rf2o 流程梳理

思路借鉴光流跟踪, 以下大概介绍流程

对于某个具体激光点P, 在极坐标系下表示为(r,theta), 激光距离表示为R(t,α), t是时间, α∈[0,N), N是一帧有多少个激光点. α的计算方法:

$$\alpha = \frac{N-1}{FOV}\,\theta + \frac{N-1}{2} = k_\alpha \theta + \frac{N-1}{2} \qquad (1)$$

其中FOV是激光量程, 单位(弧度).

假设R(t,α)是可微的, 将R(t,α)泰勒展开:

$$R(t+\Delta t, \alpha+\Delta\alpha) = R(t,\alpha) + \frac{\partial R}{\partial t}(t,\alpha)\,\Delta t + \frac{\partial R}{\partial \alpha}(t,\alpha)\,\Delta\alpha + O(\Delta t^2, \Delta\alpha^2) \qquad (2)$$

其中Δt 是两帧之间的时间差,$\Delta\alpha$ 代表坐标变化. 忽略高阶项

$$\frac{\Delta R}{\Delta t} \simeq R_t + R_\alpha \frac{\Delta\alpha}{\Delta t} \qquad (3)$$

令

$$\Delta R = R(t+\Delta t, \alpha+\Delta\alpha) - R(t,\alpha),$$
$$R_t = \frac{\partial R}{\partial t}(t,\alpha), \quad R_\alpha = \frac{\partial R}{\partial \alpha}(t,\alpha).$$

再令$\dot{r} = \Delta R/\Delta t$ 代表距离坐标系下的速度, $\dot{\alpha} = \Delta\alpha/\Delta t$ 代表scan坐标系下的速度, 得到

$$\dot{r} \simeq R_t + R_\alpha \dot{\alpha} = R_t + R_\alpha k_\alpha \, \dot{\theta} \qquad (4)$$

为了将所有点的速度统一到同一个坐标系, 把距离方位速度$(\dot{r}, \dot{\theta})$ 转换到笛卡尔坐标系表示为$(\dot{x}, \dot{y})$,

$$\dot{r} = \dot{x} \cos\theta + \dot{y} \sin\theta \qquad (5)$$

$$r \, \dot{\theta} = \dot{y} \cos\theta - \dot{x} \sin\theta \qquad (6)$$

将所有激光点打到的地方作为一个刚体, 这个刚体的运动和激光本体的运动只是符号相反, 激光点的速度和激光本体的速度转换为:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -v_{x,s} + y \, \omega_s \\ -v_{y,s} - x \, \omega_s \end{pmatrix} \qquad (7)$$

这里令$\zeta_s = (v_{x,s}, v_{y,s}, w_s)$ 是激光雷达的速度, 然后(x, y) 是激光点P的笛卡尔坐标. 把速度(5)(6)带入到(4)中, 然后加上刚体假设(7), 我们就能够把距离流约束转换成激光雷达速度的约束:

$$\left(\cos\theta + \frac{R_\alpha k_\alpha \sin\theta}{r} \right) v_{x,s} + \left(\sin\theta - \frac{R_\alpha k_\alpha \cos\theta}{r} \right) v_{y,s} + \left(x \sin\theta - y \cos\theta - R_\alpha k_\alpha \right) \omega_s + R_t = 0 \qquad (8)$$

这样的话, 我们有三个未知数,只需要三个结果点就能解出来.

VELOCITY E STIMATION

然后实际上(8)由于激光测距误差 各种动态障碍物等的影响不可能等于0. 那么对于一个速度ζ, 引入残差$\rho(\zeta)$ 来衡量(8)中的的约束.

$$\rho(\boldsymbol{\xi}) = R_t + (x\sin\theta - y\cos\theta - R_\alpha k_\alpha)\,\omega \qquad (9)$$
$$+ \left(\cos\theta + \frac{R_\alpha k_\alpha \sin\theta}{r}\right) v_x + \left(\sin\theta - \frac{R_\alpha k_\alpha \cos\theta}{r}\right) v_y$$

然后在引入一个鲁棒核函数F.

$$\boldsymbol{\xi}_M = \arg\min_{\boldsymbol{\xi}} \sum_{i=1}^{N} F(\rho_i(\boldsymbol{\xi})) \qquad (10)$$

$$F(\rho) = \frac{k^2}{2}\, ln\left(1 + \left(\frac{\rho}{k}\right)^2\right) \qquad (11)$$

F是一个柯西M, k是可调参数.相比与L1和L2,能够更好处理外点.

优化问题采用Iteratively Reweighted Least Squares (IRLS)来求解, 跟柯西核相关的权重表示为:

$$w(\rho) = \frac{1}{1 + \left(\frac{\rho}{k}\right)^2} \qquad (12)$$

A. Pre-weighting strategy

由于(7)中的刚体假设不满足或者(3)中的线性化误差, 虽然柯西核能处理一部分外点,但处理不了全部,又考虑到动态障碍物. 所以一方面我们用柯西核去降低权重,另一方面(3)中的线性化误差可以提前算出来. 引入pre-weighting过程,为了定量(2)中的误差, 展开到二阶

$$\begin{aligned} \dot{r} &= R_t + R_\alpha \dot{\alpha} + R_{2o}(\Delta t, \dot{\alpha}) + O(\Delta t^2, \dot{\alpha}) \\ R_{2o}(\Delta t, \dot{\alpha}) &= \frac{\Delta t}{2}\left(R_{tt} + R_{t\alpha}\dot{\alpha} + R_{\alpha\alpha}\dot{\alpha}^2\right) \end{aligned} \qquad (13)$$

每个点的pre-weighting函数如下:

$$\bar{w} = \frac{1}{\sqrt{\epsilon + {R_\alpha}^2 + \Delta t^2 {R_t}^2 + K_d \left({R_{\alpha\alpha}}^2 + \Delta t^2 {R_{t\alpha}}^2\right)}} \tag{14}$$

其中K_d 调节一次和二次的重要性, ε 是个小量, 放置分母为0.

所以一开始我们就可以算出所有点的加权残差, 能够加快收敛

$$\rho_i^w(\boldsymbol{\xi}) = \bar{w}_i \ \rho_i(\boldsymbol{\xi}) \qquad\qquad i \in \{1, 2...N\} \tag{15}$$

COARSE - TO -FINE SCHEME AND SCAN WARPING

这部分就是每帧激光都会构造成一个高斯金字塔, 但是降采样用的不是用高斯核,而是一个双边滤波, 保留边界信息.

然后这儿的warping指的是, 金字塔从高到低, 每一层都会根据上一层得到的旋转偏移,把激光旋转平移一下

$$\begin{pmatrix} x^w \\ y^w \\ 1 \end{pmatrix} = e^{\hat{\boldsymbol{\xi}}_p} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \ \hat{\boldsymbol{\xi}}_p = \Delta t \begin{pmatrix} 0 & -\omega_p & v_{x,p} \\ \omega_p & 0 & v_{y,p} \\ 0 & 0 & 0 \end{pmatrix} \tag{16}$$

$$R_1^w(\alpha^w) = \sqrt{(x^w)^2 + (y^w)^2}, \tag{17}$$

$$\alpha^w = k_\alpha \arctan\left(\frac{y^w}{x^w}\right) + \frac{N-1}{2} \tag{18}$$

多个点可能warp到同一个α^w , 这时只保留最近的一个

IMPLEMENTATION

主要包括两个方面, 一个是α的梯度计算, 为了处理两帧间的边界等地方,所以是两个方向间取的加权平均

$$
\begin{aligned}
R_\alpha(\alpha) &= \frac{d(\alpha+1)\,R_\alpha^-(\alpha) + d(\alpha)\,R_\alpha^+(\alpha)}{d(\alpha+1)+d(\alpha)} \qquad (19)\\
R_\alpha^- &= R(\alpha) - R(\alpha-1),\; R_\alpha^+ = R(\alpha+1) - R(\alpha)\\
d(\alpha) &= \|((x(\alpha)-x(\alpha-1), y(\alpha)-y(\alpha-1))\|
\end{aligned}
$$

第二个方面就是速度平滑,也是考虑历史速度的低通滤波. 在IRLS的协方差的特征向量空间中滤波.

$$
[(1+k_l)I + k_e E]\,\boldsymbol{\xi}^t = \boldsymbol{\xi}_M^t + (k_l I + k_e E)\boldsymbol{\xi}^{t-1} \qquad (20)
$$

其中E是特征值对角阵, Kl Ke是控制参数. I是金字塔层数.

$$
k_l = 0.05e^{-(l-1)}, \quad k_e = 15\times 10^3 e^{-(l-1)} \qquad (21)
$$

代码

变量

```
// Internal Data
//下面一堆vector就是激光金字塔, 总共5层
  std::vector<Eigen::MatrixXf> range_; // 当前帧的所有激光点距离
  std::vector<Eigen::MatrixXf> range_old_; //上一帧的距离
  std::vector<Eigen::MatrixXf> range_inter_; // 上一帧和当前帧的均值
  std::vector<Eigen::MatrixXf> range_warped_; // 每一层warp之后的距离
  std::vector<Eigen::MatrixXf> xx_;  // 笛卡尔距离
  std::vector<Eigen::MatrixXf> xx_inter_;
  std::vector<Eigen::MatrixXf> xx_old_;
  std::vector<Eigen::MatrixXf> xx_warped_;
  std::vector<Eigen::MatrixXf> yy_;
  std::vector<Eigen::MatrixXf> yy_inter_;
  std::vector<Eigen::MatrixXf> yy_old_;
  std::vector<Eigen::MatrixXf> yy_warped_;
  std::vector<Eigen::MatrixXf> transformations_; // 每一层相对与上一层的变换

  Eigen::MatrixXf range_wf_;  // 当前帧所有点距离
  Eigen::MatrixXf dtita_;  //对应公式中的(19)
  Eigen::MatrixXf dt_;     // 对应(3)中的 Rt
  Eigen::MatrixXf rtita_;  // 对应(19)中的 d(α)
  Eigen::MatrixXf normx_, normy_, norm_ang_; 没用
  Eigen::MatrixXf weights_; //对应(14)
  Eigen::MatrixXi null_;   // 每个点的距离是否为null
```

```
  Eigen::MatrixXf A_,Aw_;  // 对应(8), 建立Ax=B, Aw是IRLS每次迭代
  Eigen::MatrixXf B_,Bw_;

  MatrixS31 Var_; //3 unknowns: vx, vy, w
  IncrementCov cov_odo_;  // 每层进行IRLS后的协方差

  //std::string LaserVarName;       //Name of the topic containing the scan lasers \laser_scan
  float fps_;               //In Hz
  float fovh_;                //Horizontal FOV
  unsigned int cols_;  //总共多少点
  unsigned int cols_i_; // 当前层多少点
  unsigned int width_; // == cols
  unsigned int ctf_levels_; //多少层
  unsigned int image_level_, level_; // image_level=ctf_levels_-level_
  unsigned int num_valid_range_; // 有效点数
  unsigned int iter_irls_; // 迭代次数
  float g_mask_[5];

  double lin_speed_, ang_speed_; // 最终速度

  ros::WallDuration m_runtime_;
  ros::Time last_odom_time_, current_scan_time_;

  // 一段变换
  MatrixS31 kai_abs_;
  MatrixS31 kai_loc_;
  MatrixS31 kai_loc_old_;
  MatrixS31 kai_loc_level_;
```

主流程

```
bool CLaserOdometry2D::odometryCalculation(const sensor_msgs::LaserScan& scan)
{
  //============================================================================
  //            DIFERENTIAL  ODOMETRY  MULTILEVEL
  //============================================================================

  //copy laser scan to internal variable
  range_wf_ = Eigen::Map<const Eigen::MatrixXf>(scan.ranges.data(), width_, 1);

  ros::WallTime start = ros::WallTime::now();

  createImagePyramid(); // 创建激光金字塔

  //Coarse-to-fine scheme
  for (unsigned int i=0; i<ctf_levels_; i++)
  {
    //Previous computations
    transformations_[i].setIdentity();

    level_ = i;
    unsigned int s = std::pow(2.f,int(ctf_levels_-(i+1)));
    cols_i_ = std::ceil(float(cols_)/float(s));
    image_level_ = ctf_levels_ - i + std::round(std::log2(std::round(float(width_)/float(cols_)))) - 1;

    //1. Perform warping
```

```
    if (i == 0) // 最高层就不用warp
    {
      range_warped_[image_level_] = range_[image_level_];
      xx_warped_[image_level_]    = xx_[image_level_];
      yy_warped_[image_level_]    = yy_[image_level_];
    }
    else
      performWarping(); // warping

    //2. Calculate inter coords
    calculateCoord(); // 计算 xxx_iter 相关, 两帧平均

    //3. Find null points
    findNullPoints(); // 标记null

    //4. Compute derivatives
    calculaterangeDerivativesSurface(); //计算 Rt Rα  d(α)

    //5. Compute normals
    //computeNormals();

    //6. Compute weights
    computeWeights(); // 对应公式(14)

    //7. Solve odometry
    if (num_valid_range_ > 3)
    {
      solveSystemNonLinear(); // IRLS迭代求解
      //solveSystemOneLevel();    //without robust-function
    }
    else
    {
      /// @todo At initialization something
      /// isn't properly initialized so that
      /// uninitialized values get propagated
      /// from 'filterLevelSolution' first call
      /// Throughout the whole execution. Thus
      /// this 'continue' that surprisingly works.
      continue;
    }

    //8. Filter solution
    if (!filterLevelSolution()) return false;
  }

  m_runtime_ = ros::WallTime::now() - start;

  ROS_INFO_COND(verbose_, "[rf2o] execution time (ms): %f",
                m_runtime_.toSec()*double(1000));

  //Update poses
  PoseUpdate(); // 更新odom位姿,计算速度

  return true;
}
```