

3D 感知算法简单实现

马杰

2019 年 12 月 26 日

0.1 引言

在自动驾驶感知层面上，如何基于单目相机对道路上的动态物体进行三维空间坐标的估计？目前一些光流算法，深度学习算法或者联合的最优化算法可用于单目相机的三维感知，本文基于地平面几何设计了一个简单的算法。核心在于：ground plane 的估计。当我们能够很好的估计每一帧图像对应的 ground plane 时，物体的深度信息以及空间坐标即可得出。思路的构建受启发于 [1],[2],[3]。算法的主要思路：(1) 基于地面 ROI 的单应性矩阵估计 (SIFT 特征匹配算法)，并重构单应性矩阵，求出最佳的法向量 n ，距离 d 。(2) 由地平面几何以及 R, t 矩阵进行重投影最优化，得出优化的空间坐标以及里程计。(3) 由空间坐标以及法向量 n ，求均值，分别得出对应于左右图的 d 值 (估计值)。(4) 由估计好的地平面几何以及物体检测框求出汽车相对于相机的空间坐标。

0.2 Ground plane geometry

ground plane

给定一个相机坐标系，以及一个平面，设坐标系原点与平面的距离为 d (d 恒为正值)，平面的法向量 n 垂直于平面朝下，对于平面上相对于相机坐标系下的任何一点 $X(x,y,z)$ ，我们有 $X^T n = d$ ，如下图所示：

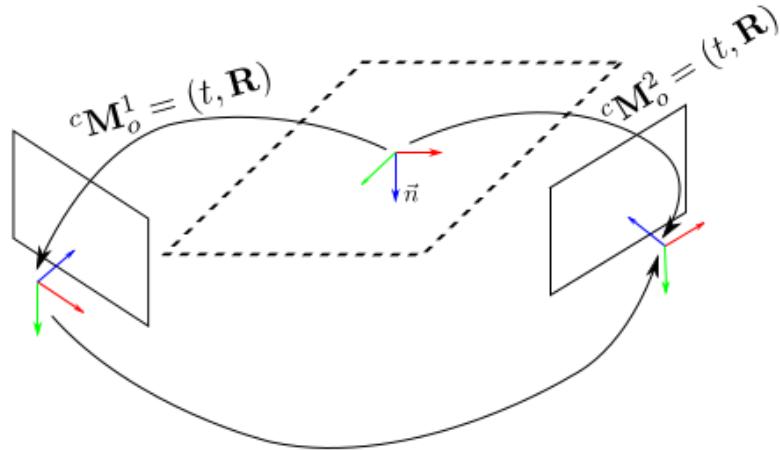


图 1：相邻两帧相机坐标系以及同一平面

单应性矩阵

单应性矩阵通常描述处于共同平面上的一些点在两张图像之间的变换关系，由 ground plane 的定义，我们可以推出单应性矩阵的定义（由于法向量 n 的方向定义不一样，所以推出的 H 矩阵表达式符号会不同）：

$$H = R + \frac{tn^T}{d}$$

(t,R) 代表了相机之间的空间转换关系。如图一所示。

对于单应性矩阵的求解，我们可以采用基于 SURF 或者 SIFT 的特征匹配方式，再结合 RANSAC 算法以及 opencv, visp 库，可以求解出对应的 H 矩阵 (cv::findHomography(matched1,matched2,RANSAC))。然后分解 H 矩阵可以得到四组 (n,R,t) 解 (cv::decomposeHomographyMat(H,K,R,t,n))，重构 H 矩阵 (H.buildfrom(R,t,n))，最后过滤掉不合适的解，得出真实解。具体步骤如下：

第一步： 对解集循环遍历，若对应的解会使图像点的空间坐标 X(X,Y,Z) 的 Z 值为负，则排除。

$$\begin{aligned} X^T n = d & \quad (n = (n_x, n_y, n_z)^T) \\ \Leftrightarrow X \cdot n_x + Y \cdot n_y + Z \cdot n_z &= d \\ x \cdot n_x + y \cdot n_y + n_z &= \frac{d}{Z} \quad ((x, y, 1)^T : normalized coordinates) \\ K^{-1}(u, v, 1)^T &= (x, y, 1)^T \end{aligned}$$

d 为距离，为正值，所以我们可以根据 $\frac{d}{Z}$ 的正负性判断 Z 值的正负，进而排除不必要的解。部分代码如图 2 所示：

第二步： 经过第一步后，若还存在多组解，则我们会选择与初始化向量 n(0,1,0) 最接近的解。代码如图三所示：

$$n = argmax(n^T \cdot n_guess)$$

```

//归一化坐标 X1 = inv(K).p
vpMatrix X1 = cvPointToNormalized(matched1_ROI);
for(unsigned int j=0;j<matched1_ROI.size();++j)
{
    for(unsigned int i=0;i<H.size();++i)
    {
        if(H[i].n.t() * X1.getCol(j) < 0)
        {
            cout << "Z为负值, 此解为无效解! ! ! " << endl;
            cout << "无效解为: "<<H[i].n.t()<<endl;
            H.erase(H.begin()+i);
            break;
        }
    }
}

```

图 2: 第一步对应的代码实现

注意: 这里需要注意时是我们从 H 矩阵分解出来的 t 一开始并不是真正的 t 矩阵, 而是 $\frac{t}{d}$, 要想得到 t 矩阵, 还需乘以预设的 d_guess 值 (1.7), $t = t \times d_{guess}$ 。

0.3 图优化 (R,t,space_coordinates)

得到 (n,d,R,t) 后 (上面求得的 n 默认为当前时刻的最优值, $d_{guess}=1.7, t = t \times d$), 我们还需通过优非线性化得到最优值 ($d_{estimated}, R, t$)。所以需要构建一个二元边的图优化模型 (如图五所示)。需要优化的点为 R, t, 以及空间坐标 X_{left}, X_{right} 。

初始空间坐标计算 通过像素坐标以及法向量 $n_{estimated}, d_{guess}$ 可求得空间坐标 X_{left} 。由上式可知:

$$Z = \frac{d_{guess}}{x.n_x + y.n_y + n_z}$$

左图像素点的三维空间坐标信息进而可知。

$$(X, Y, Z)^T = Z.(x, y, 1)^T \quad (X_{left} : space coordinate respect to left camera)$$

```

//我们假设最优解是H[0]
int idx = 0;
// 如果剩下两个解，继续检查
if(H.size() == 2)
{
    //选择与n_guess更接近的解
    if((H[0].n.t() * n_guess < H[1].n.t() * n_guess))
    {
        idx = 1;
    }
}
//如果剩下一个解，默认为最优解
if(H.size() == 1)
{
    cout << "Best solution found" << endl;
}
//得出左图n向量的估计值
n_estimated = H[idx].n;

```

图 3: 第二步对应的代码实现

```
Eigen::Vector3d translation(t_[0]*d_guess,t_[1]*d_guess,t_[2]*d_guess);
```

图 4: t 矩阵对应的代码部分

构建差误差函数 由 R, t 矩阵求得相对于右相机的空间坐标 X_right, 通过相机内参矩阵 K 投影到右图像中, 构建误差函数。

$$X_{right} = RX_{left} + t$$

$$X_{left}, R, t = argmin \frac{1}{2} \sum_{i=1}^{i=N} \|(u_i, v_i, 1)^T - sK.X_{right_i}\|^2$$

d_estimated 求解 然后可通过求均值的方式得出 d_estimated。代码如图 6 所示:

$$d_{estimated} = mean(\sum_{i=1}^{i=N} n_{estim}^T . X_{left_i})$$

左图汽车三维坐标求解 最终我们得到了左图优化后的 d_estimated, 法向量 n_estimated, 然后通过汽车检测框下边中点像素坐标即可求得左图中汽车的三维坐标。(对应公式上面已经给出) 部分代码如图 7 所示:

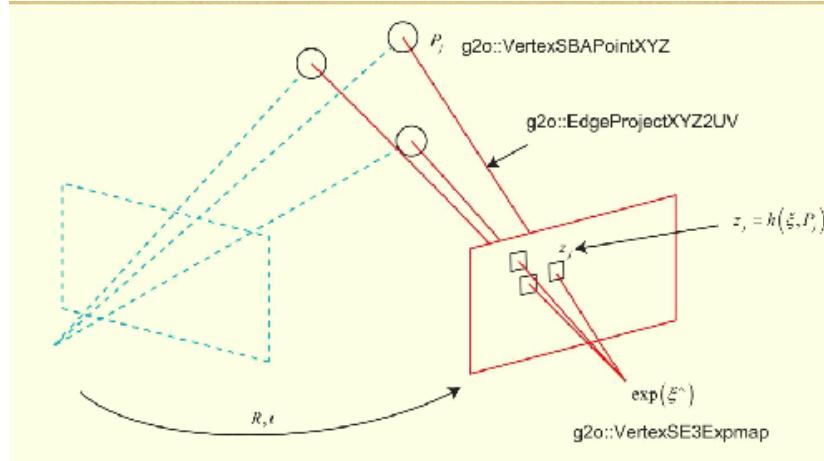


图 5: 图优化以及要引用的类函数

右图汽车三维坐标求解 目前为止, 我们已经得到了基于左图的 $d_{\text{estimated}}$ 和 $n_{\text{estimated}}$, 以及 R, t 矩阵。所以可求得基于右图的 d, n 值, 这里可认为 d, n 为右图的估计值, 因为这是由优化后的量求得的。当进入下两帧图像对比时, d, n 也可作那时左图的先验值, 再进行更精确的求解。部分代码如图 8,9 所示:

0.4 ROI

每一帧图像当中, 我们都要设置地平面的范围, 用来进行 H 矩阵的求解。这里对于 ROI 中特征的提取采取了 SIFT 算法, 因为若采取 ORB 算法, 特征很难提取到, 如图 10, 11 所示。

对应的汽车检测框我们可以从 `label.txt` 文件提取出来, 然后对应设置 ROI 的范围大小。目前为止, 代码中设置 ROI 范围大小的方式比较机械, 即选取的范围是汽车检测框下方部分区域, 如图 12 代码所示:

0.5 demo

现在我们引用 kitti 数据集中的图片进行演示。

```

//相机高度估计值函数
inline void get_d_estimated(const vpMatrix& X)
{
    vpRowVector d(X.getCols());
    for(unsigned int i=0;i<X.getCols();++i)
    {
        d[i]=n_estimated * X.getCol(i);
    }
    double d_esti = vpRowVector::mean(d);
    d_estimated = d_esti;
}

```

图 6: 左图相机高度的优化值求解

```

//得到左图中车的空间位置
inline void get_left_position(const double &xmin, const double &xmax, const double &ymax)
{
    cv::Point2f car_image;
    car_image.x = (xmin + xmax)/2.0;
    car_image.y = ymax;
    vpMatrix car_homo(3,1);
    car_homo[0][0] = car_image.x;
    car_homo[1][0] = car_image.y;
    car_homo[2][0] = 1.0;
    auto car_norm = Ki*car_homo;
    double depth =d_estimated/(n_estimated.t()*car_norm.getCol(0));
    space_coord.resize(3);
    space_coord[0] = car_norm.getCol(0)[0]*depth;
    space_coord[1] = car_norm.getCol(0)[1]*depth;
    space_coord[2] = car_norm.getCol(0)[2]*depth;
}

```

图 7: 左图汽车空间坐标求解

特征匹配, H 矩阵求解 图 15 为终端显示出的左右图片 ROI 中对应的匹配点, 但是这是经过筛选过后的, 即筛选掉错误匹配的像素点, 这样能够保证图优化部分会有较好的效果, 否则正常误差函数太多, 导致收敛出现问题。H 矩阵求解调用了 opencv 库 (如图 16 所示)。

H 矩阵解集筛选 我们会得到四组解, 然后进行筛选, 得出最终解。

空间坐标求解 目前已经得到基于左图的 n_estimated, d_duess=1.7, 以及相机内参矩阵, 调用 cvToVector3d(X1) 函数可求得左图像素点的空间坐标。

图优化 函数 VisualOdom::optim_d() 描述了图优化部分, 函数返回值为优化后的空间坐标。优化的点为相机位姿以及空间坐标, 边为投影公式。这里调

```

//得到右图中车的空间位置
inline void get_right_position(const double &xmin, const double &xmax, const double &ymax)
{
    cv::Point2f car_image;
    car_image.x = (xmin + xmax)/2.0;
    car_image.y = ymax;
    vpMatrix car_homo(3,1);
    car_homo[0][0] = car_image.x;
    car_homo[1][0] = car_image.y;
    car_homo[2][0] = 1.0;
    auto car_norm = Kj*car_homo;
    double depth = d_guess/(n_guess.t()*car_norm.getCol(0));
    space_coord_right.resize(3);
    space_coord_right[0] = car_norm.getCol(0)[0]*depth;
    space_coord_right[1] = car_norm.getCol(0)[1]*depth;
    space_coord_right[2] = car_norm.getCol(0)[2]*depth;
}

```

图 8: 右图汽车空间坐标求解函数

```

//右帧d的估计值
vpColVector X2;
vpRowVector depth_2(X1_estimated.getCols());
vpTranslationVector trans_esti;
trans_esti[0]=estimated[0];trans_esti[1]=t_estimated[1];trans_esti[2]=t_estimated[2];
for (int i =0; i< X1_estimated.getCols(); i++)
{
    X2 = R_cv*X1_estimated.getCol(i)+trans_esti;
    depth_2[i] = n_estimated.t()*X2;
}
//右帧的预测值, 因为有优化后的R, t, 以及空间坐标得到, 所以也可看作为右帧的估计值
d_guess = vpRowVector::mean(depth_2); //右帧d的估计值
n_guess = R_cv*n_estimated; //右帧n的预测值(估计值)
//取左右图检测框下边中间像素点, 并计算出左图车的三维坐标
get_left_position(xmin,xmax,ymax);
get_right_position(xmin1,xmax1,ymax1);

```

图 9: 右图 d,n 求解

用了 g2o 库中的 g2o::VertexSE3Expmap(R,t), g2o::VertexSBAPointXYZ(3D 坐标), g2o::EdgeProjectXYZ2UV(投影项)。结果如图 18 所示。

d 值估计 调用优化函数以及 get_d_estimated() 函数即可求解出相机高度的估计值。如图 19 所知。

左右图汽三维坐标 由基于左图的 d_estimated 以及 n_estimated 可求出左图汽车三维坐标。然后通过优化后的 R, t 求出基于右图的 d 和 n, 即可求出右图汽车的三维坐标。代码分别分别调用 get_left_position(xmin,xmax,ymax), get_right_position(xmin1,xmax1,ymax1) 即可。结果如图 20 所示。

终端演示的结果也存放在了 result.txt 文件中。



图 10: SIFT

图 11: ORB

```

double xmin, xmax,ymin,ymax;
xmin = bbox_im1[0]; ymin=bbox_im1[1];xmax=bbox_im1[2];ymax = bbox_im1[3];
//xmin = 296.744956; xmax = 455.226042; ymin = 161.752147; ymax=292.372804;//还要改进，读取参数
double d = xmax - xmin;
double xmin1, xmax1,ymin1,ymax1;
xmin1 = bbox_im2[0]; ymin1=bbox_im2[1];xmax1=bbox_im2[2];ymax1 = bbox_im2[3];
// xmin1 = 294.898777; xmax1 = 452.199718; ymin1 = 156.024256; ymax1=284.621269 ;
double d1 = xmax1 - xmin1;
std::vector<cv::Point2f>matched1_ROI, matched2_ROI;
for(unsigned int i=0; i<matched1.size();i++)
{
    if(matched1[i].x > (xmin-threshol*d) && matched1[i].x < (xmax+threshol*d) && matched1[i].y :
        &&matched2[i].x > (xmin1 -threshol*d1) && matched2[i].x< (xmax1+threshol*d1)&&matched2[i].y > (ymin1 -threshol*d1) && matched2[i].y< (ymax1+threshol*d1))
        &&abs(matched1[i].x-matched2[i].x)<10.0 && abs(matched1[i].y-matched2[i].y)<10.0)
    {
        matched1_ROI.push_back(matched1[i]);
        matched2_ROI.push_back(matched2[i]);
    }
}
}

```

图 12: ROI 选取方式, 检测框下方的矩形区域, 代码中的 $3 \times d$ 为检测框的宽度, threshol 为阈值, 默认为 1, 更改阈值可改变检测框大小
但是这样的选取方式会存在一些问题。具体细节后面会提到。

0.6 反思总结

算法还有很多需要改进的地方。其中, 最关键的地方在于 ROI 的智能化以及合理的选取, 因为每一帧图片对应的 ROI 可能不太一样, 或者有些帧的地平面通过 SIFT 无法提取充足的特征点进行计算。准确且充足的特征点匹配以及 ROI 的正确选取是求解地平法向量 n 的关键步骤。因为当读取后两帧图片时, ROI 范围定义不变, 计算出来的法向量 n 以及空间坐标误差比较大。

所以接下来需要改进的是 ROI 的选取方式以及特征匹配的方式, 或者地平面法向量 n 的求取方式。

抛开“基于单目相机”这一条件来说, 或许可以采用激光雷达与单目相机融合的方式进行前方车辆空间坐标的提取。首先获得单目相机与激光雷达的外参标定矩阵, 然后将稀疏的雷达点云投影至图像上, 并与检测框融合获



图 13: kitti 数据集左图

图 14: kitti 数据集右图

```

/usr/bin/ld: warning: libopencv_imgproc.so.3.3, needed by /usr/local/lib/libopencv_xfeatures2d.so.3.3.1, may conflict with libopencv_imgproc.so.3.4
/usr/bin/ld: warning: libopencv_core.so.3.3, needed by /usr/local/lib/libopencv_xfeatures2d.so.3.3.1, may conflict with libopencv_core.so.3.4
[100%] Built target 3D_perception
majie@majie-X555LD:~/master2/3D感知/project/3D_perception/build$ ./3D_perception
updated matchers 1 ls:
[147.0449, 332.9443];
149.10679, 326.00064;
153.20881, 316.35812;
153.20881, 316.35812;
157.84283, 308.89023;
160.91027, 337.67882;
164.91027, 338.80457;
172.23241, 300.48604;
172.51358, 305.48834;
179.8297, 293.4166;
179.8297, 293.4166;
213.99106, 337.09259;
234.81007, 335.09761;
327.75231, 335.09823;
332.69451, 332.04855;
342.75208, 323.61014;
444.40585, 323.45557;
444.40585, 323.45557;
528.9306, 323.0563;
553.58043, 330.98669;
555.58069, 330.98669;
580.78167, 328.34064;
601.1637, 315.08154];
updated matchers 2 ls:
[141.32335, 335.03681;
145.91807, 327.87482;
151.91221, 330.03452;
151.91221, 337.03452;
158.18552, 306.90997;
158.18586, 318.37207;
164.10532, 313.70657;
173.66284, 300.23965;
173.54618, 305.43981];
majie@majie-X555LD:~/master2/3D感知/project/3D_perception/build$
```

图 15: 匹配点

取空间坐标信息。

0.7 代码工程结构

编译 工程编译需要 opencv,g2o,visp,eigen, sophus 库。

代码结构 工程由 3D_perception.h,3D_perception.cpp,main.cpp 构成。代码中均含有必要注释。其中 bool VisualOdom::process() 为布尔类型函数，作为最终函数，输入图像以及检测框，然后获汽车汽车空间坐标信息。设置为布尔类型的原因在于当有视频流输入时，可以对视频流进循环迭代，布尔类型会起到很好的作用。

```

cout<<"updated matchers 1 is:"<<endl<<matched1_ROI<<endl;
cout<<"updated matchers 2 is:"<<endl<<matched2_ROI<<endl;
std::vector<char> inliers;
Hp = findHomography(matched1_ROI,matched2_ROI,inliers,cv::RANSAC,3);
cout<<"ROI中匹配点个数为: "<<matched1_ROI.size()<<endl;

```

图 16: H 矩阵求解

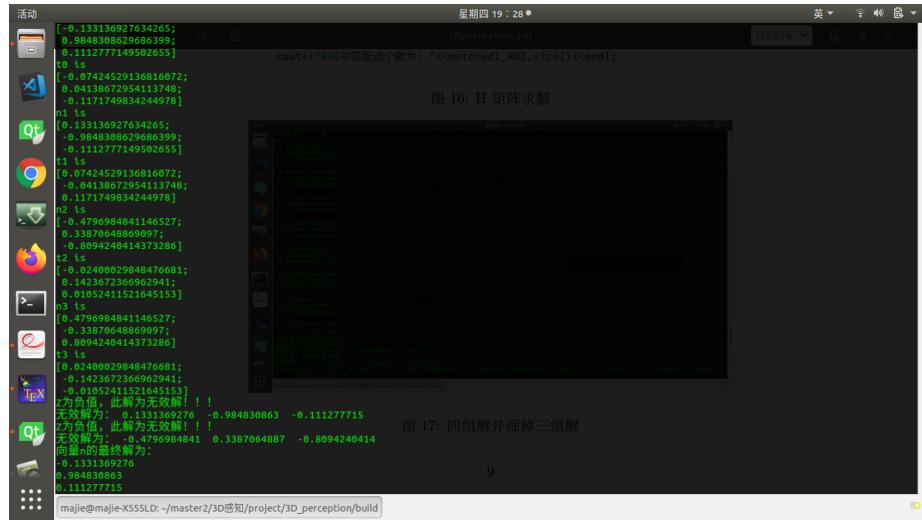


图 17: 四组解并筛选出最终 n 的解

0.8 文章参考

[1]Joint SFM and Detection Cues for Monocular 3D Localization in Road Scenes.

[2]Robust Scale Estimation in Real-Time Monocular SFM for Autonomous Driving.

[3]High Accuracy Monocular SFM and Scale Correction for Autonomous Driving.

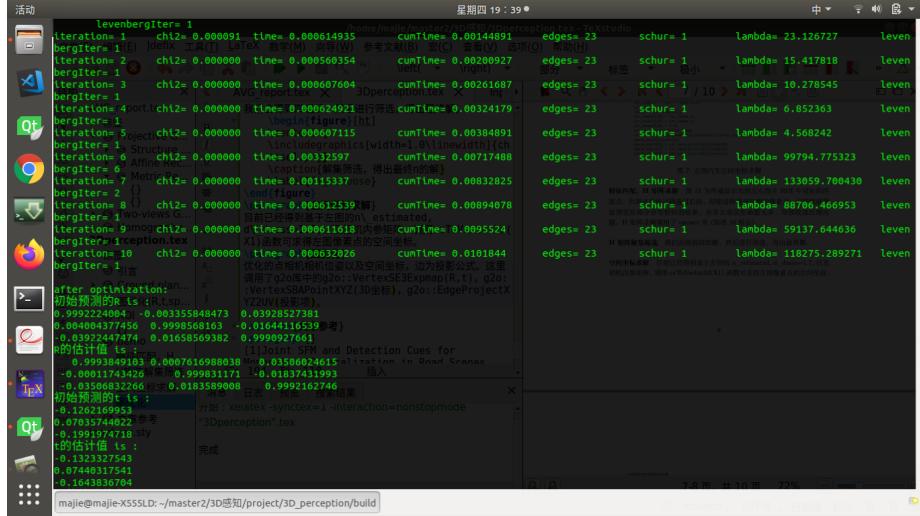


图 18: 初始的 R, t 以及优化后的 R, t

```
//图优化(优化相对于左图的空间坐标以及相机位姿)
vpMatrix X1_estimated=optim_d(pts_3d,matched2_ROI,pose,pose_init);
//得出左图高度d的估计值,d_guess = 1.7
get_d_estimated(X1_estimated);
```

图 19: 由优化后的空间坐标和 n 得出估计的相机高度值

左图车辆的三维空间位置为
x坐标为: -1.733413315
y坐标为: 0.8869898503
距离为: 5.354723341
右图车辆的三维空间位置为
x坐标为: -1.67089608
y坐标为: 0.7912862568
距离为: 5.168318659...

图 20: 左右图汽车三维坐标求解