感知规划集成

Michael Gao

1. 各模块的改进

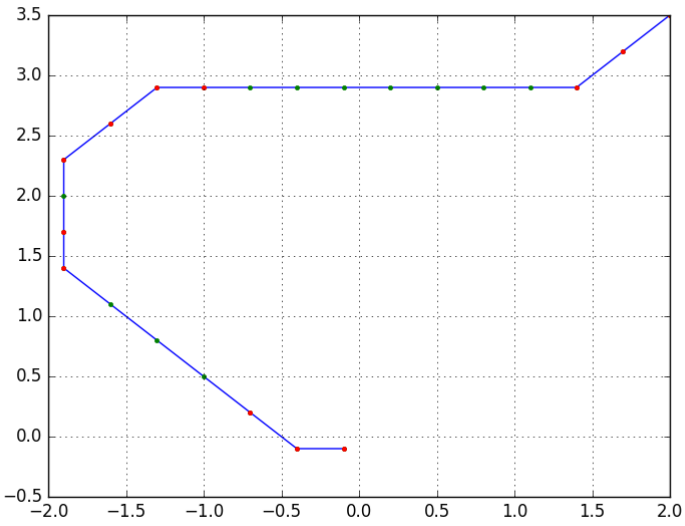
(1) A*加入路径回溯优化

提取关键路径点:

遍历从路径第2个点开始的所有节点，若当前路径点与其上一个路径点及其下一个路径点在同一直线上，则删除这个中间点。

遍历完整条路径后，留下的路径点即为关键路径点 。

为了避免出现由于两个关键点距离过长而导致小车不能快速朝向目标点方向前进，故保留正序关键点后面的一个点，具体见下图：



如图所示，红色为关键路径点，绿色为删去的冗余点。路径呈现上述形状是由于正在通过矩形边界障碍.

代码:

```
def filter_path(self, rx, ry):
    rx.reverse()
    ry.reverse()

    rxn = [rx[0]]
    ryn = [ry[0]]
    cnt = 0
    for i in range(len(rx)-2):
        if math.fabs(math.atan2(ry[i+1]-ry[i],rx[i+1]-rx[i]) -
math.atan2(ry[i+2]-ry[i+1],rx[i+2]-rx[i+1])) < 1e-3:
            cnt += 1
            if cnt == 1:
                rxn.append(rx[i+1])
                ryn.append(ry[i+1])
        else:
            cnt = 0
```

```
            rxn.append(rx[i+1])
            ryn.append(ry[i+1])

    rxn.append(rx[-1])
    ryn.append(ry[-1])

    rxn.reverse()
    ryn.reverse()

    return rxn, ryn
```

(2) A*加入权重

在A*算法中，对路径Cost加入权重：

$$F(n) = g_w * G(n) + h_w * H(n)$$

其中，G是已经走过路径cost，H是heuristic-cost.

加入权重后，若$g_w = 1, h = 0$则等价于Dijkstra算法，可找到最优路径；若$g_w = 0, h = 1$则等价于贪心算法，可加快寻找路径速度；调整权重可获得较好的寻路结果。

在`AStarPlanner`类中加入两个变量：

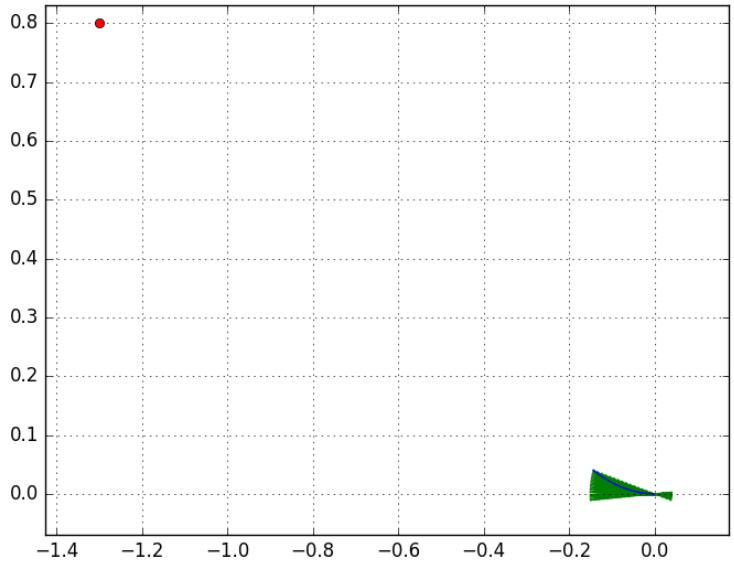
```
self.g_w = 1
self.h_w = 0.5
```

权重部分如下：

```
tnode.f = self.g_w * tnode.g + self.h_w *self.calc_dis(tnode, ngoal)
```

(3) DWA加入与目标距离部分Cost

将选择的速度角速度生成的路径末端与目标点的距离纳入Cost，以得到更优的路径：

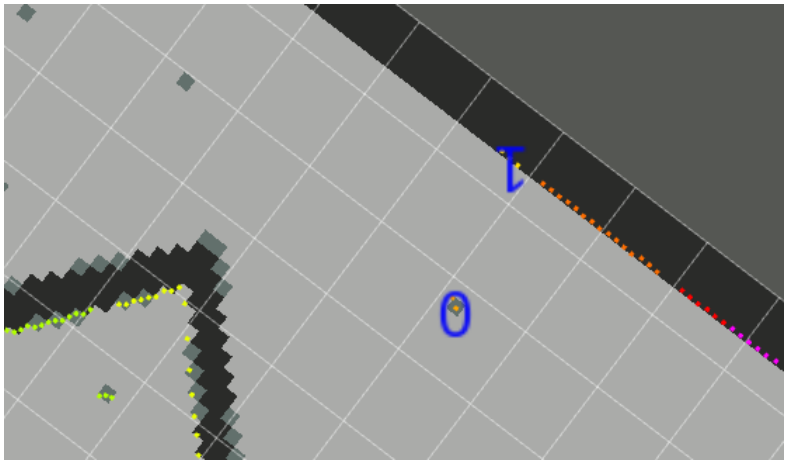


图中红色为目标点，绿色曲线为所有满足DWA约束的可能路径，蓝色曲线为最终选取路径.

```
dx = goal[0] - trajectory[-1,0]
dy = goal[1] - trajectory[-1,1]
goal_dis = math.sqrt(dx*dx+dy*dy)
cost = goal_dis
```

(4) 特征点提取改进-防止误识别

在特征点提取时通常会发生误识别，这种误识别通常是如下图所示的情况：



图中标号为1的特征是因为符合"与上下两帧离群且长度符合"的判断标准，这类误识别通常发生在墙面。且由于CPU版本激光雷达有穿墙问题，使误识别更多。

故在识别出特征之后加入**检查环节**：

若假设当前$label_n$被识别为圆柱特征，那么$label_{n-1}$与$label_{n+1}$大概率在$label_n$的以`label_dr_threshold`为半径的距离以外，此时只需检测$label_{n-2}, label_{n-3}$的最后一个点和$label_{n+2}, label_{n+3}$的第一个一个点是否距离$label_n$过近即可判断是否为此类误识别。

这种误识别判断方法可以排除绝大多数误识别，并且只需检查4个点，效率高。

```
// Check feature
int feature_label = input.intensities[i-1];

for(int prev_2_label = feature_label-3;prev_2_label<=feature_label-2;prev_2_label++){
    if(prev_2_label > 0){
        int j = i;
        while(input.intensities[j]!=prev_2_label){
            j--;
        }
        Vector2d last_point;
        angle = input.angle_min + j * input.angle_increment;
        last_point << input.ranges[j] * std::cos(angle), input.ranges[j] * std::sin(angle);
        double dis_2_pre = calc_dist(last_point, last_xy);
        if(dis_2_pre < label_dr_threshold){
            check_right = false;
            break;
        }
    }
```

```
}

for(int after_2_label = feature_label+3;after_2_label>=feature_label+2;after_2_label--){
    int k = i;
    while(input.intensities[k]!=after_2_label){
        k++;
        if(k==total_num){
            break;
        }
    }
    if(k!=total_num){
        Vector2d after_point;
        angle = input.angle_min + k * input.angle_increment;
        after_point << input.ranges[k] * std::cos(angle), input.ranges[k] * std::sin(angle);
        double dis_2_after = calc_dist(after_point, form_xy);
        if(dis_2_after < label_dr_threshold){
            check_right = false;
            break;
        }
    }
}
```

(5) ICP Odometry加入跳帧操作

由于电脑性能原因，激光雷达数据中间几帧可能出现缺失，而ICP-Odom代码中，若没有达到设定的特征匹配个数将会跳过此帧，用再下一帧特征匹配上一帧。若中间缺失帧数过多且小车运动速度快，则会出现ICP-Odom用多帧后的特征取匹配多帧前特征，且由于特征距离太远匹配失败的情况。最终导致ICP-Odom失效。

为防止这种情况，加入跳帧操作：

- 若此帧特征点数量小于`min_match`，break，且cnt记录一次
- 若此帧与上帧匹配距离在阈值内的特征点数量小于`min_match`，break，且cnt记录一次
- 若此帧成功匹配，特征点数量大于`min_match`，cnt清零
- 若cnt大于跳帧阈值，重新选择当前帧特征作为之后匹配的目标帧

这样一来可以防止ICP_Odom失效，但中间遗失的信息只能依赖EKF update补偿.

```
//Check Matched num
int match_num = src_pcn.cols();
if(i==0){
    cout<<"Matched num:"<<match_num<<endl;
}

if(match_num < min_match){
    skip_num++;
    if(skip_num>=3){
        jump_flag = true;
    }
    return;
}
skip_num = 0;
```

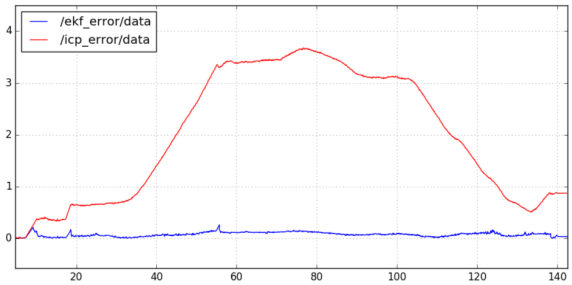
```
if(jump_flag && jump_switch)
{
    tar_pc = this->landMarksToMatrix(input);

    // init the var used in Odometry
    this->timelast = (double)ros::Time::now().toSec();
    this->setzero = true;

    cout<<"JUMP!!!!!!!!!!!!!"<<endl;
    jump_flag =false;
    return;
}
```

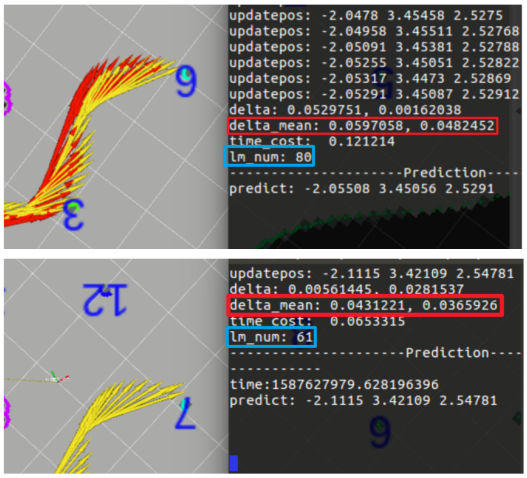
在`icp_lm.launch`文件中加入参数`<param name="jump_switch" value="1"/>`，选择开启或关闭跳帧操作.

(6) EKF误差分析

在对特征点增加误识别检查后，重新用`tf::listener`进行误差分析，平均相对误差减小了26.9%. 同时由于误识别减少，EKF状态中存储的特征数也显著减少，效果得到提升。

同时利用`rosrun rqt_plot rqt_plot`命令画出了ICP和EKF定位随时间的误差曲线，可以发现，EKF效果良好。





2.整体框架整合

(1) 地图修改

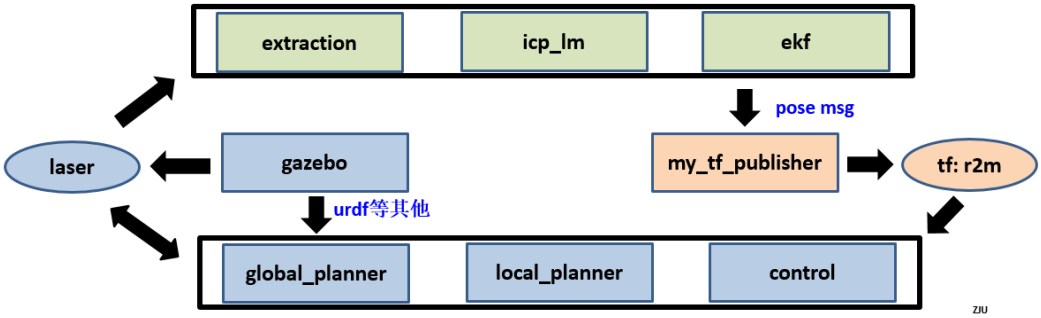
使用有圆柱特征的地图`map_pillar.png`作为框架地图，更改`\course_agv_gazebo\config\map.yaml`和`\course_agv_gazebo\models\ground_plane_for_agv\model.sdf`文件中的相关字段，替换为有圆柱特征的地图`map_pillar.png`.

由于发现GAZEBO中的圆柱特征过细，更改`model.sdf`文件修改地图高度:

```
<heightmap>
    <uri>model://ground_plane_for_agv/map/map_pillar.png</uri>
    <!--<size>20 20 0.4</size-->
    <size>20 20 1.5</size>
    <pos>0 0 0.01</pos>
</heightmap>
```

(2) 使用EKF-TF定位

用EKF的定位信息取代原来的GAZEBO真值：



修改`\course_agv_gazebo\scripts\robot_tf.py`文件，设置2个类变量：

```
self.isFirst = 1
self.isekfmode = 1
```

`isekfmode`可以选择是否开启EKF定位模式，否则小车定位使用GAZEBO真值.

添加EKF-Pub：

首先需要EKF将定位信息Publish，在EKF中添加如下代码：

```
odom_pub_tf = n.advertise<geometry_msgs::Pose>("ekf_odom_tf", 1);
```

在Pub结果时Pub EKF定位坐标：

```
geometry_msgs::Pose odom_tf;
odom_tf.position.x = status(0);
odom_tf.position.y = status(1);
odom_tf.position.z = 0.0;
odom_tf.orientation = odom_quat;

odom_pub_tf.publish(odom_tf);
```

添加EKF坐标回调函数：

在`robot_tf.py`中添加EKF坐标的订阅相关代码：

```
self.ekf_sub = rospy.Subscriber("/ekf_odom_tf", Pose, self.ekfcallback)
```

```python
def ekfcallback(self, data):
    if self.isekfmode:
        self.link_pose = data
        if self.isFirst:
            print("----------GET EKF----------")
            self.isFirst = 0
    else:
        pass
```

此时需要注意函数不能为空，必须写`pass`，否则报错.

初始位置Pub:

如果是第一帧，先Pub一个位于原点的初始位置:

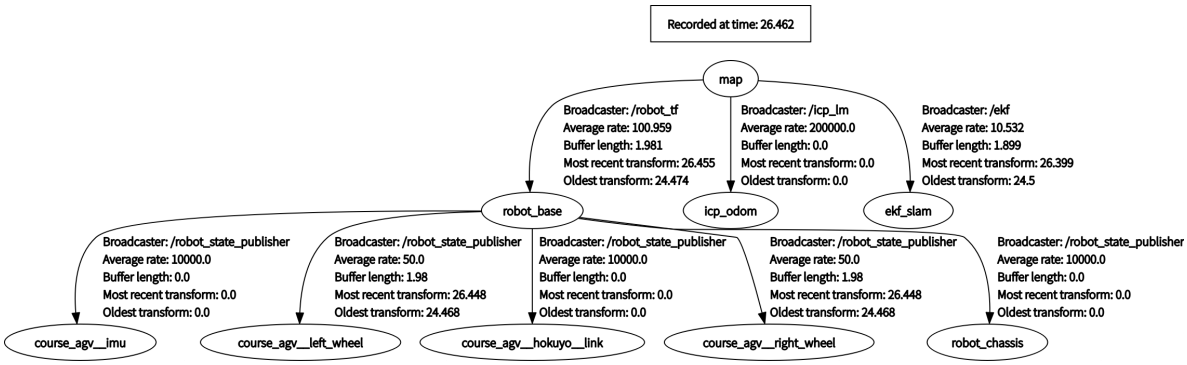
```python
if self.isFirst:
    self.tf_pub.sendTransform((0,0,0),(0,0,0,1),
      rospy.Time.now(),self.link_name,"map")
else:
    self.tf_pub.sendTransform((p.x,p.y,p.z),(o.x,o.y,o.z,o.w),
      rospy.Time.now(),self.link_name,"map")
```

统一基坐标:

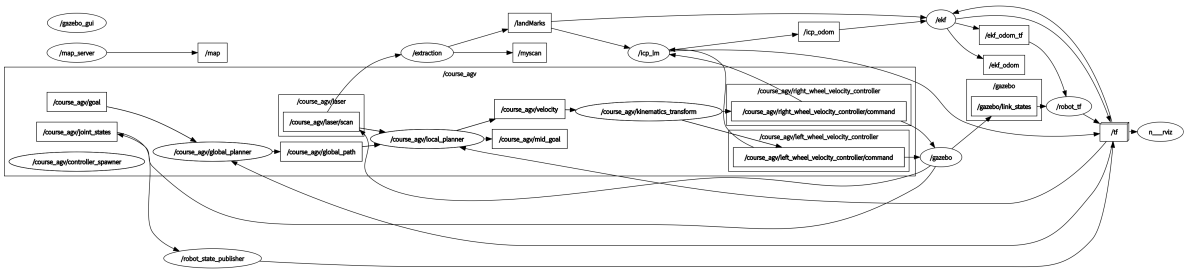
将所有节点的基坐标统一为`map`，修改`ekf.cpp`和`icp_lm.cpp`:

```cpp
geometry_msgs::Quaternion odom_quat =
tf::createQuaternionMsgFromYaw(sensor_sta(2));

geometry_msgs::TransformStamped odom_trans;
odom_trans.header.stamp = the_head.stamp;
odom_trans.header.frame_id = "map";//"world_base";
odom_trans.child_frame_id = "icp_odom";

odom_trans.transform.translation.x = sensor_sta(0);
odom_trans.transform.translation.y = sensor_sta(1);
odom_trans.transform.translation.z = 0.0;
odom_trans.transform.rotation = odom_quat;

odom_broadcaster.sendTransform(odom_trans);
```

统一后的tf树如下:



整体节点图如下：（见附件`rosgraph.png`）



3.结果与案例

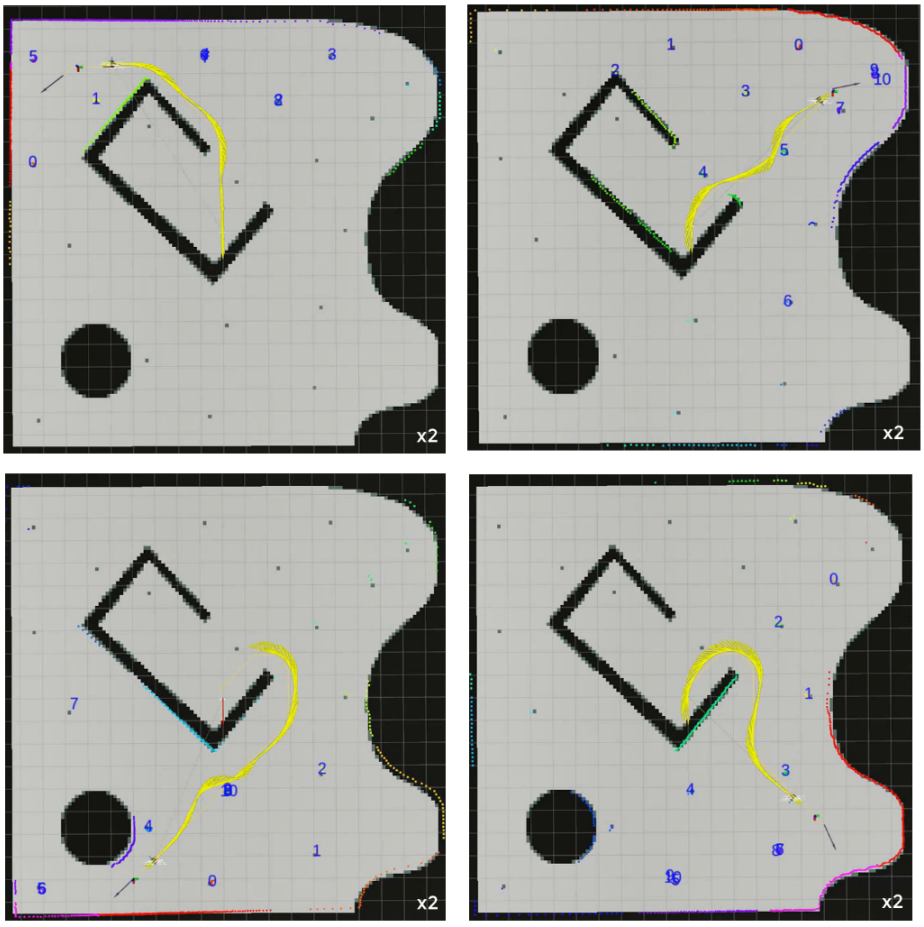
全部案例视频结果见 `vedio\case.mp4`

运行方法：

```
roslaunch course_agv_slam_task icp_all.launch
roslaunch course_agv_slam_task ekf.launch
roslaunch course_agv_nav nav_for_all.launch
```

(1) 无动态障碍物情况

无动态障碍物情况下一共展示4个案例，分别将目标点设置在地图左上、右上、左下、右下角，下图为视频结果截图：



(2) 添加动态障碍物情况

动态障碍物情况下共展示2个案例，分别进行1次动态避障和2次连续动态避障，下图为视频结果截图：