4 and cv2.contourArea(cnt) > IM_WIDTH * IM_HEIGHT * self.MIN_QUAD_AREA
_RATIO
                and self.angle_range(cnt) < self.MAX_QUAD_ANGLE_RANGE)

    def get_contour(self, rescaled_image):
        """
        Returns a numpy array of shape (4, 2) containing the vertices of the four corners
        of the document in the image. It considers the corners returned from get_corners()
        and uses heuristics to choose the four corners that most likely represent
        the corners of the document. If no corners were found, or the four corners represent
        a quadrilateral that is too small or convex, it returns the original four corners.
        """

        # these constants are carefully chosen
        MORPH = 9
        CANNY = 84
        HOUGH = 25
```

```
IM_HEIGHT, IM_WIDTH, _ = rescaled_image.shape

# convert the image to grayscale and blur it slightly
gray = cv2.cvtColor(rescaled_image, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (7, 7), 0)

# dilate helps to remove potential holes between edge segments
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (MORPH, MORPH))
dilated = cv2.dilate(gray, kernel)

# find edges and mark them in the output map using the Canny algorithm
edged = cv2.Canny(dilated, 0, CANNY)
test_corners = self.get_corners(edged)

approx_contours = []

if len(test_corners) >= 4:
    quads = []

    for quad in itertools.combinations(test_corners, 4):
        points = np.array(quad)
        points = transform.order_points(points)
        points = np.array([[p] for p in points], dtype="int32")
        quads.append(points)

    # get top five quadrilaterals by area
    quads = sorted(quads, key=cv2.contourArea, reverse=True)[:5]
    # sort candidate quadrilaterals by their angle range, which helps remove outliers
    quads = sorted(quads, key=self.angle_range)

    approx = quads[0]
    if self.is_valid_contour(approx, IM_WIDTH, IM_HEIGHT):
        approx_contours.append(approx)

    # for debugging: uncomment the code below to draw the corners and countour found
    # by get_corners() and overlay it on the image

    # cv2.drawContours(rescaled_image, [approx], -1, (20, 20, 255), 2)
    # plt.scatter(*zip(*test_corners))
    # plt.imshow(rescaled_image)
    # plt.show()

# also attempt to find contours directly from the edged image, which occasionally
# produces better results
```

```
    (_, cnts, hierarchy) = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
SIMPLE)
    cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]

    # loop over the contours
    for c in cnts:
      # approximate the contour
      approx = cv2.approxPolyDP(c, 80, True)
      if self.is_valid_contour(approx, IM_WIDTH, IM_HEIGHT):
        approx_contours.append(approx)
        break

    # If we did not find any valid contours, just use the whole image
    if not approx_contours:
      TOP_RIGHT = (IM_WIDTH, 0)
      BOTTOM_RIGHT = (IM_WIDTH, IM_HEIGHT)
      BOTTOM_LEFT = (0, IM_HEIGHT)
      TOP_LEFT = (0, 0)
      screenCnt = np.array([[TOP_RIGHT], [BOTTOM_RIGHT], [BOTTOM_LEFT], [TOP_LEFT]])

    else:
      screenCnt = max(approx_contours, key=cv2.contourArea)

    return screenCnt.reshape(4, 2)

  def interactive_get_contour(self, screenCnt, rescaled_image):
    poly = Polygon(screenCnt, animated=True, fill=False, linewidth=1)
    fig, ax = plt.subplots()
    ax.add_patch(poly)
    # ax.set_title(('Drag the corners of the box to the corners of the document. \n'
    #         'Close the window when finished.'))
    p = poly_i.PolygonInteractor(ax, poly)
    plt.imshow(rescaled_image[..., ::-1])
    # plt.plot([17], [1470], 'bo')
    plt.show()

    new_points = p.get_poly_points()[:4]
    new_points = np.array([[p] for p in new_points], dtype="int32")
    return new_points.reshape(4, 2)

  def scan(self, image_path):

    # RESCALED_HEIGHT = 500.0
    OUTPUT_DIR = 'output'
```

```
        # load the image and compute the ratio of the old height
        # to the new height, clone it, and resize it
        image = cv2.imread(image_path)

        assert(image is not None)

        # ratio = image.shape[0] / RESCALED_HEIGHT
        # orig = image.copy()
        # rescaled_image = imutils.resize(image, height=int(RESCALED_HEIGHT))

        # get the contour of the document
        # screenCnt = self.get_contour(rescaled_image)
        # screenCnt = self.get_contour(image)
        screenCnt = np.array([[200, 20], [200, 200], [20, 200], [20, 20]])
        # screenCnt = np.array([[590, 697],
        #                       [1495, 1076],
        #                       [470, 1079],
        #                       [13, 706]]
        #                      )

        if self.interactive:
            # screenCnt = self.interactive_get_contour(screenCnt, rescaled_image)
            screenCnt = self.interactive_get_contour(screenCnt, image)
            print(screenCnt)

        # apply the perspective transformation
        # warped = transform.four_point_transform(orig, screenCnt * ratio)
        warped = transform.four_point_transform(image, screenCnt)

        # # convert the warped image to grayscale
        # gray = cv2.cvtColor(warped, cv2.COLOR_BGR2GRAY)

        # # sharpen image
        # sharpen = cv2.GaussianBlur(gray, (0,0), 3)
        # sharpen = cv2.addWeighted(gray, 1.5, sharpen, -0.5, 0)

        # # apply adaptive threshold to get black and white effect
        # thresh = cv2.adaptiveThreshold(sharpen, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRES
H_BINARY, 21, 15)

        # save the transformed image
        basename = os.path.basename(image_path)
        # cv2.imwrite(OUTPUT_DIR + '/' + basename, thresh)
```

```
            cv2.imwrite(OUTPUT_DIR + '/' + basename, warped)
            print("Proccessed " + basename)

if __name__ == "__main__":
    ap = argparse.ArgumentParser()
    group = ap.add_mutually_exclusive_group(required=True)
    group.add_argument("--images", help="Directory of images to be scanned")
    group.add_argument("--image", help="Path to single image to be scanned")
    ap.add_argument("-i", action='store_true',
                    help="Flag for manually verifying and/or setting document corners")

    args = vars(ap.parse_args())
    im_dir = args["images"]
    im_file_path = args["image"]
    interactive_mode = args["i"]

    scanner = DocScanner(interactive_mode)

    valid_formats = [".jpg", ".jpeg", ".jp2", ".png", ".bmp", ".tiff", ".tif"]

    def get_ext(f): return os.path.splitext(f)[1].lower()

    # Scan single image specified by command line argument --image <IMAGE_PATH>
    if im_file_path:
        scanner.scan(im_file_path)

    # Scan all valid images in directory specified by command line argument --images <IMAGE_DIR>
    else:
        im_files = [f for f in os.listdir(im_dir) if get_ext(f) in valid_formats]
        for im in im_files:
            scanner.scan(im_dir + '/' + im)
```

2. 边缘检测模型代码

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

img = cv2.imread("/Users/haonan/本地文稿/数学建模/2019 年中国研究生数学建模竞赛赛题/C 题/附件/图4.png", 0)

img = cv2.GaussianBlur(img, (3, 3), 0)

canny = cv2.Canny(img, 50, 100)
plt.imsave('output.png', canny, cmap='gray')
plt.imshow(canny, cmap='gray')
plt.show()
```

3. 光流算法

```
import numpy as np
import cv2

# cap = cv2.VideoCapture('/Users/haonan/本地文稿/数学建模/2019 年中国研究生数学建模竞赛赛题/C 题/附件/无人机拍庄园_crop.mp4')

# params for ShiTomasi corner detection
feature_params = dict(maxCorners=100,
                      qualityLevel=0.3,
                      minDistance=7,
                      blockSize=7)

# Parameters for lucas kanade optical flow
lk_params = dict(winSize=(15, 15),
                 maxLevel=2,
                 criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Create some random colors
color = np.random.randint(0, 255, (100, 3))

# Take first frame and find corners in it
# ret, old_frame = cap.read()
old_frame = cv2.imread('/Users/haonan/workspace/frameExtract/frame/无人机拍庄园_crop.mp4/1920.jpg')
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
```

```
p0 = cv2.goodFeaturesToTrack(old_gray, mask=None, **feature_params)

# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)

# idx = 0
# while(True):
for idx in range(1921, 1941):
    # ret, frame = cap.read()
    frame = cv2.imread('/Users/haonan/workspace/frameExtract/frame/无人机拍庄园_crop.mp4/%d.jpg' % idx)
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params)

    # Select good points
    good_new = p1[st == 1]
    good_old = p0[st == 1]

    # draw the tracks
    for i, (new, old) in enumerate(zip(good_new, good_old)):
        a, b = new.ravel()
        c, d = old.ravel()
        mask = cv2.line(mask, (a, b), (c, d), color[i].tolist(), 2)
        frame = cv2.circle(frame, (a, b), 5, color[i].tolist(), -1)
    img = cv2.add(frame, mask)

    cv2.imwrite('./frames/%d.png' % idx, img)
    # idx += 1
    # cv2.imshow('frame', img)
    # k = cv2.waitKey(30) & 0xff
    # if k == 27:
    #     break

    # Now update the previous frame and previous points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)

cv2.destroyAllWindows()
# cap.release()
```

4. 视频帧提取代码

```
import cv2
import os

def grabFrame(timeInterval, fileName, videoPath):
    # 读取视频
    cap = cv2.VideoCapture(videoPath)
    print(cap.isOpened())
    if(cap.isOpened()):
        # 获取总帧数
        frameCount = cap.get(cv2.CAP_PROP_FRAME_COUNT)
        print("总帧数：%d" % frameCount)
        # 获取帧率
        fps = cap.get(cv2.CAP_PROP_FPS)
        print("帧率：%d" % fps)
        os.mkdir("frame/" + str(fileName))
        for i in range(0, int(frameCount)):
            # 设置帧的位置
            # cap.set(cv2.CAP_PROP_POS_FRAMES,i*fps*timeInterval)
            # isRead 是否读取成功
            isRead, frame = cap.read()
            if isRead:
                cv2.imwrite("frame/" + str(fileName) + "/" + str(i) + ".jpg", frame)
        cap.release()

if __name__ == '__main__':
    dir_root = r'video'
    files = os.listdir(dir_root)
    files.sort()
    for path in files:
        filepath = os.path.join(dir_root, path)
        print(filepath)
        grabFrame(1, path, filepath)
    print("done!")
```

5. 景深预测模型代码

```
# Copyright UCL Business plc 2017. Patent Pending. All rights reserved.
#
# The MonoDepth Software is licensed under the terms of the UCLB ACP-A licence
# which allows for non-commercial use only, the full terms of which are made
# available in the LICENSE file.
#
# For any other use of the software not covered by the UCLB ACP-A Licence,
# please contact info@uclb.com

from __future__ import absolute_import, division, print_function

# only keep warnings and errors
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='0'

import numpy as np
import argparse
import re
import time
import tensorflow as tf
import tensorflow.contrib.slim as slim
import scipy.misc
import matplotlib.pyplot as plt

from monodepth_model import *
from monodepth_dataloader import *
from average_gradients import *

parser = argparse.ArgumentParser(description='Monodepth TensorFlow implementation.')

parser.add_argument('--encoder',          type=str,   help='type of encoder, vgg or resnet50', default='vgg')
parser.add_argument('--image_path',       type=str,   help='path to the image', required=True)
parser.add_argument('--
checkpoint_path',  type=str,   help='path to a specific checkpoint to load', required=True)
parser.add_argument('--input_height',     type=int,   help='input height', default=256)
parser.add_argument('--input_width',      type=int,   help='input width', default=512)

args = parser.parse_args()

def post_process_disparity(disp):
    _, h, w = disp.shape
```

```
    l_disp = disp[0,:,:]
    r_disp = np.fliplr(disp[1,:,:])
    m_disp = 0.5 * (l_disp + r_disp)
    l, _ = np.meshgrid(np.linspace(0, 1, w), np.linspace(0, 1, h))
    l_mask = 1.0 - np.clip(20 * (l - 0.05), 0, 1)
    r_mask = np.fliplr(l_mask)
    return r_mask * l_disp + l_mask * r_disp + (1.0 - l_mask - r_mask) * m_disp

def test_simple(params):
    """Test function."""

    left  = tf.placeholder(tf.float32, [2, args.input_height, args.input_width, 3])
    model = MonodepthModel(params, "test", left, None)

    input_image = scipy.misc.imread(args.image_path, mode="RGB")
    original_height, original_width, num_channels = input_image.shape
    input_image = scipy.misc.imresize(input_image, [args.input_height, args.input_width], interp='lanczos')
    input_image = input_image.astype(np.float32) / 255
    input_images = np.stack((input_image, np.fliplr(input_image)), 0)

    # SESSION
    config = tf.ConfigProto(allow_soft_placement=True)
    sess = tf.Session(config=config)

    # SAVER
    train_saver = tf.train.Saver()

    # INIT
    sess.run(tf.global_variables_initializer())
    sess.run(tf.local_variables_initializer())
    coordinator = tf.train.Coordinator()
    threads = tf.train.start_queue_runners(sess=sess, coord=coordinator)

    # RESTORE
    restore_path = args.checkpoint_path.split(".")[0]
    train_saver.restore(sess, restore_path)

    disp = sess.run(model.disp_left_est[0], feed_dict={left: input_images})
    disp_pp = post_process_disparity(disp.squeeze()).astype(np.float32)

    output_directory = os.path.dirname(args.image_path)
    output_name = os.path.splitext(os.path.basename(args.image_path))[0]

    np.save(os.path.join(output_directory, "{}_disp.npy".format(output_name)), disp_pp)
```

```
    disp_to_img = scipy.misc.imresize(disp_pp.squeeze(), [original_height, original_width])
    plt.imsave(os.path.join(output_directory, "{}_disp.png".format(output_name)), disp_to_img, cmap='plasma')

    print('done!')

def main(_):

    params = monodepth_parameters(
        encoder=args.encoder,
        height=args.input_height,
        width=args.input_width,
        batch_size=2,
        num_threads=1,
        num_epochs=1,
        do_stereo=False,
        wrap_mode="border",
        use_deconv=False,
        alpha_image_loss=0,
        disp_gradient_loss_weight=0,
        lr_loss_weight=0,
        full_summary=False)

    test_simple(params)

if __name__ == '__main__':
    tf.app.run()


# Copyright UCL Business plc 2017. Patent Pending. All rights reserved.
#
# The MonoDepth Software is licensed under the terms of the UCLB ACP-A licence
# which allows for non-commercial use only, the full terms of which are made
# available in the LICENSE file.
#
# For any other use of the software not covered by the UCLB ACP-A Licence,
# please contact info@uclb.com

"""Fully convolutional model for monocular depth estimation
    by Clement Godard, Oisin Mac Aodha and Gabriel J. Brostow
    http://visual.cs.ucl.ac.uk/pubs/monoDepth/
"""

from __future__ import absolute_import, division, print_function
from collections import namedtuple
```

```
import numpy as np
import tensorflow as tf
import tensorflow.contrib.slim as slim

from bilinear_sampler import *

monodepth_parameters = namedtuple('parameters',
                        'encoder, '
                        'height, width, '
                        'batch_size, '
                        'num_threads, '
                        'num_epochs, '
                        'do_stereo, '
                        'wrap_mode, '
                        'use_deconv, '
                        'alpha_image_loss, '
                        'disp_gradient_loss_weight, '
                        'lr_loss_weight, '
                        'full_summary')

class MonodepthModel(object):
    """monodepth model"""

    def __init__(self, params, mode, left, right, reuse_variables=None, model_index=0):
        self.params = params
        self.mode = mode
        self.left = left
        self.right = right
        self.model_collection = ['model_' + str(model_index)]

        self.reuse_variables = reuse_variables

        self.build_model()
        self.build_outputs()

        if self.mode == 'test':
            return

        self.build_losses()
        self.build_summaries()

    def gradient_x(self, img):
        gx = img[:,:,:-1,:] - img[:,:,1:,:]
```

```
    return gx

def gradient_y(self, img):
    gy = img[:,:-1,:,:] - img[:,1:,:,:]
    return gy

def upsample_nn(self, x, ratio):
    s = tf.shape(x)
    h = s[1]
    w = s[2]
    return tf.image.resize_nearest_neighbor(x, [h * ratio, w * ratio])

def scale_pyramid(self, img, num_scales):
    scaled_imgs = [img]
    s = tf.shape(img)
    h = s[1]
    w = s[2]
    for i in range(num_scales - 1):
        ratio = 2 ** (i + 1)
        nh = h // ratio
        nw = w // ratio
        scaled_imgs.append(tf.image.resize_area(img, [nh, nw]))
    return scaled_imgs

def generate_image_left(self, img, disp):
    return bilinear_sampler_1d_h(img, -disp)

def generate_image_right(self, img, disp):
    return bilinear_sampler_1d_h(img, disp)

def SSIM(self, x, y):
    C1 = 0.01 ** 2
    C2 = 0.03 ** 2

    mu_x = slim.avg_pool2d(x, 3, 1, 'VALID')
    mu_y = slim.avg_pool2d(y, 3, 1, 'VALID')

    sigma_x  = slim.avg_pool2d(x ** 2, 3, 1, 'VALID') - mu_x ** 2
    sigma_y  = slim.avg_pool2d(y ** 2, 3, 1, 'VALID') - mu_y ** 2
    sigma_xy = slim.avg_pool2d(x * y , 3, 1, 'VALID') - mu_x * mu_y

    SSIM_n = (2 * mu_x * mu_y + C1) * (2 * sigma_xy + C2)
    SSIM_d = (mu_x ** 2 + mu_y ** 2 + C1) * (sigma_x + sigma_y + C2)
```

```
        SSIM = SSIM_n / SSIM_d

        return tf.clip_by_value((1 - SSIM) / 2, 0, 1)

    def get_disparity_smoothness(self, disp, pyramid):
        disp_gradients_x = [self.gradient_x(d) for d in disp]
        disp_gradients_y = [self.gradient_y(d) for d in disp]

        image_gradients_x = [self.gradient_x(img) for img in pyramid]
        image_gradients_y = [self.gradient_y(img) for img in pyramid]

        weights_x = [tf.exp(-tf.reduce_mean(tf.abs(g), 3, keep_dims=True)) for g in image_gradients_x]
        weights_y = [tf.exp(-tf.reduce_mean(tf.abs(g), 3, keep_dims=True)) for g in image_gradients_y]

        smoothness_x = [disp_gradients_x[i] * weights_x[i] for i in range(4)]
        smoothness_y = [disp_gradients_y[i] * weights_y[i] for i in range(4)]
        return smoothness_x + smoothness_y

    def get_disp(self, x):
        disp = 0.3 * self.conv(x, 2, 3, 1, tf.nn.sigmoid)
        return disp

    def conv(self, x, num_out_layers, kernel_size, stride, activation_fn=tf.nn.elu):
        p = np.floor((kernel_size - 1) / 2).astype(np.int32)
        p_x = tf.pad(x, [[0, 0], [p, p], [p, p], [0, 0]])
        return slim.conv2d(p_x, num_out_layers, kernel_size, stride, 'VALID', activation_fn=activation_fn)

    def conv_block(self, x, num_out_layers, kernel_size):
        conv1 = self.conv(x,     num_out_layers, kernel_size, 1)
        conv2 = self.conv(conv1, num_out_layers, kernel_size, 2)
        return conv2

    def maxpool(self, x, kernel_size):
        p = np.floor((kernel_size - 1) / 2).astype(np.int32)
        p_x = tf.pad(x, [[0, 0], [p, p], [p, p], [0, 0]])
        return slim.max_pool2d(p_x, kernel_size)

    def resconv(self, x, num_layers, stride):
        do_proj = tf.shape(x)[3] != num_layers or stride == 2
        shortcut = []
        conv1 = self.conv(x,         num_layers, 1, 1)
        conv2 = self.conv(conv1,     num_layers, 3, stride)
        conv3 = self.conv(conv2, 4 * num_layers, 1, 1, None)
        if do_proj:
```

```
            shortcut = self.conv(x, 4 * num_layers, 1, stride, None)
        else:
            shortcut = x
        return tf.nn.elu(conv3 + shortcut)

    def resblock(self, x, num_layers, num_blocks):
        out = x
        for i in range(num_blocks - 1):
            out = self.resconv(out, num_layers, 1)
        out = self.resconv(out, num_layers, 2)
        return out

    def upconv(self, x, num_out_layers, kernel_size, scale):
        upsample = self.upsample_nn(x, scale)
        conv = self.conv(upsample, num_out_layers, kernel_size, 1)
        return conv

    def deconv(self, x, num_out_layers, kernel_size, scale):
        p_x = tf.pad(x, [[0, 0], [1, 1], [1, 1], [0, 0]])
        conv = slim.conv2d_transpose(p_x, num_out_layers, kernel_size, scale, 'SAME')
        return conv[:,3:-1,3:-1,:]

    def build_vgg(self):
        #set convenience functions
        conv = self.conv
        if self.params.use_deconv:
            upconv = self.deconv
        else:
            upconv = self.upconv

        with tf.variable_scope('encoder'):
            conv1 = self.conv_block(self.model_input,  32, 7) # H/2
            conv2 = self.conv_block(conv1,             64, 5) # H/4
            conv3 = self.conv_block(conv2,            128, 3) # H/8
            conv4 = self.conv_block(conv3,            256, 3) # H/16
            conv5 = self.conv_block(conv4,            512, 3) # H/32
            conv6 = self.conv_block(conv5,            512, 3) # H/64
            conv7 = self.conv_block(conv6,            512, 3) # H/128

        with tf.variable_scope('skips'):
            skip1 = conv1
            skip2 = conv2
            skip3 = conv3
            skip4 = conv4
```

```
        skip5 = conv5
        skip6 = conv6

    with tf.variable_scope('decoder'):
        upconv7 = upconv(conv7,  512, 3, 2) #H/64
        concat7 = tf.concat([upconv7, skip6], 3)
        iconv7  = conv(concat7,  512, 3, 1)

        upconv6 = upconv(iconv7, 512, 3, 2) #H/32
        concat6 = tf.concat([upconv6, skip5], 3)
        iconv6  = conv(concat6,  512, 3, 1)

        upconv5 = upconv(iconv6, 256, 3, 2) #H/16
        concat5 = tf.concat([upconv5, skip4], 3)
        iconv5  = conv(concat5,  256, 3, 1)

        upconv4 = upconv(iconv5, 128, 3, 2) #H/8
        concat4 = tf.concat([upconv4, skip3], 3)
        iconv4  = conv(concat4,  128, 3, 1)
        self.disp4 = self.get_disp(iconv4)
        udisp4  = self.upsample_nn(self.disp4, 2)

        upconv3 = upconv(iconv4,  64, 3, 2) #H/4
        concat3 = tf.concat([upconv3, skip2, udisp4], 3)
        iconv3  = conv(concat3,   64, 3, 1)
        self.disp3 = self.get_disp(iconv3)
        udisp3  = self.upsample_nn(self.disp3, 2)

        upconv2 = upconv(iconv3,  32, 3, 2) #H/2
        concat2 = tf.concat([upconv2, skip1, udisp3], 3)
        iconv2  = conv(concat2,   32, 3, 1)
        self.disp2 = self.get_disp(iconv2)
        udisp2  = self.upsample_nn(self.disp2, 2)

        upconv1 = upconv(iconv2,  16, 3, 2) #H
        concat1 = tf.concat([upconv1, udisp2], 3)
        iconv1  = conv(concat1,   16, 3, 1)
        self.disp1 = self.get_disp(iconv1)

def build_resnet50(self):
    #set convenience functions
    conv   = self.conv
    if self.params.use_deconv:
        upconv = self.deconv
```

```
else:
    upconv = self.upconv

with tf.variable_scope('encoder'):
    conv1 = conv(self.model_input, 64, 7, 2) # H/2  -   64D
    pool1 = self.maxpool(conv1,           3) # H/4  -   64D
    conv2 = self.resblock(pool1,      64, 3) # H/8  -  256D
    conv3 = self.resblock(conv2,     128, 4) # H/16 -  512D
    conv4 = self.resblock(conv3,     256, 6) # H/32 - 1024D
    conv5 = self.resblock(conv4,     512, 3) # H/64 - 2048D

with tf.variable_scope('skips'):
    skip1 = conv1
    skip2 = pool1
    skip3 = conv2
    skip4 = conv3
    skip5 = conv4

# DECODING
with tf.variable_scope('decoder'):
    upconv6 = upconv(conv5,   512, 3, 2) #H/32
    concat6 = tf.concat([upconv6, skip5], 3)
    iconv6  = conv(concat6,   512, 3, 1)

    upconv5 = upconv(iconv6, 256, 3, 2) #H/16
    concat5 = tf.concat([upconv5, skip4], 3)
    iconv5  = conv(concat5,   256, 3, 1)

    upconv4 = upconv(iconv5,  128, 3, 2) #H/8
    concat4 = tf.concat([upconv4, skip3], 3)
    iconv4  = conv(concat4,   128, 3, 1)
    self.disp4 = self.get_disp(iconv4)
    udisp4  = self.upsample_nn(self.disp4, 2)

    upconv3 = upconv(iconv4,   64, 3, 2) #H/4
    concat3 = tf.concat([upconv3, skip2, udisp4], 3)
    iconv3  = conv(concat3,    64, 3, 1)
    self.disp3 = self.get_disp(iconv3)
    udisp3  = self.upsample_nn(self.disp3, 2)

    upconv2 = upconv(iconv3,   32, 3, 2) #H/2
    concat2 = tf.concat([upconv2, skip1, udisp3], 3)
    iconv2  = conv(concat2,    32, 3, 1)
    self.disp2 = self.get_disp(iconv2)
```

```
        udisp2  = self.upsample_nn(self.disp2, 2)

        upconv1 = upconv(iconv2,  16, 3, 2) #H
        concat1 = tf.concat([upconv1, udisp2], 3)
        iconv1  = conv(concat1,   16, 3, 1)
        self.disp1 = self.get_disp(iconv1)

    def build_model(self):
        with slim.arg_scope([slim.conv2d, slim.conv2d_transpose], activation_fn=tf.nn.elu):
            with tf.variable_scope('model', reuse=self.reuse_variables):

                self.left_pyramid  = self.scale_pyramid(self.left,  4)
                if self.mode == 'train':
                    self.right_pyramid = self.scale_pyramid(self.right, 4)

                if self.params.do_stereo:
                    self.model_input = tf.concat([self.left, self.right], 3)
                else:
                    self.model_input = self.left

                #build model
                if self.params.encoder == 'vgg':
                    self.build_vgg()
                elif self.params.encoder == 'resnet50':
                    self.build_resnet50()
                else:
                    return None

    def build_outputs(self):
        # STORE DISPARITIES
        with tf.variable_scope('disparities'):
            self.disp_est  = [self.disp1, self.disp2, self.disp3, self.disp4]
            self.disp_left_est  = [tf.expand_dims(d[:,:,:,0], 3) for d in self.disp_est]
            self.disp_right_est = [tf.expand_dims(d[:,:,:,1], 3) for d in self.disp_est]

        if self.mode == 'test':
            return

        # GENERATE IMAGES
        with tf.variable_scope('images'):
            self.left_est  = [self.generate_image_left(self.right_pyramid[i], self.disp_left_est[i])  for i in range(4)]
            self.right_est = [self.generate_image_right(self.left_pyramid[i], self.disp_right_est[i]) for i in range(4)]

        # LR CONSISTENCY
```

```
        with tf.variable_scope('left-right'):
            self.right_to_left_disp = [self.generate_image_left(self.disp_right_est[i], self.disp_left_est[i])  for i in r
ange(4)]
            self.left_to_right_disp = [self.generate_image_right(self.disp_left_est[i], self.disp_right_est[i]) for i in r
ange(4)]

        # DISPARITY SMOOTHNESS
        with tf.variable_scope('smoothness'):
            self.disp_left_smoothness  = self.get_disparity_smoothness(self.disp_left_est,  self.left_pyramid)
            self.disp_right_smoothness = self.get_disparity_smoothness(self.disp_right_est, self.right_pyramid)

    def build_losses(self):
        with tf.variable_scope('losses', reuse=self.reuse_variables):
            # IMAGE RECONSTRUCTION
            # L1
            self.l1_left = [tf.abs( self.left_est[i] - self.left_pyramid[i]) for i in range(4)]
            self.l1_reconstruction_loss_left  = [tf.reduce_mean(l) for l in self.l1_left]
            self.l1_right = [tf.abs(self.right_est[i] - self.right_pyramid[i]) for i in range(4)]
            self.l1_reconstruction_loss_right = [tf.reduce_mean(l) for l in self.l1_right]

            # SSIM
            self.ssim_left = [self.SSIM( self.left_est[i],  self.left_pyramid[i]) for i in range(4)]
            self.ssim_loss_left  = [tf.reduce_mean(s) for s in self.ssim_left]
            self.ssim_right = [self.SSIM(self.right_est[i], self.right_pyramid[i]) for i in range(4)]
            self.ssim_loss_right = [tf.reduce_mean(s) for s in self.ssim_right]

            # WEIGTHED SUM
            self.image_loss_right = [self.params.alpha_image_loss * self.ssim_loss_right[i] + (1 - self.params.alph
a_image_loss) * self.l1_reconstruction_loss_right[i] for i in range(4)]
            self.image_loss_left  = [self.params.alpha_image_loss * self.ssim_loss_left[i]  + (1 - self.params.alpha_
image_loss) * self.l1_reconstruction_loss_left[i]  for i in range(4)]
            self.image_loss = tf.add_n(self.image_loss_left + self.image_loss_right)

            # DISPARITY SMOOTHNESS
            self.disp_left_loss  = [tf.reduce_mean(tf.abs(self.disp_left_smoothness[i]))  / 2 ** i for i in range(4)]
            self.disp_right_loss = [tf.reduce_mean(tf.abs(self.disp_right_smoothness[i])) / 2 ** i for i in range(4)]
            self.disp_gradient_loss = tf.add_n(self.disp_left_loss + self.disp_right_loss)

            # LR CONSISTENCY
            self.lr_left_loss  = [tf.reduce_mean(tf.abs(self.right_to_left_disp[i] - self.disp_left_est[i]))  for i in rang
e(4)]
            self.lr_right_loss = [tf.reduce_mean(tf.abs(self.left_to_right_disp[i] - self.disp_right_est[i])) for i in ran
ge(4)]
            self.lr_loss = tf.add_n(self.lr_left_loss + self.lr_right_loss)
```

```
        # TOTAL LOSS
        self.total_loss = self.image_loss + self.params.disp_gradient_loss_weight * self.disp_gradient_loss + se
lf.params.lr_loss_weight * self.lr_loss

    def build_summaries(self):
        # SUMMARIES
        with tf.device('/cpu:0'):
            for i in range(4):
                tf.summary.scalar('ssim_loss_' + str(i), self.ssim_loss_left[i] + self.ssim_loss_right[i], collections=sel
f.model_collection)
                tf.summary.scalar('l1_loss_' + str(i), self.l1_reconstruction_loss_left[i] + self.l1_reconstruction_loss_
right[i], collections=self.model_collection)
                tf.summary.scalar('image_loss_' + str(i), self.image_loss_left[i] + self.image_loss_right[i], collection
s=self.model_collection)
                tf.summary.scalar('disp_gradient_loss_' + str(i), self.disp_left_loss[i] + self.disp_right_loss[i], collect
ions=self.model_collection)
                tf.summary.scalar('lr_loss_' + str(i), self.lr_left_loss[i] + self.lr_right_loss[i], collections=self.model_
collection)
                tf.summary.image('disp_left_est_' + str(i), self.disp_left_est[i], max_outputs=4, collections=self.mod
el_collection)
                tf.summary.image('disp_right_est_' + str(i), self.disp_right_est[i], max_outputs=4, collections=self.m
odel_collection)

                if self.params.full_summary:
                    tf.summary.image('left_est_' + str(i), self.left_est[i], max_outputs=4, collections=self.model_colle
ction)
                    tf.summary.image('right_est_' + str(i), self.right_est[i], max_outputs=4, collections=self.model_co
llection)
                    tf.summary.image('ssim_left_'  + str(i), self.ssim_left[i],  max_outputs=4, collections=self.model_
collection)
                    tf.summary.image('ssim_right_' + str(i), self.ssim_right[i], max_outputs=4, collections=self.model
_collection)
                    tf.summary.image('l1_left_'  + str(i), self.l1_left[i],  max_outputs=4, collections=self.model_collec
tion)
                    tf.summary.image('l1_right_' + str(i), self.l1_right[i], max_outputs=4, collections=self.model_coll
ection)

            if self.params.full_summary:
                tf.summary.image('left',  self.left,   max_outputs=4, collections=self.model_collection)
                tf.summary.image('right', self.right,  max_outputs=4, collections=self.model_collection)


# Copyright UCL Business plc 2017. Patent Pending. All rights reserved.
```

```
#
# The MonoDepth Software is licensed under the terms of the UCLB ACP-A licence
# which allows for non-commercial use only, the full terms of which are made
# available in the LICENSE file.
#
# For any other use of the software not covered by the UCLB ACP-A Licence,
# please contact info@uclb.com

"""Monodepth data loader.
"""

from __future__ import absolute_import, division, print_function
import tensorflow as tf

def string_length_tf(t):
  return tf.py_func(len, [t], [tf.int64])

class MonodepthDataloader(object):
    """monodepth dataloader"""

    def __init__(self, data_path, filenames_file, params, dataset, mode):
        self.data_path = data_path
        self.params = params
        self.dataset = dataset
        self.mode = mode

        self.left_image_batch  = None
        self.right_image_batch = None

        input_queue = tf.train.string_input_producer([filenames_file], shuffle=False)
        line_reader = tf.TextLineReader()
        _, line = line_reader.read(input_queue)

        split_line = tf.string_split([line]).values

        # we load only one image for test, except if we trained a stereo model
        if mode == 'test' and not self.params.do_stereo:
            left_image_path  = tf.string_join([self.data_path, split_line[0]])
            left_image_o  = self.read_image(left_image_path)
        else:
            left_image_path  = tf.string_join([self.data_path, split_line[0]])
            right_image_path = tf.string_join([self.data_path, split_line[1]])
            left_image_o  = self.read_image(left_image_path)
            right_image_o = self.read_image(right_image_path)
```

```
        if mode == 'train':
            # randomly flip images
            do_flip = tf.random_uniform([], 0, 1)
            left_image  = tf.cond(do_flip > 0.5, lambda: tf.image.flip_left_right(right_image_o), lambda: left_imag
e_o)
            right_image = tf.cond(do_flip > 0.5, lambda: tf.image.flip_left_right(left_image_o),  lambda: right_ima
ge_o)

            # randomly augment images
            do_augment  = tf.random_uniform([], 0, 1)
            left_image, right_image = tf.cond(do_augment > 0.5, lambda: self.augment_image_pair(left_image, rig
ht_image), lambda: (left_image, right_image))

            left_image.set_shape( [None, None, 3])
            right_image.set_shape([None, None, 3])

            # capacity = min_after_dequeue + (num_threads + a small safety margin) * batch_size
            min_after_dequeue = 2048
            capacity = min_after_dequeue + 4 * params.batch_size
            self.left_image_batch, self.right_image_batch = tf.train.shuffle_batch([left_image, right_image],
                    params.batch_size, capacity, min_after_dequeue, params.num_threads)

        elif mode == 'test':
            self.left_image_batch = tf.stack([left_image_o,  tf.image.flip_left_right(left_image_o)],  0)
            self.left_image_batch.set_shape( [2, None, None, 3])

            if self.params.do_stereo:
                self.right_image_batch = tf.stack([right_image_o,  tf.image.flip_left_right(right_image_o)],  0)
                self.right_image_batch.set_shape( [2, None, None, 3])

    def augment_image_pair(self, left_image, right_image):
        # randomly shift gamma
        random_gamma = tf.random_uniform([], 0.8, 1.2)
        left_image_aug  = left_image  ** random_gamma
        right_image_aug = right_image ** random_gamma

        # randomly shift brightness
        random_brightness = tf.random_uniform([], 0.5, 2.0)
        left_image_aug  =  left_image_aug * random_brightness
        right_image_aug = right_image_aug * random_brightness

        # randomly shift color
        random_colors = tf.random_uniform([3], 0.8, 1.2)
```

```
        white = tf.ones([tf.shape(left_image)[0], tf.shape(left_image)[1]])
        color_image = tf.stack([white * random_colors[i] for i in range(3)], axis=2)
        left_image_aug  *= color_image
        right_image_aug *= color_image

        # saturate
        left_image_aug  = tf.clip_by_value(left_image_aug,  0, 1)
        right_image_aug = tf.clip_by_value(right_image_aug, 0, 1)

        return left_image_aug, right_image_aug

    def read_image(self, image_path):
        # tf.decode_image does not return the image size, this is an ugly workaround to handle both jpeg and png
        path_length = string_length_tf(image_path)[0]
        file_extension = tf.substr(image_path, path_length - 3, 3)
        file_cond = tf.equal(file_extension, 'jpg')

        image  = tf.cond(file_cond, lambda: tf.image.decode_jpeg(tf.read_file(image_path)), lambda: tf.image.de
code_png(tf.read_file(image_path)))

        # if the dataset is cityscapes, we crop the last fifth to remove the car hood
        if self.dataset == 'cityscapes':
            o_height    = tf.shape(image)[0]
            crop_height = (o_height * 4) // 5
            image  =  image[:crop_height,:,:]

        image  = tf.image.convert_image_dtype(image,  tf.float32)
        image  = tf.image.resize_images(image,  [self.params.height, self.params.width], tf.image.ResizeMethod.
AREA)

        return image


# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
# See the License for the specific language governing permissions and
# limitations under the License.

from __future__ import absolute_import, division, print_function
import tensorflow as tf

def average_gradients(tower_grads):

    average_grads = []
    for grad_and_vars in zip(*tower_grads):
    # Note that each grad_and_vars looks like the following:
    #   ((grad0_gpu0, var0_gpu0), ... , (grad0_gpuN, var0_gpuN))
        grads = []
        for g, _ in grad_and_vars:
            # Add 0 dimension to the gradients to represent the tower.
            expanded_g = tf.expand_dims(g, 0)

            # Append on a 'tower' dimension which we will average over below.
            grads.append(expanded_g)

        # Average over the 'tower' dimension.
        grad = tf.concat(axis=0, values=grads)
        grad = tf.reduce_mean(grad, 0)

        # Keep in mind that the Variables are redundant because they are shared
        # across towers. So .. we will just return the first tower's pointer to
        # the Variable.
        v = grad_and_vars[0][1]
        grad_and_var = (grad, v)
        average_grads.append(grad_and_var)
    return average_grads


import numpy as np
import matplotlib.pyplot as plt
import scipy.misc

from PIL import Image

img = Image.open('图 3_kitti_res.png')
dep = np.load('图 3_kitti_res.npy')
dep = scipy.misc.imresize(dep.squeeze(), [img.height, img.width])
coords = np.where(dep == np.amax(dep))
print(coords)
```

```
# y, x = 1470, 17
plt.subplot(121)
# plt.title('[%d, %d]' % (x, y))
plt.imshow(img)
plt.plot(coords[1], coords[0], 'ro')
plt.subplot(122)
plt.imshow(dep, cmap='gray')
plt.plot(coords[1], coords[0], 'ro')
plt.show()


# Copyright 2016 The TensorFlow Authors. All Rights Reserved.
# Copyright 2017 Modifications Clement Godard.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
=

from __future__ import absolute_import, division, print_function
import tensorflow as tf

def bilinear_sampler_1d_h(input_images, x_offset, wrap_mode='border', name='bilinear_sampler', **kwargs):
    def _repeat(x, n_repeats):
        with tf.variable_scope('_repeat'):
            rep = tf.tile(tf.expand_dims(x, 1), [1, n_repeats])
            return tf.reshape(rep, [-1])

    def _interpolate(im, x, y):
        with tf.variable_scope('_interpolate'):

            # handle both texture border types
            _edge_size = 0
            if _wrap_mode == 'border':
                _edge_size = 1
```

```
            im = tf.pad(im, [[0, 0], [1, 1], [1, 1], [0, 0]], mode='CONSTANT')
            x = x + _edge_size
            y = y + _edge_size
        elif _wrap_mode == 'edge':
            _edge_size = 0
        else:
            return None

        x = tf.clip_by_value(x, 0.0,  _width_f - 1 + 2 * _edge_size)

        x0_f = tf.floor(x)
        y0_f = tf.floor(y)
        x1_f = x0_f + 1

        x0 = tf.cast(x0_f, tf.int32)
        y0 = tf.cast(y0_f, tf.int32)
        x1 = tf.cast(tf.minimum(x1_f,  _width_f - 1 + 2 * _edge_size), tf.int32)

        dim2 = (_width + 2 * _edge_size)
        dim1 = (_width + 2 * _edge_size) * (_height + 2 * _edge_size)
        base = _repeat(tf.range(_num_batch) * dim1, _height * _width)
        base_y0 = base + y0 * dim2
        idx_l = base_y0 + x0
        idx_r = base_y0 + x1

        im_flat = tf.reshape(im, tf.stack([-1, _num_channels]))

        pix_l = tf.gather(im_flat, idx_l)
        pix_r = tf.gather(im_flat, idx_r)

        weight_l = tf.expand_dims(x1_f - x, 1)
        weight_r = tf.expand_dims(x - x0_f, 1)

        return weight_l * pix_l + weight_r * pix_r

    def _transform(input_images, x_offset):
        with tf.variable_scope('transform'):
            # grid of (x_t, y_t, 1), eq (1) in ref [1]
            x_t, y_t = tf.meshgrid(tf.linspace(0.0,   _width_f - 1.0,  _width),
                                   tf.linspace(0.0 , _height_f - 1.0 , _height))

            x_t_flat = tf.reshape(x_t, (1, -1))
            y_t_flat = tf.reshape(y_t, (1, -1))
```

```
        x_t_flat = tf.tile(x_t_flat, tf.stack([_num_batch, 1]))
        y_t_flat = tf.tile(y_t_flat, tf.stack([_num_batch, 1]))

        x_t_flat = tf.reshape(x_t_flat, [-1])
        y_t_flat = tf.reshape(y_t_flat, [-1])

        x_t_flat = x_t_flat + tf.reshape(x_offset, [-1]) * _width_f

        input_transformed = _interpolate(input_images, x_t_flat, y_t_flat)

        output = tf.reshape(
            input_transformed, tf.stack([_num_batch, _height, _width, _num_channels]))
        return output

    with tf.variable_scope(name):
        _num_batch    = tf.shape(input_images)[0]
        _height       = tf.shape(input_images)[1]
        _width        = tf.shape(input_images)[2]
        _num_channels = tf.shape(input_images)[3]

        _height_f = tf.cast(_height, tf.float32)
        _width_f  = tf.cast(_width,  tf.float32)

        _wrap_mode = wrap_mode

        output = _transform(input_images, x_offset)
        return output
```