

**智能机器人课程设计结题报告**

**题 目 基于Nav2与YOLOv5的巡航搜救机器人**

**学生姓名 梁文杰、陈宸、林沛麟**

**学 号 20224001003 20224001071 20224001045**

**专业班级 \_ 22级人工智能一班**

**导师姓名 \_\_ 冯刚 \_\_**

**二〇二四 年 十二 月**

目 录

[一、 项目概述 3](#_Toc185685121)

[1.1 引言 3](#_Toc185685122)

[1.2 实现思路 3](#_Toc185685123)

[1.3 得分点 5](#_Toc185685124)

[二、 项目实现 6](#_Toc185685125)

[2.1 模型构建 6](#_Toc185685126)

[2.1.1 机器人模型构建 6](#_Toc185685127)

[2.1.2 世界模型的构建 8](#_Toc185685128)

[2.2 SLAM建图 10](#_Toc185685129)

[2.3 创新点的实现 15](#_Toc185685130)

[2.3.1 YOLOv5搭载 15](#_Toc185685131)

[2.3.2 巡航任务实现 21](#_Toc185685132)

[三、 项目总结 29](#_Toc185685133)

[四、参考文献 30](#_Toc185685134)

1. 项目概述

## 引言

本项目旨在实现基于ROS2框架在gazebo与Rviz仿真环境下的机器人巡航与导航搜救任务。具体目标包括：自行设计构建机器人模型，利用Carto实现地图模型的构建并保存；实现YOLOv5在ROS2的搭载，使YOLOv5订阅摄像机话题实现机器人摄像机的图像识别功能；编写巡航任务文件，使机器人在自行构建的地图文件中，基于Nav2利用rclpy包多线程完成经过多个目标点的巡航任务，若在巡航任务中摄像机检测到人体模型则前往人所在位置，并立即中断当前巡航任务返回初始点。该项目简单模拟了现实生活中自然灾害情境下搜救机器人的工作任务。

## 1.2 实现思路

**1.实现机器人的组装与配置**

首先基于我们的任务，机器人需要能够完成一般的建图、导航、图像识别任务，故我们为机器人配置了差速控制插件并正确分配其物理属性使其能够在仿真环境中正常运动，同时为机器人配置雷达插件以实现建图工作，配置摄像头使其能够完成图像识别功能。按照要求编写完urdf后为机器人配置相关功能包的依赖。

**2.实现地图环境**

按照实际任务的需求，我们需要编写room.world文件创建复杂的仿真环境，包括障碍物和目标人体模型以检验导航任务的完成效果，并测试机器人在巡航过程中能够识别并响应目标的任务。

**3.实现SLAM**

首先安装Cartographer与相关软件包，并配置SLAM节点，创建ROS2的launch文件，配置SLAM节点订阅激光雷达数据并发布地图。启动SLAM节点后，在Gazebo仿真环境中运行机器人进行建图，并保存生成的地图文件

**4.搭载YOLO**

首先需要安装YOLOv5代码库并安装所需的为ros配置相关的Python依赖项。然后需要编写一个YOLO ROS2节点，用于加载YOLOv5模型，订阅摄像头话题，并进行图像识别。

之后我们需要进行测试，确保YOLO节点能够正常运行，并在接收到摄像头图像后进行目标检测，输出识别结果

**5.设计集成导航与识别的巡航任务**

我们需要编写巡航任务脚本，基于rclpy编写多线程ROS2节点使机器人在仿真环境中基于Nav2巡航，经过多个目标点。同时还需要实现检测响应的条件模块，在巡航过程中，如果摄像头检测到人体模型，则终止当前巡航任务的线程，发布新的线程，前往检测到的人体位置。完成救援任务后，机器人返回到起始位置，模拟搜救任务完成后的状态。

## 1.3 得分点

**总得分：170分**

**使用仿真环境（30分）:**我们使用Gazebo与Rviz模拟软件对机器人系统的功能进行编程和演示。

**基于传感器的行为（20分）:**我们编写的机器人具有雷达传感器，导航任务实质上基于了雷达传感器数据做出了反应。

**自定义节点开发（20分）:**我们编写了使ROS2搭载YOLOv5功能包的结点。

**导航算法（20分）:**机器人基于Nav2功能包自带的导航算法在环境中有效地移动或规划路径。

**集成人工智能（40分）:**编写的图像识别结点订阅机器人摄像头发布的内容并基于YOLOv5模型完成识别任务，YOLOv5集成了卷积神经网络等深度学习技术实现了图像识别功能。

**使用地图（20分）:**我们自行构建了地图的urdf文件，并在导航过程中利用SLAM导出的包含地图数据的.yaml文件执行任务。

**地图绘制（20分）**:机器人基于Cartographer功能包实现了对地图文件的SLAM，绘制得到的地图数据用于导航。

1. 项目实现

## 2.1 模型构建

首先我们在~/目录下创建工作空间目录RobotProject。

### 2.1.1 机器人模型构建

在工作空间src目录下构建robot\_description功能包，保存机器人、世界模型等功能包。

机器人模型的设计目标是能在一般平面上平滑运动并完成后续的导航任务。首先需要满足机器人的每一个关节具有正确的物理属性，需要为每个关节配置对应的惯性矩阵与质量。

构建的机器人主体为0.20\*0.20\*0.12的正方体，含有4个轮子，其中后两轮为驱动轮，形状为高度为0.04，半径为0.032的圆柱体。前两轮为半径为0.016的球体支撑轮，不提供驱动功能。

机器人每个部分对应信息如下：

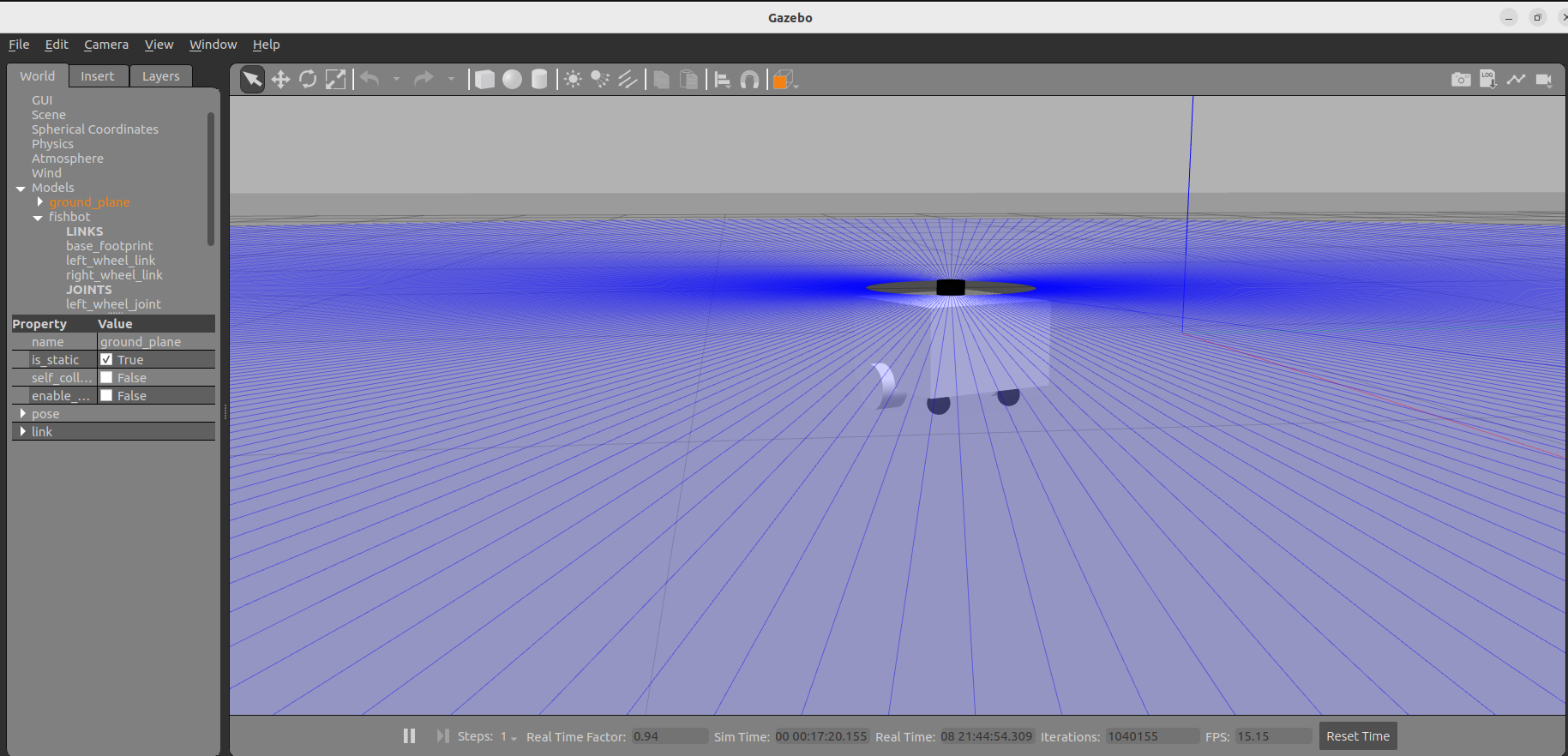
|  |  |  |  |
| --- | --- | --- | --- |
| **关节** | **类型** | **Parent link** | **Child link** |
| **Base\_joint** | fixed | base\_footprint | base\_link |
| **laser\_joint** | fixed | base\_link | laser\_link |
| **imu\_joint** | fixed | base\_link | imu\_link |
| **left\_wheel\_joint** | continuous | base\_link | left\_wheel\_link |
| **right\_wheel\_joint** | continuous | base\_link | right\_wheel\_link |
| **caster\_joint** | fixed | base\_link | caster\_link |
| **caster\_joint1** | fixed | base\_link | caster\_link1 |

为驱动机器人运动，在urdf文件中为机器人配置了差速控制插件，同时为机器人编写了雷达传感器插件以完成建图导航任务并配置雷达范围等参数。最后为机器人配置了摄像机以完成图像识别工作。

由于在中期概要设计部分已展示差速控制插件与雷达的实现，此处只展示摄像机插件的构建。

|  |
| --- |
| <gazebo reference="camera\_link">  <sensor name="camera\_depth" type="depth">  <update\_rate>20</update\_rate>  <camera>  <horizontal\_fov>1.047198</horizontal\_fov>  <image>  <width>640</width>  <height>480</height>  <format>R8G8B8</format>  </image>  <clip>  <near>0.05</near>  <far>3</far>  </clip>  </camera>  <plugin name="camera\_controller" filename="libgazebo\_ros\_camera.so">  <always\_on>true</always\_on>  <update\_rate>10.0</update\_rate>  <camera\_name>camera</camera\_name>  <frame\_name>camera\_link</frame\_name>  </plugin>  </sensor>  </gazebo> |

机器人模型在gazebo中可视化如下：



### 2.1.2 世界模型的构建

在robot\_description下创建world目录，在目录下编写room.world文件来实现迷宫世界模型的构建。

Room.world定义了一个名为“default”的仿真世界，包含了多种元素，如光源、地面、墙壁和一个虚拟人物模型。整个环境创建了一个迷宫房间模型，结构复杂且具有多个墙面。

同时还我们还设置了复杂的环境要素，如光源为一个具有阴影和特定光照参数的方向性光源，设定了重力（-9.8）和一个微弱的磁场。机器人运动的地面是一个静态的灰色平面，其材质设定为“Gazebo/Grey”带有摩擦和接触参数。迷宫由多个墙壁组成，每个墙壁都有自己的碰撞和可视属性，以及不同的大小和角度，同时每面墙壁设定了具体的几何形状（如长方体）和位置，能够与其他元素发生物理交互。

最后加入搜救任务的目标：人体模型。我们采用url引用的方式导入该模型并设定对应的坐标。

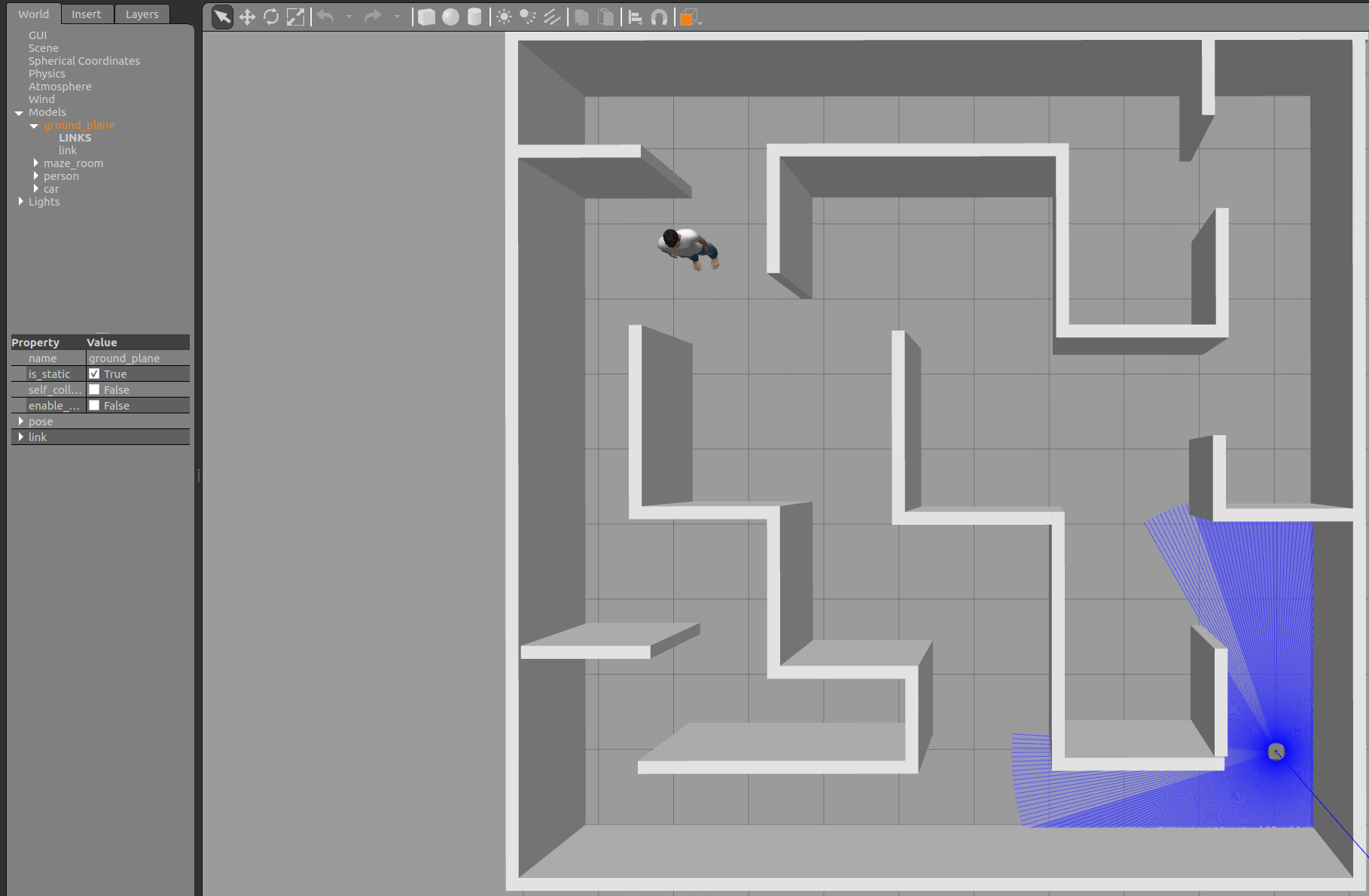
|  |
| --- |
| <model name='person'>  <static>true</static>  <include>  <uri>model://person\_standing</uri>  </include>  <pose>6.617711 7.591491 0 0 0 -1.423432</pose>  </model> |

最后设置了世界的摄像机参数，使项目参与人员能够更加灵活的观察调控记录模型与机器人的位置。

在launch目录下编写gazebo.launch.py，使机器人模型与世界模型能够加载入gazebo环境内。结果如下：

|  |
| --- |
| import os  from launch import LaunchDescription  from launch.actions import ExecuteProcess  from launch\_ros.actions import Node  from launch\_ros.substitutions import FindPackageShare  def generate\_launch\_description():  robot\_name\_in\_model = 'car'  package\_name = 'robot\_description'  urdf\_name = "car.urdf"  world\_name='room.world'  ld = LaunchDescription()  pkg\_share = FindPackageShare(package=package\_name).find(package\_name)  urdf\_model\_path = os.path.join(pkg\_share, f'urdf/{urdf\_name}')  gazebo\_world\_path = os.path.join(pkg\_share, f'world/{world\_name}')  # Start Gazebo server  start\_gazebo\_cmd = ExecuteProcess(  cmd=['gazebo', '--verbose','-s', 'libgazebo\_ros\_init.so', '-s', 'libgazebo\_ros\_factory.so', gazebo\_world\_path],  output='screen')  # Launch the robot  spawn\_entity\_cmd = Node(  package='gazebo\_ros',  executable='spawn\_entity.py',  arguments=['-entity', robot\_name\_in\_model, '-file', urdf\_model\_path ], output='screen')  # Start Robot State publisher  start\_robot\_state\_publisher\_cmd = Node(  package='robot\_state\_publisher',  executable='robot\_state\_publisher',  arguments=[urdf\_model\_path]  )  # Launch RViz  start\_rviz\_cmd = Node(  package='rviz2',  executable='rviz2',  name='rviz2',  output='screen',  # arguments=['-d', default\_rviz\_config\_path]  )  ld.add\_action(start\_gazebo\_cmd)  ld.add\_action(spawn\_entity\_cmd)  ld.add\_action(start\_robot\_state\_publisher\_cmd)  # ld.add\_action(start\_rviz\_cmd)  return ld |

通过命令行运行launch.py文件，在gazebo中模型可视化结果如下：



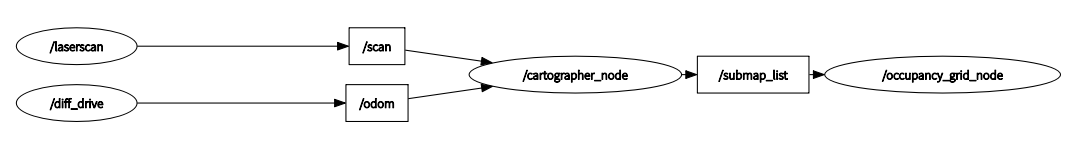
## 2.2 SLAM建图

在工作空间src目录下创建robot\_mapping功能包，因为建图时需要使用Rviz并实现cartographer相关配置，我们在功能包目录下创建rviz，config，launch，map目录。

在config目录下添加配置文件map\_2d.lua，在建图时可以使机器人的雷达按照配置文件进行工作，对应参数解释详见注释。

|  |
| --- |
| include "map\_builder.lua"  include "trajectory\_builder.lua"  options = {  map\_builder = MAP\_BUILDER,  trajectory\_builder = TRAJECTORY\_BUILDER,  map\_frame = "map",  tracking\_frame = "base\_link",  -- base\_link改为odom,发布map到odom之间的位姿态  published\_frame = "odom",  odom\_frame = "odom",  -- true改为false，不用提供里程计数据  provide\_odom\_frame = false,  -- false改为true，仅发布2D位资  publish\_frame\_projected\_to\_2d = true,  -- false改为true，使用里程计数据  use\_odometry = true,  use\_nav\_sat = false,  use\_landmarks = false,  -- 0改为1,使用一个雷达  num\_laser\_scans = 1,  -- 1改为0，不使用多波雷达  num\_multi\_echo\_laser\_scans = 0,  -- 10改为1，1/1=1等于不分割  num\_subdivisions\_per\_laser\_scan = 1,  num\_point\_clouds = 0,  lookup\_transform\_timeout\_sec = 0.2,  submap\_publish\_period\_sec = 0.3,  pose\_publish\_period\_sec = 5e-3,  trajectory\_publish\_period\_sec = 30e-3,  rangefinder\_sampling\_ratio = 1.,  odometry\_sampling\_ratio = 1.,  fixed\_frame\_pose\_sampling\_ratio = 1.,  imu\_sampling\_ratio = 1.,  landmarks\_sampling\_ratio = 1.,  }  -- false改为true，启动2D SLAM  MAP\_BUILDER.use\_trajectory\_builder\_2d = true  -- 0改成0.10,比机器人半径小的都忽略  TRAJECTORY\_BUILDER\_2D.min\_range = 0.10  -- 30改成3.5,限制在雷达最大扫描范围内，越小一般越精确些  TRAJECTORY\_BUILDER\_2D.max\_range = 3.5  -- 5改成3,传感器数据超出有效范围最大值  TRAJECTORY\_BUILDER\_2D.missing\_data\_ray\_length = 3.  -- true改成false,不使用IMU数据  TRAJECTORY\_BUILDER\_2D.use\_imu\_data = false  -- false改成true,使用实时回环检测来进行前端的扫描匹配  TRAJECTORY\_BUILDER\_2D.use\_online\_correlative\_scan\_matching = true  -- 1.0改成0.1,提高对运动的敏感度  TRAJECTORY\_BUILDER\_2D.motion\_filter.max\_angle\_radians = math.rad(0.1)  -- 0.55改成0.65,Fast csm的最低分数，高于此分数才进行优化。  POSE\_GRAPH.constraint\_builder.min\_score = 0.65  --0.6改成0.7,全局定位最小分数，低于此分数则认为目前全局定位不准确  POSE\_GRAPH.constraint\_builder.global\_localization\_min\_score = 0.7  -- 设置0可关闭全局SLAM  -- POSE\_GRAPH.optimize\_every\_n\_nodes = 0  return options |

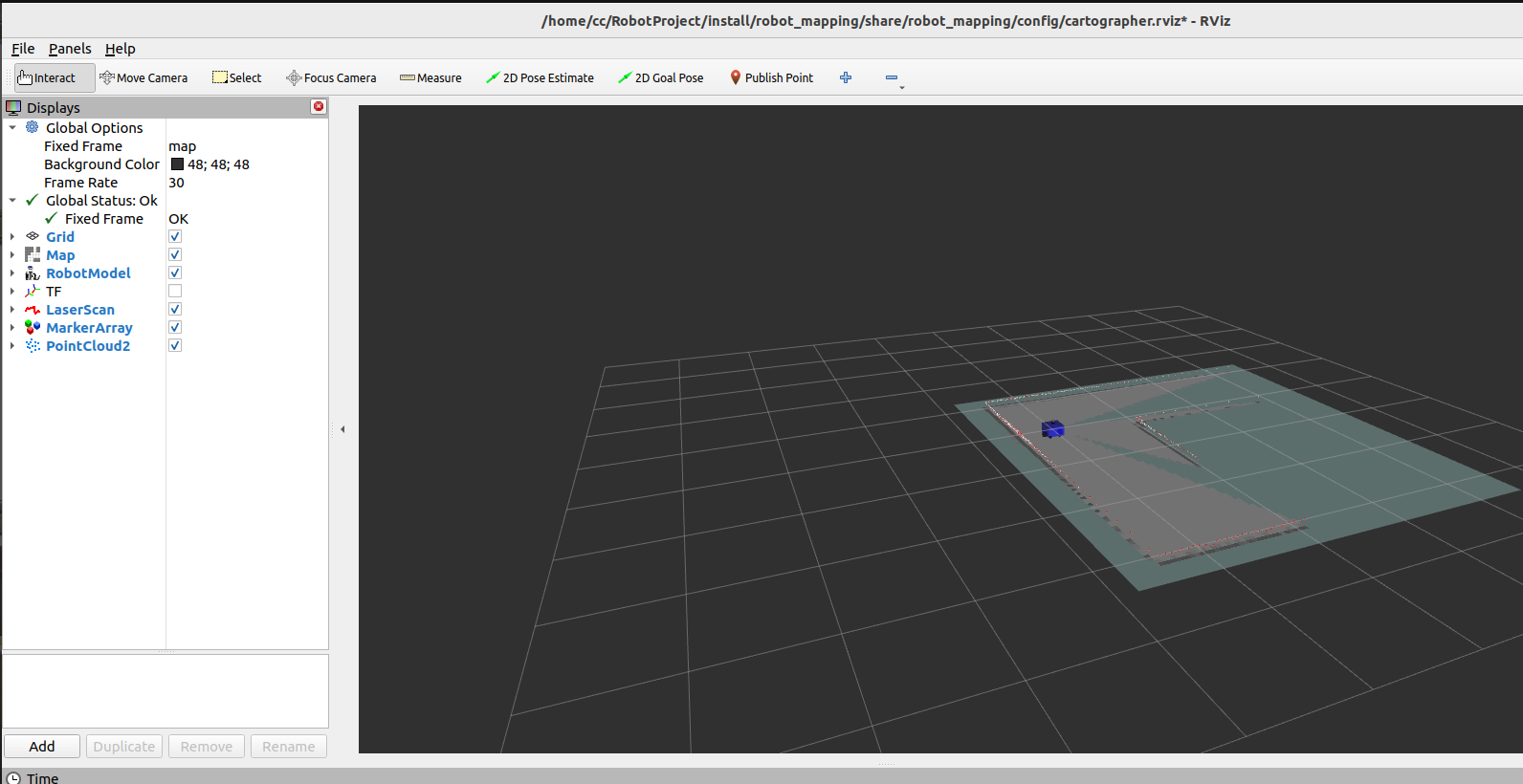
接下来需编写对应的launch文件，为实现SLAM，cartographer需要加入以下两个结点：**/cartographer\_node节点**用于从/scan和/odom话题接收数据进行计算，输出/submap\_list数据；**/occupancy\_grid\_node节点**用于接收/submap\_list子图列表，然后将其拼接成map并发布



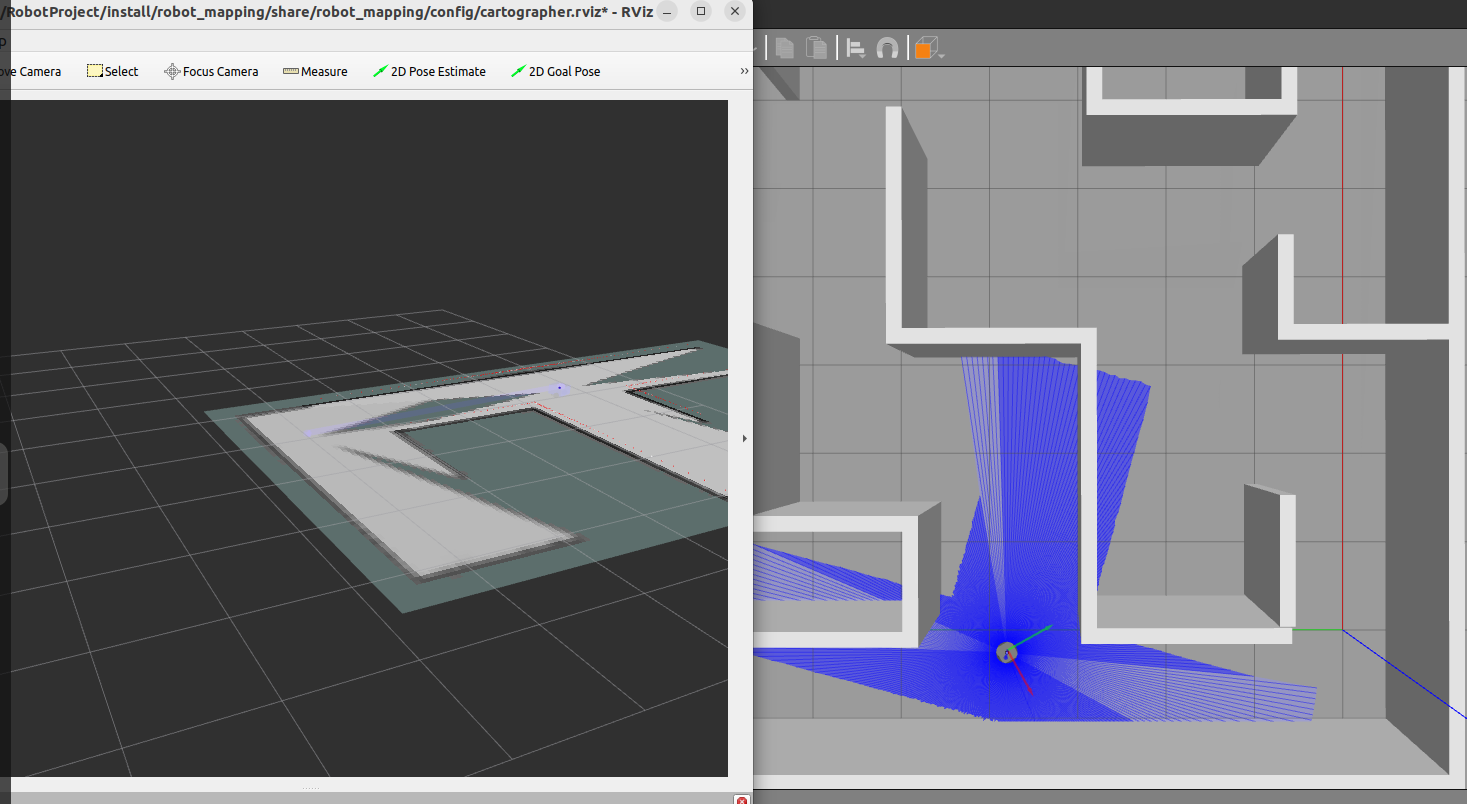
在加入以上结点同时还需加入启动rviz的结点，cartographer.launch.py文件如下

|  |
| --- |
| import os  from launch import LaunchDescription  from launch.substitutions import LaunchConfiguration  from launch\_ros.actions import Node  from launch.actions import IncludeLaunchDescription  from launch.launch\_description\_sources import PythonLaunchDescriptionSource  from launch\_ros.substitutions import FindPackageShare  from ament\_index\_python.packages import get\_package\_share\_directory  def generate\_launch\_description():  # 定位功能包地址  pkg\_share = FindPackageShare(package='robot\_mapping').find('robot\_mapping')  # 是否使用仿真时间，设置成true  use\_sim\_time = LaunchConfiguration('use\_sim\_time', default='true')  # 地图的分辨率  resolution = LaunchConfiguration('resolution', default='0.05')  # 地图的发布周期  publish\_period\_sec = LaunchConfiguration('publish\_period\_sec', default='1.0')  # 配置文件夹路径  configuration\_directory = LaunchConfiguration('configuration\_directory',default= os.path.join(pkg\_share, 'config') )  # 配置文件  configuration\_basename = LaunchConfiguration('configuration\_basename', default='map\_2d.lua')  rviz\_config\_dir = os.path.join(pkg\_share, 'config')+"/cartographer.rviz"  print(f"rviz config in {rviz\_config\_dir}")  cartographer\_node = Node(  package='cartographer\_ros',  executable='cartographer\_node',  name='cartographer\_node',  output='screen',  parameters=[{'use\_sim\_time': use\_sim\_time}],  arguments=['-configuration\_directory', configuration\_directory,  '-configuration\_basename', configuration\_basename])  cartographer\_occupancy\_grid\_node = Node(  package='cartographer\_ros',  executable='cartographer\_occupancy\_grid\_node',  name='cartographer\_occupancy\_grid\_node',  output='screen',  parameters=[{'use\_sim\_time': use\_sim\_time}],  arguments=['-resolution', resolution, '-publish\_period\_sec', publish\_period\_sec])  rviz\_node = Node(  package='rviz2',  executable='rviz2',  name='rviz2',  arguments=['-d', rviz\_config\_dir],  parameters=[{'use\_sim\_time': use\_sim\_time}],  output='screen')  gazebo\_robot\_launch=IncludeLaunchDescription(  PythonLaunchDescriptionSource([os.path.join(  get\_package\_share\_directory('robot\_description'),'launch','gazebo.launch.py'  )])  )  ld = LaunchDescription()  ld.add\_action(cartographer\_node)  ld.add\_action(cartographer\_occupancy\_grid\_node)  ld.add\_action(rviz\_node)  ld.add\_action(gazebo\_robot\_launch)  return ld |

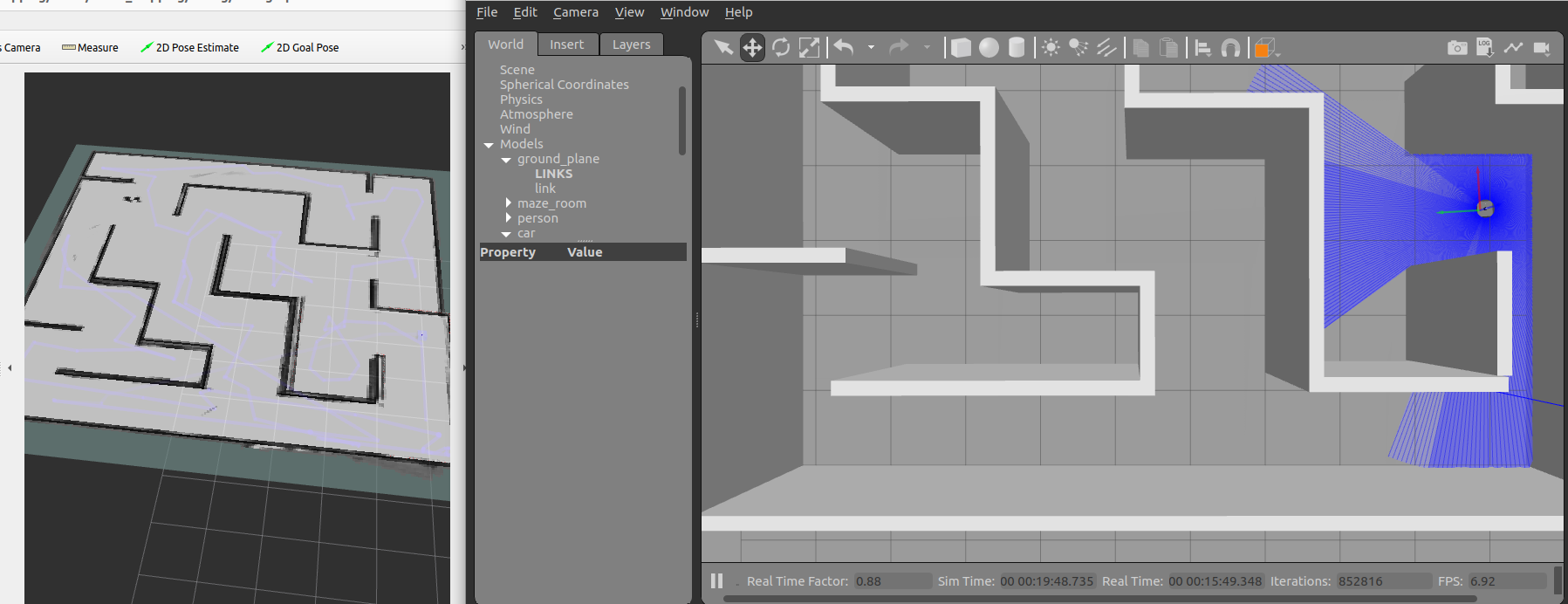
修改CmakeLists.txt添加安装指令后编译启动先前编写的robot\_description中的launch文件以启动gazebo，接着启动cartographer.launch.py，添加地图相关插件后，rviz中初始状态如下。



在新命令行中使用键盘操控机器人运动进行建图，建图过程中地图变化如下，白色部分为可行走的部分，黑色部分为障碍物轮廓。



操控机器人行走经过地图一周后，得到了如下的建图结果。



利用nav2\_map\_server将地图保存至本地，得到了map.pgm和map.yaml，至此建图的工作已全部完成。

## 2.3 创新点的实现

### 2.3.1 YOLOv5搭载

为实现机器人对相机发布话题的图像识别功能，我们需要创建一个ROS2节点，接收图像信息并调用YOLOv5进行检测，返回识别对象的信息。首先在工作空间src目录下创建yolo\_detection功能包，在目录下创建同名目录，并在同名目录下编写yolo\_detect\_2d.py，在其中实现ROS2节点的编写。

首先初始化节点并声明设备参数（如CPU或CUDA设备），默认值为CPU；声明YOLOv5模型参数，默认使用 yolov5s 模型。还需要声明图像话题和相机信息话题，默认值分别为 /camera/image\_raw 和 /camera/camera\_info；声明相机信息文件路径参数，默认使用预定义的配置文件路径；最后需要声明显示检测结果和发布检测结果图像的参数。

|  |
| --- |
| # Create a ROS 2 Node class YoloV5Ros2.  class YoloV5Ros2(Node):  def \_\_init\_\_(self):  super().\_\_init\_\_('yolo\_detection')  self.get\_logger().info(f"Current ROS 2 distribution: {ros\_distribution}")  self.last\_time = time.time()  # Declare ROS parameters.  self.declare\_parameter("device", "cpu", ParameterDescriptor(  name="device", description="Compute device selection, default: cpu, options: cuda:0"))  self.declare\_parameter("model", "yolov5s", ParameterDescriptor(  name="model", description="Default model selection: yolov5s"))  self.declare\_parameter("image\_topic", "/camera/image\_raw", ParameterDescriptor(  name="image\_topic", description="Image topic, default: /camera/image\_raw"))    self.declare\_parameter("camera\_info\_topic", "/camera/camera\_info", ParameterDescriptor(  name="camera\_info\_topic", description="Camera information topic, default: /camera/camera\_info"))  # Read parameters from the camera\_info topic if available, otherwise, use the file-defined parameters.  self.declare\_parameter("camera\_info\_file", f"{package\_share\_directory}/config/camera\_info.yaml", ParameterDescriptor(  name="camera\_info", description=f"Camera information file path, default: {package\_share\_directory}/config/camera\_info.yaml"))  # Default to displaying detection results.  self.declare\_parameter("show\_result", False, ParameterDescriptor(  name="show\_result", description="Whether to display detection results, default: False"))  # Default to publishing detection result images.  self.declare\_parameter("pub\_result\_img", True, ParameterDescriptor(  name="pub\_result\_img", description="Whether to publish detection result images, default: True")) |

从对应路径加载yolov5模型。

|  |
| --- |
| # Load model  model\_path = package\_share\_directory + "/config/" + self.get\_parameter('model').value + ".pt"  device = self.get\_parameter('device').value  self.yolov5 = YOLOv5(model\_path=model\_path, device=device) |

创建两个发布节点，分别用于发布检测结果和检测结果图像。同时需要创建两个订阅节点，第一个节点作为图像订阅节点，负责订阅图像话题并调用image\_callback回调函数；第二个节点用于订阅相机信息，并调用camera\_info\_callback回调函数。

|  |
| --- |
| # Create publishers.  self.yolo\_result\_pub = self.create\_publisher(  Detection2DArray, "yolo\_result", 10)  self.result\_msg = Detection2DArray()  self.result\_img\_pub = self.create\_publisher(Image, "result\_img", 10)  # subscribe to depth information for 3D cameras  image\_topic = self.get\_parameter('image\_topic').value  self.image\_sub = self.create\_subscription(  Image, image\_topic, self.image\_callback, 10) |

其中为实现订阅图像话题，需要获取当前时间并计算时间差，记录图像回调频率。为了能够使yolov5对图像信息进行处理，需要使用 cv\_bridge 将图像消息转换为OpenCV格式。经过模型预测后，需解析检测结果，包括目标物体的类别、位置坐标和置信度。最后将检测结果绘制在图像上，并根据需要显示和发布检测结果图像。

|  |
| --- |
| def image\_callback(self, msg: Image):  # get current time  current\_time = time.time()  time\_diff = current\_time - self.last\_time  self.last\_time = current\_time  self.get\_logger().info(f'Image callback frequency: {1.0 / time\_diff:.2f} Hz')  # Detect and publish results  image = self.bridge.imgmsg\_to\_cv2(msg)  detect\_result = self.yolov5.predict(image)  self.get\_logger().info(str(detect\_result))  self.result\_msg.detections.clear()  self.result\_msg.header.frame\_id = "camera"  self.result\_msg.header.stamp = self.get\_clock().now().to\_msg()  # Parse the results.  predictions = detect\_result.pred[0]  boxes = predictions[:, :4] # x1, y1, x2, y2  scores = predictions[:, 4]  categories = predictions[:, 5]  for index in range(len(categories)):  name = detect\_result.names[int(categories[index])]  detection2d = Detection2D()  detection2d.id = name  x1, y1, x2, y2 = boxes[index]  x1 = int(x1)  y1 = int(y1)  x2 = int(x2)  y2 = int(y2)  center\_x = (x1+x2)/2.0  center\_y = (y1+y2)/2.0  if ros\_distribution=='galactic':  detection2d.bbox.center.x = center\_x  detection2d.bbox.center.y = center\_y  else:  detection2d.bbox.center.position.x = center\_x  detection2d.bbox.center.position.y = center\_y  detection2d.bbox.size\_x = float(x2-x1)  detection2d.bbox.size\_y = float(y2-y1)  obj\_pose = ObjectHypothesisWithPose()  obj\_pose.hypothesis.class\_id = name  obj\_pose.hypothesis.score = float(scores[index])  # px2xy  world\_x, world\_y = px2xy(  [center\_x, center\_y], self.camera\_info["k"], self.camera\_info["d"], 1)  obj\_pose.pose.pose.position.x = world\_x  obj\_pose.pose.pose.position.y = world\_y  detection2d.results.append(obj\_pose)  self.result\_msg.detections.append(detection2d)  # Draw results.  if self.show\_result or self.pub\_result\_img:  cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)  cv2.putText(image, f"{name}({world\_x:.2f},{world\_y:.2f})", (x1, y1),  cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 1)  cv2.waitKey(1)  # Display results if needed.  if self.show\_result:  cv2.imshow('result', image)  cv2.waitKey(1)  # Publish result images if needed.  if self.pub\_result\_img:  result\_img\_msg = self.bridge.cv2\_to\_imgmsg(image, encoding="bgr8")  result\_img\_msg.header = msg.header  self.result\_img\_pub.publish(result\_img\_msg)  if len(categories) > 0:  self.yolo\_result\_pub.publish(self.result\_msg) |

在另一个订阅者节点中接收相机参数信息，并更新存储的相机信息。

|  |
| --- |
| def camera\_info\_callback(self, msg: CameraInfo):  # Get camera parameters through a callback function.  self.camera\_info['k'] = msg.k  self.camera\_info['p'] = msg.p  self.camera\_info['d'] = msg.d  self.camera\_info['r'] = msg.r  self.camera\_info['roi'] = msg.roi  # self.camera\_info\_sub.destroy() destroy会导致spin过程反复生成并杀死进程报错 |

编写完搭载YOLOv5的ROS2节点后，为使导航任务中能够准确获取目标的坐标，需要将识别出的像素坐标转换为相机坐标系的二维坐标，为了将相机图像中的像素坐标转换为相机坐标系下的二维坐标，我们通常需要进行两步：畸变校正和坐标转换。

我们引入相机的内参矩阵，其中，和是焦距（以像素为单位），和是主点坐标（通常是图像的中心）

为实现畸变的矫正，我们需要使用畸变参数进行计算，其一般包括径向和切向畸变参数:

其中，, , 是径向畸变系数，，是切向畸变系数。

假设未校正的像素坐标为 ，校正后的像素坐标为

则校正的过程如下：

其中，。实际操作中我们使用OpenCV中的函数进行畸变校正。最后实现坐标转换，经过畸变校正后，像素坐标 转换为相机坐标系下的二维坐标 的过程如下：

其中，是深度值，根据先前设定的相机参数，此处设定为1。

实现cv\_tool.py与yolo\_detect\_2d放置在同一目录下，具体内容如下：

|  |
| --- |
| import cv2  import numpy as np  # Camera intrinsic matrix K  K = [[602.7175003324863, 0, 351.305582038406],  [0, 601.6330312976042, 240.0929104708551],  [0, 0, 1]]  # Camera distortion parameters D  D = [0.06712174262966401, -0.2636999208734844,  0.006484443443073637, 0.01111161327049835, 0]  # Define a function px2xy to convert pixel coordinates to 2D coordinates in camera coordinate system  def px2xy(point, camera\_k, camera\_d, z=1.0):  MK = np.array(camera\_k, dtype=float).reshape(3, 3)  MD = np.array(camera\_d, dtype=float)  point = np.array(point, dtype=float)    # Use OpenCV's cv2.undistortPoints function to correct distortion  pts\_uv = cv2.undistortPoints(point, MK, MD) \* z  # Return 2D coordinates in the camera coordinate system  return pts\_uv[0][0]  # print(px2xy([0, 0], K, D, 1)) |

配置相关依赖后，我们已经实现了ROS2搭载YOLOv5的工作，确保了自定义的节点能够订阅摄像机发布的内容并返回图像识别后的结果。

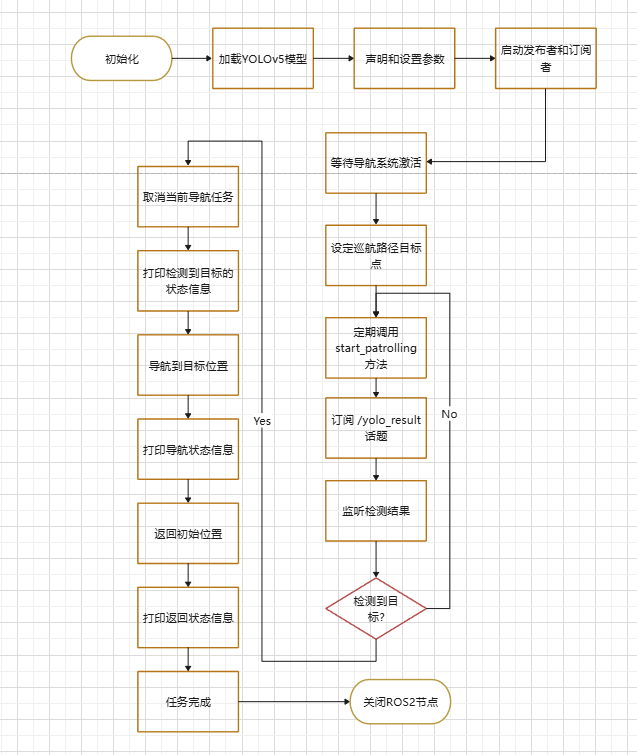
### 2.3.2 巡航任务实现

在工作目录下创建robot\_navigating包，在其目录下创建launch，maps，param，rviz，config目录。

下载安装nav2后需要配置对应参数，将nav2\_bringup下的默认参数复制至config目录下。将先前构建的地图文件map.pgm，map.yaml放置在map目录下，然后在launch目录下编写navigating.launch.py。

|  |
| --- |
| import os  from ament\_index\_python.packages import get\_package\_share\_directory  from launch import LaunchDescription  from launch.actions import IncludeLaunchDescription  from launch.launch\_description\_sources import PythonLaunchDescriptionSource  from launch.substitutions import LaunchConfiguration  from launch\_ros.actions import Node  def generate\_launch\_description():  robot\_navigation\_dir = get\_package\_share\_directory('robot\_navigation')  nav\_bringup\_dir = get\_package\_share\_directory('nav2\_bringup')  use\_sim\_time = LaunchConfiguration('use\_sim\_time', default='true')  map\_yaml\_path = LaunchConfiguration('map',default=os.path.join(robot\_navigation\_dir,'maps','map.yaml'))  nav\_param\_path = LaunchConfiguration('params\_file',default=os.path.join(robot\_navigation\_dir,'param','robot\_nav.yaml'))  rviz\_config\_dir = os.path.join(nav\_bringup\_dir,'rviz','nav2\_default\_view.rviz')  #传入地图路径与nav2参数文件  nav\_bringup\_launch = IncludeLaunchDescription(  PythonLaunchDescriptionSource([nav\_bringup\_dir,'/launch','/bringup\_launch.py']),  launch\_arguments={  'map': map\_yaml\_path,  'use\_sim\_time': use\_sim\_time,  'params\_file': nav\_param\_path}.items(),  )  rviz\_node = Node(  package='rviz2',  executable='rviz2',  name='rviz2',  arguments=['-d', rviz\_config\_dir],  parameters=[{'use\_sim\_time': use\_sim\_time}],  output='screen'  )  gazebo\_robot\_launch=IncludeLaunchDescription(  PythonLaunchDescriptionSource([os.path.join(  get\_package\_share\_directory('robot\_description'),'launch','gazebo.launch.py'  )])  )  return LaunchDescription([nav\_bringup\_launch,rviz\_node,gazebo\_robot\_launch]) |

完成了导航包的构建后，我们需要设计巡航任务，即在地图中经过多个目标点，在遍历多个目标点过程中若摄像头检测到人体，则前往人的目标位置，后返回初始位置，预设的工作包实现流程如下：



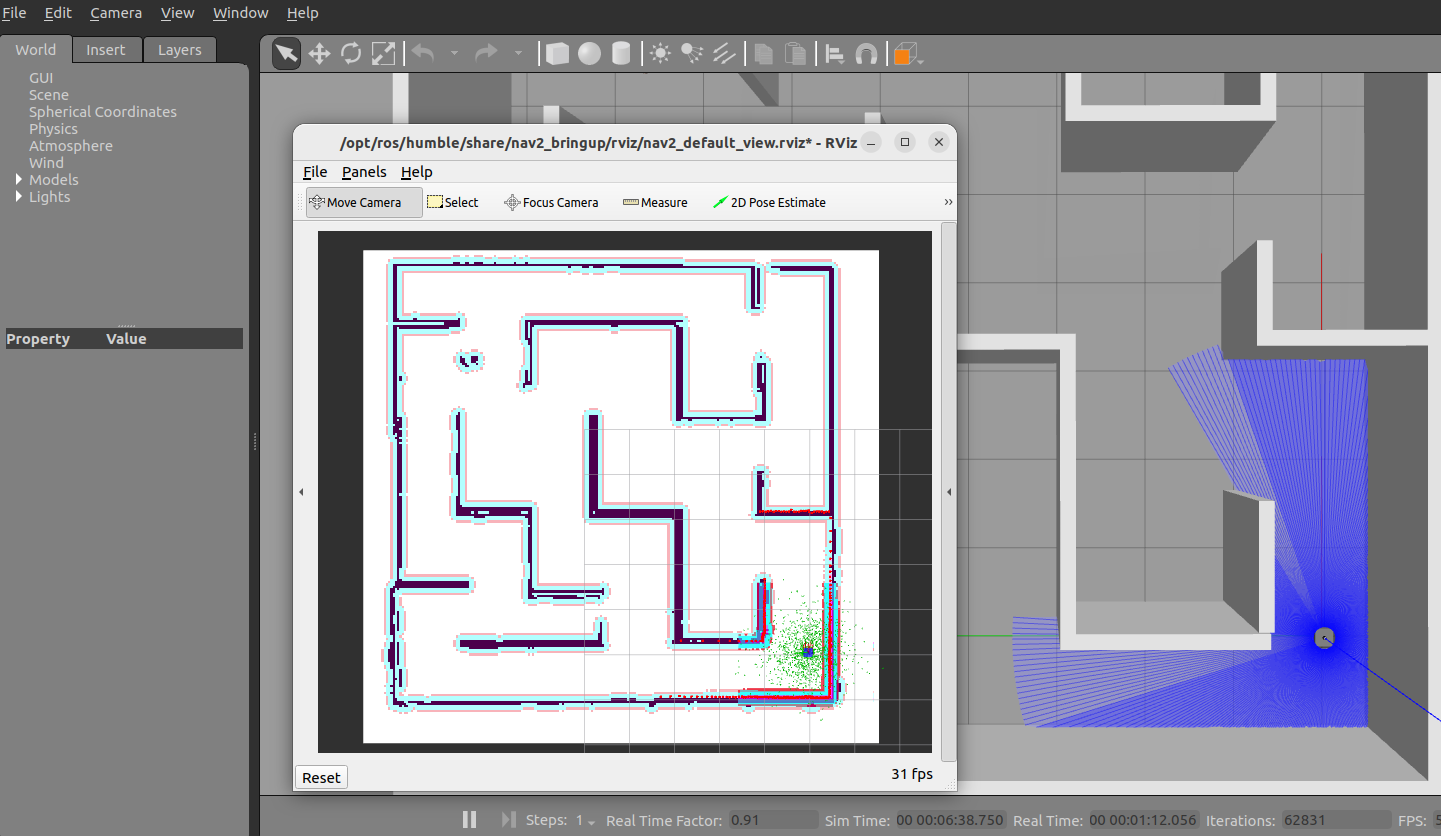
实现上述流程的关键代码如下：

|  |
| --- |
| class ResultSubscriber(Node):  def \_\_init\_\_(self):  super().\_\_init\_\_('result\_subscriber')  self.group=ReentrantCallbackGroup()  self.subscription = self.create\_subscription(Detection2DArray, '/yolo\_result', self.listener\_callback, 10,callback\_group=self.group)  self.find\_person = False  self.navigator = BasicNavigator()  # 等待导航启动完成  self.navigator.waitUntilNav2Active()  self.waypoint = [[4.42,6.0,1.0],[7.56,0.16,1.0],[7.94,6.95,-1.0],[-0.41,8.62,1.0],[0.0,0.0,1.0]]  self.point\_index = 0  self.person\_point, self.return\_point = [6.1, 7.5], [0.0, 0.0]  # 使用线程启动巡航  self.patrolling=self.create\_timer(2,self.start\_patrolling,callback\_group=self.group)  self.executor=MultiThreadedExecutor(2)  def start\_patrolling(self):  if not self.find\_person and self.navigator.isTaskComplete():  point = self.waypoint[self.point\_index]  self.nav2pose(point[0], point[1], point[2])  self.point\_index = (self.point\_index + 1) % len(self.waypoint)  def nav2pose(self, x, y, w):  goal\_pose = PoseStamped()  goal\_pose.header.frame\_id = 'map'  goal\_pose.header.stamp = self.navigator.get\_clock().now().to\_msg()  goal\_pose.pose.position.x = x  goal\_pose.pose.position.y = y  goal\_pose.pose.orientation.w = w  self.navigator.goToPose(goal\_pose)  while not self.navigator.isTaskComplete():  feedback = self.navigator.getFeedback()  self.navigator.get\_logger().info(  f'预计: {Duration.from\_msg(feedback.estimated\_time\_remaining).nanoseconds / 1e9:.1f} s 后到达')  if Duration.from\_msg(feedback.navigation\_time) > Duration(seconds=600.0):  self.navigator.cancelTask()  break  time.sleep(1.5) # 避免过于频繁地循环  result = self.navigator.getResult()  if result == TaskResult.SUCCEEDED:  self.navigator.get\_logger().info('导航结果：成功')  elif result == TaskResult.CANCELED:  self.navigator.get\_logger().warn('导航结果：被取消')  elif result == TaskResult.FAILED:  self.navigator.get\_logger().error('导航结果：失败')  else:  self.navigator.get\_logger().error('导航结果：返回状态无效')  return result  def listener\_callback(self, msg: Detection2DArray):  if self.find\_person:  return  for detection in msg.detections:  object\_name = detection.results[0].hypothesis.class\_id  if object\_name == 'person':  self.find\_person = True  self.navigator.cancelTask()  # discover  self.printContext('Robot discovers person! Robot is on its way!')  self.nav2pose(self.person\_point[0], self.person\_point[1], 1.0)  # returning  self.printContext(' The robot is returning! ')  result=self.nav2pose(self.return\_point[0], self.return\_point[1], 1.0)  if result==TaskResult.SUCCEEDED:  self.printContext('Successfully returned')  rclpy.shutdown()  return  def printContext(self, context):  space\_len = 10  str = '#' + ' ' \* space\_len + context + ' ' \* space\_len + '#'  print(Fore.GREEN + '#' \* len(str))  print(Fore.GREEN + str)  print(Fore.GREEN + '#' \* len(str))  print(Style.RESET\_ALL)  def main():  rclpy.init()  result\_subscriber = ResultSubscriber()  rclpy.spin(result\_subscriber)  result\_subscriber.destroy\_node()  rclpy.shutdown() |

编译所有功能包后，首先启动导航节点

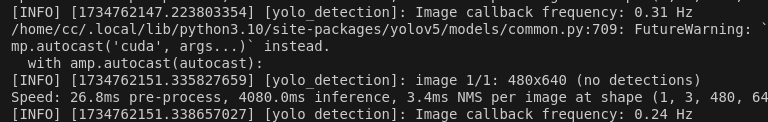
|  |
| --- |
| ~/RobotProject/src$ ros2 launch robot\_navigation navigating.launch.py |

设置2Dpoint estimate为机器人的初始导航方向，展示RobotModel后，在Rviz和gazebo中初始情况如下图：

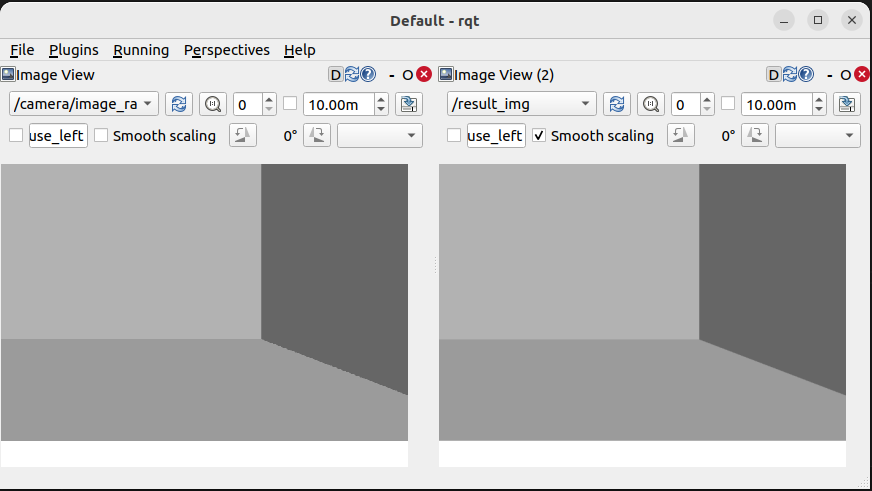


然后在新的命令窗口中启动yolov5的ros2节点订阅摄像机内容

|  |
| --- |
| ~/RobotProject$ ros2 run yolo\_detection yolo\_detect\_2d |



新建命令窗口，启动rqt，添加对应话题的plugins。

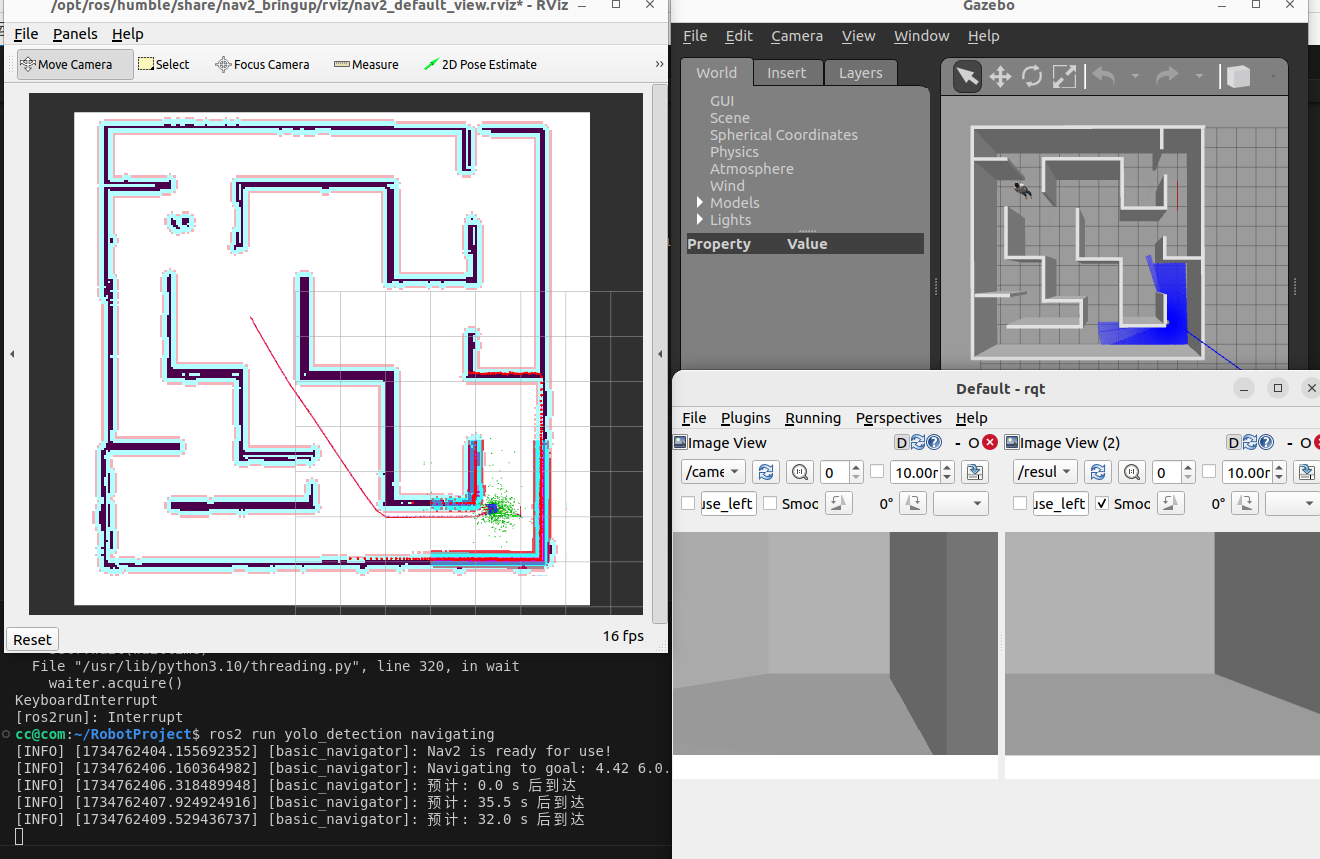


已经得到了对应的摄像机画面。

新建命令窗口，启动编写的巡航任务节点。正式开始我们的巡航任务。

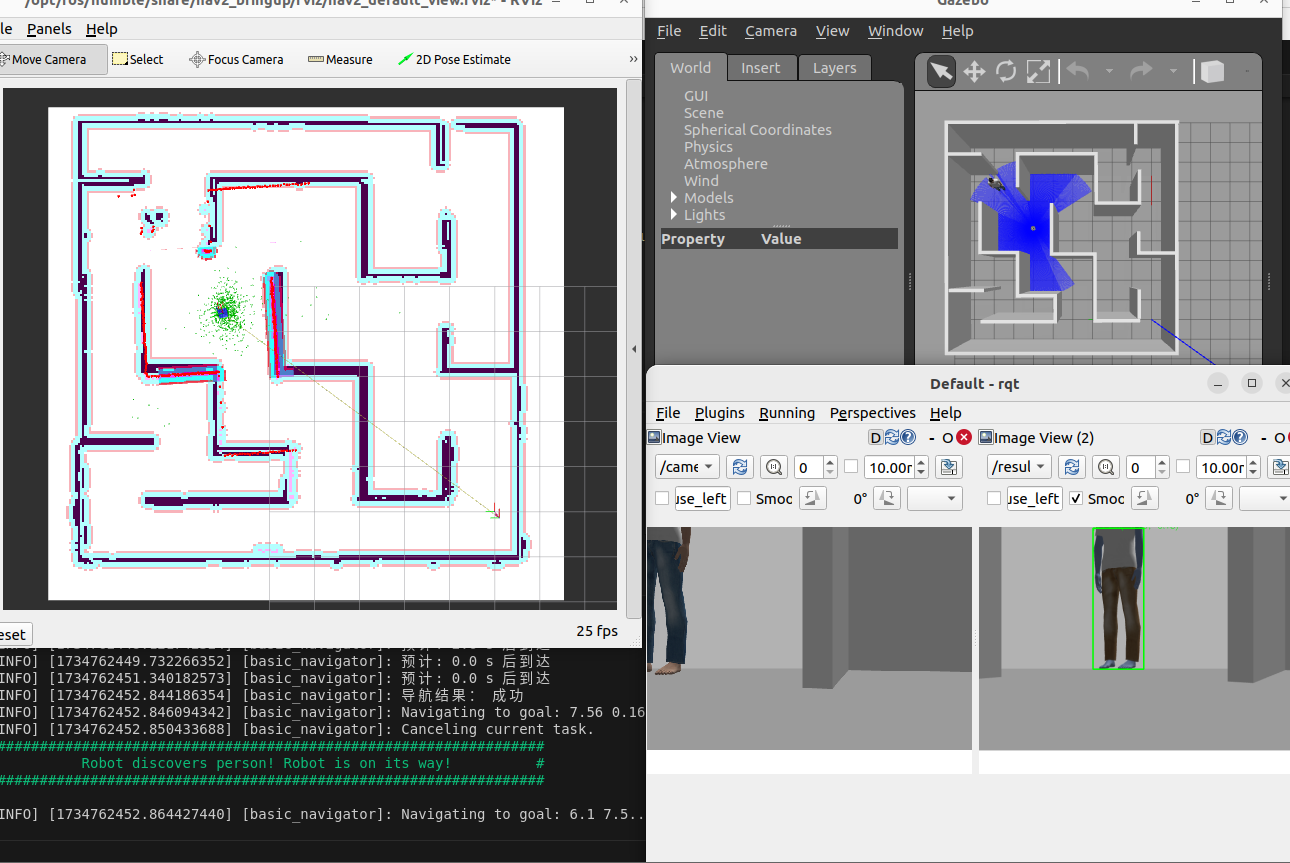
|  |
| --- |
| ~/RobotProject$ ros2 run yolo\_detection navigating |

我们设置了以下5个巡航点，[4.42,6.0,1.0],[7.56,0.16,1.0],[7.94,6.95,-1.0],[-0.41,8.62,1.0],[0.0,0.0,1.0],使机器人尽可能遍历整个地图。

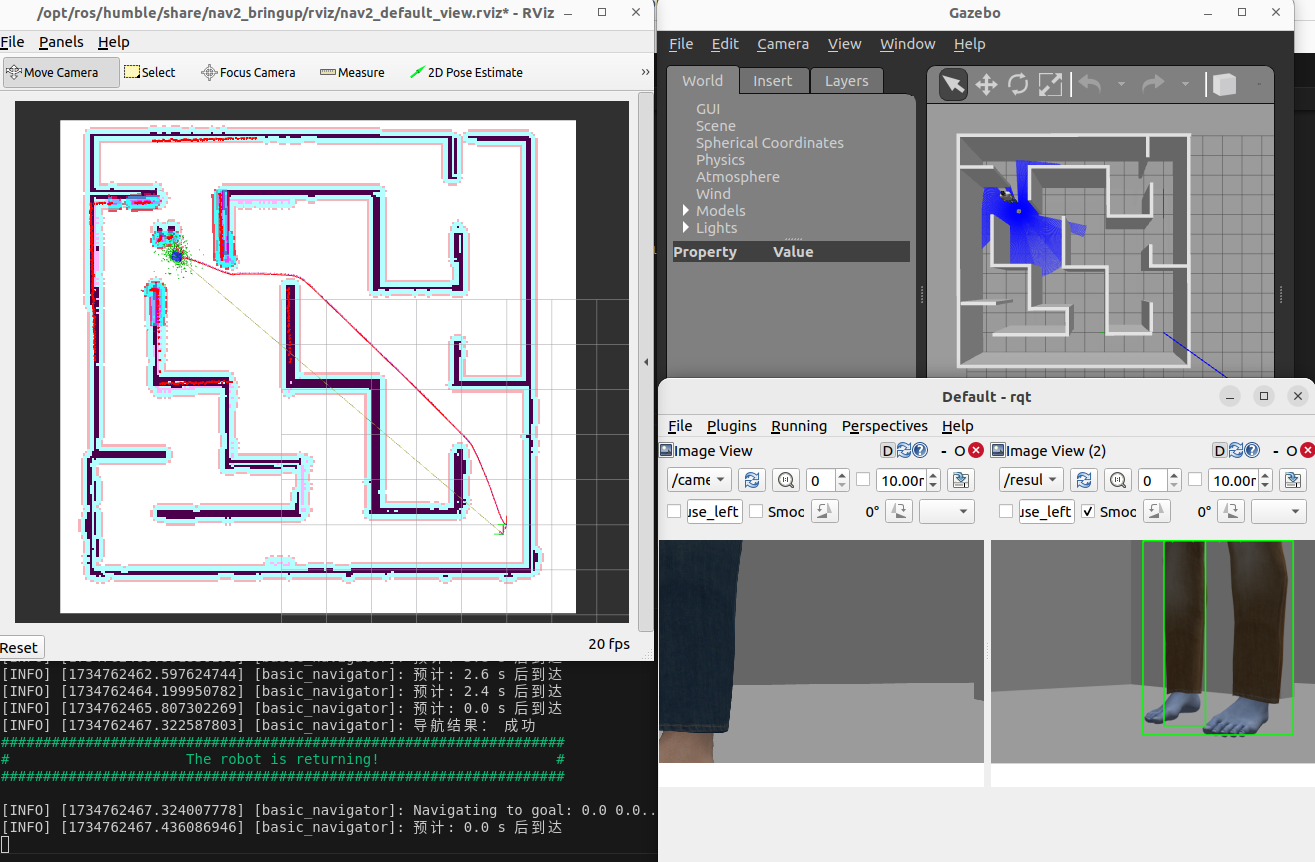




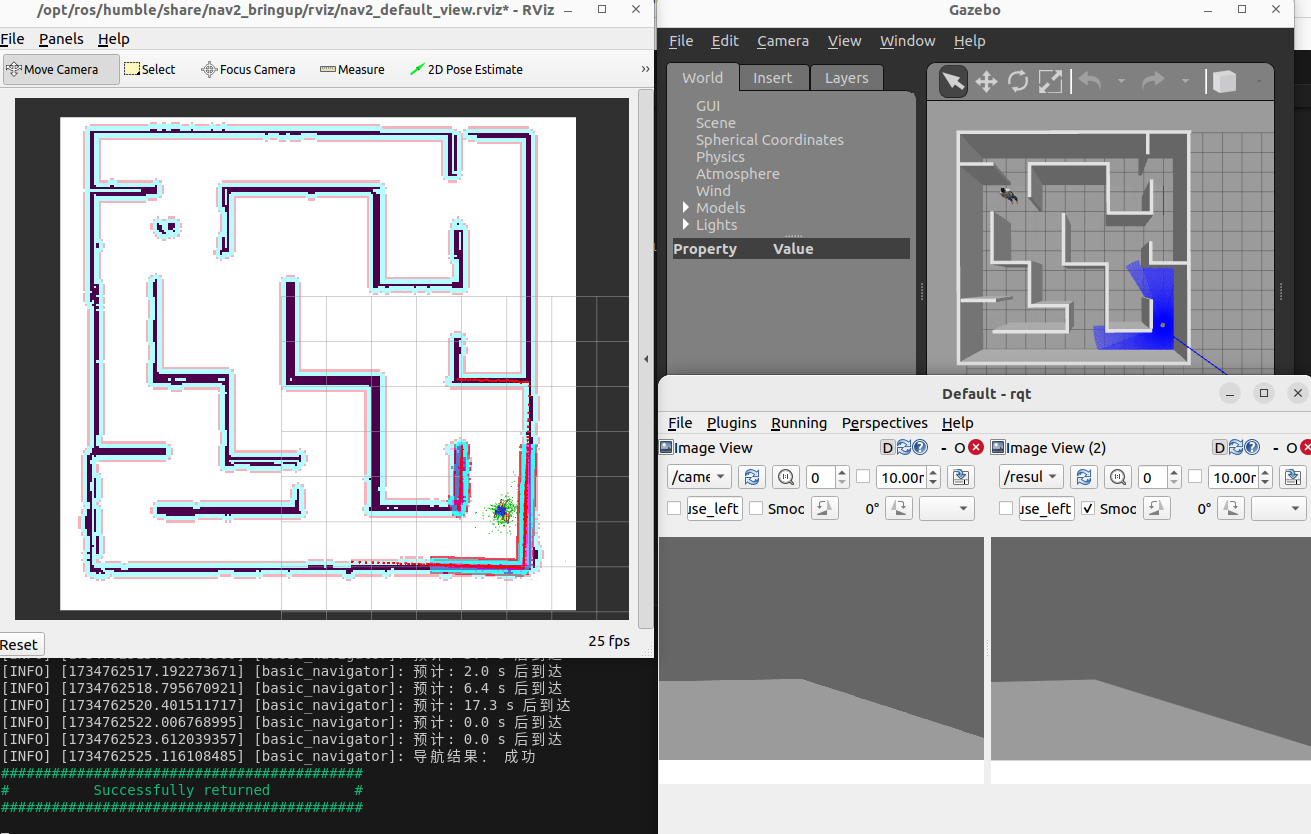
机器人行走至第一个途径点过程中摄像机识别到了人，前往人的位置。



到达人体模型位置后，机器人执行返航任务。



机器人成功返航至初始点，巡航任务成功结束。



1. 项目总结

本项目成功实现了基于Nav2与YOLOv5的巡航搜救机器人，其主要任务是在仿真环境中进行巡航、导航及目标识别，模拟了自然灾害下的搜救工作。这一系统集成了多个先进的技术，使得机器人能够在复杂的环境中有效地完成既定任务。

在创新点方面，本项目结合了激光雷达、摄像头等多种传感器的数据，有效提升了环境感知的精确性。机器人能够在巡航过程中实时检测目标并调整任务，为机器人在动态环境中工作提供了强大的支持。通过使用ROS2的多线程处理，使得系统能够同时执行导航和目标检测，提升了整体工作效率。

受限于虚拟机性能，图像识别响应速度较慢，返回结果有时不及时，在未来的工作中可考虑在多样的实际环境中测试并提升机器人的适应性，增加更多的传感器以应对复杂场景。随着深度学习技术的发展，也可持续更新YOLO模型以提高目标检测的准确率与实时性。除此之外我们也可以探索更为高效的任务调度和智能决策机制，以提升机器人在各种突发情况中的反应能力与处理效率。

本项目的成功实施不仅展示了各项技术的集成应用能力，也为未来的科研与实际应用打下了良好的基础。通过不断的优化与创新，巡航搜救机器人将能够在更广泛的领域中发挥其重要作用，尤其是在紧急救援等关键场景中

四、参考文献

**[1]** https://fishros.com/d2lros2/#/

**[2]** https://github.com/fishros/yolov\_ros2

**[3]** Ella, S., & Hossain, M. (2020). A Survey of Robot Navigation Techniques using SLAM. *International Journal of Advanced Computer Science and Applications*, 11(5), 187-194.

**[4]** Chen, H., & Hsiao, C. (2021). A Review of Object Detection Technologies for Autonomous Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 22(3), 1695-1710.