ICP

Michael Gao

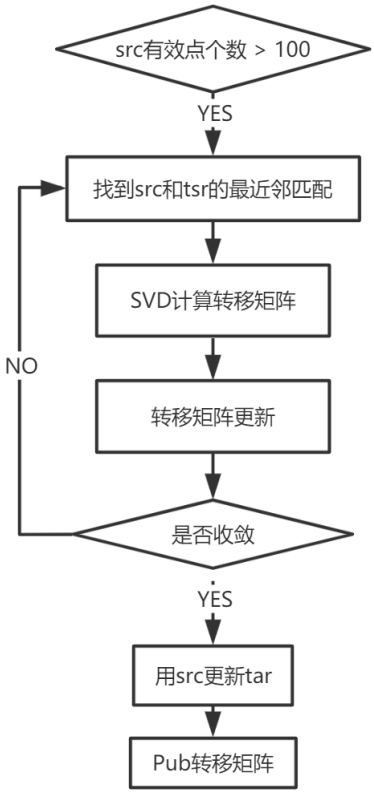
实验目标

- 填补`icp.cpp`，实现点到点ICP方法
- 填补`extraction.cpp`和`icp_lm.cpp`，先提取柱体特征点，然后点到点ICP

实验步骤

(1) 点到点ICP

`process()`部分主要LOOP流程：



a. 最近邻匹配 findNearest()

在利用两帧激光点云数据计算两帧之间的转移矩阵之前，需要先建立两帧之间的匹配关系，ICP方法默认：距当前帧任意一点P_c最近的上一帧中的点P_l为P_c的匹配点。

利用`pcl::KdTreeFLANN`库中的kd树完成此函数：

```
NeighBor icp::findNearest(pcl::KdTreeFLANN<pcl::PointXYZ>::Ptr kdtreeLast,
const Eigen::MatrixXd &src)
```

`pcl::KdTreeFLANN<pcl::PointXYZ>::Ptr kdtreeLast`为上一帧所有点建立的kd树；

`Eigen::MatrixXd &src`为当前帧的所有点；

由于pcl库中的kd树是强类型的，故需要先把src点云转换成`pcl::PointXYZ`格式.

```
for(int i = 0; i < src.cols(); i++){
    pcl::PointXYZ tpoint;
    tpoint.x = src(0,i);
    tpoint.y = src(1,i);
    tpoint.z = 0.0;
}
```

调用API进行最近邻搜索：

```
std::vector<int> pointFindIndex;
std::vector<float> pointFindDistance;
int len = 1;
kdtreeLast->nearestKSearch(tpoint, len, pointFindIndex, pointFindDistance);
```

b. SVD计算齐次转移矩阵 getTransform（）

先求得两个点集质心位置：

$$\mathbf{p} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{p}_i, \mathbf{p}' = \frac{1}{n}\sum_{i=1}^{n}\mathbf{p}'_i$$

计算去质心坐标：

$$\mathbf{q}_i = \mathbf{p}_i - \mathbf{p}, \mathbf{q}'_i = \mathbf{p} - \mathbf{p}'_i$$

根据以下优化问题计算旋转矩阵：

$$\mathbf{R}^* = \arg\min_{\mathbf{R}} \frac{1}{2}\sum_{i=1}^{n}\|\mathbf{q}_i - \mathbf{R}\mathbf{q}'_i\|_2^2$$

再根据R计算t：

$$\mathbf{t}^* = \mathbf{p} - \mathbf{R}^*\mathbf{p}'$$

由数学推导可得：

定义矩阵W：

$$\mathbf{W} = \sum_{i=1}^{n}\mathbf{q}_i\mathbf{q}_i^T$$

对W进行SVD分解：

$$\mathbf{W} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

可求得旋转矩阵与位置平移向量：

$$\mathbf{R} = \mathbf{V}\mathbf{U}^\mathbf{T}$$
$$\boldsymbol{t} = \boldsymbol{p} - \boldsymbol{R}\boldsymbol{p}'$$

Eigen库中可直接利用函数进行SVD分解：

```
// SVD on W
Eigen::JacobiSVD<Eigen::Matrix2d> svd(W, Eigen::ComputeFullU |
Eigen::ComputeFullV);
Eigen::Matrix2d U = svd.matrixU();
Eigen::Matrix2d V = svd.matrixV();

Eigen::Matrix2d R_12 = V* (U.transpose());
Eigen::Vector2d T_12 = Eigen::Vector2d(tx_mean, ty_mean) - R_12 *
Eigen::Vector2d(sx_mean, sy_mean);
```

c. 转移矩阵更新

每次迭代需要累计转移矩阵，并将src用累计的转移矩阵变换：

```
T_all = R_12 * T_all + T_12;
R_all = R_12 * R_all;
Transform_acc.block(0,0,2,2) = R_all;
Transform_acc.block(0,2,2,1) = T_all;
src_pc = Transform_acc*src_pc_copy;
```

注意:

需要计算的转移矩阵为T_{pose}:

$$Pose_{src} = T_{pose} * Pose_{tar}$$

而点的运动正是车的逆运动：

$$Point_{src} = T_{point} * Point_{tar}$$

$$T_{point} = T_{pose}^{-1}$$

故：

$$T_{pose} * Point_{src} = Point_{tar}$$

所有此处只需计算src下的点云到tar下的点云的转移矩阵，即为小车位姿全局转移矩阵.

d. 改进：使用轮速里程计优化ICP初值

由于纯ICP方法长期漂移过于严重，考虑可能是ICP点匹配关系错误，故利用轮速里程计改进ICP初值，优化ICP方法.

首先需要订阅左右轮速：

```
leftv_sub = n.subscribe("/course_agv/left_wheel_velocity_controller/command", 1, &icp::leftv_callback, this);
rightv_sub = n.subscribe("/course_agv/right_wheel_velocity_controller/command", 1, &icp::rightv_callback, this);

void icp::leftv_callback(const std_msgs::Float64::ConstPtr& msg){
    this->leftv = msg->data;
}
void icp::rightv_callback(const std_msgs::Float64::ConstPtr& msg){
    this->rightv = msg->data;
}
```

根据运动学，在两帧之间迭代计算(x,y)的位移和yaw角的变化：

$$\dot{\mathbf{X}}_I = R(\theta)^{-1}\dot{\mathbf{X}}_R = R(\theta)^{-1}\begin{bmatrix} \frac{r\dot{\varphi}_1}{2} + \frac{r\dot{\varphi}_2}{2} \\ 0 \\ \frac{r\dot{\varphi}_1}{2l} + \frac{-r\dot{\varphi}_2}{2l} \end{bmatrix}$$

```
double vx = (this->leftv + this->rightv)/(2.0*this->rx);
double vw = (this->rightv - this->leftv)/(2.0*this->rw);

this->dx += vx*dt*cos(sensor_sta(2)+this->dtheta);
this->dy += vx*dt*sin(sensor_sta(2)+this->dtheta);
this->dtheta += vw*dt;
```

在`calcinitRT(Eigen::Matrix2d& R_all, Eigen::Vector2d& T_all)`函数中进一步处理，生成ICP初值：

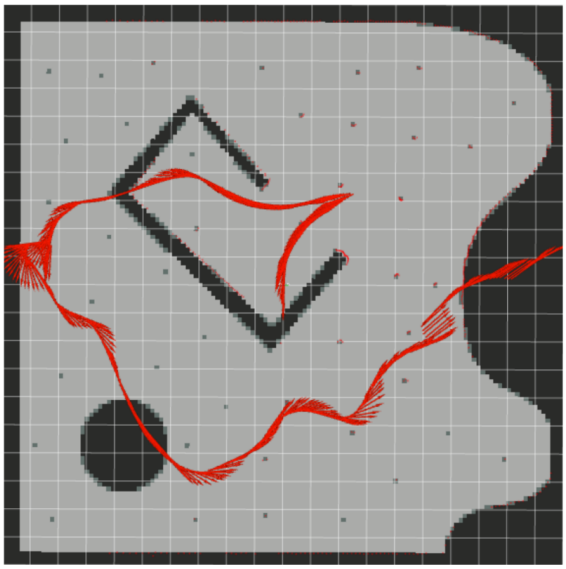
```
// Use the wheel_v odometry to set the init tranform Matrix
calcinitRT(R_all, T_all); //!! YOU CAN COMMENT IT
Transform_acc.block(0,0,2,2) = R_all;
Transform_acc.block(0,2,2,1) = T_all;
src_pc = Transform_acc*src_pc_copy;
```

Result

运行结果如下：

运行`icp.launch`文件

整体路径：(左图为加轮速里程计前，右图为加轮速里程计后)




可以看到，加轮速里程计后改进效果明显.

根据TF坐标分析，加轮速里程计前终点位置的绝对误差为：

$$\Delta x + \Delta y = 4.94m$$

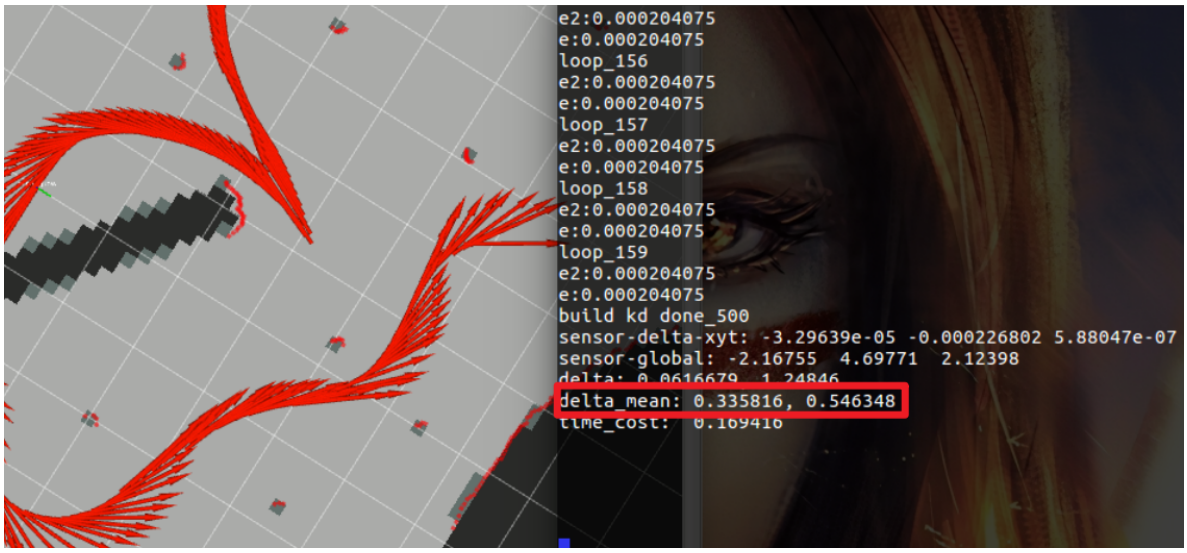
加轮速里程计后终点位置的绝对误差为：

$$\Delta x + \Delta y = 1.81m$$

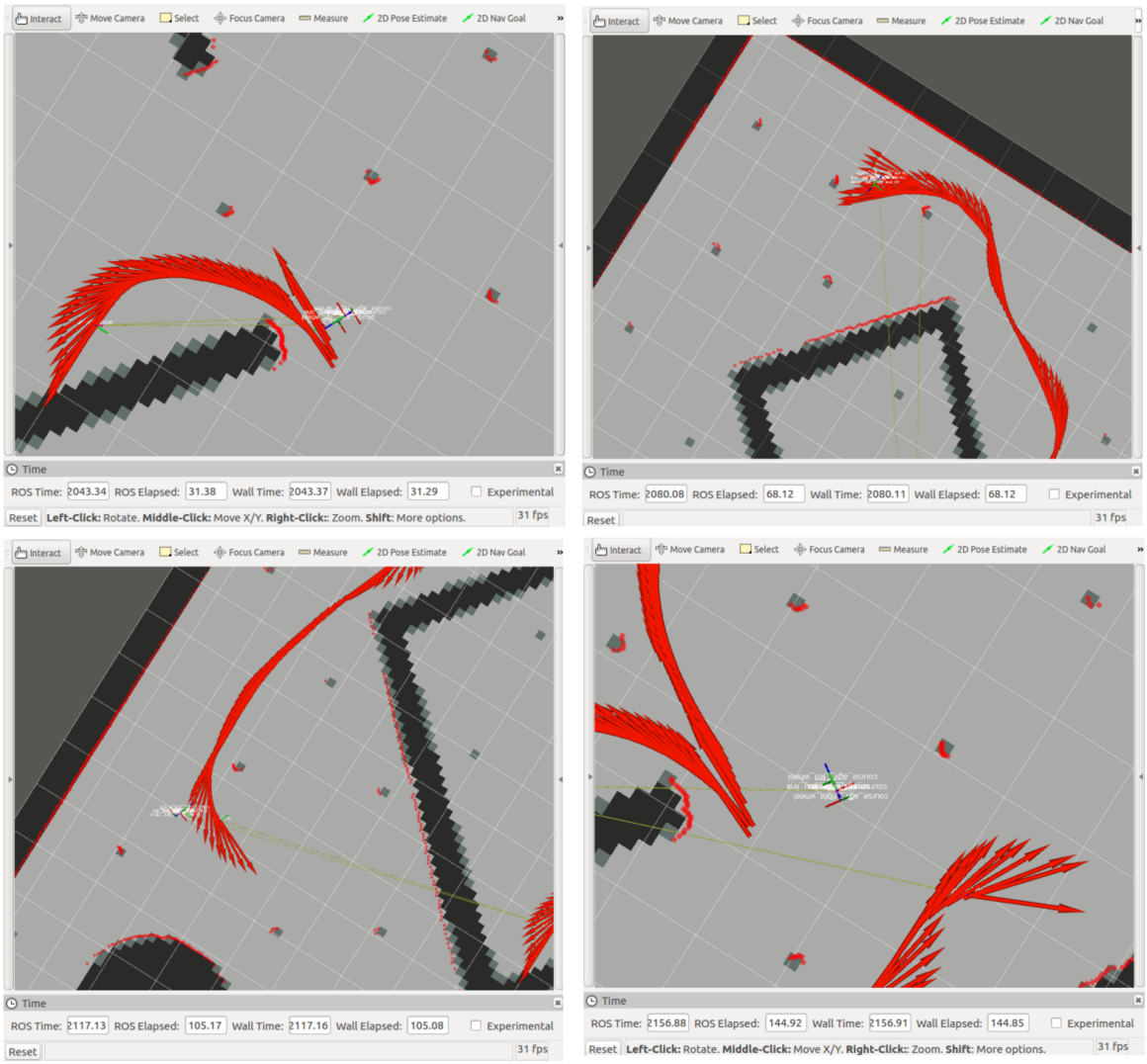
但中间过程的实际误差存在大于终点位置误差的情况.

通过`tf::listener`获取真值计算差值得到，[轮速里程计+ICP_Odom]全程平均误差：

$$|\bar{dx}| = 0.3358$$
$$|\bar{dy}| = 0.5463$$



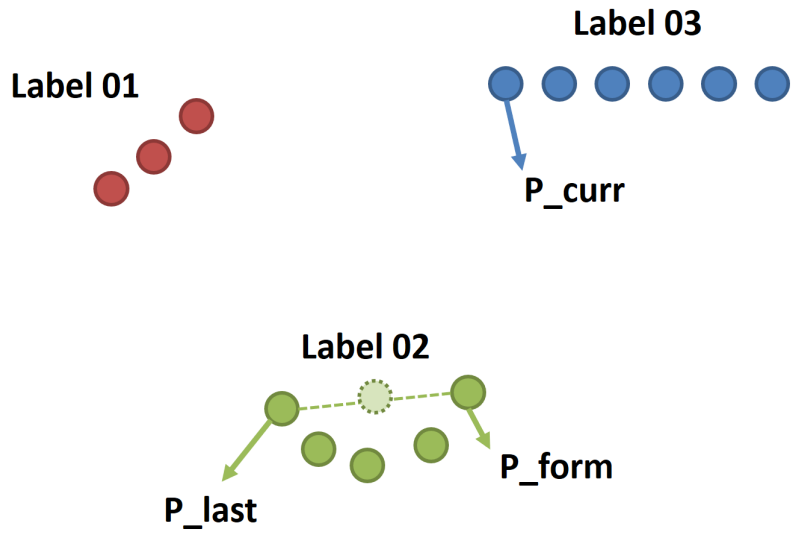
加轮速里程计后ICP导航中间过程：



(2) 柱体KeyPoints提取

首先将所有点云分成小簇，若此点和上一点距离超过阈值，则此点开启新的一簇.

每一族顺序分配一个Label，在遍历每个Label，可提取出当前簇的起始点，进而提取出上一簇的起始点P_{last}和P_{form}:



令:

$$dis = Distance(P_{last}, P_{form})$$

若:

$$dis < RadiusMaxTh$$

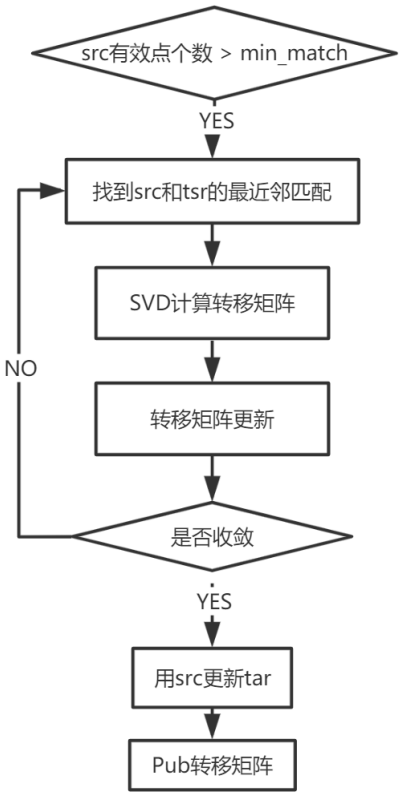
且簇中点数大于`landMark_min_pt`

则判定该簇为圆柱特征，且起始点与终止点中点为圆柱特征坐标.

```
for(int i=0; i<total_num; i++)
{
    if(input.intensities[i] == exp_label){

        cout<<"label:"<<input.intensities[i]-1 <<endl;
        angle = input.angle_min + i * input.angle_increment;
        curr_xy << input.ranges[i] * std::cos(angle), input.ranges[i] *
std::sin(angle);

        if(firstlast){
            last_xy = curr_xy;
            firstlast = false;
            exp_label++;
            continue;
        }
        angle = input.angle_min + (i-1) * input.angle_increment;
        form_xy << input.ranges[i-1] * std::cos(angle), input.ranges[i-1] *
std::sin(angle);
        dis = calc_dist(last_xy, form_xy);
        cout<<"dis"<<dis<<endl;

        if(dis < radius_max_th && len >= landMark_min_pt){
            landMarks.id.push_back(cnt);
            cnt++;

            landMarks.position_x.push_back((last_xy(0)+form_xy(0))/2);
            landMarks.position_y.push_back((last_xy(1)+form_xy(1))/2);
        }

        last_xy = curr_xy;
        exp_label++;
        len = 1;
    }
    else{
        len++;
    }
}
```

(3) LandMark ICP

LandMark ICP 主要LOOP流程如下:



与ICP主要区别为src和tar换成了圆柱特征的坐标.

a. 最近邻匹配 findNearest()

同ICP一样，先建立两帧之间的匹配关系。此处因为是对圆柱特征建立匹配关系，特征量较少为10个左右，故不再采用kd树，直接遍历查找最近点：

```
for(int i = 0; i < src_pos.cols(); i++){
    temp_pos = tar_pos;
    temp_pos.colwise() -= src_pos.col(i);
    distance = temp_pos.array().square().colwise().sum();
    minNum = distance.minCoeff(&minInd);

    if(minNum >= dis_th)
    continue;
    dist.push_back(sqrt(minNum));
    src_ind.push_back(i);
    tar_ind.push_back(minInd);
}
```

b. SVD计算齐次转移矩阵 getTransform（）

同ICP.

c. 转移矩阵更新

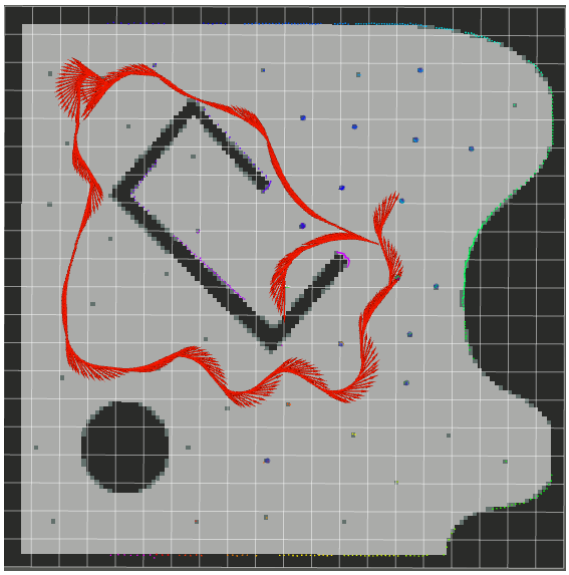
同ICP.

Result

运行结果如下：

运行`icp_all.launch`文件

整体路径：



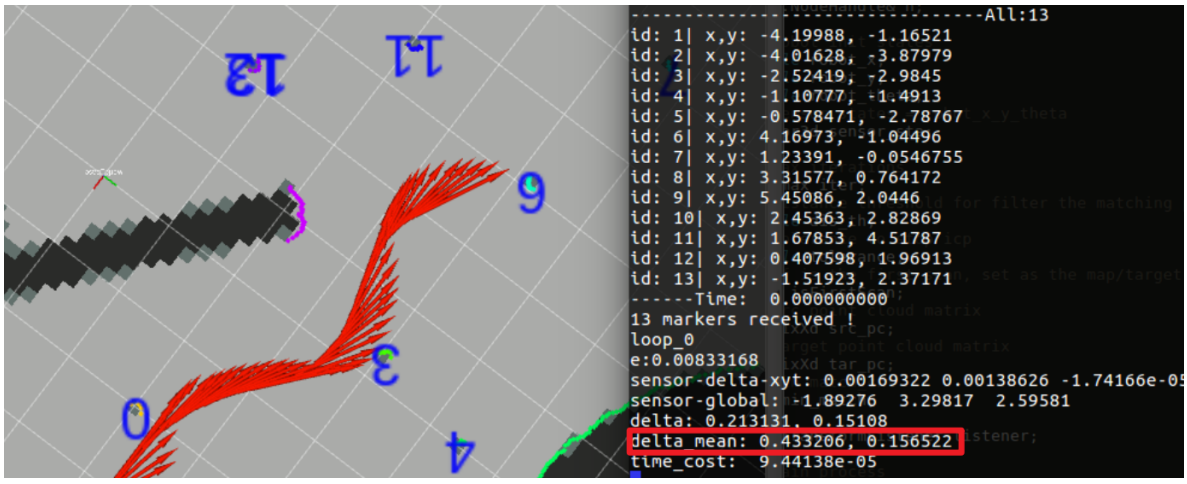
根据TF坐标分析，加轮速里程计前终点位置的绝对误差为：

$$\Delta x + \Delta y = 0.34m$$

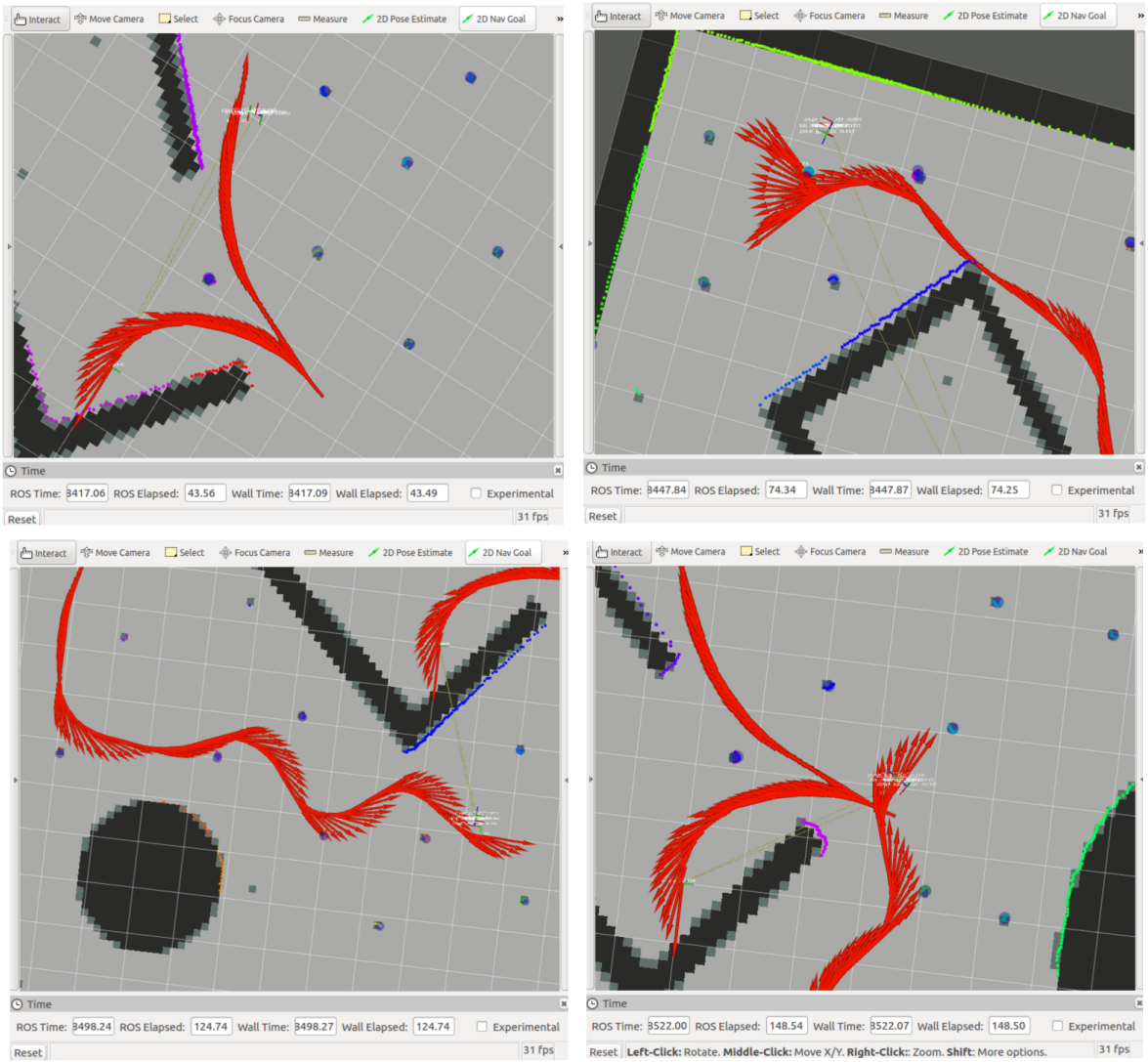
但中间过程的实际误差存在大于终点位置误差的情况.

通过`tf::listener`获取真值计算差值得到，[轮速里程计+ICP_Odom]全程平均误差：

$$|\bar{dx}| = 0.4332$$
$$|\bar{dy}| = 0.1565$$



ICP_LM导航中间过程：



此处rviz使用的配置文件为`course_agv1.rviz`，它选择了新的LaserScan话题进行可视化，同一个Lable显示为同一种颜色.