视觉 SLAM 理论与实践 - 作业 6

peng00bo00

June 27, 2020

1. 1. 代码可参见./code/optical_flow.cpp

   1) 光流法一共分为 4 种：{ additive, compositional } × { forward, inverse }
   2) warp 是对像素发生运动的描述，如计算光流时认为像素仅发生 2 个方向的平移，此时 warp 为 2 个方向上的平移向量
   3) forward 是在第二张图像 I 上面计算图像的梯度，而 inverse 则是在模板图像 T 上计算。由于在迭代时会更新 Warp 参数，因此 forward 方法在每一步迭代时需要重新计算而 inverse 则无需重复计算。

   2.

   1) 误差为两幅图像对应点的灰度之差

   $$e = I_2(x + \Delta x, y + \Delta y) - I_1(x, y) \tag{1}$$

   2) 误差对应的雅可比矩阵为

   $$J = \nabla I_2(x + \Delta x, y + \Delta y) \tag{2}$$

   计算得到的光流如 Fig.1所示

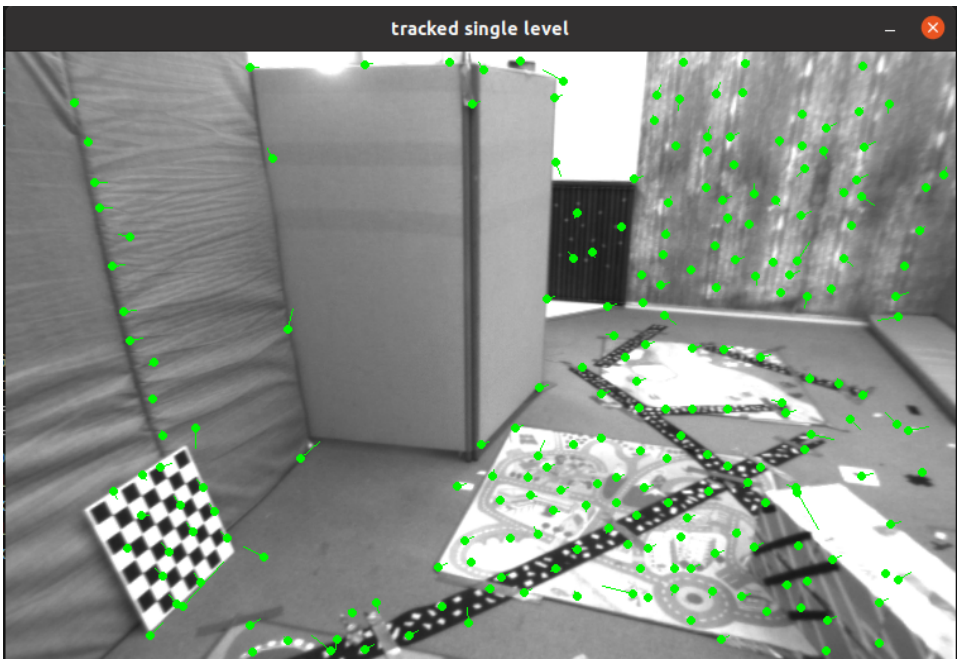

Figure 1: 单层光流

3. 计算得到的光流如 Fig.2所示
4.

   1) coarse-to-fine 是指利用金字塔自顶向下逐步计算光流。在计算每一层的光流时把上一层计算得到的光流作为该层计算的初始值，从而提高光流计算的精度。
   2) 光流金字塔是指利用金字塔子顶向下逐步计算，并把上一层计算得到的光流作为该层计算的初始值。这样保证即使像素点发生较大的运动仍然可以进行计算。而特征金字塔则是为了获得描述子的尺度不变性，从而在不同的尺度下进行计算。二者的目的有所不同。
   计算得到的光流如 Fig.3,Fig.4,Fig.5所示，结果显示使用金字塔后光流的计算结果要明显优于单层光流，而与 OpenCV 计算得到的光流接近，在某些点上甚至要优于 OpenCV 的计算结果。

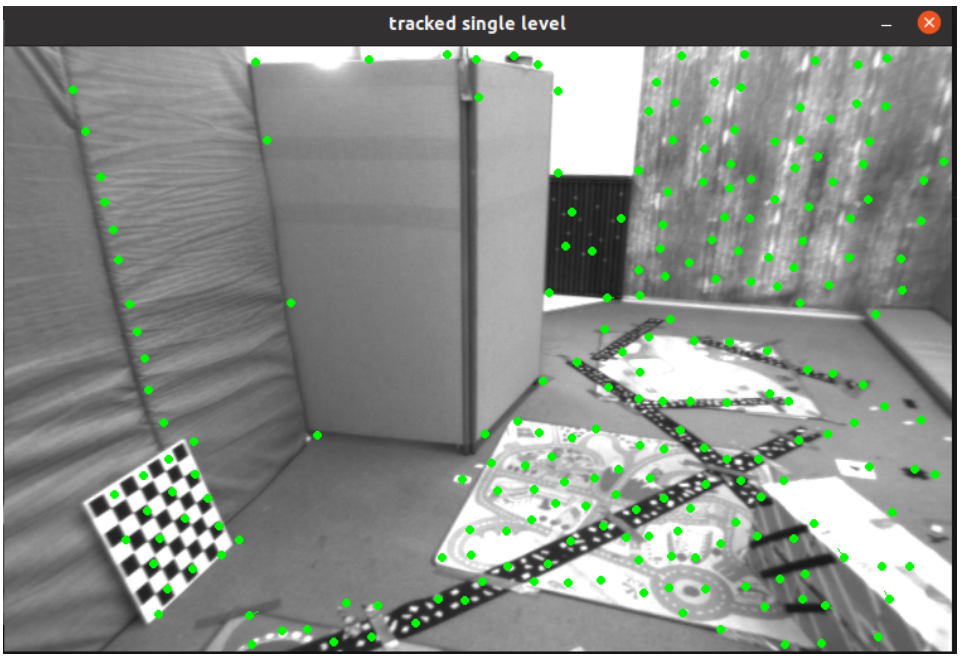

Figure 2: 反向单层光流

5.

1) 根据灰度不变假设，计算比较图像块的灰度之差是合理的。但在实际场景中同一个点的颜色在不同相机位置下是有可能发生变化的，因此需要放宽灰度不变假设或通过预处理的方式使同一个点在不同相机位置下灰度尽可能一致。
2) 理论上讲图像块越大计算结果越稳定也更能够适应较大的运动，当图像块取到和整张图像一样大小时相当于计算稠密光流。本例中使用 16×16 图像块对单层光流的计算结果有改善，但对金字塔的计算结果不明显。
3) 理论上讲金字塔层数越多缩放倍率越大顶层的尺度越大，越能适应较大的运动，计算结果也越好。

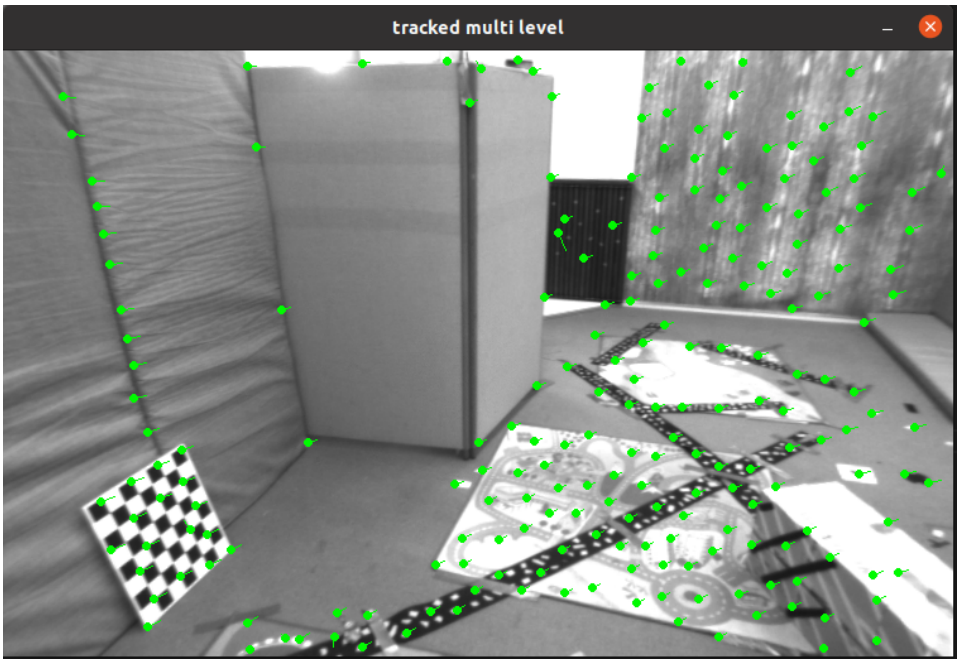

Figure 3: 金字塔光流

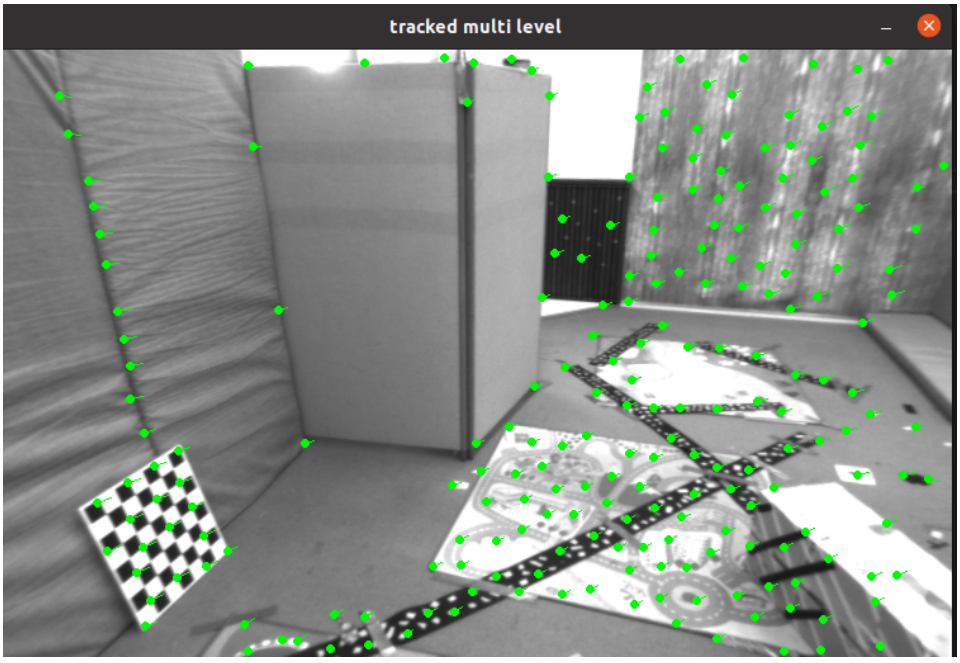

Figure 4: 金字塔反向光流

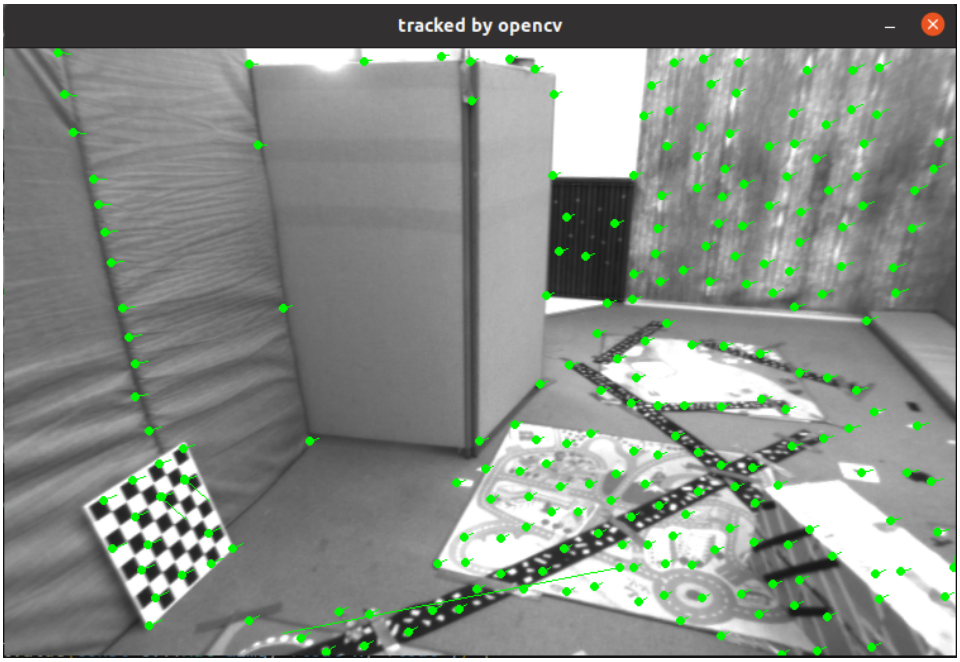

Figure 5: OpenCV 计算光流

2. 1. 代码可参见./code/direct_method.cpp

1) 误差项为同一像素点在两张图像上的灰度差异，即光度误差。

2) 雅可比矩阵为 6×1 维向量，其表达式为

$$\begin{aligned} J &= -\frac{\partial I_2}{\partial u}\frac{\partial u}{\partial \xi} \\ &= -\begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & -\frac{f_x XY}{Z^2} & f_x + \frac{f_x X^2}{Z^2} & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Z}{Z^2} & -f_y - \frac{f_y Y^2}{Z^2} & \frac{f_y XY}{Z^2} & \frac{f_y X}{Z} \end{bmatrix}^T \begin{bmatrix} \nabla_x I_2 \\ \nabla_y I_2 \end{bmatrix} \end{aligned} \tag{3}$$

3) 理论上讲窗口可以取任意大小，这是因为直接法是计算光度误差对相机位姿的导数而不像光流法去求解方程，因此对于窗口大小没有要求，但窗口越大计算量越大会影响到计算的效率。窗口也可以只取一个点，但此时会带来较大的误差。

2. 计算得到的平移量为

$$t_1 = \begin{bmatrix} -0.00183496 \\ -0.00183496 \\ -0.725144 \end{bmatrix}, t_2 = \begin{bmatrix} 0.00741157 \\ -0.00132109 \\ -1.4707 \end{bmatrix}, t_3 = \begin{bmatrix} 0.0070642 \\ 0.00378672 \\ -2.20885 \end{bmatrix}, t_4 = \begin{bmatrix} 0.00786921 \\ 0.00281236 \\ -2.99537 \end{bmatrix}, t_5 = \begin{bmatrix} 0.0188695 \\ -0.0102822 \\ -3.7929 \end{bmatrix} \tag{4}$$

3.

1) 我认为直接法是可以提出类似 inverse, compositional 的概念。

2) 首先可以通过缩小窗口大小来提高效率；其次如果认为窗口内像素对应的是同一个三维空间点则可以简化呀科比矩阵的计算，此时在同一个窗口内 $\frac{\partial u}{\partial \xi}$ 为定值仅进行一次计算即可。

3) 假设了同一个点在不同相机位置下成像的灰度值不变，同时假设了同一个点在两次成像时自身的位置不变

4) 直接法并没有对像素点进行任何的假设，因此理论上可以使用任意点进行计算。但由于计算雅可比矩阵时需要计算图像梯度，因此对于图像梯度较大的点（如角点）往往会的到更为理想的结果。本例中对于非角点仍然能够得到基本正确的结果。

5) 特征点法首先计算图像中的特征点然后通过特征匹配和对极几何来估计相机的姿态，而直接法则通过最小化光度误差来估计相机的姿态。

直接法优点

i. 无需计算特征点和描述子，速度要远快于特征点法
ii. 仅依赖与图像梯度而不依赖于图像特征，因此可以适应特征缺失或是重复性纹理的场合
iii. 可以建立稠密地图

直接法缺点

i. 待优化函数为非凸函数，因此计算结果取决于初始值的选取
ii. 像素之间没有区分度，在选点较少时性能下降明显
iii. 灰度不变的假设过强，实际中很难满足

3. 代码可参见./code/disparity.cpp
提取到的角点和利用光流计算得到的视差可参见 Fig.6, Fig.7。与实际视差图进行对比发现大部分角点的水平视差均在 10% 以内，误差大于 10% 的点约占总体的 20%

Figure 6: 角点提取

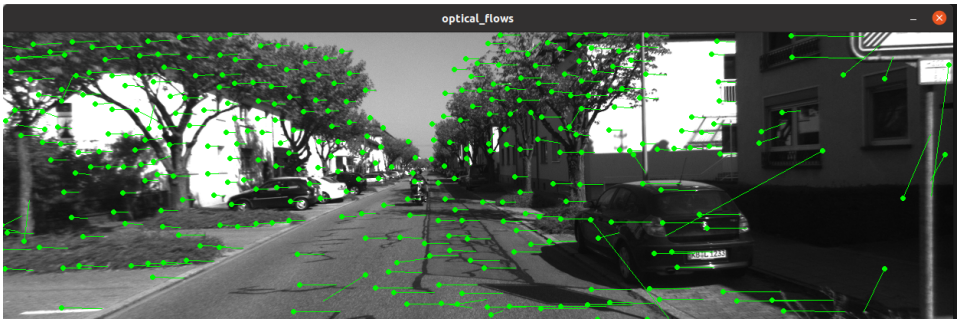

Figure 7: 光流计算视差