ROBOWORKS

Robofleet User Manual

Prepared by: Wayne Liu & Janette Lin
1 May 2024
Version #: 20240501

SUMMARY

This document mainly explains the usage of the multi-robot formation function package named wheeltec_multi.

This document is divided into four parts:
- The first part is mainly about the introduction of the multi-robot formation method;
- the second part mainly describes the ROS multi-machine communication settings, including ROS construction of multi-machine communication and the problems that may be encountered in the process of ROS communication;
- the third part mainly describes the operation steps of multi-machine time synchronization;
- the fourth part expounds the specific use of the multi-machine formation function package.

The purpose of this document is an introduction to multi-agent robotic system and allows user to start the multi-robot formation project quickly.

TABLE OF CONTENTS

INTRODUCTION TO MULTI-AGENT ALGORITHMS

1.1 Multi-agent formation algorithms

This ROS package presents a typical problem of multi-agents in collaborative control during a formation drive. This tutorial lays a foundation for future development on this topic. Formation control algorithm refers to an algorithm that controls multiple agents to form a specific formation to perform a task. Collaboration refers to the cooperation between multiple agents using a certain constraint relationship to complete a task. Take the multi-robot formation drive as an example, collaboration means that multiple robots form a desired formation together. Its essence is a certain mathematical relationship is satisfied between the positions of each robot. Formation methods are mainly divided into centralized formation control and distributed formation control. Centralized formation control methods mainly include virtual structure method, graphical theory method and model predictive method. Distributed formation control methods mainly include leader-follower method, behaviour-based method and virtual structure method.

This ROS package applies the leader-follower method in distributed formation control method to execute the multi-robot formation drive. One robot in the formation is designated as the leader, and other robots are designated as slaves to follow the leader. The algorithm uses the movement trajectory of the leading robot to set the coordinates to be tracked by the following robots with certain direction and speed. By correcting the position deviations from the tracking coordinates, the followers eventually will reduce the deviation between the follower and the expected tracking coordinates to zero in order to achieve the objectives of formation drive. In this way, the algorithm is relatively less complicated.

1.2 Obstacle avoidance algorithms

A common obstacle avoidance algorithm is the artificial potential field method. The movement of the robot in a physical environment is regarded as a movement in a virtual artificial force field. The nearest obstacle is identified by LiDAR. The obstacle provides a repulsive force field to generate repulsion to the robot and the target point provides a gravitational field to generate gravitational force to the robot. In this way, it controls the motion of the robot under the combined action of repulsion and attraction.

This ROS package is an improvement based on the artificial potential field method. Firstly, the formation algorithm calculates the linear and angular velocity of the Slave follower. Then it increases or decreases the linear and angular velocity according to the obstacle avoidance requirements.When the distance between the Slave follower and the obstacle is closer, the repulsion force of the obstacle to the Slave follower is greater. Meanwhile the change of the linear velocity and the angular velocity variations are greater. When the obstacle is closer to the front of the Slave follower, the repulsion of the obstacle to the Slave follower becomes greater (the front repulsion is the biggest and the side repulsion is the smallest). As a result, the variations of the linear velocity and the angular velocity are greater. Through the artificial potential field method, it improves a solution when a robot could stop responding in front of an obstacle. This serves a purpose of better obstacle avoidance.

MULTI-AGENT COMMUNICATION SETUP

Multi-agent communication is one of the key steps to complete a multi-robot formation. When the relative positions of multiple robots are unknown, the robots need to share each other's information through communication to facilitate the establishment of connections. ROS distributed architecture and network communications are very powerful. It is not only convenient for inter-process communication, but also for communication between different devices. Through network communication, all nodes can run on any computer. The main tasks such as data processing are completed on the host side. The slave machines are responsible for receiving environmental data collected by various sensors. The host here is the manager that runs the Master node in ROS. The current multi-agent communication framework is through a node manager and a parameter manager to handle communications among multiple robots.

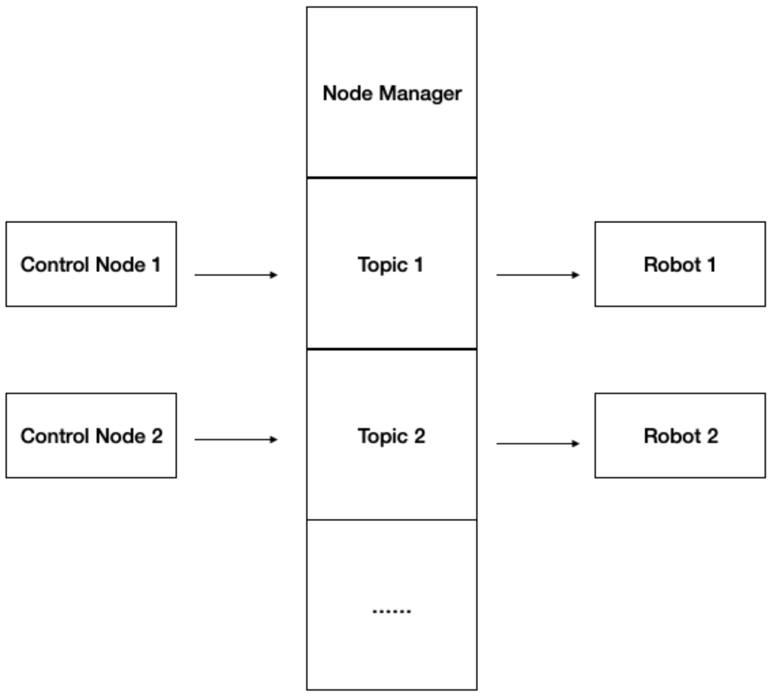


Fig 2-0-1 ROS Multi-agent Communication Framework

2.1 The steps to set up multi-agent communications

2.1.1 Set up ROS Controls in the same network

There are 2 ways of setting up Master/Slave ROS Controls under the same network.

Option 1:

The Master Host creates a local wifi by running the Master node manager. Generally, one of the robots who is designated as the master creates this wifi network. Other robots or virtual machines join this wifi network as slaves.

Options 2:

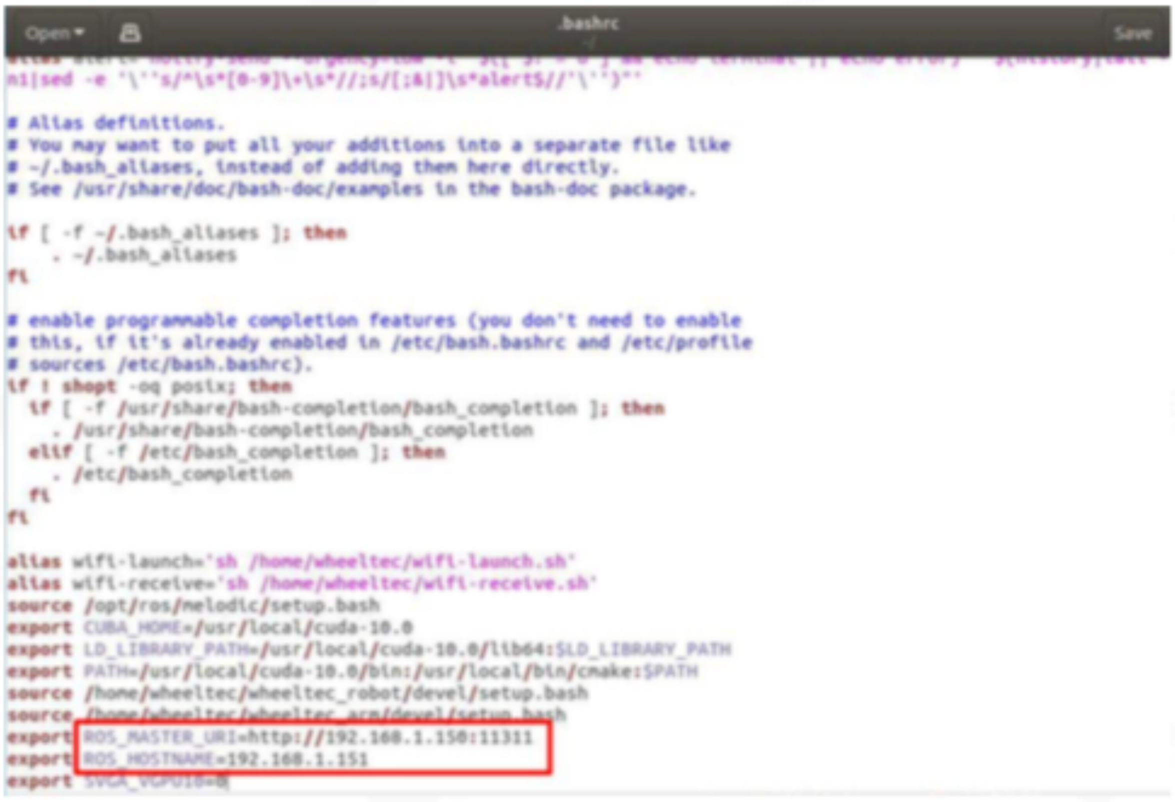
The local wifi network is provided by a third-party router as an information relay center. All robots are connected to the same router. The router can also be used without internet connection. Select one of the robots as the master and run the Master node manager. The other robots are designated as slaves and run the master node manager from the master.

The decision on which option to choose depends on your project requirements. If the number of robots that need to communicate is not at a large amount, Option 1 is recommended since it saves cost and it is easy to set up. When the number of robots is at large quantity, Option 2 is recommended. The constraint on computing power of the ROS master control and limited onboard wifi bandwidth can easily cause delays and network disruptions. A router can easily fix these issues.

Please note that when performing multi-agent communication, if the virtual machine is used as a ROS slave, its network mode needs to be set to bridge mode.

2.1.2 Configure Master/Slave environment variables

After all the ROS masters are all in the same network, the environment variables for multi-agent communication need to be set. This environment variable is configured in the .bashrc file in the main directory. Run the gedit ~/.bashrc command to launch it. Please note that both the .bashrc files of the master and the slave in multi-agent communication need to be configured. What needs to be changed are the IP addresses at the end of the file. The two lines of are ROS_MASTER_URI and ROS_HOSTNAME, as shown in Figure 2-1-4. The ROS_MASTER_URI and ROS_HOSTNAME of the ROS host are both local IPs. The ROS_MASTER_URI in the ROS slave .bashrc file needs to be changed to the host's IP address while ROS_HOSTNAME remains as a local IP address.



```
[illegible]
n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

alias wifi-launch='sh /home/wheeltec/wifi-launch.sh'
alias wifi-receive='sh /home/wheeltec/wifi-receive.sh'
source /opt/ros/melodic/setup.bash
export CUDA_HOME=/usr/local/cuda-10.0
export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64:$LD_LIBRARY_PATH
export PATH=/usr/local/cuda-10.0/bin:/usr/local/bin/cmake:$PATH
source /home/wheeltec/wheeltec_robot/devel/setup.bash
source /home/wheeltec/wheeltec_arm/devel/setup.bash
export ROS_MASTER_URI=http://192.168.1.150:11311
export ROS_HOSTNAME=192.168.1.151
export SVGA_VGPU10=0
```

Fig 2-1-4 Environment Configuration File .bashrc

ROS multi-machine communication is not constrained by ROS release version. In the process of multi-machine communication, one should be aware of the following:

1) The operation of the ROS slave program depends on the ROS master program of the ROS master device. The ROS master program must launch first on the master device before executing the slave program on the slave device.

2) The IP addresses of the master and slave machines in multi-machine communication need to be in the same network. This means the IP address and the subnet mask are under the same network.

3) ROS_HOSTNAME in the environment configuration file .bashrc is not recommended to use localhost. It is recommended to use a specific IP address.

4) In the case that the slave IP address is not set correctly, the slave device can still access the ROS master but cannot input control information.

5) If the virtual machine participates in the multi-agent communication, its network mode needs to be set to bridge mode. Static IP cannot be selected for the network connection.

6) Multi-machine communication cannot view or subscribe to topics of message data type that do not exist locally.

7) You can use the Little Turtle simulation demo to verify whether the communication between the robots is successful:

a. Run from the master

```
roscore                                   #launch ROS services
rosrun turtlesim turtlesim_node           #launch turtlesim interface
```

b. Run from the slave

```
rosrun turtlesim turtle_teleop_key        #launch keyboard control node for turtlesim
```

If you can manipulate the turtle movements from the keyboard on the slave, it means the master/slave communication has been established successfully.

2.2 Automatic Wifi connection in ROS

The below procedures explain how to configure the robot to automatically connect to the host network or router network.

2.2.1 Automatic Wifi connection setup for Jetson Nano

1. Connect Jetson Nano via VNC remote tool or directly to the computer screen. Click on wifi icon on the top right corner then click "Edit Connections..”

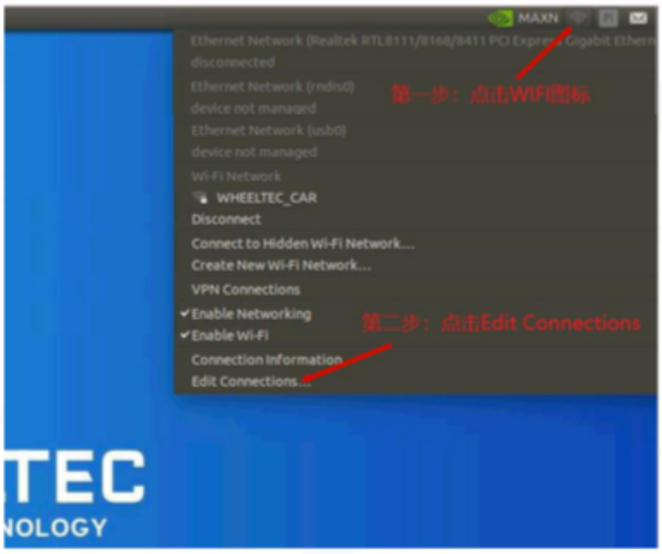


Fig 2-2-1 Jetson Nano Connection to Wifi

2. Click the + button in Network Connections:

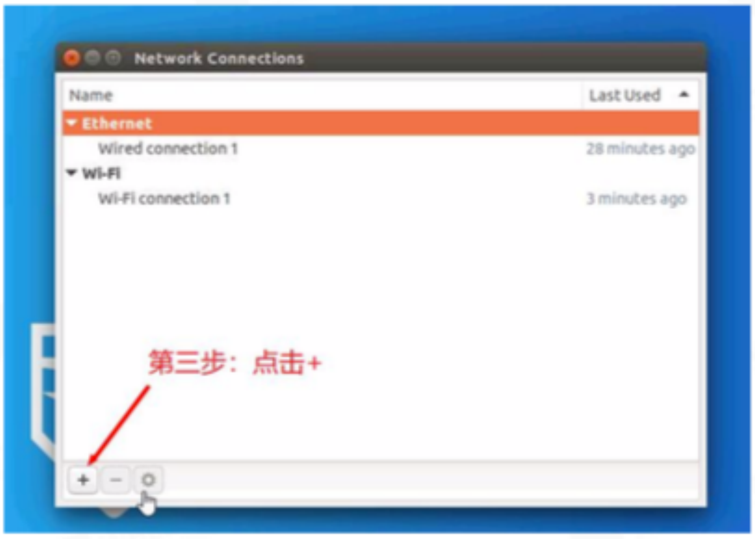


Fig 2-2-2 Jetson Nano Connection to Wifi

3. Under the “Choose a Connection Type” window, click the drop-down menu and click “Create…” button:

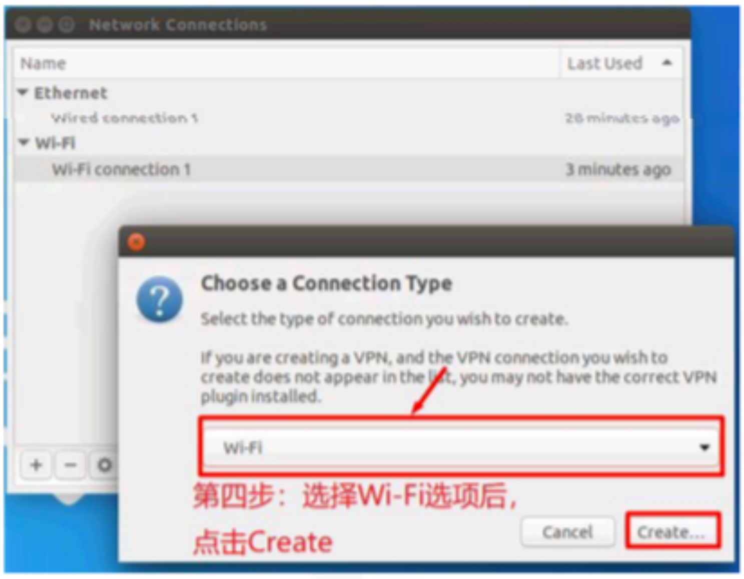
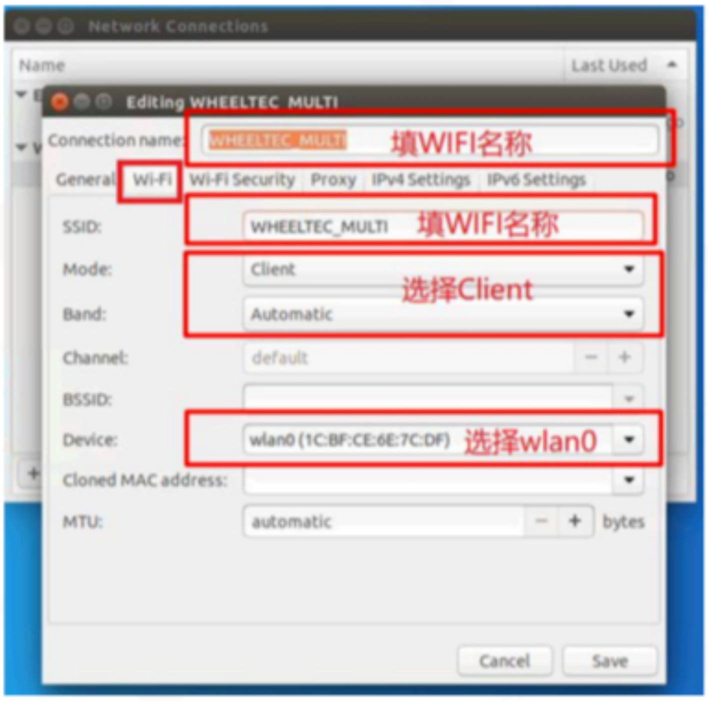

Network Connections
Name
Last Used
Ethernet
Wired connection 1
28 minutes ago
Wi-Fi
Wi-Fi connection 1
3 minutes ago
Choose a Connection Type
Select the type of connection you wish to create.
If you are creating a VPN, and the VPN connection you wish to create does not appear in the list, you may not have the correct VPN plugin installed.
Wi-Fi
第四步：选择Wi-Fi选项后，
点击Create
Cancel
Create…

Fig 2-2-3 Jetson Nano Connection to Wifi

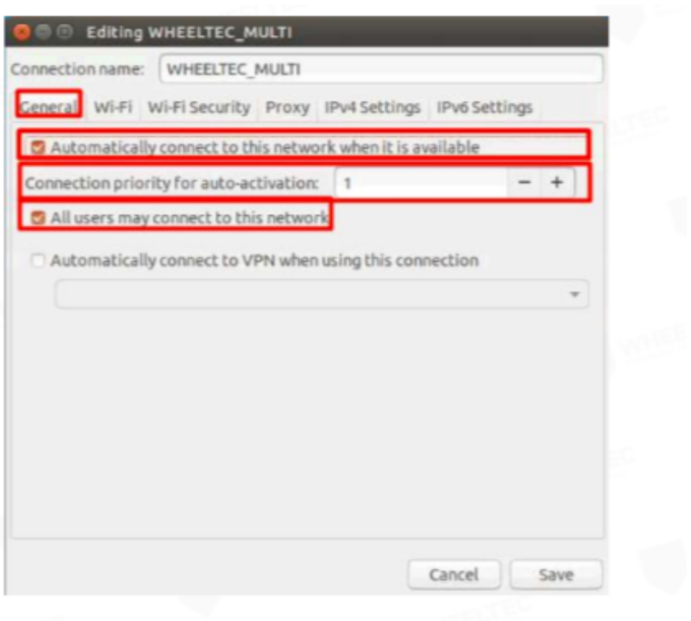
4. In Control Panel, click Wifi option. Enter the Wifi name to connect in “Connection Name” and SSID fields. Select “Client” in “Mode” dropdown menu and select “wlan0” in “Device” dropdown menu.



Connection name: WHEELTEC_MULTI
SSID: WHEELTEC_MULTI
Mode: Client
Band: Automatic
Channel: default
BSSID:
Device: wlan0 (1C:BF:CE:6E:7C:DF)
Cloned MAC address:
MTU: automatic bytes

Fig 2-2-4 Jetson Nano Connection to Wifi

5. In Control Panel, click the "General" option and check "Automatically connect to this network…”. Set the connection priority to 1 in the "Connection priority for auto-activation" option. Check "All users may connect to this network” option. When the option is set to 0 in "Connection priority for auto-activation” for other wifi, this means this is the preferred wifi network in the past.



Editing WHEELTEC_MULTI

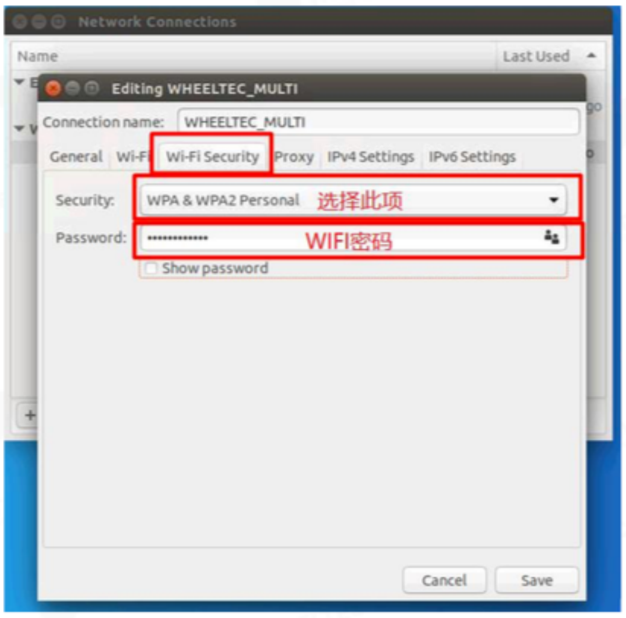
Connection name: WHEELTEC_MULTI

General | Wi-Fi | Wi-Fi Security | Proxy | IPv4 Settings | IPv6 Settings

- [✓] Automatically connect to this network when it is available
- Connection priority for auto-activation: 1
- [✓] All users may connect to this network
- [ ] Automatically connect to VPN when using this connection

Fig 2-2-5 Jetson Nano Connection to Wifi

6. Click “Wi-Fi Security” option in Control Panel. Select “WPA & WPA2 Personal” in “Security” field. Then enter the Wifi password in “Password” field.



Editing WHEELTEC_MULTI

Connection name: WHEELTEC_MULTI

General | Wi-Fi | Wi-Fi Security | Proxy | IPv4 Settings | IPv6 Settings

Security: WPA & WPA2 Personal 选择此项

Password: ••••••••••••• WIFI密码

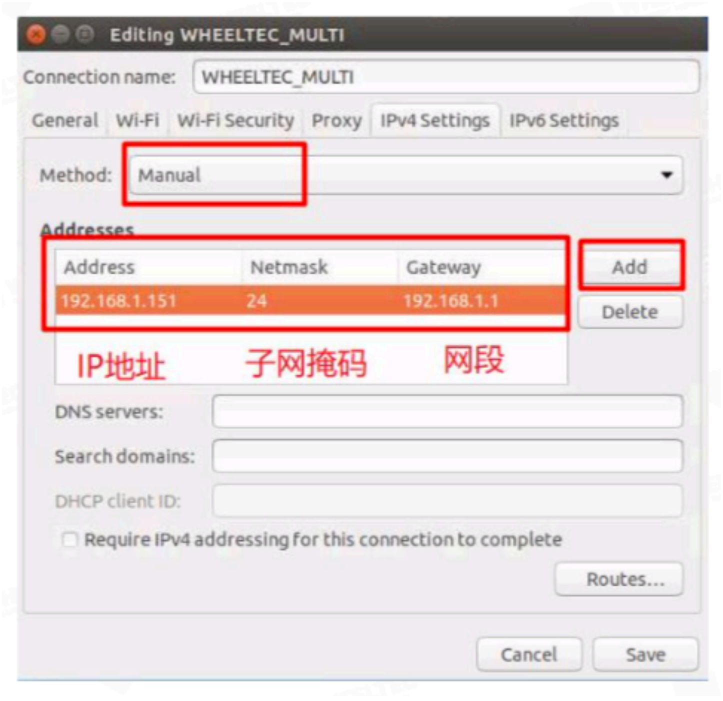
- [ ] Show password

Fig 2-2-6 Jetson Nano Connection to Wifi

Note:

If the robot cannot automatically connect to the wifi network after booting when the wifi priority is set to 0, it may be caused by a problem of weak wifi signal. In order to avoid this problem, you can choose to delete all the wifi options that have been connected in the past. Only keep the wifi network created by the host or the router.

Click the "IPv4 Settings" option in the network settings control panel. Select "Manual" option in the “Method" field. Then click "Add", fill in the IP address of the slave machine in “Address" field. Fill in "24" in “Netmask" field. Fill in the IP network segment in "Gateway". Change the last three digits of the IP network segment to "1". The main purpose of this step is to fix the IP address. After this is completed for the first time, the IP address will remain unchanged when connecting to the same WIFI subsequently.



Editing WHEELTEC_MULTI

Connection name: WHEELTEC_MULTI

Method: Manual

Addresses

Address	Netmask	Gateway
192.168.1.151	24	192.168.1.1

Fig 2-2-7 Jetson Nano Connection to Wifi

After all settings are configured, click "save" to save the settings. After the saving is successful, the robot will automatically connect to the network of the host or router when it is powered on.

Note:

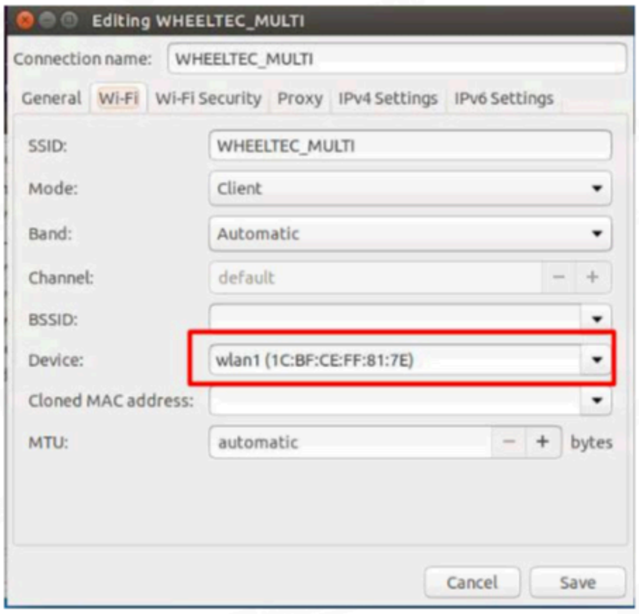
1) The IP address set here needs to be the same as the IP address set in the .bashrc file in Section 2.1.
2) The IP address of the master and each slave must be unique.
3) The master and slave IP addresses need to be in the same network segment.
4) You must wait for the host or router to send out WiFi signal before the slave robot can be powered on and automatically connect to the WiFi network.
5) After the setting is configured, if the robot cannot automatically connect to the WiFi when it is turned on, please plug and unplug the network card and try connecting again.

2.2.2 Automatic Wifi connection setup for Raspberry Pi

The procedure for Raspberry Pi is the same as Jetson Nano.

2.2.3 Automatic Wifi connection setup for Jetson TX1

The setup in Jetson TX1 is almost the same as in Jetson Nano with one exception that Jetson TX1 should select the device of "wlan1" in "Device" in the network settings control panel.



Connection name: WHEELTEC_MULTI

SSID: WHEELTEC_MULTI

Mode: Client

Band: Automatic

Channel: default

BSSID:

Device: wlan1 (1C:BF:CE:FF:81:7E)

Cloned MAC address:

MTU: automatic bytes

Fig 2-2-8 Jetson Nano Connection to Wifi

MULTI-AGENT SYNCHRONISATION SETUP

In the multi-agent formation project, the multi-agent time synchronization setting is a crucial step. In the process of the formation, many problems will be caused due to the asynchronous system time of each robot. Multi-agent time synchronization is divided into two situations, namely, the situation that both the master and slave robots are connected to the network and the situation that both are disconnected from the network.

3.1 Successful master/slave network connection

After the multi-agent communication is configured, if the master and slave machines can successfully connect to the network, they will automatically synchronize network time. In this case, no further actions are required to achieve time synchronization.

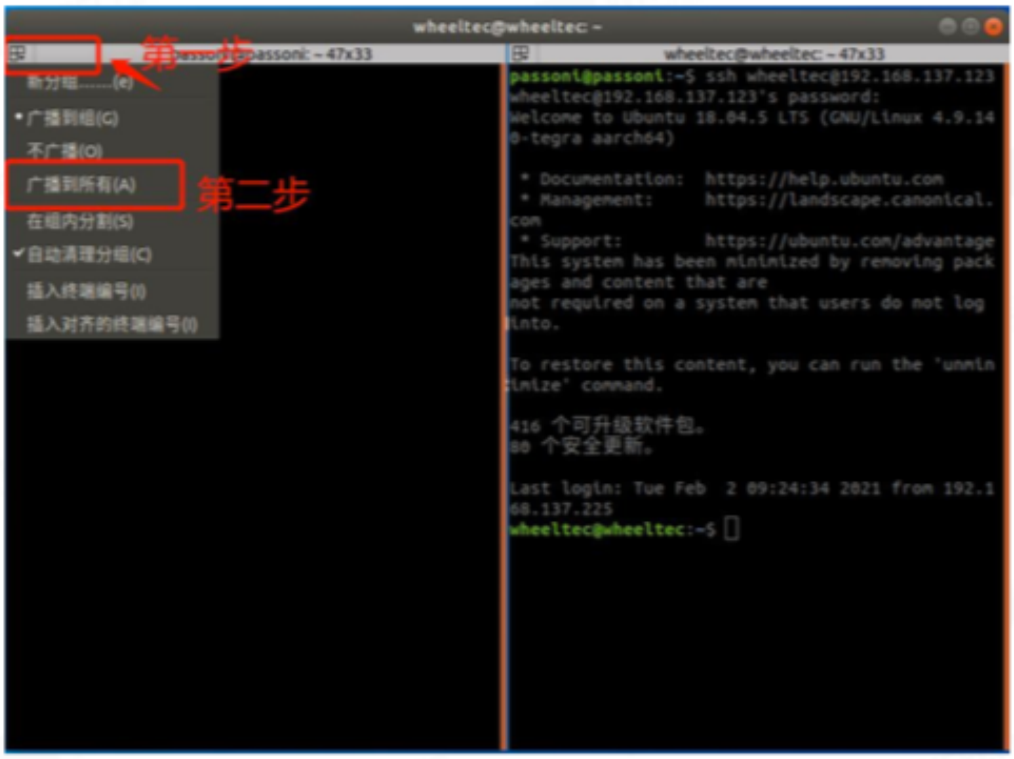
3.2 Troubleshooting network dis-connections

After the multi-agent communication is configured, if the master and slave devices cannot successfully connect to the network, it is necessary to manually synchronize the time. We will use the date command to complete the time setting.

First, install the terminator tool. From the terminator tool, use the window splitting tool to place the control terminals of the master and slave into the same terminal window (right-click to set a split window, and log in to the master and slave machines by ssh in different windows).

```
sudo apt-get install terminator            # Download terminator to split terminal window
```

Click the button on the top left, select the option [Broadcast to all]/[Broadcast all], enter the following command. Then use the terminator tool to set the same time for the master and slave.



```
passoni@passoni:~$ ssh wheeltec@192.168.137.123
wheeltec@192.168.137.123's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 4.9.14
0-tegra aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.
com
 * Support:        https://ubuntu.com/advantage
This system has been minimized by removing pack
ages and content that are
not required on a system that users do not log
into.

To restore this content, you can run the 'unmin
imize' command.

416 个可升级软件包。
80 个安全更新。

Last login: Tue Feb  2 09:24:34 2021 from 192.1
68.137.225
wheeltec@wheeltec:~$ 
```

Fig 3-2-1 Steps to enter Broadcast Command

```
sudo date -s "2022-01-30 15:15:00"                # Manual time setup
```

MULTI-AGENT ROS PACKAGE

4.1 ROS Package Introduction

4.1.1 Set up slave name

In the wheeltec_multi function package, it is necessary to set a unique name for each slave robot in order to avoid errors. For example, No. 1 for slave1 and No. 2 for slave2 etc..

The purpose of setting different names is to group running nodes and distinguish them by different namespaces. For example, the radar topic of slave 1 is: /slave1/scan, and the LiDAR node of slave 1 is: /slave1/laser.

4.1.2 Set up slave coordinates

The wheeltec_multi package can implement custom formations. When different formations are required, just modify the desired coordinates of the slave robots. Slave_x and slave_y are the x and y coordinates of the slave with the master as the original reference point. The front of the master is the positive direction of the x coordinate, and the left side is the positive direction of the y coordinate. After the setting is completed, a TF coordinate slave1 will be issued as the expected coordinate of the slave.

If there are one master and two slaves, the following formation can be set:

1) Horizontal formation: You can set the coordinates of the slave on the left to: slave_x:0, slave_y: 0.8, and the coordinates of the slave on the right to: slave_x:0, slave_y:-0.8.

2) Column formation: The coordinates of one slave can be set to: slave_x:-0.8, slave_y:0, and the coordinates of the other slave can be set to: slave_x:-1.8, slave_y:0.

3) Triangular formation: The coordinates of one slave can be set to: slave_x:-0.8, slave_y: 0.8, and the coordinates of the other slave can be set to: slave_x:-0.8, slave_y:-0.8.

Other formations can be customized as needed.

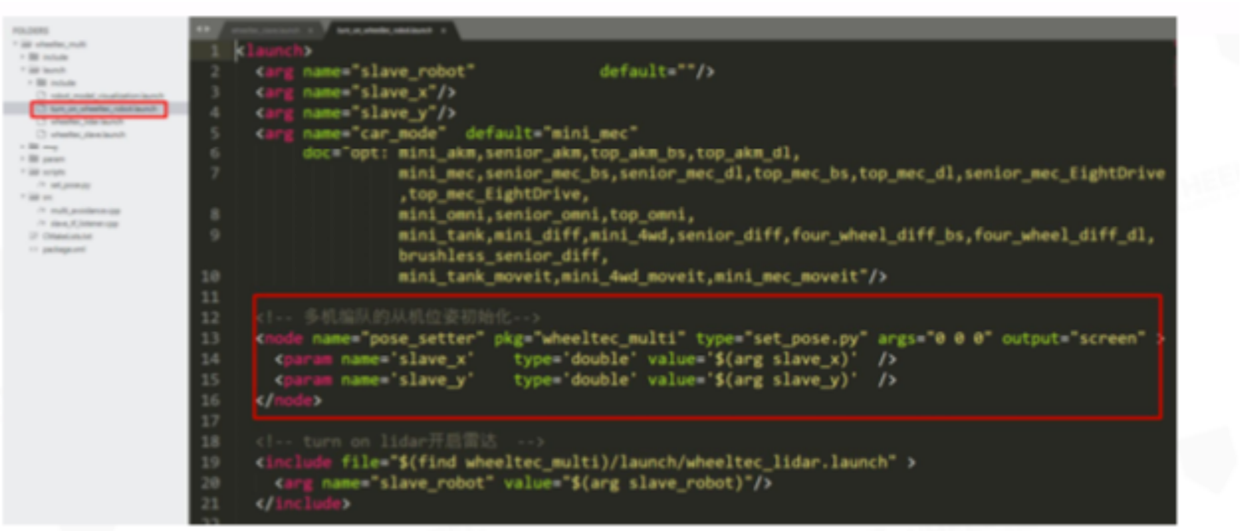
Note:

The recommended distance between the two robots is set to 0.8, and it is recommended not to be lower than 0.6. The distance between the slaves and the master is recommended to be set below 2.0. The farther it is from the master, the greater the linear speed of the slave is when the master is turning. Due to the limitation of the maximum speed, the speed of the slave will deviate if it does not meet the requirements. The robot formation will become chaotic.

4.1.3 Initialisation of the slave position

The initial position of the slave is at the expected coordinates by default. Before running the program, just place the slave robot close to its expected coordinates to complete the initialization.

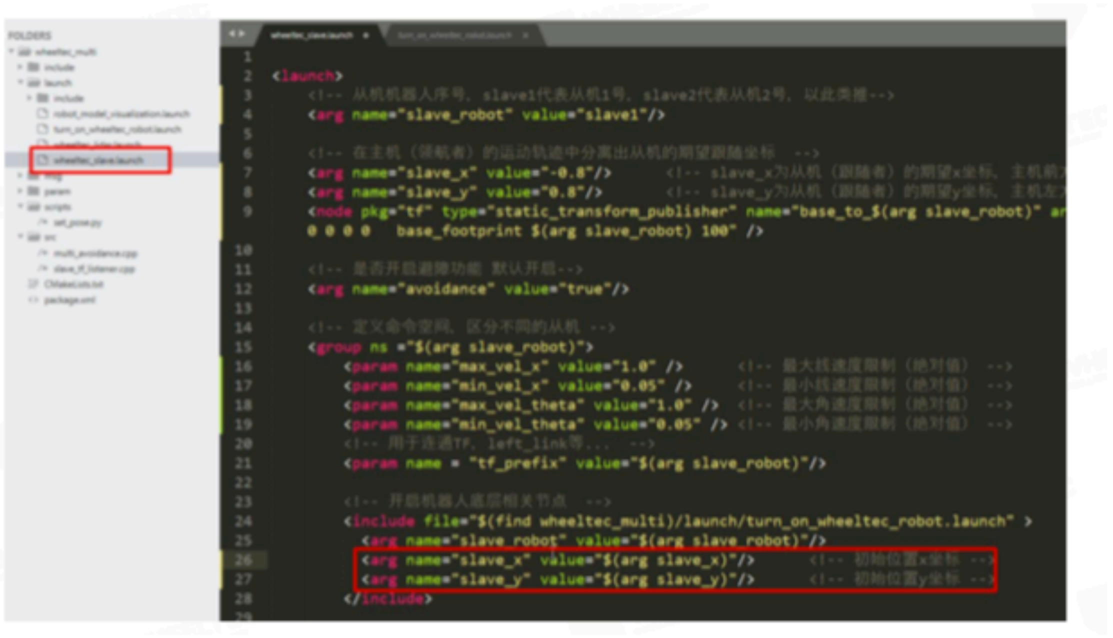
This function is implemented by the pose_setter node in the file named turn_on_wheeltec_robot.launch in the wheeltec_multi package, as shown in Figure 4-1-3.



```
<launch>
  <arg name="slave_robot"            default=""/>
  <arg name="slave_x"/>
  <arg name="slave_y"/>
  <arg name="car_mode"  default="mini_mec"
      doc="opt: mini_akm,senior_akm,top_akm_bs,top_akm_dl,
                mini_mec,senior_mec_bs,senior_mec_dl,top_mec_bs,top_mec_dl,senior_mec_EightDrive
                ,top_mec_EightDrive,
                mini_omni,senior_omni,top_omni,
                mini_tank,mini_diff,mini_4wd,senior_diff,four_wheel_diff_bs,four_wheel_diff_dl,
                brushless_senior_diff,
                mini_tank_moveit,mini_4wd_moveit,mini_mec_moveit"/>

  <!-- 多机编队的从机位姿初始化-->
  <node name="pose_setter" pkg="wheeltec_multi" type="set_pose.py" args="0 0 0" output="screen" >
    <param name='slave_x'    type='double' value='$(arg slave_x)'  />
    <param name='slave_y'    type='double' value='$(arg slave_y)'  />
  </node>

  <!-- turn on lidar开启雷达  -->
  <include file="$(find wheeltec_multi)/launch/wheeltec_lidar.launch" >
    <arg name="slave_robot" value="$(arg slave_robot)"/>
  </include>
```

Fig 4-1-3 Initialisation Node of the Slave Position

If the user wants to customize the initial position of the slave, he or she only needs to set the slave_x and slave_y values as shown in Figure 4-1-4 in wheeltec_slave.launch. The slave_x and slave_y values will be passed to turn_on_wheeltec_robot.launch and assigned to the pose_setter node. Just place the robot in a custom position before running the program.



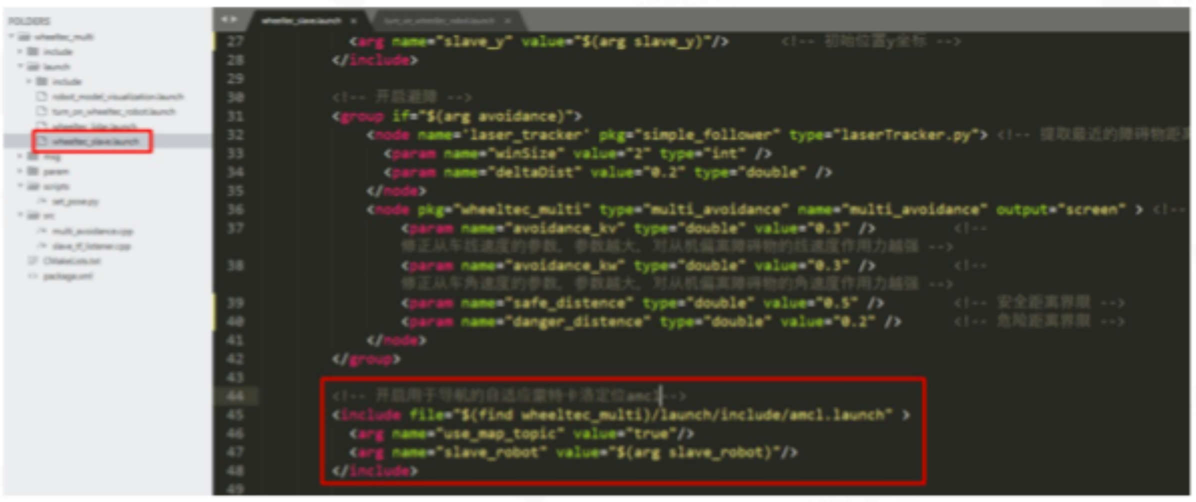
```

<launch>
    <!-- 从机机器人序号，slave1代表从机1号，slave2代表从机2号，以此类推-->
    <arg name="slave_robot" value="slave1"/>

    <!-- 在主机（领航者）的运动轨迹中分离出从机的期望跟随坐标  -->
    <arg name="slave_x" value="-0.8"/>      <!-- slave_x为从机（跟随者）的期望x坐标，主机前
    <arg name="slave_y" value="0.8"/>       <!-- slave_y为从机（跟随者）的期望y坐标，主机左
    <node pkg="tf" type="static_transform_publisher" name="base_to_$(arg slave_robot)" ar
    0 0 0 0   base_footprint $(arg slave_robot) 100" />

    <!-- 是否开启避障功能 默认开启-->
    <arg name="avoidance" value="true"/>

    <!-- 定义命令空间，区分不同的从机 -->
    <group ns ="$(arg slave_robot)">
        <param name="max_vel_x" value="1.0" />      <!-- 最大线速度限制（绝对值） -->
        <param name="min_vel_x" value="0.05" />     <!-- 最小线速度限制（绝对值） -->
        <param name="max_vel_theta" value="1.0" /> <!-- 最大角速度限制（绝对值） -->
        <param name="min_vel_theta" value="0.05" /> <!-- 最小角速度限制（绝对值） -->
        <!-- 用于连通TF，left_link等...  -->
        <param name = "tf_prefix" value="$(arg slave_robot)"/>

        <!-- 开启机器人底层相关节点  -->
        <include file="$(find wheeltec_multi)/launch/turn_on_wheeltec_robot.launch" >
          <arg name="slave_robot" value="$(arg slave_robot)"/>
          <arg name="slave_x" value="$(arg slave_x)"/>       <!-- 初始位置x坐标 -->
          <arg name="slave_y" value="$(arg slave_y)"/>       <!-- 初始位置y坐标 -->
        </include>
```

Fig 4-1-4 Slave initial position can be customised.

4.1.4 Position Configuration

In a multi-agent formation, the first problem to be solved is the positioning of the master and the slave. The master will construct a 2D map first. After creating and saving the map, run the 2D navigation package and use the adaptive Monte Carlo positioning algorithm (amcl positioning) in the 2D navigation package to configure the positioning of the master.

Since the master and the slaves are in the same network and share the same node manager, the master has launched the map from the 2D navigation package, all the slaves can use the same map under the same node manager. Therefore, the slave does not need to create a map. In wheeltec_slave.launch, run Monte Carlo positioning (amcl positioning), the slaves can configure their positions by using the map created by the master.



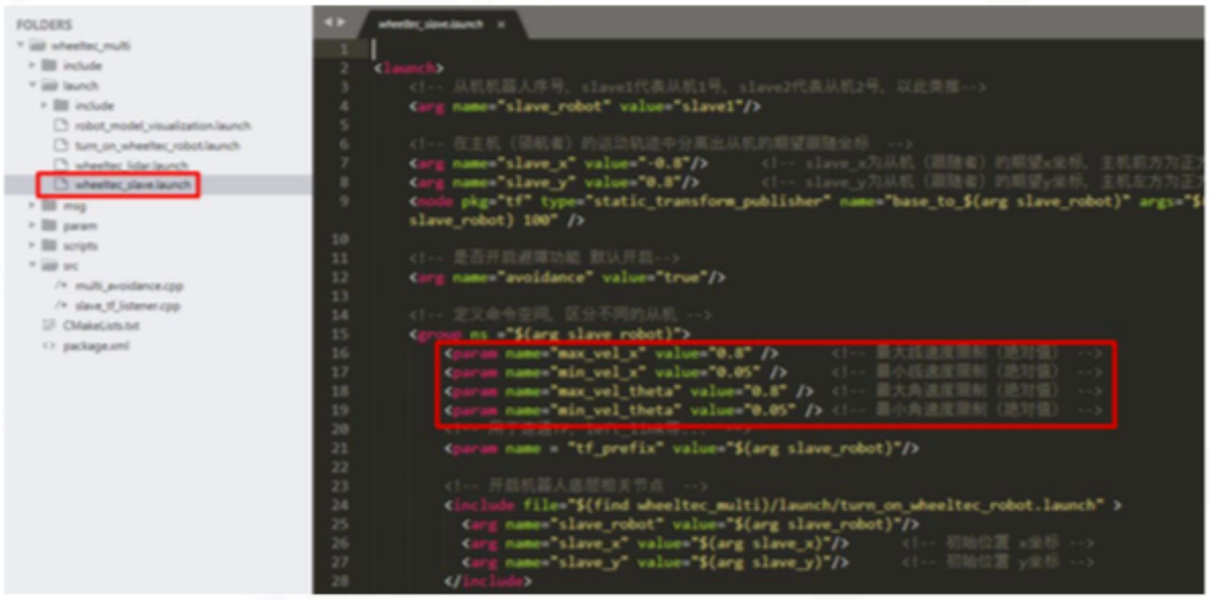
```
      <arg name="slave_y" value="$(arg slave_y)"/>      <!-- 初始位置y坐标 -->
    </include>

    <!-- 开启避障 -->
    <group if="$(arg avoidance)">
      <node name='laser_tracker' pkg="simple_follower" type="laserTracker.py"> <!-- 提取最近的障碍物距
        <param name="winSize" value="2" type="int" />
        <param name="deltaDist" value="0.2" type="double" />
      </node>
      <node pkg="wheeltec_multi" type="multi_avoidance" name="multi_avoidance" output="screen" > <!--
        <param name="avoidance_kv" type="double" value="0.3" />       <!--
        修正从车线速度的参数，参数越大，对从机偏离障碍物的线速度作用力越强 -->
        <param name="avoidance_kw" type="double" value="0.3" />       <!--
        修正从车角速度的参数，参数越大，对从机偏离障碍物的角速度作用力越强 -->
        <param name="safe_distence" type="double" value="0.5" />      <!-- 安全距离界限 -->
        <param name="danger_distence" type="double" value="0.2" />    <!-- 危险距离界限 -->
      </node>
    </group>

    <!-- 开启用于导航的自适应蒙特卡洛定位amcl -->
    <include file="$(find wheeltec_multi)/launch/include/amcl.launch" >
      <arg name="use_map_topic" value="true"/>
      <arg name="slave_robot" value="$(arg slave_robot)"/>
    </include>
```

Fig 4-1-5 Slave Positioning Node

4.1.5 How to create formation and maintain formation

In the process of formation movement, the master movement can be controlled by Rviz, keyboard, remote control and other methods. The slave calculates its speed through the slave_tf_listener node in order to control its movement and achieve the goal of the formation.

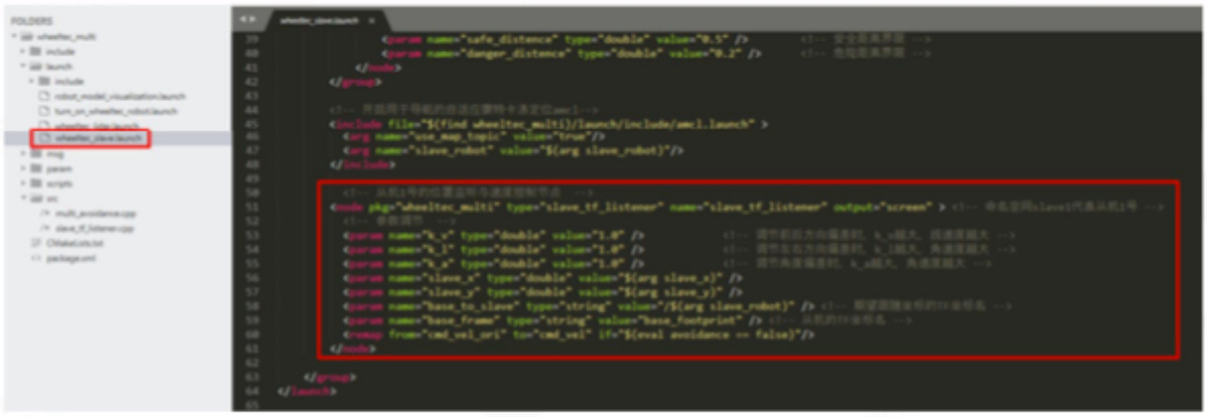
The slave_tf_listener node limits the slave speed to avoid the excessive speed by the node calculation, which will cause a series of impacts. The specific value can be modified in wheeltec_slave.launch.



```

<launch>
    <!-- 从机机器人序号，slave1代表从机1号，slave2代表从机2号，以此类推-->
    <arg name="slave_robot" value="slave1"/>

    <!-- 在主机（领航者）的运动轨迹中分离出从机的期望跟随坐标  -->
    <arg name="slave_x" value="-0.8"/>       <!-- slave_x为从机（跟随者）的期望x坐标，主机前方为正方
    <arg name="slave_y" value="0.8"/>        <!-- slave_y为从机（跟随者）的期望y坐标，主机左方为正方
    <node pkg="tf" type="static_transform_publisher" name="base_to_$(arg slave_robot)" args="$(
    slave_robot) 100" />

    <!-- 是否开启避障功能 默认开启-->
    <arg name="avoidance" value="true"/>

    <!-- 定义命名空间，区分不同的从机 -->
    <group ns ="$(arg slave_robot)">
        <param name="max_vel_x" value="0.8" />      <!-- 最大线速度限制（绝对值）  -->
        <param name="min_vel_x" value="0.05" />     <!-- 最小线速度限制（绝对值）  -->
        <param name="max_vel_theta" value="0.8" />  <!-- 最大角速度限制（绝对值）  -->
        <param name="min_vel_theta" value="0.05" /> <!-- 最小角速度限制（绝对值）  -->
        <!-- [illegible] -->
        <param name = "tf_prefix" value="$(arg slave_robot)"/>

        <!-- 开启机器人底层相关节点  -->
        <include file="$(find wheeltec_multi)/launch/turn_on_wheeltec_robot.launch" >
          <arg name="slave_robot" value="$(arg slave_robot)"/>
          <arg name="slave_x" value="$(arg slave_x)"/>      <!-- 初始位置 x坐标 -->
          <arg name="slave_y" value="$(arg slave_y)"/>      <!-- 初始位置 y坐标 -->
        </include>
```

Fig 4-1-6 Speed Limitation

The relevant parameters of the formation algorithm are as follows:

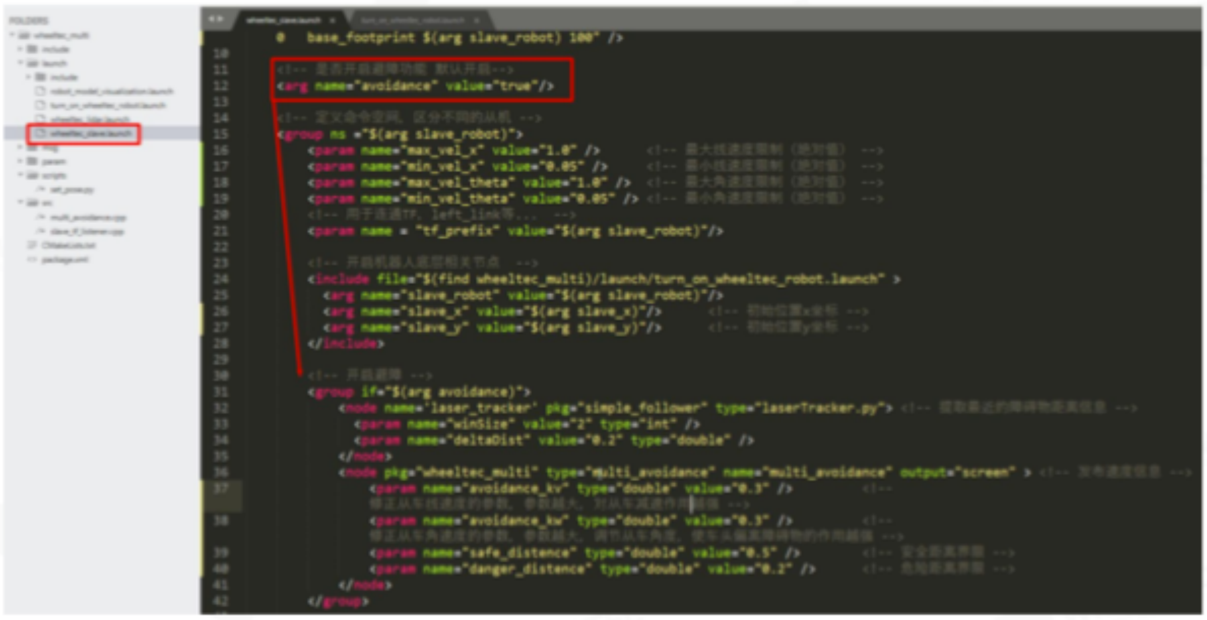


```
            <param name="safe_distence" type="double" value="0.5" />       <!-- 安全距离界限 -->
            <param name="danger_distence" type="double" value="0.2" />     <!-- 危险距离界限 -->
        </node>
    </group>

    <!-- [illegible]amcl-->
    <include file="$(find wheeltec_multi)/launch/include/amcl.launch" >
      <arg name="use_map_topic" value="true"/>
      <arg name="slave_robot" value="$(arg slave_robot)"/>
    </include>

      <!-- [illegible]  -->
    <node pkg="wheeltec_multi" type="slave_tf_listener" name="slave_tf_listener" output="screen" > <!-- [illegible] -->
      <!-- [illegible]  -->
      <param name="k_v" type="double" value="1.0" />          <!-- [illegible] -->
      <param name="k_l" type="double" value="1.0" />          <!-- [illegible] -->
      <param name="k_a" type="double" value="1.0" />          <!-- [illegible] -->
      <param name="slave_x" type="double" value="$(arg slave_x)" />
      <param name="slave_y" type="double" value="$(arg slave_y)" />
      <param name="base_to_slave" type="string" value="/$(arg slave_robot)" /> <!-- [illegible] -->
      <param name="base_frame" type="string" value="base_footprint" /> <!-- [illegible] -->
      <remap from="cmd_vel_ori" to="cmd_vel" if="$(eval avoidance == false)"/>
    </node>

  </group>
</launch>
```

Fig 4-1-7 Formation Algorithm

4.1.6 Obstacle avoidance in formation

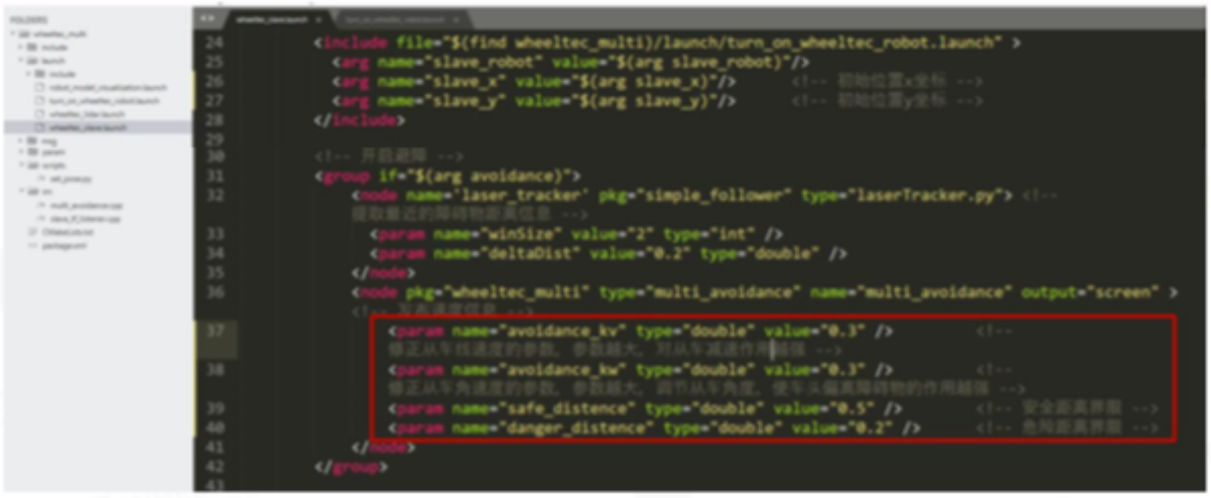
In a multi-agent formation, the master can use the move_base node to complete obstacle avoidance. However, the initialization of the slave does not use move_base node. At this point, the multi_avoidance node needs to be called in the slave program. The obstacle avoidance node is enabled by default in the package. If necessary, avoidance can be set to “false” to disable the obstacle avoidance node.



```
    0   base_footprint $(arg slave_robot) 100" />

<!-- 是否开启避障功能 默认开启-->
<arg name="avoidance" value="true"/>

<!-- 定义命名空间，区分不同的从机 -->
<group ns ="$(arg slave_robot)">
    <param name="max_vel_x" value="1.0" />       <!-- 最大线速度限制（绝对值） -->
    <param name="min_vel_x" value="0.05" />      <!-- 最小线速度限制（绝对值） -->
    <param name="max_vel_theta" value="1.0" />  <!-- 最大角速度限制（绝对值） -->
    <param name="min_vel_theta" value="0.05" /> <!-- 最小角速度限制（绝对值） -->
    <!-- 用于连通TF, left_link等...  -->
    <param name = "tf_prefix" value="$(arg slave_robot)"/>

    <!-- 开启机器人底层相关节点  -->
    <include file="$(find wheeltec_multi)/launch/turn_on_wheeltec_robot.launch" >
      <arg name="slave_robot" value="$(arg slave_robot)"/>
      <arg name="slave_x" value="$(arg slave_x)"/>       <!-- 初始位置x坐标 -->
      <arg name="slave_y" value="$(arg slave_y)"/>       <!-- 初始位置y坐标 -->
    </include>

    <!-- 开启避障 -->
    <group if="$(arg avoidance)">
        <node name='laser_tracker' pkg="simple_follower" type="laserTracker.py"> <!-- 提取最近的障碍物距离信息 -->
          <param name="winSize" value="2" type="int" />
          <param name="deltaDist" value="0.2" type="double" />
        </node>
        <node pkg="wheeltec_multi" type="multi_avoidance" name="multi_avoidance" output="screen" > <!-- 发布速度信息 -->
            <param name="avoidance_kv" type="double" value="0.3" />          <!--
            修正从车线速度的参数，参数越大，对从车减速作用越强 -->
            <param name="avoidance_kw" type="double" value="0.3" />          <!--
            修正从车角速度的参数，参数越大，调节从车角度，使车头偏离障碍物的作用越强 -->
            <param name="safe_distence" type="double" value="0.5" />         <!-- 安全距离界限 -->
            <param name="danger_distence" type="double" value="0.2" />       <!-- 危险距离界限 -->
        </node>
    </group>
```

Fig 4-1-8 Obstacle Avoidance Node for the Slaves

Some relevant parameters of the obstacle avoidance node are shown in the figure below, where safe_distance is the obstacle safe distance limit, and danger_distance is the obstacle dangerous distance limit. When the obstacle is within safe_distance and danger_distance, the slave adjusts its position to avoid the obstacle. When the obstacle is within danger_distance, the slave will drive away from the obstacle.



```
    <include file="$(find wheeltec_multi)/launch/turn_on_wheeltec_robot.launch" >
      <arg name="slave_robot" value="$(arg slave_robot)"/>
      <arg name="slave_x" value="$(arg slave_x)"/>       <!-- 初始位置x坐标 -->
      <arg name="slave_y" value="$(arg slave_y)"/>       <!-- 初始位置y坐标 -->
    </include>

    <!-- 开启避障 -->
    <group if="$(arg avoidance)">
        <node name='laser_tracker' pkg="simple_follower" type="laserTracker.py"> <!--
        提取最近的障碍物距离信息 -->
          <param name="winSize" value="2" type="int" />
          <param name="deltaDist" value="0.2" type="double" />
        </node>
        <node pkg="wheeltec_multi" type="multi_avoidance" name="multi_avoidance" output="screen" >
        <!-- 发布速度信息 -->
            <param name="avoidance_kv" type="double" value="0.3" />          <!--
            修正从车线速度的参数，参数越大，对从车减速作用越强 -->
            <param name="avoidance_kw" type="double" value="0.3" />          <!--
            修正从车角速度的参数，参数越大，调节从车角度，使车头偏离障碍物的作用越强 -->
            <param name="safe_distence" type="double" value="0.5" />         <!-- 安全距离界限 -->
            <param name="danger_distence" type="double" value="0.2" />       <!-- 危险距离界限 -->
        </node>
    </group>
```

Fig 4-1-9 Obstacle Avoidance Parameters for the Slaves

4.2 Operation Procedure

4.2.1 Enter execution command

Preparations before starting multi-agent formation:

- The master and slave connect to the same network and set up multi-agent communication correctly
- The master builds a 2D map in advance and saves it
- The master is placed at the starting point of the map, and the slave is placed near the initialization position (the default slave formation position)
- After logging in to Jetson Nano/Raspberry Pi remotely, perform time synchronization.

```
sudo date -s “2022-04-01 15:15:00”
```

Step 1: Open 2D map from the master.

```
roslaunch turn_on_wheeltec_robot navigation.launch
```

Step 2: Run formation program from all the slaves.

```
roslaunch wheeltec_multi wheeltec_slave.launch
```

Step 3: Open keyboard control node from the master or use joystick to remote control the master movement.

```
roslaunch wheeltec_robot_rc keyboard_teleop.launch
```

Step 4: (Optional) Observe the robot movements from Rviz.

```
rviz
```

4.2.2 Note:

1. Be sure to complete the time synchronization operation before executing the program.

2. When controlling the master of a multi-agent formation, the angular velocity should not be too fast. Recommended linear speed is 0.2m/s, angular speed degree below 0.3rad/s. When the master is making a turn, the farther the slave is from the master, the greater the linear speed is required. Because of the limit on the linear speed and angular speed in the package, when the slave car cannot reach the required speed, the formation will be chaotic. Overall, the excessive linear speed can easily damage the robot.

3. When the number of slave is more than one, due to the limited on-board wifi bandwidth of the ROS host, it is easy to cause significant delays and disconnection of the multi-agent communication. Using a router can solve this problem well.

4. The TF tree of the multi-robot formation (2 slaves) is: rqt_tf_tree

5. The node relationship diagram of the multi-robot formation (2 slaves) is: rqt_graph