**机器狗二次开发系统**

**验收材料**

日期：2025年9月22日

**目 录**

[§1 项目基本情况](#_Toc118860314)

[§2 项目进度审核](#_Toc118860315)

[2.1 项目实施进度情况](#_Toc118860316)

[2.2 项目变更情况](#_Toc118860317)

[2.3 项目投资结算情况](#_Toc118860318)

[§3 项目验收计划](#_Toc118860319)

[3.1 项目验收原则](#_Toc118860320)

[3.2 项目验收方式](#_Toc118860321)

[3.3 项目验收内容](#_Toc118860322)

[§4 项目验收情况汇总](#_Toc118860323)

[4.1 项目验收情况汇总表](#_Toc118860324)

[4.2 项目验收附件明细](#_Toc118860325)

[4.3 专家组验收意见](#_Toc118860326)

[§5 项目验收结论](#_Toc118860327)

[5.1 开发单位结论](#_Toc118860328)

[5.2 建设单位结论](#_Toc118860329)

[§6 附件](#_Toc118860330)

[6.1 附件一：软件平台验收单](#_Toc118860331)

[6.2 附件二：功能模块验收单](#_Toc118860332)

[6.3 附件三：项目文档验收单](#_Toc118860333)

[6.4 附件四：硬件设备验收单](#_Toc118860334)

# 

# §1 项目基本情况

|  |  |
| --- | --- |
| **项目名称：** | 基于宇树Go2机器狗的语音交互和智能导航系统的二次开发项目 |
| **项目合同甲方：** | 杭州淮源科技有限公司 |
| **项目合同乙方：** | 逻极科技（宁波）有限公司 |
| **项目开工时间：** | 2025年4月 |
| **项目竣工时间：** | 2025年9月 |
| **项目验收日期：** | 2025年9月18日 |

# §2 系统基本情况

## 2.1 系统架构

系统总体分成三个部分，分别是机器人运动控制与智能导航模块、智能语音交互模块和网页展示与控制模块。

其中机器人运动控制与智能导航模块、智能语音交互模块需要在机器狗外置算力设备上进行运行，网页展示与控制模块需要部署在服务器上，为其他设备访问网页提供支持。

## 2.2 机器人运动控制与智能导航模块

该模块需要运行在ubuntu系统下，需要ROS2机器人操作系统环境，边缘计算设备需要拥有4GB以上运行内存，芯片架构推荐为x86，核心频率推荐为3Ghz以上，需要至少一个有线网口以连接机器狗。安装系统推荐为ubuntu22.04.5，安装ROS2系统推荐为humble。

以下是机器人运动控制与智能导航模块介绍：

使用unitree\_sdk2功能包获得与机器狗的交互接口；使用dds\_rostopic功能包订阅机器狗发布的dds信号，并发布为标准ros消息（包含狗头相机图像发布和激光雷达信号发布）；

Input：dds、230.1.1.1:1720

Output：SMX/Go2Lidar、SMX/Go2Camera

使用sport\_control功能包读取控制手柄或其他包的控制指令，调用交互接口控制狗的运动；

Input：/joy、SMX/SportCmd、/cmd\_vel、SMX/Odom

Output：SMX/JoystickCmd、SMX/GimbalAngleCmd

使用fusion\_estimator功能包实现基于运动学的位置估计，并输出机器狗的里程计消息；

Input：dds、SMX/JoystickCmd

Output：SMX/Estimation、SMX/Odom、SMX/Odom\_2D

使用message\_handle功能包做进一步的信号处理（做激光雷达的tf转换，把点云信号转化为二维scan信号）。

Input：/SMX/Go2Lidar、/SMX/Odom、/SMX/Odom\_2D

Output：/SMX/Scan

使用unitree\_sdk2功能包获得与机器狗的交互接口；

使用dds\_rostopic功能包订阅机器狗发布的dds信号，并发布为标准ros消息（包含狗头相机图像发布和激光雷达信号发布）；

Input：dds、230.1.1.1:1720

Output：SMX/Go2Lidar、SMX/Go2Camera

使用sport\_control功能包读取控制手柄或其他包的控制指令，调用交互接口控制狗的运动；

Input：/joy、SMX/SportCmd、/cmd\_vel、SMX/Odom

Output：SMX/JoystickCmd、SMX/GimbalAngleCmd

使用fusion\_estimator功能包实现基于运动学的位置估计，并输出机器狗的里程计消息；

Input：dds、SMX/JoystickCmd

Output：SMX/Estimation、SMX/Odom、SMX/Odom\_2D

使用message\_handle功能包做进一步的信号处理（做激光雷达的tf转换，把点云信号转化为二维scan信号）。

Input：/SMX/Go2Lidar、/SMX/Odom、/SMX/Odom\_2D

Output：/SMX/Scan

使用mqtt协议实现网页和语音指令对机器狗的控制；

使用SLAM技术实现机器狗在陌生环境下进行地图构建和预建地图的智能巡逻和导航。

以下是机器人运动控制与智能导航模块部署说明：

Use Ubuntu 22.04, ROS2 Humble

需要安装以下库，命令行如下：

sudo apt install ros-humble-joy ros-humble-nav2-msgs ros-humble-slam-toolbox ros-humble-nav2-bringup python3-pip libopencv-dev ros-humble-cv-bridge ros-humble-image-transport ros-humble-compressed-image-transport

部署的文件目录，命令行如下：

mkdir -p ~/ros2\_ws/LeggedRobot/src && cd ~/ros2\_ws/LeggedRobot/src

拉取文件，命令行如下：

git clone --recursive git@github.com:luckyCharm1123/go2-raspi.git

需要注意：

# 1. 搜索工程中的所有 /home/ubuntu/ros2\_ws/LeggedRobot，替换为您的路径

# 2. 把 src/Ros2Go2Estimator/config.yaml 中的所有 br0 替换为您的网卡名，如 enxf8e43b808e06

完成后需要进行构建，命令行如下：

colcon build

启动命令如下：

ros2 launch sport\_control go2\_launch.py

补充说明：

系统使用config.yaml进行话题名称设置。

切换两足、四足无需在估计器内做模式切换;

目前没有调整参数做补偿，工程使用时可进一步提升精度;

dds\_rostopic包将宇树dds提供的信息转换和发布为标准ros2话题;

fusion\_estimator包发布对应“base\_link”的话题SMX/Odom和对应“base\_link\_2D”的话题SMX/Odom\_2D;

message\_handle包完成SMX/Odom和SMX/Odom\_2D的tf，此外，将frame“utlidar\_lidar”的pointcloud2转换为“base\_link\_2D”话题SMX/Scan;

sport\_control包读取joystick输入和其他指令，使用unitree\_sdk2提供的接口控制机器狗;

使用SLAM Toolbox建图时额外ros2 launch sport\_control slam\_launch.py;

使用Nav2导航时额外ros2 launch sport\_control nav\_launch.py;

使用Amov机架跟踪时额外ros2 launch sport\_control g1\_launch.py;

SLAM Toolbox目前是纯里程计建图，请擅长SLAM的同志自行把地图匹配加进去;

Nav2同样请自行调整，加载的地图记得改成自己的;

也适用于Ubuntu20.04 foxy系统，把apt install的软件改为-foxy-即可;

## 2.3 项目投资结算情况

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **款 项** | **金 额**  **（万元）** | **备注** |
| **1** |  |  |  |
| **2** |  |  |  |
| **合 计** | |  |  |

## 2.4 网页展示与控制模块

该部分分为后台控制模块与前台展示模块，后台控制部分基于宇树官方sdk，通过cmake进行编译，通过websocket进行通信，达到网页端可以远程控制机器人的目的。

### 2.4.1 控制模块

### 系统要求

* 操作系统: Ubuntu 18.04+ 或其他Linux发行版
* 编译器: GCC 7.0+ (需要C++17支持)
* CMake: 3.10+ C代码：3.10+
* SDK: Unitree SDK2 开发工具包：Unitree SDK2

编译安装

克隆并编译项目：

mkdir -p build && cd build

cmake ..

make

运行程序：

./cpp\_project

CMakeLists.txt 配置说明

如果编译失败，请检查CMakeLists.txt文件配置：

cmake\_minimum\_required(VERSION 3.10)

project(cpp\_project)

# 设置C++标准（SDK2需要C++17）

set(CMAKE\_CXX\_STANDARD 17)

set(CMAKE\_CXX\_STANDARD\_REQUIRED True)

# 查找Unitree SDK2

find\_package(unitree\_sdk2 REQUIRED)

# 添加包含目录

include\_directories(${PROJECT\_SOURCE\_DIR}/include)

# 添加所有源文件

file(GLOB SOURCES

${PROJECT\_SOURCE\_DIR}/src/\*.cpp

${PROJECT\_SOURCE\_DIR}/src/utils/\*.cpp

${PROJECT\_SOURCE\_DIR}/src/service/\*.cpp

${PROJECT\_SOURCE\_DIR}/src/core/client/\*.cpp

)

# 创建可执行文件

add\_executable(cpp\_project ${SOURCES})

# 链接Unitree SDK2

target\_link\_libraries(cpp\_project PRIVATE unitree\_sdk2 pthread)

⚠️ 注意: 如果有新的源文件需要添加到项目中，直接在src目录下添加文件即可。

🎯 项目特性

多任务管理: 支持WebSocket连接、状态监控、姿态控制、媒体控制等多个任务

WebSocket通信: 远程控制机器人姿态和灯光等功能

LiDAR定位: 通过点云数据获取机器人位置信息

模块化设计: 清晰的代码结构，易于扩展和维护

跨平台支持: 支持多种Linux发行版

📡 WebSocket控制文档

项目集成了以下任务到线程池中，支持通过WebSocket进行远程控制：

websocket\_connection\_task: WebSocket连接管理任务

websocket\_monitor\_task: WebSocket状态监控任务

pose\_task: 机器人姿态控制任务

obstacle\_avoidance\_task: 障碍物规避运动控制任务

media\_task: 媒体控制任务（包含灯光控制）

WebSocket 消息格式

系统使用嵌套JSON消息格式进行通信：

{

"type": "private",

"to": "target\_client\_id",

"message": "{\"target\":\"任务名称\",\"command\":\"具体命令\",\"type\":\"消息类型\"}"

}

机器人姿态控制 (pose\_task)

pose\_task任务负责控制机器人的基本姿态和动作，包括坐立、站立、趴下以及特殊动作等。

支持的命令

坐下命令:

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"pose\_task\",\"command\":\"sit\",\"type\":\"robot\_command\"}"

}

站立命令:

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"pose\_task\",\"command\":\"stand\",\"type\":\"robot\_command\"}"

}

趴下命令:

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"pose\_task\",\"command\":\"standdown\",\"type\":\"robot\_command\"}"

}

站高命令:

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"pose\_task\",\"command\":\"standup\",\"type\":\"robot\_command\"}"

}

停止命令:

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"pose\_task\",\"command\":\"stop\",\"type\":\"robot\_command\"}"

}

问候动作:

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"pose\_task\",\"command\":\"hello\",\"type\":\"robot\_command\"}"

}

比心动作:

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"pose\_task\",\"command\":\"heart\",\"type\":\"robot\_command\"}"

}

获取机器人状态:

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"pose\_task\",\"command\":\"get\_state\",\"type\":\"robot\_command\"}"

}

命令说明

命令 功能描述 对应的SportMode

sit 机器人坐下 SportMode::sit 运动模式：：坐

stand 机器人站立（恢复站立） SportMode::recovery\_stand

运动模式：：recovery\_stand

standdown 机器人趴下 SportMode::stand\_down

运动模式：：stand\_down

standup 机器人站高 SportMode::stand\_up 运动模式：：stand\_up

stop 停止当前动作 SportMode::stop\_move

运动模式：：stop\_move

hello 执行问候动作 SportMode::stop\_move

运动模式：：stop\_move

heart 执行比心动作 SportMode::stop\_move

运动模式：：stop\_move

get\_state 获取当前机器人状态信息 -

使用示例

基本姿态控制序列:

import asyncio

import websockets

import json

async def pose\_control\_demo():

uri = "ws://192.168.31.132:8765"

async with websockets.connect(uri) as websocket:

# 设置客户端ID

await websocket.send(json.dumps({

"type": "set\_id",

"id": "pose\_control\_client"

}))

# 1. 先让机器人站立

command = {

"type": "private",

"to": "go2\_robot\_001",

"message": json.dumps({

"target": "pose\_task",

"command": "stand",

"type": "robot\_command"

})

}

await websocket.send(json.dumps(command))

await asyncio.sleep(3) # 等待3秒

# 2. 执行问候动作

command["message"] = json.dumps({

"target": "pose\_task",

"command": "hello",

"type": "robot\_command"

})

await websocket.send(json.dumps(command))

await asyncio.sleep(3)

# 3. 执行比心动作

command["message"] = json.dumps({

"target": "pose\_task",

"command": "heart",

"type": "robot\_command"

})

await websocket.send(json.dumps(command))

await asyncio.sleep(3)

# 4. 让机器人坐下

command["message"] = json.dumps({

"target": "pose\_task",

"command": "sit",

"type": "robot\_command"

})

await websocket.send(json.dumps(command))

# 运行示例

asyncio.run(pose\_control\_demo())

注意事项

动作间隔: 建议在连续动作之间留出2-3秒的间隔，确保前一个动作完全执行

状态检查: 使用get\_state命令可以获取机器人的详细状态信息

安全考虑: 在执行动作前确保机器人周围有足够的空间

错误处理: 所有命令都会返回执行结果，0表示成功，非0表示失败

障碍物规避运动控制 (obstacle\_avoidance\_task)

支持的命令

直接移动控制:

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"move\",\"x\":0.5,\"y\":0.0,\"yaw\":0.0,\"type\":\"obstacle\_avoidance\_command\"}"

}

方向移动命令:

// 前进

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"forward\",\"speed\":0.3,\"type\":\"obstacle\_avoidance\_command\"}"

}

// 后退

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"backward\",\"speed\":0.3,\"type\":\"obstacle\_avoidance\_command\"}"

}

// 左移

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"left\",\"speed\":0.3,\"type\":\"obstacle\_avoidance\_command\"}"

}

// 右移

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"right\",\"speed\":0.3,\"type\":\"obstacle\_avoidance\_command\"}"

}

旋转控制:

// 左转

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"turn\_left\",\"speed\":0.5,\"type\":\"obstacle\_avoidance\_command\"}"

}

// 右转

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"turn\_right\",\"speed\":0.5,\"type\":\"obstacle\_avoidance\_command\"}"

}

停止和状态:

// 停止移动

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"stop\",\"type\":\"obstacle\_avoidance\_command\"}"

}

// 查询状态

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"status\",\"type\":\"obstacle\_avoidance\_command\"}"

}

// 退出任务

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"obstacle\_avoidance\_task\",\"command\":\"exit\",\"type\":\"obstacle\_avoidance\_command\"}"

}

参数说明

x: 前后移动速度 (-1.0 到 1.0，正值向前)

y: 左右移动速度 (-1.0 到 1.0，正值向左)

yaw: 旋转角速度 (-1.0 到 1.0，正值逆时针)

speed: 移动速度 (0.0 到 1.0，建议0.1-0.5)

安全限制

所有速度参数都有安全限制，超出范围会被自动调整

系统会自动处理障碍物检测和规避

紧急停止命令会立即停止所有运动

灯光控制 (media\_task)

增亮命令

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"media\_task\",\"command\":\"light\_up\",\"type\":\"light\_command\"}"

}

减暗命令

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"media\_task\",\"command\":\"light\_down\",\"type\":\"light\_command\"}"

}

设置亮度等级

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"media\_task\",\"command\":\"set\_brightness\",\"level\":5,\"type\":\"light\_command\"}"

}

获取当前亮度

{

"type": "private",

"to": "go2\_robot\_001",

"message": "{\"target\":\"media\_task\",\"command\":\"get\_brightness\",\"type\":\"light\_command\"}"

}

部署指南

此部分用于开发环境的本地部署。机器人部署请参考下一节。

本地开发环境

确保安装所有依赖

编译项目：mkdir build && cd build && cmake .. && make

运行测试：make test

🤖 机器人部署

快速部署（一键脚本）

环境检查：

./check\_deployment\_env.sh

自动部署：

./deploy\_to\_robot.sh

手动部署步骤

1. 准备离线依赖包

# 如果还没有创建依赖包，运行：

./create\_complete\_offline\_package.sh

2. 传输文件到机器人

# 传输依赖包

scp -r robot\_offline\_deps/ unitree@<robot\_ip>:~/

# 传输项目源码

rsync -avz --exclude='build/' . unitree@<robot\_ip>:~/robot\_control/

3. 在机器人上安装依赖

# SSH连接到机器人

ssh unitree@<robot\_ip>

# 安装依赖

cd ~/robot\_offline\_deps

sudo ./scripts/install\_all\_deps.sh

# 验证安装

./scripts/verify\_all\_deps.sh

4. 编译和运行

# 编译项目

cd ~/robot\_control

mkdir -p build && cd build

cmake ..

make -j$(nproc)

# 运行程序

./cpp\_project

部署文件说明

deploy\_to\_robot.sh: 自动化部署脚本

check\_deployment\_env.sh: 部署前环境检查

DEPLOYMENT\_GUIDE.md: 详细部署指南

robot\_offline\_deps/: 离线依赖包目录

注意事项

机器人需要至少500MB可用空间

首次部署需要约10-15分钟

确保机器人网络连接正常

建议配置SSH密钥免密登录

### 2.4.2 网页展示模块

# Go2 机器人控制器 Web 界面

[![React](https://img.shields.io/badge/React-19.1.0-blue.svg)](https://reactjs.org/)

[![TypeScript](https://img.shields.io/badge/TypeScript-5.8.3-blue.svg)](https://www.typescriptlang.org/)

[![Vite](https://img.shields.io/badge/Vite-6.3.5-yellow.svg)](https://vitejs.dev/)

[![Ant Design](https://img.shields.io/badge/Ant%20Design-5.25.4-red.svg)](https://ant.design/)

一个基于 React + TypeScript + Ant Design 的 Go2 机器人远程控制 Web 界面，支持实时运动控制、状态监控和数据分析。

## ✨ 功能特性

### 🎮 机器人控制

- \*\*运动控制\*\*: 支持前进、后退、转向、精确位移控制

- \*\*姿态控制\*\*: 站立、坐下、趴下、站高、问候、比心等动作

- \*\*动作序列\*\*: 预设动作序列管理和执行

- \*\*实时控制\*\*: WebSocket 实时通信，支持远程控制

### 📊 状态监控

- \*\*实时状态\*\*: 机器人运行状态、电池电量、温度监控

- \*\*传感器数据\*\*: 陀螺仪、加速度计、气压等传感器数据

- \*\*电池状态\*\*: 电池电量、温度、充电状态监控

- \*\*系统状态\*\*: CPU、内存、存储使用情况

### 📈 数据分析

- \*\*运动轨迹\*\*: 机器人运动轨迹可视化

- \*\*性能分析\*\*: 运动性能数据统计和分析

- \*\*历史记录\*\*: 控制命令和状态变化历史

### 👥 用户管理

- \*\*用户认证\*\*: 登录/登出功能

- \*\*权限管理\*\*: 不同用户权限控制

- \*\*操作日志\*\*: 用户操作记录和审计

### ⚙️ 系统设置

- \*\*网络配置\*\*: WebSocket 连接参数设置

- \*\*机器人参数\*\*: 运动参数、安全限制等配置

- \*\*界面设置\*\*: 主题、语言等个性化设置

## 🚀 快速开始

### 环境要求

- Node.js >= 18.0.0

- npm >= 9.0.0 或 yarn >= 1.22.0

### 安装依赖

```bash

# 克隆项目

git clone https://github.com/hellolukeding/go2-controller-web.git

cd go2-controller-web

# 安装依赖

npm install

```

### 配置环境变量

复制环境变量配置文件：

```bash

cp .env.example .env

```

编辑 `.env` 文件，配置 WebSocket 连接参数：

```env

# WebSocket 远程控制配置

VITE\_WEBSOCKET\_HOST=152.32.128.33

VITE\_WEBSOCKET\_PORT=8765

VITE\_WEBSOCKET\_URL=ws://152.32.128.33:8765

# 机器人配置

VITE\_ROBOT\_ID=go2\_robot\_001

VITE\_CONTROLLER\_ID=robot\_controller

# 连接配置

VITE\_WEBSOCKET\_RECONNECT\_INTERVAL=5000

VITE\_WEBSOCKET\_HEARTBEAT\_INTERVAL=30000

VITE\_WEBSOCKET\_TIMEOUT=10000

```

### 启动开发服务器

```bash

# 启动开发服务器

npm run dev

# 构建生产版本

npm run build

# 预览生产版本

npm run preview

```

访问 `http://localhost:3000` 即可使用应用。

## � 部署指南

### 本地部署

#### 1. 构建生产版本

```bash

# 构建生产版本

npm run build

```

构建完成后，会在 `dist/` 目录下生成静态文件。

#### 2. 预览构建结果

```bash

# 预览生产版本

npm run preview

```

#### 3. 使用静态服务器部署

可以使用任何静态文件服务器来部署构建后的文件：

```bash

# 使用 Python 内置服务器

python -m http.server 8080 -d dist

# 或使用 Node.js serve

npx serve dist -p 8080

# 或使用 nginx

# 将 dist/ 目录下的文件复制到 nginx 的静态文件目录

```

### Docker 部署

#### 1. 创建 Dockerfile

在项目根目录创建 `Dockerfile`：

```dockerfile

# 使用 Node.js 官方镜像作为构建环境

FROM node:18-alpine AS builder

# 设置工作目录

WORKDIR /app

# 复制 package.json 和 package-lock.json

COPY package\*.json ./

# 安装依赖

RUN npm ci --only=production

# 复制源代码

COPY . .

# 构建应用

RUN npm run build

# 使用 nginx 作为生产环境

FROM nginx:alpine

# 复制构建好的文件到 nginx 静态文件目录

COPY --from=builder /app/dist /usr/share/nginx/html

# 复制 nginx 配置（可选）

# COPY nginx.conf /etc/nginx/nginx.conf

# 暴露端口

EXPOSE 80

# 启动 nginx

CMD ["nginx", "-g", "daemon off;"]

```

#### 2. 创建 .dockerignore

```dockerfile

node\_modules

npm-debug.log

dist

.env

.env.local

.env.production

.git

.gitignore

README.md

```

#### 3. 构建和运行 Docker 镜像

```bash

# 构建镜像

docker build -t go2-controller-web .

# 运行容器

docker run -d -p 8080:80 --name go2-controller go2-controller-web

# 访问 http://localhost:8080

```

### Nginx 部署

#### 1. 配置 nginx

创建 nginx 配置文件 `/etc/nginx/sites-available/go2-controller-web`：

```nginx

server {

listen 80;

server\_name your-domain.com; # 替换为您的域名

# 静态文件根目录

root /var/www/go2-controller-web/dist;

index index.html;

# 处理 React Router 的客户端路由

location / {

try\_files $uri $uri/ /index.html;

}

# 缓存静态资源

location ~\* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {

expires 1y;

add\_header Cache-Control "public, immutable";

}

# 安全头

add\_header X-Frame-Options "SAMEORIGIN" always;

add\_header X-XSS-Protection "1; mode=block" always;

add\_header X-Content-Type-Options "nosniff" always;

add\_header Referrer-Policy "no-referrer-when-downgrade" always;

add\_header Content-Security-Policy "default-src 'self' http: https: data: blob: 'unsafe-inline'" always;

# 日志

access\_log /var/log/nginx/go2-controller-web.access.log;

error\_log /var/log/nginx/go2-controller-web.error.log;

}

```

#### 2. 启用站点

```bash

# 创建符号链接

sudo ln -s /etc/nginx/sites-available/go2-controller-web /etc/nginx/sites-enabled/

# 测试配置

sudo nginx -t

# 重载 nginx

sudo systemctl reload nginx

```

#### 3. SSL 配置 (推荐)

使用 Let's Encrypt 获取免费 SSL 证书：

```bash

# 安装 certbot

sudo apt install certbot python3-certbot-nginx

# 获取 SSL 证书

sudo certbot --nginx -d your-domain.com

# 自动续期

sudo crontab -e

# 添加: 0 12 \* \* \* /usr/bin/certbot renew --quiet

```

### Apache 部署

#### 1. 配置 Apache

创建 Apache 配置文件 `/etc/apache2/sites-available/go2-controller-web.conf`：

```apache

<VirtualHost \*:80>

ServerName your-domain.com

DocumentRoot /var/www/go2-controller-web/dist

<Directory "/var/www/go2-controller-web/dist">

AllowOverride All

Require all granted

# 处理 React Router

RewriteEngine On

RewriteBase /

RewriteRule ^index\.html$ - [L]

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule . /index.html [L]

# 缓存头

<FilesMatch "\.(js|css|png|jpg|jpeg|gif|ico|svg)$">

Header set Cache-Control "max-age=31536000, public"

</FilesMatch>

</Directory>

# 日志

ErrorLog ${APACHE\_LOG\_DIR}/go2-controller-web\_error.log

CustomLog ${APACHE\_LOG\_DIR}/go2-controller-web\_access.log combined

</VirtualHost>

```

#### 2. 启用站点和模块

```bash

# 启用重写模块

sudo a2enmod rewrite

# 启用站点

sudo a2ensite go2-controller-web

# 重启 Apache

sudo systemctl restart apache2

```

### 云服务部署

#### Vercel 部署

1. 安装 Vercel CLI：

```bash

npm i -g vercel

```

2. 部署到 Vercel：

```bash

vercel

```

3. 配置环境变量：

```bash

vercel env add VITE\_WEBSOCKET\_HOST

vercel env add VITE\_WEBSOCKET\_PORT

# ... 添加其他环境变量

```

#### Netlify 部署

1. 连接 GitHub 仓库到 Netlify

2. 设置构建命令：`npm run build`

3. 设置发布目录：`dist`

4. 配置环境变量

#### GitHub Pages 部署

1. 安装 gh-pages：

```bash

npm install --save-dev gh-pages

```

2. 添加部署脚本到 package.json：

```json

{

"scripts": {

"deploy": "gh-pages -d dist"

}

}

```

3. 部署：

```bash

npm run deploy

```

### 生产环境配置

#### 环境变量

在生产环境中，确保设置以下环境变量：

```bash

# WebSocket 配置

VITE\_WEBSOCKET\_HOST=your-production-websocket-host

VITE\_WEBSOCKET\_PORT=8765

VITE\_WEBSOCKET\_URL=wss://your-production-websocket-host:8765

# 机器人配置

VITE\_ROBOT\_ID=your-robot-id

VITE\_CONTROLLER\_ID=your-controller-id

# 安全配置

VITE\_API\_BASE\_URL=https://your-api-domain

```

#### 性能优化

1. \*\*启用 gzip 压缩\*\*：

```nginx

# nginx 配置

gzip on;

gzip\_vary on;

gzip\_min\_length 1024;

gzip\_types text/plain text/css text/xml text/javascript application/javascript application/xml+rss application/json;

```

2. \*\*设置适当的缓存头\*\*

3. \*\*使用 CDN\*\* 加速静态资源分发

#### 监控和日志

- 配置服务器访问日志

- 设置错误监控 (如 Sentry)

- 配置性能监控

## �📁 项目结构

```

src/

├── components/ # 通用组件

│ ├── dashboard/ # 仪表盘组件

│ │ ├── MotionControl.tsx # 运动控制

│ │ ├── RealTimeStatus.tsx # 实时状态

│ │ ├── DataAnalysis.tsx # 数据分析

│ │ ├── UserManagement.tsx # 用户管理

│ │ ├── SystemSettings.tsx # 系统设置

│ │ └── ActionSequenceManager.tsx # 动作序列管理

│ ├── video/ # 视频组件

│ │ └── VideoPlayer.tsx # 视频播放器

│ └── WebSocketStatus.tsx # WebSocket 状态显示

├── pages/ # 页面组件

│ ├── Login/ # 登录页面

│ │ └── index.tsx

│ └── home/ # 主页

│ ├── index.tsx # 主页布局

│ └── subpage/ # 子页面

│ ├── Dashboard.tsx # 仪表盘

│ ├── MotionControlPage.tsx

│ ├── PostureControlPage.tsx

│ ├── ActionSequencePage.tsx

│ ├── RealTimeStatusPage.tsx

│ ├── SensorDataPage.tsx

│ ├── BatteryStatusPage.tsx

│ ├── DataAnalysisPage.tsx

│ ├── UserManagementPage.tsx

│ └── SystemSettingsPage.tsx

├── hooks/ # React Hooks

│ └── useWebSocket.ts # WebSocket Hook

├── router/ # 路由配置

│ ├── index.ts

│ └── routers.tsx

├── utils/ # 工具函数

│ ├── websocket-service.ts # WebSocket 服务

│ ├── websocket-config.ts # WebSocket 配置

│ ├── config-manager.ts # 配置管理器

│ └── websocket-config.ts # WebSocket 配置

├── App.tsx # 应用入口

├── main.tsx # 主入口文件

└── vite-env.d.ts # Vite 类型声明

```

## 🔧 技术栈

### 前端框架

- \*\*React 19\*\*: 用户界面库

- \*\*TypeScript\*\*: 类型安全的 JavaScript

- \*\*Vite\*\*: 快速的构建工具

### UI 组件库

- \*\*Ant Design 5\*\*: 企业级 UI 组件库

- \*\*Tailwind CSS\*\*: 实用优先的 CSS 框架

### 状态管理

- \*\*React Hooks\*\*: 内置状态管理

- \*\*Context API\*\*: 全局状态管理

### 网络通信

- \*\*WebSocket\*\*: 实时双向通信

- \*\*MQTT\*\*: 轻量级消息协议 (可选)

### 地图和可视化

- \*\*Leaflet\*\*: 交互式地图

- \*\*Chart.js\*\*: 图表可视化 (集成在 Ant Design 中)

### 视频和媒体

- \*\*WebRTC\*\*: 实时音视频通信

- \*\*Janus Gateway\*\*: WebRTC 网关

## 🌐 WebSocket 通信协议

### 消息格式

系统使用嵌套 JSON 消息格式进行通信：

```json

{

"type": "private",

"to": "target\_client\_id",

"message": "{\"target\":\"任务名称\",\"command\":\"具体命令\",\"type\":\"消息类型\"}"

}

```

### 支持的任务类型

#### 1. 姿态控制 (pose\_task)

- `sit`: 坐下

- `stand`: 站立

- `standdown`: 趴下

- `standup`: 站高

- `stop`: 停止

- `hello`: 问候动作

- `heart`: 比心动作

- `get\_state`: 获取状态

#### 2. 运动控制 (obstacle\_avoidance\_task)

- `move`: 精确移动 (x, y, yaw)

- `forward`: 前进

- `backward`: 后退

- `left`: 左移

- `right`: 右移

- `turn\_left`: 左转

- `turn\_right`: 右转

- `stop`: 停止

- `status`: 获取状态

#### 3. 媒体控制 (media\_task)

- `light\_up`: 增亮

- `light\_down`: 减暗

- `set\_brightness`: 设置亮度

- `get\_brightness`: 获取亮度

### 连接配置

- \*\*服务器地址\*\*: `152.32.128.33:8765`

- \*\*机器人 ID\*\*: `go2\_robot\_001`

- \*\*控制器 ID\*\*: `robot\_controller`

- \*\*自动重连\*\*: 5 秒间隔

- \*\*心跳间隔\*\*: 30 秒

## 🎯 使用指南

### 1. 登录系统

访问应用后，首先进行用户登录。

### 2. 连接机器人

确保 WebSocket 连接正常（右上角显示绿色状态）。

### 3. 运动控制

- 使用方向按钮进行基本移动

- 使用滑块进行速度调节

- 使用精确控制进行毫米级位移

### 4. 状态监控

- 查看实时状态和传感器数据

- 监控电池和系统状态

- 分析运动轨迹和性能数据

### 5. 系统设置

- 配置网络参数

- 调整机器人参数

- 个性化界面设置

## 🔒 安全说明

- 所有控制命令都会验证 WebSocket 连接状态

- 敏感操作需要用户确认

- 支持操作日志记录和审计

- 网络通信使用加密传输

# §3 项目验收计划

## 3.1 项目验收原则

1. 审查提供验收的各类文档的正确性、完整性和统一性，审查文档是否齐全、合理；
2. 审查项目功能是否达到了合同规定的要求；
3. 审查项目有关服务指标是否达到了合同的要求；
4. 审查项目投资以及实施进度的情况；
5. 对项目的技术水平做出评价，并得出项目的验收结论。

## 3.2 项目验收方式

{记录项目验收的组织方式和参与验收工作的人员情况}

|  |  |  |  |
| --- | --- | --- | --- |
| **验收人员** | **所属单位** | **所属角色** | **相关职责** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## 3.3 项目验收内容

1. 硬件设备验收；
2. 软件平台验收；
3. 应用系统验收；
4. 项目文档验收；
5. 项目服务响应（如售后服务、问题相应等方面）验收。

# §4 项目验收情况汇总

## 4.1 项目验收情况汇总表

|  |  |  |  |
| --- | --- | --- | --- |
| **验收项** | **验收意见** | | **备注** |
| **通过** | **不通过** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **总体意见：**  **项目验收组长（签字）** | | | |
| **未通过理由：**  **项目验收组长（签字）** | | | |

## 4.2 项目验收附件明细

1、软件平台验收单（见附件一）。

2、功能模块验收单（见附件二）。

3、项目文档验收单（见附件三）。

4、硬件设备验收单（见附件四）。

## 4.3 专家组验收意见

|  |
| --- |
| **专家组长（签字）** |

# §5 项目验收结论

## 5.1 开发单位结论

|  |
| --- |
| **开发单位（签章）** |

## 5.2 建设单位结论

|  |
| --- |
| **建设单位（签章）** |

# §6 附件

## 6.1 附件一：软件平台验收单

**验收人：**

**验收时间：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **软件类型** | **软件名称** | **验收结果** | **备注**  **（机器的IP地址等）** |
| **1** |  |  |  |  |
| **2** |  |  |  |  |
| **3** |  |  |  |  |

## 6.2 附件二：功能模块验收单

**验收人：**

**验收时间：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **功能模块** | **验收内容** | **合同要求** | **验收结果** |
| **1** |  |  |  |  |
| **2** |  |  |  |  |
| **3** |  |  |  |  |
| **4** |  |  |  |  |
| **5** |  |  |  |  |
| **6** |  |  |  |  |
| **7** |  |  |  |  |

## 6.3 附件三：项目文档验收单

**验收人：**

**验收时间：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **文档名称** | **用 途** | **验收结果** | **备注** |
| **1** |  |  |  |  |
| **2** |  |  |  |  |
| **3** |  |  |  |  |
| **4** |  |  |  |  |
| **5** |  |  |  |  |
| **6** |  |  |  |  |

## 6.4 附件四：硬件设备验收单

**验收人：**

**验收时间：**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **硬件名称** | **基本用途** | **型号** | **配置情况** | **验收结果** | **备注**  **（机器的IP地址等）** |
| **1** |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |