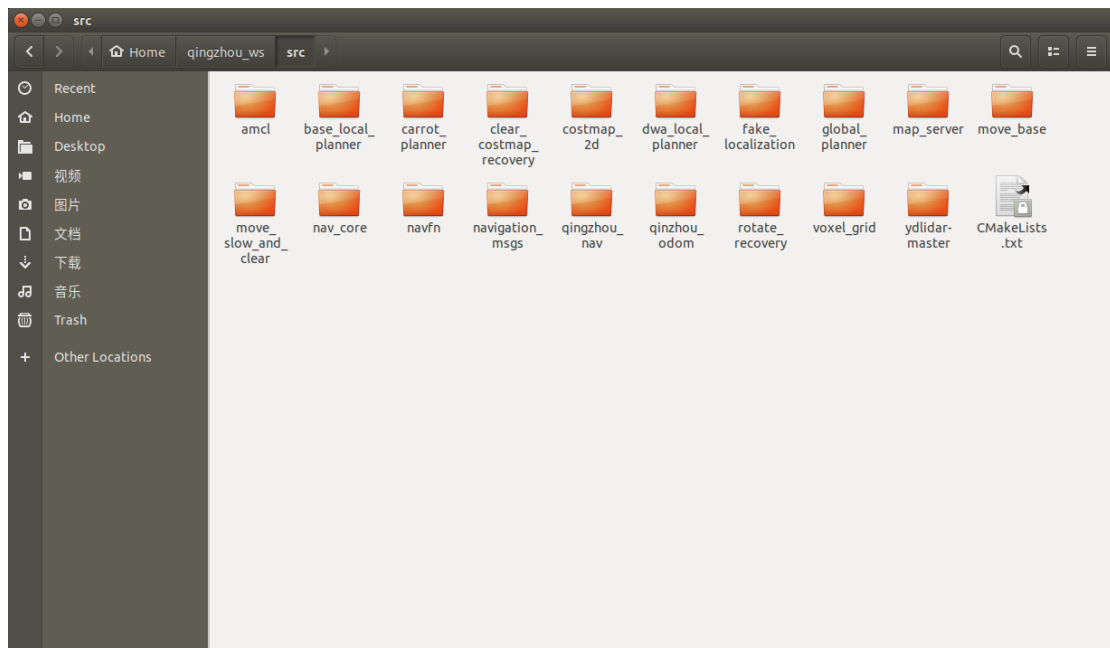


轻舟机器人导航包 qingzhou_ws 的建立及使用

AI 航团队

学习前两节内容后，大家一定对 ros 小车的自动驾驶有了较深的理解，想要迫不及待的在自己的轻舟机器人上边进行实战演习了，接下来将结合我们开源的 qingzhou_ws 导航功能包，带领大家快速的为手上的轻舟机器人赋予智慧的大脑。



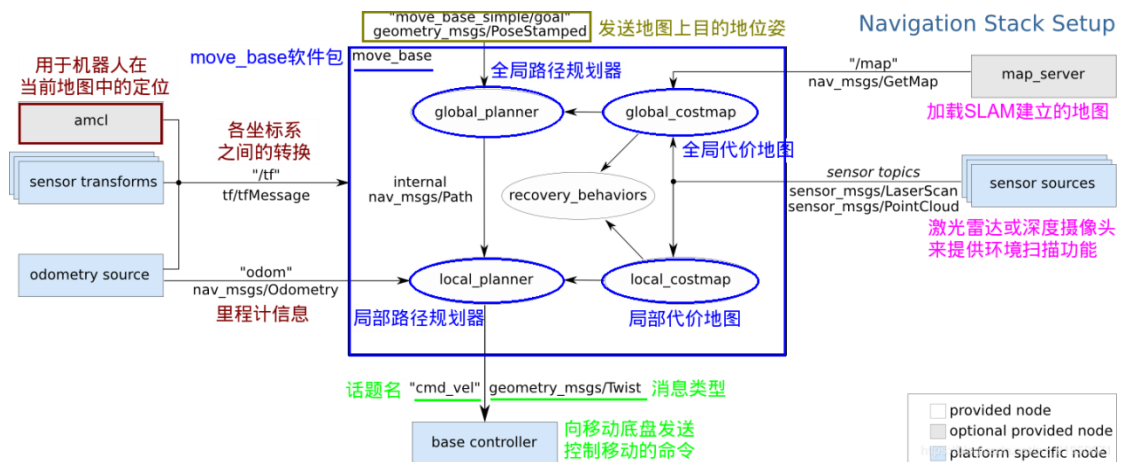
说明：如上图所示，qingzhou_ws 工作空间下，有很多功能包，大部分是 navigation 栈下的开源包，不需要做任何修改。其中的（1）qingzhou_odom 包为我们的自己建立，主要完成数据的通信及解算等；（2）qingzhou_nav 包为我们建立的导航功能的参数配置包，主要完成导航参数的配置、地图的保存、launch 启动文件等；（3）ydlidar-master 包为激光雷达驱动包，提供 sensor_msgs/LaserScan 消息。

1. 轻舟机器人控制系统的架构：

最底层：机器人本身的电机驱动部分，该部分使用的是单片机 stm32 实现的，主要完成：（1）通过串口接收工控机输出的期望速度速度和期望转角，通过分解得到舵机和左右后轮驱动电机的控制量，进而实现 PID 控速。（2）定时采样 IMU 模块值和电机码盘值，通过串口上传给工控机。当然 PID 控速这一部分也可以放到工控机 ROS 端，这样的话，工控机串口输出的是直接的 PWM 值而不是之前的期望速度了。

中间通信层：工控机和底层电机的控制通信，以及将传感器信息发布给 ROS 的通信。这一层主要通过串口收集左右轮速度值，用航迹推演法将左右轮速度转化为机器人的 x 轴方向速度和机器人的旋转速度，然后发布/odom 主题，好让 ROS 的相应 package 收到这个消息，进行机器人位置的估计。同时，这一部分还要关注 ROS 相应部分发出的机器人控制指令，转化为左右轮的期望速度，再通过串口传给 stm32，这一层是自己写程序实现，也是学习 ROS 移动机器人的基础。

决策层：就是与导航有关的了，建立地图和定位，然后用 `move_base` 根据你发布的传感器信息做出路径规划以及机器人的速度和转向控制。这一部分为 ROS 相应的 package 已经完成，我们只需要调用即可。



图中我们可以看到 `move_base` package 的输入和输出。要使得它能运行起来，我们就得构建好这些输入和输出。本节内容涉及中间通信层和决策层内容，最底层默认已经实现。

(1) 必要的输入：

`goal`：期望机器人在地图中的目标位置。

`tf`：各个坐标系之间的转换关系。（具体 `/map frame --> /odom frame`，`/odom frame --> /base_link frame`）

`odom`：根据机器人左右轮速度推算出的航向信息（即 `/odom` 坐标系中机器人 `x,y` 坐标以及航向角 `yaw`）

`LaserScan`：激光传感器的信息，用于定位。

(2) 输出：

`cmd_vel`：在 `cmd_vel` 这个主题上发布 `Twist` 消息，这个消息包含的就是机器人的期望前进速度和转向速度。

至此，我们已经熟悉了 `move_base` 的各种接口，它订阅了什么消息，会发布什么消息都已经清楚了。因此让 `move_base` 控制实际的机器人最主要的就是要解决实际机器人如何发布这些消息给 `move_base`，以及如何接受 `move_base` 发出的消息。

2. 轻舟机器人配置

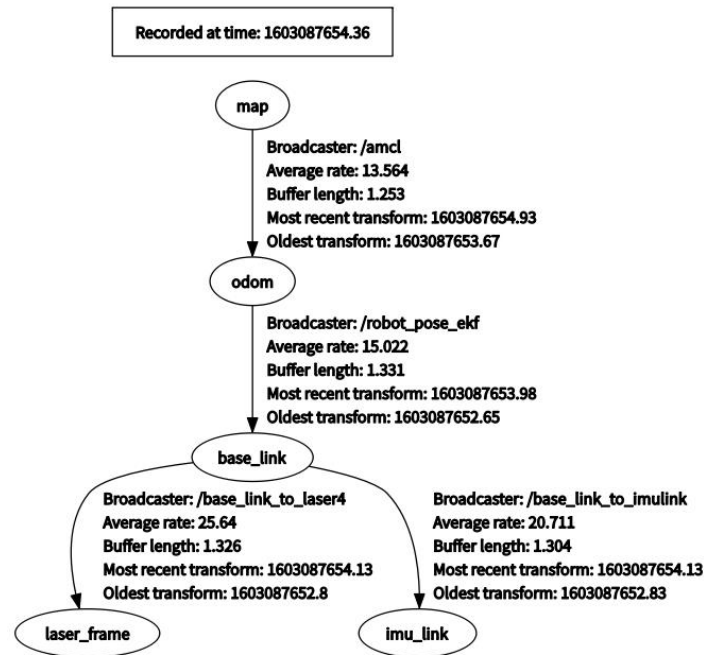
2.1 ROS

首先，请确保你的机器人安装了 ROS 框架。

2.2、tf 变换(sensortransforms)

导航功能包集需要机器人不断使用 `tf` 发布有关坐标系之间的关系的信息。下图为轻舟机器人的 `TF` 变换树，也是一个移动机器人最基本的 `tf` 变换关系，可使用如下指令查看你的 `tf` 变换树。

```
roslaunch rqt_tf_tree rqt_tf_tree
```



轻舟机器人上的主要传感器为 IMU 模块和激光雷达，两个传感器相对于车体位置都是静态的，所以示例程序采用 `static_transform_publisher` 工具的功能是发布两个参考系之间的静态坐标变换，两个参考系一般不发生相对位置变化。

根据以上静态 TF 发布方式，可将轻舟机器人 TF 变换写入 launch 文件中，底盘到激光雷达的 TF 变换如下：

```
<node pkg="tf" type="static_transform_publisher" name="base_link_to_laser4"
  args="0.12 0.0 0.08 0.0 0.0 0.0 /base_link /laser_frame 40" />
```

底盘到 IMU 的 TF 变换如下：

```
<node pkg="tf" type="static_transform_publisher" name="base_link_to_imulink"
  args="0.40 0.0 0.08 0.0 0.0 0.0 /base_link /imu_link 50" />
```

2.3、传感器信息(sensor sources)

导航功能包集使用来自传感器的信息避开现实环境中的障碍物，轻舟机器人采用单线激光雷达，在 ROS 上不断发布 `sensor_msgs/LaserScan` 消息，完成传感器信息的输入提供。`qingzhou_ws` 工作空间中的 `ydliar-master` 包实现了本功能，在启动激光雷达的 `ydliar_node` 节点后，将会发布 `sensor_msgs/LaserScan` 消息。`ydliar.launch` 文件内容如下：

```
ydliar.launch (~/.qingzhou_ws/src/qingzhou_nav/launch) - gedit
ydliar.launch
~/.qingzhou_ws/src/qingzhou_nav/launch

<launch>
  <node name="ydliar_node" pkg="ydliar" type="ydliar_node" output="screen">
    <param name="port" type="string" value="/dev/ydliar"/>
    <param name="baudrate" type="int" value="115200"/>
    <param name="frame_id" type="string" value="laser_frame"/>
    <param name="angle_fixed" type="bool" value="true"/>
    <param name="low_exposure" type="bool" value="false"/>
    <param name="heartbeat" type="bool" value="false"/>
    <param name="resolution_fixed" type="bool" value="true"/>
    <param name="angle_min" type="double" value="-180"/>
    <param name="angle_max" type="double" value="180"/>
    <param name="range_min" type="double" value="0.08"/>
    <param name="range_max" type="double" value="16.0"/>
    <param name="ignore_array" type="string" value="" />
    <param name="samp_rate" type="int" value="9"/>
    <param name="frequency" type="double" value="7"/>
  </node>
  <node pkg="tf" type="static_transform_publisher" name="base_link_to_laser4"
    args="0.12 0.0 0.08 0.0 0.0 0.0 /base_link /laser_frame 40" />
</launch>
```

2.4、里程计信息(odometrysource)

导航功能包要求机器人发布 `nav_msgs/Odometry` 格式的里程计信息，同时也要发布相应的 `tf` 变换 `odom-to-base_link`。里程计包含 2 方面的信息，一方面是位姿（位置 `x,y` 和转角 `yaw`），另一方面是速度（前进速度 `Vx,Vy` 和转向速度 `Vth`）。

里程计信息的获取可以通过编码器获取，也可以通过 IMU 获取，而轻舟机器人采用的是更为准确的将两种数据进行融合，得到更准确的里程计信息，采用 `robot_pose_ekf` 包开启扩展卡尔曼滤波器生成机器人姿态。http://wiki.ros.org/robot_pose_ekf

订阅的话题：

(1) `odom(nav_msgs/Odometry)`（编码器）

用到 topic 中的信息：

`odom->header.stamp` 时间戳
`odom->pose.pose.orientation` 四元数
`odom->pose.pose.position.x` x 位置
`odom->pose.pose.position.y` y 位置
`odom->pose.covariance` 协方差矩阵 6×6 主对角元素

(2) `imu_data (sensor_msgs/Imu)`（IMU）

用到 topic 中的信息：

`imu->header.stamp` 时间戳
`imu->orientation` 四元数
`imu->orientation_covariance` 3×3 四元数的协方差矩阵的主对角线元素

发布的话题：


`robot_pose_ekf/odom_combined`

注意：`/odom` 和 `/robot_pose_ekf/odom_combined` 消息类型不同，前者是 `nav_msgs/Odometry`，后者是 `geometry_msgs/PoseWithCovarianceStamped`。两者的区别：后者的内容是前者的一部分。

提供的 `tf`：

`odom_combined` \rightarrow `base_footprint`

注意：我们要在 `launch` 文件中进行更改，分别改为 `odom` 和 `base_link`。



```

*ekf.launch (~/.qinzhou_ws/src/qinzhou_odom/robot_pose_ekf/launch) - gedit
Open  *ekf.launch
~/.qinzhou_ws/src/qinzhou_odom/robot_pose_ekf/launch

<?xml version="1.0" ?>
<launch>

  <node pkg="robot_pose_ekf" type="robot_pose_ekf" name="robot_pose_ekf">
    <param name="output_frame" value="odom"/>
    <param name="base_footprint_frame" value="base_link"/>
    <param name="freq" value="30.0"/>
    <param name="sensor_timeout" value="1.0"/>
    <param name="odom_used" value="true"/>
    <param name="imu_used" value="true"/>
    <param name="vo_used" value="false"/>
    <param name="gps_used" value="true"/>
    <param name="debug" value="true"/>
    <param name="self_diagnose" value="false"/>
  </node>
  <!--
  <node pkg="tf" type="static_transform_publisher" name="base_link_to_imulink"
    args="0.40 0.0 0.08 0.0 0.0 0.0 /base_link /imu_link 50" />-->
</launch>

```

现在我们已经知道使用 `robot_pose_ekf` 进行数据融合, 我们需要发布 `odom` 和 `imu` 数据。
示例代码如下:

(1) 根据编码器值累计形成里程计程序

```
if(encoderLeft > 220 || encoderLeft < -220) encoderLeft = 0;
if(encoderRight > 220 || encoderRight < -220) encoderRight = 0;
//encoderLeft = -encoderLeft;
encoderRight = -encoderRight;

detEncode = (encoderLeft + encoderRight)/2;           //求编码器平均值
detdistance = detEncode/ticksPerMeter;
detth = (encoderRight - encoderLeft)*2*PI/ticksPer2PI; //计算当前角度 通过标定获得ticksPer2PI

linearSpeed = detdistance/velDeltaTime;
angularSpeed = detth/velDeltaTime;

if(detdistance != 0){
    x += detdistance * cos(th);           //x坐标
    y += detdistance * sin(th);           //y坐标
}
if(detth != 0){
    th += detth;                           //总角度
}
```

其中里程计的标定参数如下:

需要调整的是两个量是 (其他默认, 不需要调整):

`<param name="ticksPerMeter" value = "2363" type = "int"/>` `<!-- 每移动一米需要读取多少个脉冲值 -->`

`<param name="ticksPer2PI" value = "4477" type = "int"/>` `<!-- 每移动360度需要读取多少个脉冲值 -->`

(2) 发布 odom

```
nav_msgs::Odometry odom;           //创建nav_msgs:
odom.header.stamp = current_time;
odom.header.frame_id = "odom";
odom.child_frame_id = "base_link";

//set the position
odom.pose.pose.position.x = x;
odom.pose.pose.position.y = y;
odom.pose.pose.position.z = 0.0;
odom.pose.pose.orientation = odom_quat;

odom.twist.twist.linear.x = linearSpeed;           //线速度
odom.twist.twist.linear.y = 0;
odom.twist.twist.linear.z = 0;
odom.twist.twist.angular.x = 0;
odom.twist.twist.angular.y = 0;
odom.twist.twist.angular.z = angularSpeed;         //角速度
```

(3) 发布 imu 数据

```
void actuator::pub_9250() {
    sensor_msgs::Imu imuMsg;
    sensor_msgs::MagneticField magMsg;

    ros::Time current_time = ros::Time::now();

    imuMsg.header.stamp = current_time;
    imuMsg.header.frame_id = "imu_link";
    imuMsg.angular_velocity.x = gyroX;
    imuMsg.angular_velocity.y = gyroY;
    imuMsg.angular_velocity.z = gyroZ;
    imuMsg.angular_velocity_covariance = {
        0.04,0.0,0.0,
        0.0,0.04,0.0,
        0.0,0.0,0.04
    };
}
```

```

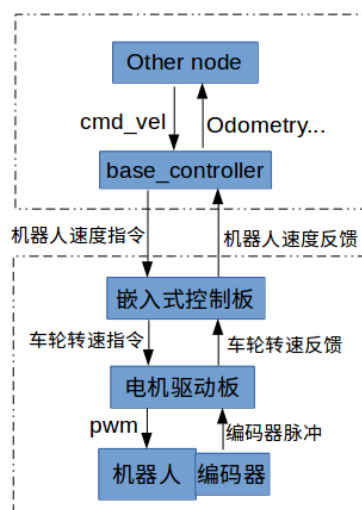
imuMsg.linear_acceleration.x = accelX;
imuMsg.linear_acceleration.y = accelY;
imuMsg.linear_acceleration.z = accelZ;
imuMsg.linear_acceleration_covariance = {
    0.04,0.0,0.0,
    0.0,0.04,0.0,
    0.0,0.0,0.04
};
pub_imu.publish(imuMsg); //发布imuMsg

magMsg.header.stamp = current_time;
magMsg.header.frame_id = "base_link";
magMsg.magnetic_field.x = magX;
magMsg.magnetic_field.y = magY;
magMsg.magnetic_field.z = magZ;
magMsg.magnetic_field_covariance = {
    0.0,0.0,0.0,
    0.0,0.0,0.0,
    0.0,0.0,0.0
};
pub_mag.publish(magMsg); //发布magMsg
}

```

2.5、基座控制器(base controller)

导航功能包集假定它可以通过话题"cmd_vel"发布 [geometry_msgs/Twist](#) 类型的消息，这个消息基于机器人的基座坐标系，它传递的是运动命令。这意味着必须有一个节点订阅"cmd_vel"话题，轻舟机器人采用的是将该话题上的速度和角度直接下发给 stm32 驱动板，由驱动板完成解算得到电机控制 PWM 命令。



2.6、地图 (map_server)

在导航过程中，地图并不是必须的，此时相当于是在一个无限大的空地上进行导航，并没有任何障碍物。但是考虑到实际情况，在我们使用导航的过程中还是要一个地图的。当 stm32 驱动板代码编写完善，且以上 2.1—2.5 都设置完成以后，建议使用开源 [gmapping](#) 算法进行建图测试，rviz 中会显示正在建立地图，则以机器人自主导航的准备工作已经完成，接下来我们开始定位和路径规划相关内容。

3. 轻舟机器人定位系统

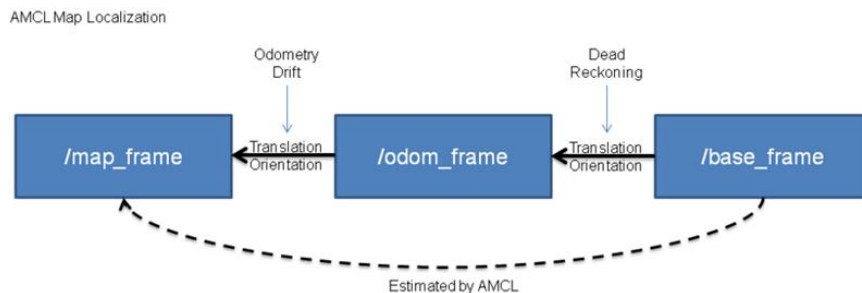
轻舟机器人的定位是使用 qingzhou_ws 中的 amcl 包完成的，此包不需要修改，但是需要我們进行一些配置。

定位程序是使用的粒子滤波获取最佳定位点，该点称为 Mp (point on map)，它是相对于

地图 map 上的坐标,也就是 base_link 相对 map 上的坐标。

odom 的原点是机器人启动时刻的位置,它在 map 上的位置或转换矩阵是未知的。但是 AMCL 可以根据最佳粒子的位置推算出 odom->map 的 tf 转换信息并发布到 tf 主题上。因为 base_link->odom 的 tf 转换信息是每时每刻都在发布的,所以它是已知的。

- 已知 tf 转换:
map->base_link
base_link->odom
- 下面的公式就可以推算:
map->odom = map->base_link - base_link->odom



我们在 amcl.launch 中对轻舟机器人的定位参数进行了设置,大家在使用过程中,可以尝试更改设置以达到更好的效果。

```

amcl.launch (-/qingzhou_ws/src/qingzhou_nav/launch) - gedit
amcl.launch
~/qingzhou_ws/src/qingzhou_nav/launch
<?xml version="1.0" ?>
<launch>
  <node pkg="amcl" type="amcl" name="amcl" output="screen">
    <!-- Publish scans from best pose at a max of 10 Hz -->
    <param name="odom_model_type" value="diff"/>
    <param name="odom_alpha5" value="0.1"/>
    <param name="transform_tolerance" value="0.2" />
    <param name="gui_publish_rate" value="10.0"/> <!-- Maximum rate (Hz) at which scans and paths are published for visualization, -1.0 to disable. -->
    <param name="laser_max_beams" value="60"/>
    <param name="min_particles" value="500"/>
    <param name="max_particles" value="800"/>
    <param name="kld_err" value="0.05"/>
    <param name="kld_z" value="0.99"/>
    <param name="odom_alpha1" value="0.3"/> <!-- Specifies the expected noise in odometry's rotation estimate from the rotational component of the robot's motion -->
    <param name="odom_alpha2" value="1.2"/> <!-- Specifies the expected noise in odometry's rotation estimate from translational component of the robot's motion -->
    <!-- translation std dev, m -->
    <param name="odom_alpha3" value="3.7"/> <!-- Specifies the expected noise in odometry's translation estimate from the translational component of the robot's motion -->
    <param name="odom_alpha4" value="0.3"/> <!-- Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion -->
    <param name="laser_z_hit" value="0.5"/>
    <param name="laser_z_short" value="0.05"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.5"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_model_type" value="likelihood_field"/>
    <!-- <param name="laser_model_type" value="beam"/> -->
    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d" value="0.1"/> <!-- Translational movement required before performing a filter update. -->
    <param name="update_min_a" value="0.1"/> <!-- Rotational movement required before performing a filter update. 0.1 represents 5.7 degrees -->
    <param name="odom_frame_id" value="odom"/>
    <param name="resample_interval" value="1"/> <!-- Number of filter updates required before resampling. -->
    <!-- Increase tolerance because the computer can get quite busy -->
    <param name="transform_tolerance" value="1.0"/> <!-- Default 0.1; time with which to post-date the transform that is published, to indicate that this transform is not yet fully initialized -->
    <param name="recovery_alpha_slow" value="0.001"/> <!-- Exponential decay rate for the slow average weight filter, used in deciding when to recover by adding an old particle -->
    <param name="recovery_alpha_fast" value="0.1"/> <!-- Exponential decay rate for the fast average weight filter, used in deciding when to recover by adding a new particle -->
  </node>
</launch>

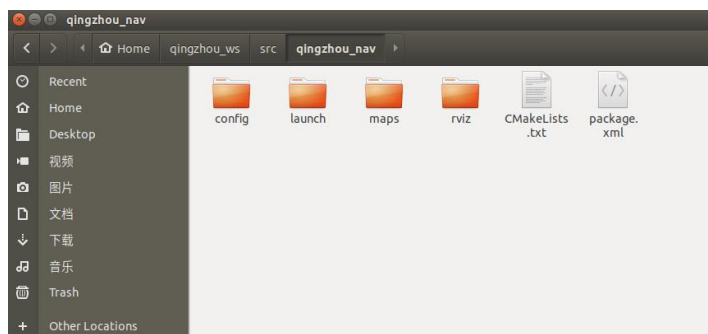
```

4、导航功能包集的配置

4.1、创建一个功能包

首先,我们创建一个软件包,用来保存我们所有的配置文件和启动文件。这个软件包需要包含所有用于实现机器人配置小节所述以来,就如其以依赖导航功能包集高级接口

我们可以看到 qingzhou_ws 空间中包含的 qingzhou_ws 包,此包为我们配置导航功能以及相关文件的存储。其中 config 文件夹内存放的是配置 move_base 相关的配置文件,launch 文件夹中存放的是我们导航及建图时的启动文件,maps 文件夹用于存放建好的地图文件,rviz 文件夹用于存放 rviz 配置参数。



4.2 创建机器人启动配置文件

现在，我们有了一个存放所有配置文件和启动文件的工作空间，我们会创建一个 `roslaunch` 文件来启动所有的硬件以及发布机器人所需的 `tf`。在 `launch` 文件夹中，我们可以看到 `qingzhou_bringup.launch` 文件，这就轻舟机器人的启动 `launch` 文件。

```
qingzhou_bringup.launch (~/.qingzhou_ws/src/qingzhou_nav/launch) - gedit
Open Save

<?xml version="1.0" ?>
<launch>
  <node name="qingzhou_bringup" pkg="qingzhou_bringup" type="qingzhou_bringup" output="screen" respawn="true"/>
  <param name="mcuserialport" value="/dev/stm32board" type="string"/>
  <param name="mcubaudrate" value="115200" type="int"/>

  <param name="calibrate_lineSpeed" value="0" type="int"/>
  <param name="calibrate_angularSpeed" value="0" type="int"/>
  <param name="ticksPerMeter" value="1200" type="int"/>
  <param name="ticksPer2PI" value="3059" type="int"/>
  <remap from="imu" to="raw" />

  <node pkg="imu_calib" type="apply_calib" name="apply_calib" output="screen" respawn="false">
    <!-- <remap from="imu" to="raw" /> -->
    <param name="calib_file" value="$(find imu_calib)/../launch/imu_calib.yaml" />
    <param name="calibrate_gyros" value="true" />
    <remap from="corrected" to="raw/data_raw" />
  </node>

  <node pkg="imu_filter_madgwick" type="imu_filter_node" name="imu_filter_madgwick" output="screen" respawn="true">
    <param name="fixed_frame" value="odom" />
    <param name="use_mag" value="false" />
    <param name="publish_tf" value="false" />
    <param name="use_magnetic_field_msg" value="true" />
    <param name="world_frame" value="enu" />
    <remap from="raw/data" to="imu_data" />
  </node>

  <node pkg="robot_pose_ekf" type="robot_pose_ekf" name="robot_pose_ekf" output="screen" respawn="true">
    <param name="output_frame" value="odom" />
    <param name="base_footprint_frame" value="base_link" />
    <param name="freq" value="30.0" />
    <param name="sensor_timeout" value="1.0" />
    <param name="odom_used" value="true" />
    <param name="imu_used" value="true" />
    <param name="vo_used" value="false" />
    <param name="gps_used" value="false" />
    <param name="debug" value="false" />
    <param name="self_diagnose" value="false" />
  </node>

  <node pkg="tf" type="static_transform_publisher" name="base_link_to_imulink"
    args="0.40 0.0 0.08 0.0 0.0 0.0 /base_link /imu_link 50" />
</launch>
```

4.3 配置代价地图 (local_costmap) & (global_costmap)

导航功能包集需要两个代价地图来保存世界中的障碍物信息。一张代价地图用于规划，在整个环境中创建长期的规划，另一个用于局部规划与避障。有一些参数两个地图都需要，而有一些则各不相同。因此，对于代价地图，有三个配置项: `common` 配置项, `global` 配置项和 `local` 配置项。

(1) 共同配置(local_costmap) & (global_costmap)

导航功能包集使用代价地图存储障碍物信息。为了使这个过程更合理，我们需要指出要监听的传感器的话题，以更新数据。我们创建一个名为 `costmap_common_params.yaml` 的文件，内容如下：


```

*costmap_common_params.yaml (~/.qingzhou_ws/src/qingzhou_nav/config) - gedit
Open  *costmap_common_params.yaml
~/.qingzhou_ws/src/qingzhou_nav/config

#map_type: voxel
#map_type: costmap
robot_radius: 0.25

footprint: [[0.23,0.19],[0.25,0],[0.23,-0.19],[-0.23, -0.19],[-0.23,0.19]]
recovery_behavior_enabled: true

controller_frequency: 5

obstacle_layer:
  enabled: true
  combination_method: 1
  track_unknown_space: false
  origin_z: 0.0
  z_voxels: 20
  z_resolution: 0.1
  unknown_cost_value: 0
  unknown_threshold: 8
  mark_threshold: 0
  publish_voxel_map: false

  obstacle_range: 6.0
  raytrace_range: 7.0
  #footprint: [[-0.3,-0.4],[-0.3,1.03],[-0.0,1.03],[0.3, 1.03],[0.3,-0.4]]
  robot_radius: 0.25
  inflation_radius: 0.25
  #inf_is_valid: true
  max_obstacle_height: 0.40
  min_obstacle_height: 0.03
  controller_frequency: 5
  observation_sources: scan
  #point_cloud_sensor

  scan: {sensor_frame: base_link, observation_persistence: 0.0,
max_obstacle_height: 0.3, min_obstacle_height: 0.05, data_type: LaserScan, topic: /scan,
marking: true,clearing: true}

inflation_layer:
  enabled: true
  cost_scaling_factor: 5.0 # exponential rate at which the obstacle cost drops off (default: 10)
  inflation_radius: 0.25 # max. distance from an obstacle at which costs are incurred for planning paths.

static_layer:
  enabled: true

```

以上各个参数的含义需要大家需要弄清楚,可以网上查询,之后尝试调试出更合适参数,主要参数包括:

“obstacle_range”参数决定了引入障碍物到代价地图的传感器读书的最大范围;“raytrace_range”参数确定的空白区域内光线追踪的范围;Footprint 设置机器人轮廓,还需设置代价地图膨胀半径。

“observation_sources”参数定义了一系列传递空间信息给代价地图的传感器。每个传感器定义在下一行。

(2) 全局配置(global_costmap)

下面我们将创建一个存储特定的全局代价地图配置选项的文件。新建一个文件: global_costmap_params.yaml:

```

global_costmap_params.yaml (~/.qingzhou_ws/src/qingzhou_nav/config) - gedit
Open  global_costmap_params.yaml
~/.qingzhou_ws/src/qingzhou_nav/config

global_costmap:
  global_frame: map
  robot_base_frame: base_link
  update_frequency: 1.0
  #static_map: true

  rolling_window: false
  resolution: 0.05

  transform_tolerance: 1.0
  ##map_type: costmap
  ##map_type: voxel
  ##inflation_radius: 0.1
  plugins:
    - {name: static_layer, type: "costmap_2d::StaticLayer"}
    - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
    - {name: inflation_layer, type: "costmap_2d::InflationLayer"}

```

“global_frame”参数定义了代价地图运行所在的坐标帧。在这种情况下,我们会选择/map frame。“robot_base_frame”参数定义了代价地图参考的的机器地毯的坐标帧。“update_frequency”参数决定了代价地图更新的频率。“static_map”参数决定代价地图是否根据 map_server 提供的地图初始化。如果你不使用现有的地图, 设为 false。

(3) 本地配置(local_costmap)

下面我们将创建一个存储特定的本地代价地图配置选项的文件。新建一个文件: localal_costmap_params.yaml 并粘贴以下内容:

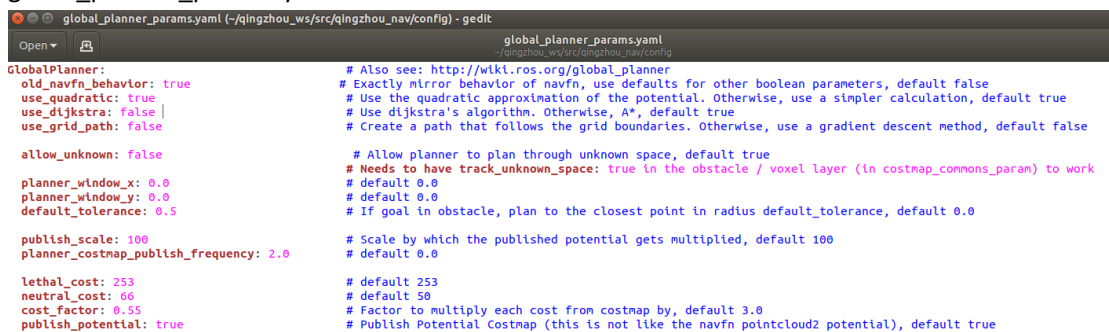


```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 3.0
  publish_frequency: 3.0
  #static_map: false
  rolling_window: true
  width: 2.5
  height: 2.5
  resolution: 0.05
  plugins:
  - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
  - {name: inflation_layer, type: "costmap_2d::InflationLayer"}
```

“global_frame”, “robot_base_frame”, “update_frequency”, “static_map”参数与全局配置意义相同。“publish_frequency”参数决定了代价地图发布可视化信息的频率。将“rolling_window”参数设置为 true, 意味着随着机器人在限时世界里移动, 代价地图会保持以机器人作为中心。“width”、“height”、“resolution”参数分别设置代价地图的宽度(米)、高度(米)和分辨率(米/单元)。注意, 这里的分辨率和你的静态地图的分辨率可能不同, 但我们通常把他们设成一样的。

4.4 Global Planner 配置

Global_Planner 将指定用于 move_base 的全局规划器插件名称。新建一个名为 golbal_planner_params.yaml 的文件, 并加入以下内容。



```
GlobalPlanner:
  old_navfn_behavior: true
  use_quadratic: true
  use_dijkstra: false
  use_grid_path: false
  allow_unknown: false
  planner_window_x: 0.0
  planner_window_y: 0.0
  default_tolerance: 0.5
  publish_scale: 100
  planner_costmap_publish_frequency: 2.0
  lethal_cost: 253
  neutral_cost: 60
  cost_factor: 0.55
  publish_potential: true

# Also see: http://wiki.ros.org/global_planner
# Exactly mirror behavior of navfn, use defaults for other boolean parameters, default false
# Use the quadratic approximation of the potential. Otherwise, use a simpler calculation, default true
# Use dijkstra's algorithm. Otherwise, A*, default true
# Create a path that follows the grid boundaries. Otherwise, use a gradient descent method, default false
# Allow planner to plan through unknown space, default true
# Needs to have track_unknown_space: true in the obstacle / voxel layer (in costmap_commons_param) to work
# default 0.0
# If goal in obstacle, plan to the closest point in radius default_tolerance, default 0.0
# Scale by which the published potential gets multiplied, default 100
# default 0.0
# default 253
# default 50
# Factor to multiply each cost from costmap by, default 3.0
# Publish Potential Costmap (this is not like the navfn pointcloud2 potential), default true
```

default_tolerance: 当设置的目的地被障碍物占据时, 需要以该参数为半径寻找到最近的点作为新目的地。use_dijkstra: 设置为 true, 将使用 dijkstra 算法, 否则使用 A*算法。

4.4 DWA Local Planner 配置

DWA_local_planner 负责根据高层规划计算速度命令并发送给机器人基座。DWA 作为 ROS 内的局部路径规划算法, 里面有许多参数需要配置。新建一个名 dwa_local_planner_params.yaml 的文件, 内容如下:

```
dwa_local_planner_params.yaml (~/.qingzhou_ws/src/qingzhou_nav/config) - gedit
dwa_local_planner_params.yaml
~/.qingzhou_ws/src/qingzhou_nav/config

DWAPlannerROS:

  max_vel_x: 0.44 # 0.55
  min_vel_x: 0.32

  max_vel_y: 0.0
  min_vel_y: 0.0

  max_vel_trans: 0.6 # choose slightly less than the base's capability
  min_vel_trans: 0.05 # this is the min trans velocity when there is negligible rotational velocity
  trans_stopped_vel: 0.1

  # Warning!
  # do not set min_trans_vel to 0.0 otherwise dwa will always think translational velocities
  # are non-negligible and small in place rotational velocities will be created.

  max_vel_theta: 0.8 # choose slightly less than the base's capability
  min_vel_theta: 0.45 # this is the min angular velocity when there is negligible translational velocity
  theta_stopped_vel: 0.1

  acc_lim_x: 18.0 # maximum is theoretically 2.0, but we
  acc_lim_theta: 18.0
  acc_lim_y: 0.0 # diff drive robot

# Goal Tolerance Parameters
yaw_goal_tolerance: 0.8 # 0.05
xy_goal_tolerance: 0.3 # 0.10
latch_xy_goal_tolerance: false

# Forward Simulation Parameters
sim_time: 2.1 # 1.7
vx_samples: 8 # 3
vy_samples: 1 # diff drive robot, there is only one sample
vtheta_samples: 20 # 20

# Trajectory Scoring Parameters
path_distance_bias: 35.0 # 32.0 - weighting for how much it should stick to the global path plan
goal_distance_bias: 26.0 # 24.0 - weighting for how much it should attempt to reach its goal
occdist_scale: 0.6 # 0.01 - weighting for how much the controller should avoid obstacles
forward_point_distance: 0.2 # 0.325 - how far along to place an additional scoring point
stop_time_buffer: 0.2 # 0.2 - amount of time a robot must stop in before colliding for a valid traj.
scaling_speed: 0.25 # 0.25 - absolute velocity at which to start scaling the robot's footprint
max_scaling_factor: 0.2 # 0.2 - how much to scale the robot's footprint when at speed.

# Oscillation Prevention Parameters
oscillation_reset_dist: 0.05 # 0.05 - how far to travel before resetting oscillation flags

# Debugging
publish_traj_pc: true
publish_cost_grid_pc: true
global_frame_id: odom
```

以上这些参数包括多机器人的 x 及 y 方向上加速度、平移速度、旋转速度最大值和最小值设置等一系列参数，参数的好坏决定着我们的机器人的稳定与否。

4.5 为导航功能包创建一个 Launch 启动文件

现在我们已经有了所有的配置文件，我们还需要在一个启动文件中一起启动他们，创建一个名为 `qingzhou_move_base.launch` 的文件，内容如下：

```
qingzhou_move_base.launch (~/.qingzhou_ws/src/qingzhou_nav/launch) - gedit
qingzhou_move_base.launch
~/.qingzhou_ws/src/qingzhou_nav/launch

<?xml version="1.0" ?>
<launch>
  <master auto="start"/>

  <include file="$(find qingzhou_nav)/launch/ydlidar.launch" />
  <node name="map_server" pkg="map_server" type="map_server" args="$(find qingzhou_nav)/maps/test20201120.yaml"
  output="screen"/>

  <include file="$(find qingzhou_nav)/launch/amcl.launch" />

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

    <param name="base_global_planner" value="global_planner/GlobalPlanner" />
    <param name="planner_frequency" value="1.0" />
    <param name="planner_patience" value="5.0" />

    <param name="base_local_planner" value="dwa_local_planner/DWAPlannerROS" />
    <param name="controller_frequency" value="5.0" />
    <param name="controller_patience" value="5.0" />

    <rosparam file="$(find qingzhou_nav)/config/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find qingzhou_nav)/config/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find qingzhou_nav)/config/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find qingzhou_nav)/config/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find qingzhou_nav)/config/dwa_local_planner_params.yaml" command="load" />
    <rosparam file="$(find qingzhou_nav)/config/global_planner_params.yaml" command="load" />

  </node>
```

启动文件中,我们更改地图服务器使指向你的已有的地图,启动雷达节点 `ydlidar.launch`, 启动定位 `amcl.launch` 文件, 启动 `move_base` 节点, 并且将我们的相关配置文件, 我们选择 A*全局路径规划和 DWA 局部路径规划, 快来测试你手中的轻舟机器人吧!