

# SE3TRACKER

## 快速使用

下载依赖文件<https://cloud.momenta.ai/s/lGltB7HKfdADXm2>，解压并替换掉相应的软链接。

### 定位

```
mkdir build  
cd build  
cmake ..  
make -j4  
mkdir result  
./test
```

结果图像会保存在result文件夹

### 生成地图

从点云地图，生成算法依赖的地图数据。

```
cd src  
python se3track.py  
mv *.tiff ../mapdata  
mv map.yaml ../mapdata
```

### 添加噪声测试

对VO输入添加噪声，测试算法鲁棒性。

```
cd build  
for ((i=0; i<30; i++)); do ./testnoise; mv result.png result_${i}.png; done
```

查看生成的result\_{0-29}.png图像。

### 原理

### 地图

对于任意点云地图，可以将其抽象为 $D(x, y), M(x, y)$ 两个函数， $x, y$ 表示地图坐标系的x, y坐标， $D(x, y)$ 返回坐标 $(x, y)$ 所对应的深度信息， $M(x, y)$ 返回 $(x, y)$ 对应的label信息。所以，可以将地图点认为是一个2D流形，可以用 $x, y$ 两个变量描述。

函数 $D(x, y), M(x, y)$ 对 $x, y$ 的偏导数分别用 $D'_x, D'_y, M'_x, M'_y$ 表示。

## 观测

车辆采集的拼接图是对地图的一个观测，地图上的一个点 $X(X = [x, y, D(x, y)]^T)$ 在图像上坐标为 $u$ ，观测方程为：

$$\Pi(H^i(Rc^i(RX + T) + Tc^i)) = u^k, \quad i = 1, 2, 3, 4$$

其中， $\Pi(\cdot)$ 为将3D点到2D的投影函数， $Rc^i, Tc^i, H^i$ 表示第*i*个相机外参的旋转、平移、单应， $R, T$ 表示从地图坐标系到车体坐标系的 $SE(3)$ 变换。

对拼接图做语义分割， $X$ 和 $u$ 应当有相同的label，用 $C(u)$ 表示当前帧中 $u$ 坐标对应的label，有：

$$M(x, y) = C(u)$$

## 优化

在这个系统中，待求解参数有 $R, T, x, y$ ，考虑到观测点的数量很大，为了防止在迭代时的未知数过多导致计算量过大，我们将 $x, y$ 当做 $R, T$ 的隐函数，求解以下优化方程：

$$R, T = \arg \min_{R, T} \sum_{k=1..N} ||M(x^k(R, T), y^k(R, T)) - C(u^k)|| s.t. \quad \Pi(H^i(Rc^i(RX^k(R, T) + T) + Tc^i)) = u^k$$

$i$ 表示第*i*个相机， $k$ 表示第*k*个观测点，共*N*个观测。

通过隐函数 $\Pi(H^i(Rc^i(RX^k(R, T) + T) + Tc^i)) = u^k$ ，可以出 $x, y$ 对 $R, T$ 的偏导数：

$$\begin{aligned} \begin{bmatrix} \frac{\partial x^k}{\partial R} \\ \frac{\partial y^k}{\partial R} \end{bmatrix} &= -(\Pi'^{(k)} H R c^i R \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ D'_x^{(k)} & D'_y^{(k)} \end{bmatrix})^{-1} \Pi'^{(k)} H R c^i X^k \\ \begin{bmatrix} \frac{\partial x^k}{\partial T} \\ \frac{\partial y^k}{\partial T} \end{bmatrix} &= -(\Pi'^{(k)} H R c^i R \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ D'_x^{(k)} & D'_y^{(k)} \end{bmatrix})^{-1} \Pi'^{(k)} H R c^i \end{aligned}$$

上式中需要使用到 $X^k$ ，在方程 $\Pi(H^i(Rc^i(RX^k + T) + Tc^i)) = u^k$ 中，给定 $R, T$ 的情况下，共有两个约束， $x^k, y^k$ 两个自由度，可以作为一个非线性方程对每个点独立求解。

令 $f^k = M(x^k(R, T), y^k(R, T)) - C(u^k)$ ，则 $\frac{\partial f}{\partial R} = M'_x^{(k)} \frac{\partial x^k}{\partial R} + M'_y^{(k)} \frac{\partial y^k}{\partial R}$ ,  $\frac{\partial f}{\partial T} = M'_x^{(k)} \frac{\partial x^k}{\partial T} + M'_y^{(k)} \frac{\partial y^k}{\partial T}$ ，可以构建雅克比矩阵，使用L-M算法求解。

## 实现

在具体实现时有一些细节需要提及。一方面是将点云地图生成我们需要的格式 $D(x, y), M(x, y)$ , 另一方面是快速求解每个点在地图上的坐标 $x^k, y^k$ 。

## 生成地图

地面形成的地图是3D空间中的2D流行，可以用抽象函数 $D(x, y)$ 表示，即通过2个自由度的变量可以得到地图中的3D坐标点。实际实现中，我们简单的将 $x, y$ 取为地图点的 $x, y$ 坐标， $D(x, y)$ 返回 $(x, y)$ 点的 $z$ 值。在目前实现中，简单的将 $D, M$ 描述成两个矩阵，每个 $(x, y)$ 坐标对应矩阵中的一个值。这样选取 $x, y$ 计算较为简单，但是会造成不支持地库跨层。重新定义 $x, y$ ，比如定义为对地图降维（例如使用isomap算法）后的平面坐标，可以解决多层地库的问题。

由于点云地图的稀疏性，生成的矩阵地图 $D$ 会存在很多的无效区域，需要进行修补。在此，使用的是使用 $3 \times 3$ 窗口卷积，窗口中有效点的平均值作为窗口中心点的值。循环50次，将地图拓宽50个像素。

```
for i in range(50):
    m = D!=0
    m = filter2D(m, ones(3,3))
    m[m==0] = 1
    D = filter2D(D, ones(3,3))/m
```

## 反向投影

从当前帧坐标到地图坐标的反向投影，可以看作一个每个点独立的非线性方程，但是直接使用高斯牛顿问题的方法求解计算复杂度较大。这里采用引入变量 $s^k$ ，将 $\Pi(H^i(Rc^i(RX^k + T) + Tc^i)) = u^k$ 转变为线性方程：

$$H^i(Rc^i(RX^k + T) + Tc^i) = s^k \begin{bmatrix} u^k \\ 1 \end{bmatrix}$$

化简得：

$$X^k + \underbrace{R^T T + T^T R c^{iT} T c^i}_{\hat{T}} = s^k \underbrace{(R^T R c^{iT} H^{-1} \begin{bmatrix} u^k \\ 1 \end{bmatrix})}_{\hat{u}^k}$$

在车辆的body系，地面点的 $z$ 轴应该接近0， $(RX^k + T)^z = 0 = (s^k R \hat{u} - R \hat{T} + T)^z$ ，上标 $x, y, z$ 分别表示xyz方向上的分量，可以得到 $s^k$ 的初值。在给定 $s^k$ 的情况下， $x^k = (s^k \hat{u}^k - \hat{T})^x, y^k = (s^k \hat{u}^k - \hat{T})^y, X^k = [x^k, y^k, D^k]^T$ ， $D^k$ 表示 $D(x^k, y^k)$ ，对 $D(x, y)$ 做一阶近似，有：

$$\underbrace{\begin{bmatrix} 1 & 0 & \hat{T}^x \\ 0 & 1 & \hat{T}^y \\ D_x^k & D_y^k & D^k - D_x^k x^k - D_y^k y^k + \hat{T}^z \end{bmatrix}}_{A^k} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = s^k \hat{u}^k$$

更新 $s^k = \frac{1}{((A^k)^{-1} \hat{u})^z}$

迭代 $s^k$ 至收敛， $u^k$ 对应的地图点坐标为 $s^k\hat{u}^k - \hat{T}$ 。

## 测试

---

### 评价标准

使用将当前帧投影到地图上，和地图的重合程度作为评估标准。目前还是通过人眼来做判断，是否当前帧投影到了合适的位置。

### 功能测试

在苏州兆润数据集，从176帧跑到1999帧，其中包括进入地库的下坡，并在地库b1层跑了两圈。

在整个过程中，每一帧在地图上的投影，都和地图有较大的重合，跟踪良好。

在时间上，平均时间7.79ms，最大时间23.25ms。

### 鲁棒性测试

在本系统中，使用了VO提供两帧间的差值( $\Delta pose$ )，在每次计算前，都在上一帧的结果上加上 $\Delta pose$ 作为初值。考虑到VO可能存在的误差，这里对每一帧的 $\Delta pose$ 加了一个随机扰动。对 $\Delta pose$ 的轴角扰动在±5°内，平移的扰动在±10cm内。

共测试了30次，每次连续跟踪1000帧，使用最后一帧到地图投影的重合程度作为评价标准，共有3次失败。认为在VO存在误差时候，有90%的概率成功跟踪。