模板跟踪算法详细解读与改进

Yuanlizheng

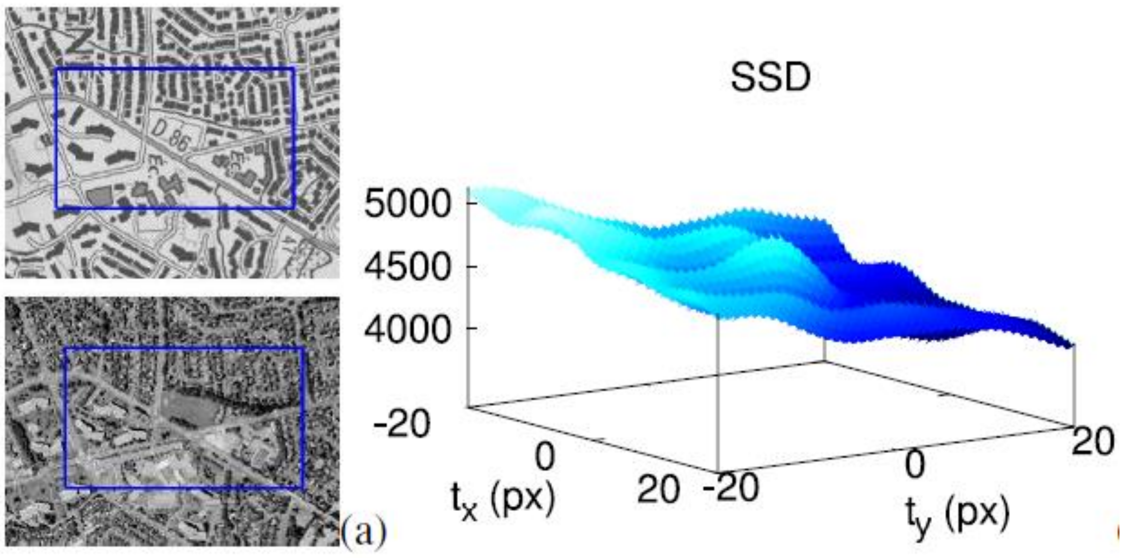
1.概述

模板跟踪算法，最重要的表达式就是，

$$\hat{H}_t = \arg\min_H \sum_{x \in ROI} \left(I^*(x) - I_t(w(x,H)) \right)^2$$

其中，I^* 表示的是关键帧图像，x 表示在关键帧的 ROI 区域的每个像素点，I_t 表示当前帧的图像。w 表示 H 矩阵把 x 坐标映射到当前帧上的操作。其实就是，通过调节 H 矩阵的值，使得上面的 SSD 表达式最小。

两幅图匹配的效果如下图所示，



在程序里面实现，总共分为初始化，优化，更新等几个部分。

2.初始化与优化表达式

从 main_BYD.cpp 的 tracker.initClick 函数进入，初始化的主要的计算过程在 vpTemplateTrackerSSDInverseCompositional.cpp 的 initCompInverse 函数里面。

首先，用三角形选择参考区，然后，生成图像金字塔，对每一层中的参考区的像素，都计算其关于 H 矩阵的扰动 ΔH 的导数。

对于每一项像素残差 e，其表达式展开如下，

$$e = I^*(x) - I_t(w(x, H\cdot\Delta H)) = I^*(x) - I_t\left(\begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & 1 \end{bmatrix} \begin{bmatrix} 1+\Delta H_{11} & \Delta H_{12} & \Delta H_{13} \\ \Delta H_{21} & 1+\Delta H_{22} & \Delta H_{23} \\ \Delta H_{31} & \Delta H_{32} & 1 \end{bmatrix} x \right)$$

从上面表达式可以看出，因为每次迭代之后，x 在图片 I_t 上的投影位置都会发生改变，所以雅克比矩阵 $J = \frac{\partial e}{\partial \Delta H} = -\frac{\partial I_t}{\partial \Delta H}$，不是固定的，所以每次优化迭代，或者在新的图片 I_t 的时候，都要重新计算一次雅克比矩阵，计算量蛮大的。

所以，为了加速计算，改成了逆向模型。逆向模型的思想就是，把关于目标区域的求导，转换成关于参考区域的求导。因为参考帧 o 和当前帧 t 之间的位置关系 H_{to} 是相对的。所以，要获得一个比较好的相对位置，可以是参考帧位置保持单位帧 $H_{oo} = I$ 固定不动，而当前帧移动，变成 $\Delta H_{t't} H_{to}$；也可以是当前帧保持 H_{to} 固定不动，而参考帧移动，变成 $H_{o'o} + \Delta H_{o'o}$。逆向模型就是采用的后者。而参考区域，总是在参考帧上面，而且是固定不变的。所以，改成逆向模型，可以极大地减少计算量。这种逆向模型的做法，在 svo 里面也有使用，参考《SVO 详细解读》。

每次迭代，都用逆向模型优化，然后把结果更新到 H 矩阵上，再进行下一次的迭代。

逆向模型的表达式如下，

$$
\begin{aligned}
e &= I^*\left(w\left(x, H_{o'o}+\Delta H_{o'o}\right)\right) - I_t\left(w\left(x, H_{to}\right)\right) = I^*\left(\begin{bmatrix} 1+\Delta H_{o'o,11} & \Delta H_{o'o,12} & \Delta H_{o'o,13} \\ \Delta H_{o'o,21} & 1+\Delta H_{o'o,22} & \Delta H_{o'o,23} \\ \Delta H_{o'o,31} & \Delta H_{o'o,32} & 1 \end{bmatrix}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}\right) - I_t\left(w\left(x, H_{to}\right)\right) \\
&= I^*\left(p_x = \frac{u+\Delta H_{o'o,11}u+\Delta H_{o'o,12}v+\Delta H_{o'o,13}}{\Delta H_{o'o,31}u+\Delta H_{o'o,32}v+1}, p_y = \frac{\Delta H_{o'o,21}u+v+\Delta H_{o'o,22}v+\Delta H_{o'o,23}}{\Delta H_{o'o,31}u+\Delta H_{o'o,32}v+1}\right) - I_t\left(w\left(x, H_{to}\right)\right) \\
&= I^*\left(p_x, p_y\right) - I_t\left(w\left(x, H_{to}\right)\right)
\end{aligned}
$$

其中，$H_{o'o}$ 的初值为单位阵，即$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$。$p\left(p_x, p_y\right)$表示在 I^* 上的新映射的坐标。

所以，雅克比矩阵 J 的计算表达式如下，

$$
\begin{aligned}
J &= \frac{\partial e}{\partial \Delta H} = \frac{\partial I^*}{\partial \Delta H} = \frac{\partial I^*}{\partial p}\frac{\partial p}{\partial \Delta H} \\
&= \frac{\partial I^*}{\left[\partial p_x \quad \partial p_y\right]}\frac{\partial p}{\partial\left[\Delta H_{11} \quad \Delta H_{12} \quad \Delta H_{13} \quad \Delta H_{21} \quad \Delta H_{22} \quad \Delta H_{23} \quad \Delta H_{31} \quad \Delta H_{32}\right]}
\end{aligned}
$$

其中，ΔH 为扰动，初值全部为零。迭代优化之后，也是非常接近于零的数。所以，在具体关于 ΔH 每个元素求导时，ΔH 中的非被求导的元素，可以直接当成零来看待。

所以，$\frac{\partial p_x}{\partial \Delta H}$ 的计算表达式如下，

$$\frac{\partial p_x}{\partial \Delta H_{11}} = \frac{u}{\Delta H_{o'o,31}u+\Delta H_{o'o,32}v+1} = u$$

$$\frac{\partial p_x}{\partial \Delta H_{12}} = \frac{v}{\Delta H_{o'o,31}u+\Delta H_{o'o,32}v+1} = v$$

$$\frac{\partial p_x}{\partial \Delta H_{13}} = \frac{1}{\Delta H_{o'o,31}u+\Delta H_{o'o,32}v+1} = 1$$

$$\frac{\partial p_x}{\partial \Delta H_{21}} = 0$$

$$\frac{\partial p_x}{\partial \Delta H_{22}} = 0$$

$$\frac{\partial p_x}{\partial \Delta H_{23}} = 0$$

$$\frac{\partial p_x}{\partial \Delta H_{31}} = -\left(u+\Delta H_{o'o,11}u+\Delta H_{o'o,12}v+\Delta H_{o'o,13}\right)\left(\Delta H_{o'o,31}u+\Delta H_{o'o,32}v+1\right)^{-2}u = -u^2$$

$$\frac{\partial p_x}{\partial \Delta H_{32}} = -\left(u+\Delta H_{o'o,11}u+\Delta H_{o'o,12}v+\Delta H_{o'o,13}\right)\left(\Delta H_{o'o,31}u+\Delta H_{o'o,32}v+1\right)^{-2}v = -uv$$

其中，$\frac{\partial p_y}{\partial \Delta H}$ 的计算表达式如下，

$$\frac{\partial p_y}{\partial \Delta H_{11}} = 0$$

$$\frac{\partial p_y}{\partial \Delta H_{12}} = 0$$

$$\frac{\partial p_y}{\partial \Delta H_{13}} = 0$$

$$\frac{\partial p_y}{\partial \Delta H_{21}} = \frac{u}{\Delta H_{o'o,31}u+\Delta H_{o'o,32}v+1} = u$$

$$\frac{\partial p_y}{\partial \Delta H_{22}} = \frac{v}{\Delta H_{o'o,31}u + \Delta H_{o'o,32}v + 1} = v$$

$$\frac{\partial p_y}{\partial \Delta H_{23}} = \frac{1}{\Delta H_{o'o,31}u + \Delta H_{o'o,32}v + 1} = 1$$

$$\frac{\partial p_y}{\partial \Delta H_{31}} = -\left(\Delta H_{o'o,21}u + v + \Delta H_{o'o,22}v + \Delta H_{o'o,23}\right)\left(\Delta H_{o'o,31}u + \Delta H_{o'o,32}v + 1\right)^{-2} u = -uv$$

$$\frac{\partial p_y}{\partial \Delta H_{32}} = -\left(\Delta H_{o'o,21}u + v + \Delta H_{o'o,22}v + \Delta H_{o'o,23}\right)\left(\Delta H_{o'o,31}u + \Delta H_{o'o,32}v + 1\right)^{-2} v = -v^2$$

整理之后，表达式如下所示，

$$\frac{\partial p}{\partial \Delta H} = \begin{bmatrix} u & v & 1 & 0 & 0 & 0 & -u^2 & -uv \\ 0 & 0 & 0 & u & v & 1 & -uv & -v^2 \end{bmatrix}$$

用 $d_x = \frac{\partial I^*}{\partial p_x}$，$d_y = \frac{\partial I^*}{\partial p_y}$ 表示，其实就是参考帧图像 I^* 在 p 点的 x 方向和在 y 方向上的梯度，而这个梯度，是随着参考帧一起保持固定的。则雅克比 J 表达式整理如下，

$$\begin{aligned} J &= \frac{\partial I^*}{\partial p}\frac{\partial p}{\partial \Delta H} = \begin{bmatrix} d_x & d_y \end{bmatrix}\begin{bmatrix} u & v & 1 & 0 & 0 & 0 & -u^2 & -uv \\ 0 & 0 & 0 & u & v & 1 & -uv & -v^2 \end{bmatrix} \\ &= \begin{bmatrix} d_x u & d_x v & d_x & d_y u & d_y v & d_y & -d_x u^2 - d_y uv & -d_x uv - d_y v^2 \end{bmatrix} \end{aligned}$$

因为最后优化的时候，要用的是高斯牛顿法。所以，可以提前计算好高斯牛顿法中需要的矩阵。

正常的高斯牛顿法的做法是，参考《SVO 详细解读》，把所有的 $J_1 \cdots J_n$ 组合成一个大的雅克比矩阵 $\mathbf{J}$，把所有的 $e_1 \cdots e_n$ 组合成一个大的矩阵 $\mathbf{e}$，它们的对应关系如下，

$$\begin{bmatrix} J_1 \\ J_2 \\ \vdots \\ J_n \end{bmatrix} \Delta H = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \Rightarrow$$
$$\mathbf{J}\Delta H = \mathbf{e}$$

使用高斯牛顿法的优化公式，得到，

$$\Delta H = -\left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}^T\mathbf{e}$$

然后，把 $-\left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}^T$ 提前计算好保存起来的。

但是，在程序里面，跟普通的高斯牛顿优化方法不同，在程序里面是对每一项残差 $e_1 \cdots e_n$ 都分别计算。把 $\Delta H = -\left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}^T\mathbf{e}$ 按照每一项残差分离出来，每项单独计算。比如，针对 e_n 的计算过程如下，

$$\begin{aligned} \Delta H_1 &= -\left(\mathbf{J}^T\mathbf{J}\right)^{-1} {J_1}^T e_1 \\ &\vdots \\ \Delta H_n &= -\left(\mathbf{J}^T\mathbf{J}\right)^{-1} {J_n}^T e_n \end{aligned}$$

在程序里面，针对每一项残差，都把其对应的 $-\left(\mathbf{J}^T\mathbf{J}\right)^{-1} {J_n}^T$ 保存起来。最终的 $\Delta H = \Delta H_1 + \Delta H_2 \cdots \Delta H_n$，最终结果是跟普通的高斯牛顿法一样的，只不过是把矩阵拆开来计算了。但是，程序里那样的做法，会导致每个残差都要单独算一次，通过 for 循环来实现，相比普通方法计算量更大，更耗时。也许应该改成正常的高斯牛顿法，用矩阵乘法，一次性计算。

3.跟踪

从 main_BYD.cpp 的 tracker.track(I)进入，跟踪主要的计算过程在 vpTemplateTracker.cpp 的 trackPyr 函数。

每新来一张图像，都用上一张图像算出来的 H 矩阵作为初值，把参考帧上的参考区域往当前帧上映射。首先，映射到当前帧的金字塔的最高层优化。每一层都拿上一层优化出来的 H 矩阵作为初值。然后逐层往下，一直优化到最下面的一层。

但是，在不同的层数，相应的 H 矩阵是不同的。

参考《视觉 SLAM 十四讲》的 7.3.3 的单应矩阵表达式，

$$H = K\left(R - \frac{tn^T}{d}\right)K^{-1}$$

设$M=R-\frac{tn^T}{d}$，则上面的表达式可以转换为，

$$
\begin{aligned}
H&=\begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \\
&=KMK^{-1}=\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}\begin{bmatrix} 1/f_x & 0 & -c_x/f_x \\ 0 & 1/f_y & -c_y/f_y \\ 0 & 0 & 1 \end{bmatrix} \\
&=\begin{bmatrix} (c_xM_{31}+f_xM_{11})/f_x & (c_xM_{32}+f_xM_{12})/f_y & c_xM_{33}+f_xM_{13}-c_x(c_xM_{31}+f_xM_{11})/f_x-c_y(c_xM_{32}+f_xM_{12})/f_y \\ (c_yM_{31}+f_yM_{21})/f_x & (c_yM_{32}+f_yM_{22})/f_y & c_yM_{33}+f_yM_{23}-c_x(c_yM_{31}+f_yM_{21})/f_x-c_y(c_yM_{32}+f_yM_{22})/f_y \\ M_{31}/f_x & M_{32}/f_y & M_{33}-c_xM_{31}/f_x-c_yM_{32}/f_y \end{bmatrix}
\end{aligned}
$$

在高一层的金字塔图像上，K'会变成，

$$
K'==\begin{bmatrix} f_x/2 & 0 & c_x/2 \\ 0 & f_y/2 & c_y/2 \\ 0 & 0 & 1 \end{bmatrix}
$$

所以，高一层的金字塔图像上，对应的单应矩阵H'会变成，

$$
\begin{aligned}
&H'=K'MK'^{-1} \\
&=\begin{bmatrix} (c_xM_{31}+f_xM_{11})/f_x & (c_xM_{32}+f_xM_{12})/f_y & \frac{1}{2}\left(c_xM_{33}+f_xM_{13}-c_x(c_xM_{31}+f_xM_{11})/f_x-c_y(c_xM_{32}+f_xM_{12})/f_y\right) \\ (c_yM_{31}+f_yM_{21})/f_x & (c_yM_{32}+f_yM_{22})/f_y & \frac{1}{2}\left(c_yM_{33}+f_yM_{23}-c_x(c_yM_{31}+f_yM_{21})/f_x-c_y(c_yM_{32}+f_yM_{22})/f_y\right) \\ 2M_{31}/f_x & 2M_{32}/f_y & M_{33}-c_xM_{31}/f_x-c_yM_{32}/f_y \end{bmatrix} \\
&=\begin{bmatrix} H_{11} & H_{12} & \frac{1}{2}H_{13} \\ H_{21} & H_{22} & \frac{1}{2}H_{23} \\ 2H_{31} & 2H_{32} & H_{33} \end{bmatrix}
\end{aligned}
$$

于是，就得到了在不同层的金子塔图像上的对应的单应矩阵H。

然后，按照第 2 部分中的方法，计算每一个投影点的残差。对于每一项残差，都代入$\Delta H_n=-\left(\mathbf{J}^T\mathbf{J}\right)^{-1}J_n{}^Te_n$中，得到最终的$\Delta H=\Delta H_1+\Delta H_2\cdots\Delta H_n$。

在得到ΔH，也就是得到了$\Delta H_{o'o}$，然后把扰动更新在目标量$\hat{H}_{to}$上。

$$
\hat{H}_{to}=H_{to'}=H_{to}H_{oo'}=H_{to}\left(\hat{H}_{o'o}\right)^{-1}=H_{to}\left(H_{o'o}+\Delta H_{o'o}\right)^{-1}=H_{to}\left(I+\Delta H_{o'o}\right)^{-1}
$$

每迭代一次，都更新一次$\hat{H}_{to}$。然后把$\hat{H}_{to}$作为H_{to}再加入到下一次的迭代中。直到迭代次数达到阈值，或者迭代后，平均每个三角形顶点的移动距离，小于阈值。

优化结束，得到最终的$\hat{H}_{to}$。

4.改进方向

visp1.0 版本和 2.0 版本，是要增加抗遮挡的功能:

visp1.0 版本。要读入每帧对应的 mask。能达到这样的效果，就算跟踪时 ROI 被遮挡，程序也不会挂掉。（因为跟踪对 mask 的精度的要求不高，所以为了速度的话，可以把 1920x1080 的图片 resize 成原来面积的四分之一，送给 panet，得到 mask 之后再把 mask 恢复成原来的大小。）

visp2.0 版本，也需要读入 mask。能达到的效果是，在关键帧参考区域 ROI 的时候，不必再避开人，可以一次把整个平面都选出来，就算把人框在 ROI 里面，也不会对跟踪结果有影响。

visp3.0 和 4.0 版本，要做的是，对速度进行提升:

visp3.0 版本是，对特征点法并行化，得到每帧初值，传给 visp 并行优化 H 矩阵。这方法可以用来处理纹理丰富的视频，理论上是只用花 1 帧的时间。同时，为了提升鲁棒性，也许可以把特征点残差和模板法的残差都加到一个损失函数里面，进行联合优化。

visp4.0 版本是，对于纹理不丰富的视频，仍然只能靠 visp 一帧帧跟踪。需要先测试模板跟踪的各个部分的耗时，针对最耗

时的部分进行优化。如果是第 2 部分中指出的优化方法耗时的话，则先改成普通的高斯牛顿法，看看效果。再改成 ceres 优化库 +LM 方法来提速。

参考文献

1.Dame A . Accurate Real-time Tracking Using Mutual Information[C]// IEEE International Symposium on Mixed & Augmented Reality. IEEE, 2010.

Yuanlizheng

2019 年 1 月 1 日