Voice Control Multi-Point Navigation

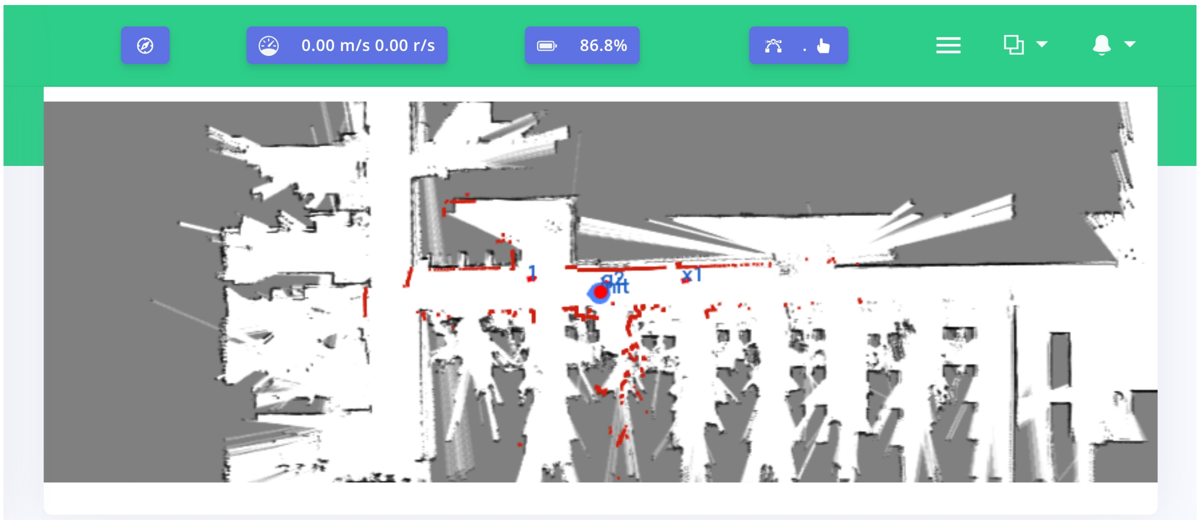
The voice control multi-point navigation function is to control the vehicle to move to the specified navigation point through voice control. We have preset 3 points in the plug-in, so when voice control is used, the vehicle can be controlled to reach positions 1, 2, and 3.

1. Prerequisites for operation

1. The map of the environment has been built.
2. Add three points d1, d2, and d3 to the environment map.
3. The navigation mode has been started.
4. The initialization positioning of the vehicle has been completed.

The above 4 steps can be completed using the Navrobo App on your mobile phone. (For specific operations, please refer to which chapter of the mobile app)

The following figure shows the initialization of the vehicle and the addition of three points.



2. Start

Note: The [SWB] mid-range of the aircraft remote control has the [emergency stop] function of this gameplay

- To start control, you need to first turn the SWB button to the upper gear position (control command mode) to release the remote control

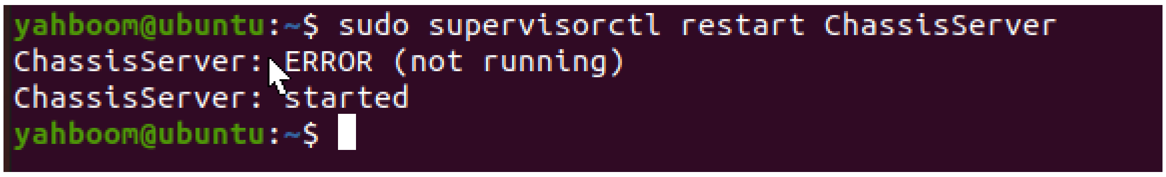
2.1. Code path

```
/home/yahboom/wukong-robot/voice_ctrl_point.py
```

2.2. Run command

First, close all other terminals. Open a command to restart the self-starting chassis service

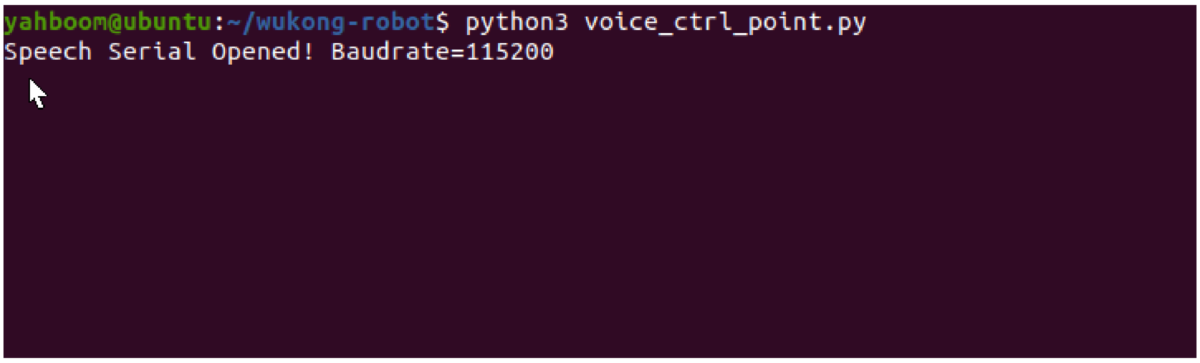
```
sudo supervisorctl restart ChassisServer
```



```
yahboom@ubuntu:~$ sudo supervisorctl restart ChassisServer
ChassisServer: ERROR (not running)
ChassisServer: started
yahboom@ubuntu:~$
```

Start the multi-point navigation function and enter the following command,

```
cd wukong-robot/
python3 voice_ctrl_point.py
```



```
yahboom@ubuntu:~/wukong-robot$ python3 voice_ctrl_point.py
Speech Serial Opened! Baudrate=115200
```

Say "Hello, Xiaoya" to NAVROBOT, and wait until the module broadcasts the reply "Yes", then according to the table below, you can switch/control the movement state of the car by voice. For example, the following is "Navigate to position 1".

2.3, Voice array communication table

Functional words	Voice recognition module results	Voice broadcast content
Go to the point A	19	OK, I'm going to the point A.
Go to the point B	20	OK, I'm going to the point B.
Go to the point C	21	OK,I'm going to the point C.

3, Writing of multi-point navigation plug-in

The code path of the multi-point navigation plug-in is:

```
/home/yahboom/wukong-robot/voice_ctrl_point.py
```

Code analysis:

```
# -*- coding: utf-8-*-
# Start the map building plug-in
import time
import socket
import subprocess
from Speech_Lib import Speech
from robot import logging
from dao import DaoCurrentTask, DaoHistoryTask
from robot.sdk.AbstractPlugin import AbstractPlugin
import requests
import json
```

```python
logger = logging.getLogger(__name__)
spe = Speech()

class Plugin():

    SLUG = "navrobo_nav_point"

    def __init__(self):
        self.url = "http://127.0.0.1:8888/yahboom/yahboomStartRunPoint"
        self.word = 0
        self.dct = DaoCurrentTask()



    def handle(self, x):
        self.word = x
        url_get = "http://127.0.0.1:8888/yahboom/yahboomRobotInfo"
        url_stop = "http://127.0.0.1:8888/yahboom/yahboomDeleteRunningTasks"



        response_get = requests.get(url_get)
        map_name=json.loads(response_get.text).get('data').get('map_name')

        try:
            if speech_r == 19 :

                bool_stop = requests.post(url_stop, data={})
                time.sleep(3)

                self.dct.delete_all_task_info()

                poseTaskData = {"map_name": map_name,
            "group_name": "default",
            "task_type": "pose", "loops": 1,
            "tasks": [{"name": 1,
              "type": "pose",
              "Delay_before_task": 0,
              "Delay_after_task": 0,
              "speak": "任务已经完成了",
              "arm_task": [0]},]}

                posedata = {"data": json.dumps(poseTaskData)}
                response = requests.post(self.url, data=posedata)
                spe.void_write(self.word)

                print(response.text)




            elif speech_r == 20 :
                bool_stop = requests.post(url_stop, data={})
                time.sleep(3)

                self.dct.delete_all_task_info()
                poseTaskData = {"map_name": map_name,
            "group_name": "default",
            "task_type": "pose", "loops": 1,
            "tasks": [{"name": "g2",
              "type": "pose",
              "Delay_before_task": 0,
              "Delay_after_task": 0,
              "speak": "任务已经完成了",
              "arm_task": [0]},]}

                posedata = {"data": json.dumps(poseTaskData)}
                response = requests.post(self.url, data=posedata)
                print(response.text)

                spe.void_write(self.word)

            elif speech_r == 21 :
                bool_stop = requests.post(url_stop, data={})
                time.sleep(3)

                self.dct.delete_all_task_info()


                poseTaskData = {"map_name": map_name,
            "group_name": "default",
            "task_type": "pose", "loops": 1,
            "tasks": [{"name": "x1",
              "type": "pose",
              "Delay_before_task": 0,
              "Delay_after_task": 0,
              "speak": "任务已经完成了",
              "arm_task": [0]},]}

                posedata = {"data": json.dumps(poseTaskData)}

                response = requests.post(self.url, data=posedata)

                print(response.text)

                spe.void_write(self.word)


        except Exception as e:
            logger.error(e)

    # def isValid(self,x):
    #     self.word = x
    #     return self.word



if __name__ == "__main__":
    wukong = Plugin()
    while 1:
        time.sleep(0.05)
        speech_r = spe.speech_read()
        if speech_r != 999:
            print(speech_r)
        wukong.handle(speech_r)
```

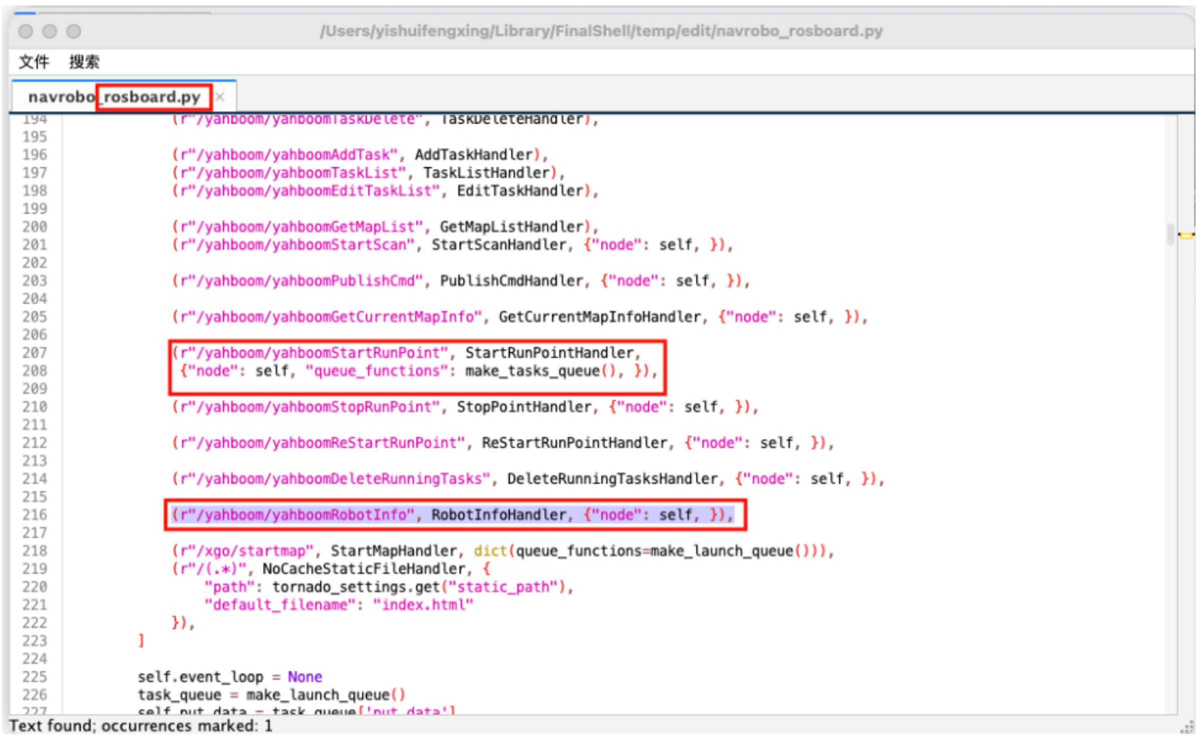
Let's take a look at how the navigation interface is handled in rosboard

The interface entry is as follows, in the rosboard.py file

```
(r"/yahboom/yahboomStartRunPoint", StartRunPointHandler,
             {"node": self, "queue_functions": make_tasks_queue(), }),
```

```
(r"/yahboom/yahboomRobotInfo", RobotInfoHandler, {"node": self, }),
```

As shown in the following figure:


/Users/yishuifengxing/Library/FinalShell/temp/edit/navrobo_rosboard.py
文件
搜索
navrobo_rosboard.py

```
            (r"/yahboom/yahboomTaskDelete", TaskDeleteHandler),

            (r"/yahboom/yahboomAddTask", AddTaskHandler),
            (r"/yahboom/yahboomTaskList", TaskListHandler),
            (r"/yahboom/yahboomEditTaskList", EditTaskHandler),

            (r"/yahboom/yahboomGetMapList", GetMapListHandler),
            (r"/yahboom/yahboomStartScan", StartScanHandler, {"node": self, }),

            (r"/yahboom/yahboomPublishCmd", PublishCmdHandler, {"node": self, }),

            (r"/yahboom/yahboomGetCurrentMapInfo", GetCurrentMapInfoHandler, {"node": self, }),

            (r"/yahboom/yahboomStartRunPoint", StartRunPointHandler,
             {"node": self, "queue_functions": make_tasks_queue(), }),

            (r"/yahboom/yahboomStopRunPoint", StopPointHandler, {"node": self, }),

            (r"/yahboom/yahboomReStartRunPoint", ReStartRunPointHandler, {"node": self, }),

            (r"/yahboom/yahboomDeleteRunningTasks", DeleteRunningTasksHandler, {"node": self, }),

            (r"/yahboom/yahboomRobotInfo", RobotInfoHandler, {"node": self, }),

            (r"/xgo/startmap", StartMapHandler, dict(queue_functions=make_launch_queue())),
            (r"/(.*)", NoCacheStaticFileHandler, {
                "path": tornado_settings.get("static_path"),
                "default_filename": "index.html"
            }),
        ]

        self.event_loop = None
        task_queue = make_launch_queue()
        self.put_data = task_queue['put_data']
Text found; occurrences marked: 1
```

The corresponding handler code is as follows

```
StartRunPointHandler.py:

class StartRunPointHandler(tornado.web.RequestHandler):

    def options(self, *args, **kwargs):
        # Set allowed request headers and methods
        self.set_header("Access-Control-Allow-Origin", "*")
        # self.set_header("Access-Control-Allow-Headers", "Access-Control-Allow-
Headers, Origin,Accept, X-Requested-With, Content-Type, Access-Control-Request-
Method, Access-Control-Request-Headers")
        self.set_header("Access-Control-Allow-Headers", "*")
        self.set_header("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE,
OPTIONS")
```

```python
        self.set_header("Access-Control-Allow-Credentials", "false")
        self.set_header('Referrer-Policy', 'strict-origin-when-cross-origin')
        self.set_status(204)  # Returning 204 indicates that the OPTIONS request 
was successfully processed

    def set_default_headers(self):
        self.set_header("Access-Control-Allow-Origin", "*")
        self.set_header("Access-Control-Allow-Headers", "Content-Type")
        self.set_header("Access-Control-Allow-Methods", "GET, POST, OPTIONS")

    def initialize(self, node, queue_functions):
        """store the instance of the ROS node that created this 
StopPerformHandler
        so we can access it later"""
        self.node = node
        self.put_data = queue_functions["put_data"]
        self.get_data = queue_functions["get_data"]

    def run_point_in_thread(self, task_list, map_name, group_name, loops):
        self.node.run_point(task_list, map_name, group_name, loops)

    @gen.coroutine
    def async_run_point(self, task_list, map_name, group_name, loops):
        thread = threading.Thread(target=self.run_point_in_thread, args=
(task_list, map_name, group_name, loops))
        thread.start()
        yield gen.sleep(0)  # Give up CPU control so that other coroutines can 
continue to execute
        # yield self.node.run_point(world_point)

    @gen.coroutine
    def post(self):
        try:
            conn = sqlite3.connect(dbPath)
            c = conn.cursor()
            c.execute("SELECT * FROM current_task ", )
            results = c.fetchall()
            if len(results) > 0:
                re_data = {"msg": "There is a task currently being executed"}
                self.write(json.dumps(re_data))
                return
        except sqlite3.IntegrityError as e:
            re_data = {"msg": str(e)}
            self.write(json.dumps(re_data))
            return
        data_post = self.get_argument('data', None)
        decompress_data = json.loads(data_post)
        map_name = decompress_data["map_name"]
        tasks = decompress_data["tasks"]
        task_type = decompress_data["task_type"]
        loops = decompress_data["loops"]
        group_name = decompress_data["group_name"]

        point_list = []
        data = self.node.sub_current_map_info()
        for task in tasks:
            if task["type"] == "pose":
                # try:
                conn = sqlite3.connect(dbPath)
                c = conn.cursor()
                print("/yahboom/yahboomStartRunPoint:", task)

 print("*********************/yahboom/yahboomStartRunPoint:::*******************
*********")
                print(type(task["name"]))
                print(map_name)
                c.execute("SELECT * FROM points WHERE mapName=? AND pointName=?",
(map_name, str(task["name"])))

 print("*********************/yahboom/yahboomStartRunPoint:::*******************
*********")
                results = c.fetchall()

 print("*********************/yahboom/yahboomStartRunPoint222:::****************
************")
                if results:
                    for result in results:
                        print(result)
                        world_point = web_to_world(result[3], result[4],
data["origin_x"],
                                                           data["origin_y"],
data["resolution"], data["height"])
                        point = {"id": result[0], "pose": [result[3], result[4],
result[5]], "createdAt": result[6],
                                 "x": result[3], "y": result[4], "name":
result[2], "angle": result[5],
                                 "world_point": world_point,
                                 "type": "pose", "Delay_before_task":
task["Delay_before_task"],
                                 "Delay_after_task": task["Delay_after_task"],
"speak": task["speak"],
                                 "arm_task": [0]}
                        point_list.append(point)
                    c.close()
                    conn.close()

                # except sqlite3.IntegrityError as e:
                #     re_data = {"msg": str(e)}
                #     print("这里会有报错吗：：", re_data)
                #     self.write(json.dumps(re_data))
                #     return

            else:
                try:
                    conn = sqlite3.connect(dbPath)
                    c = conn.cursor()
                    print("/yahboom/yahboomStartRunPoint:", task)
                    print(str(task["name"]))
                    print(task["name"])
                    print(type(str(task["name"])))
                    print(type(task["name"]))
 print("*********************/yahboom/yahboomStartRunPoint::::******************
*********")
                    task_name = task["name"]
                    c.execute("SELECT * FROM paths WHERE mapName=? AND
pathName=?", (str(map_name), str(task_name)))
                    results = c.fetchall()
                    if results:
                        for result in results:
                            print(result)
                            world_point_start = web_to_world(result[3],
result[4], data["origin_x"],
                                                             data["origin_y"],
data["resolution"], data["height"])

                            world_point_end = web_to_world(result[5], result[6],
data["origin_x"],
                                                           data["origin_y"],
data["resolution"], data["height"])

                            world_point_contr = web_to_world(result[9],
result[10], data["origin_x"],
                                                             data["origin_y"],
data["resolution"], data["height"])

                            bs = BezierPoints()
                            control_points =
np.array([[world_point_start["world_x"], world_point_start["world_y"]],

[world_point_contr["world_x"], world_point_contr["world_y"]],

[world_point_end["world_x"], world_point_end["world_y"]]])
                            bs_num = bs.bessel_resolution(result[3], result[4],
result[9], result[10])
                            evaluated_x, evaluated_y =
bs.evaluate_bezier_curve(control_points, int(bs_num))
                            print("*****************")
                            print(bs_num)
                            print(evaluated_x, evaluated_y)
                            print("*****************")
                            path_list = []
                            for j in range(len(evaluated_x)):
                                path_point = {"x": evaluated_x[j], "y":
evaluated_y[j]}
                                path_list.append(path_point)
                            path = {"id": result[0],
                                    "pose": [result[3], result[4], result[5],
result[6], result[9], result[10],
                                             result[7]],
                                    "createdAt": result[8], "startX": result[3],
"startY": result[4],
                                    "endX": result[5],
                                    "endY": result[6], "name": result[2],
"angle": result[7], "controlX": result[9],
                                    "path_pose": path_list,
                                    "controlY": result[10], "active": False,
"world_point_start": world_point_start,
                                    "world_point_end": world_point_end,
"world_point_contr": world_point_contr,
                                    "type": "path", "Delay_before_task":
task["Delay_before_task"],
                                    "Delay_after_task": task["Delay_after_task"],
"speak": task["speak"],
                                    "arm_task": [0]}

                            point_list.append(path)
                        c.close()
                        conn.close()

                except sqlite3.IntegrityError as e:
                    re_data = {"msg": str(e)}
                    self.write(json.dumps(re_data))
                    return
                pass

        yield self.async_run_point(point_list, map_name, group_name, loops)
        self.write(json.dumps(point_list))
        return
```

Code for RobotInfoHandler:

```
class RobotInfoHandler(tornado.web.RequestHandler):
    def options(self, *args, **kwargs):
        # Set allowed request headers and methods
        self.set_header("Access-Control-Allow-Origin", "*")
        # self.set_header("Access-Control-Allow-Headers", "Access-Control-Allow-
Headers, Origin,Accept, X-Requested-With, Content-Type, Access-Control-Request-
Method, Access-Control-Request-Headers")
        self.set_header("Access-Control-Allow-Headers", "*")
        self.set_header("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE,
OPTIONS")
        self.set_header("Access-Control-Allow-Credentials", "false")
        self.set_header('Referrer-Policy', 'strict-origin-when-cross-origin')
        self.set_status(204)

    def set_default_headers(self):
        self.set_header("Access-Control-Allow-Origin", "*")
        self.set_header("Access-Control-Allow-Headers", "Content-Type")
        self.set_header("Access-Control-Allow-Methods", "GET, POST, OPTIONS")


    def initialize(self, node):
        """store the instance of the ROS node that created this
StopPerformHandler
        so we can access it later"""
        self.node = node

    @gen.coroutine
    def get(self):
        robot_info = self.node.get_robot_current_info()
        re_data = {"msg": "ok", "data": robot_info}
        self.write(json.dumps(re_data))
```