

可变换姿态的六足机器人

欧阳金笛；韩宇轩；马嘉

第一部分 设计概述

1.1 设计目的

六足机器人旨在应对野外探索和搜索任务中的复杂地形和环境。通过可变换姿态、图像处理和目标识别，机器人提供灵活的机动性、增强的探测能力和稳定的运动性。它能在六足和轮足形态之间切换，适应不同地形。同时，利用图像处理和目标识别技术，机器人能识别、跟踪目标物体，提高探测能力和任务执行效果。这些使得六足机器人在野外搜寻、救援和勘测等领域发挥重要作用，应对复杂场景的挑战。

1.2 应用领域

六足机器人可应用于搜寻救援领域。通过图像处理和目标识别技术，六足机器人能够快速识别废墟中的生命迹象和被困者位置，并提供实时的遥感图像和数据支持。此外，六足机器人可以携带急救设备、药物，为被困者提供基本的医疗援助。它的稳定运动性和机动性使其能够在各种复杂地形条件下行动，并快速运送救援物资。机器人的快速响应和灵活性可以确保紧急救援行动的高效进行。

1.3 主要技术特点

- 1) 实现六足与轮足的机械设计与运动控制，使用几何法进行六足运动学逆解、并通过改进的复合摆线轨迹减小竖直方向的足端冲击力与水平方向的惯性力。
- 2) G4 作为主控元件，通过串口、IIC 与多个模块通信，实现舵机、电机等多个器件的操控。
- 3) 通过 esp32、MQTT 服务器、微信小程序实现实时通信与远程控制。

1.4 关键性能指标

- 1) 资源利用：硬件资源方面，整合了多个设备和模块进行开发，占用 3 个 IIC 和 3 个串口资源。
- 2) 运动性能：机器人重 4.5kg 左右，六足状态下运动速度达到 0.2m/s。通过改进的符合摆线，足端与地面接触前冲击力降至最低。轮足状态下速度达到 0.4m/s。
- 3) 指令及图像传输速率和延迟：视频传输帧率不低于 20fps。考虑到网络因素指令及图像的延迟在 0.5s 左右。

- 4) 人体识别准确率：人体识别模型准确度不低于 90%。
- 5) MQTT 通信服务质量：mqtt 的指令发送需要确保能够收到，并且不能重复发送，因此需要满足 Qos2 的规范。

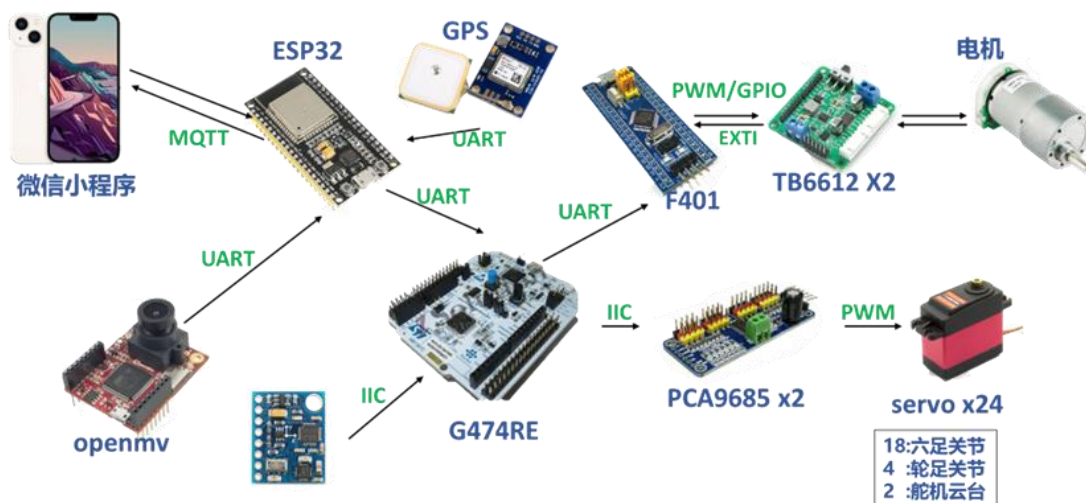
1.5 主要创新点

- 1) 通过 MQTT 协议实现机器人-服务器-微信小程序三方通讯，只要有 4G 信号便可以进行遥控操作，实现了真正意义上的远程操控。
- 2) 机器人姿态、跨步长度、高度、直行方向、旋转方向及曲率均可任意调整。
- 3) 实现了六足和轮足之间的转换，可以适应不同地形。
- 4) 机器人能够通过 IMU 实现自平衡。
- 5) 通过小程序画图，可实现机器人路径规划。
- 6) 结合视觉功能，能够进行人体识别。

第二部分 系统组成及功能说明

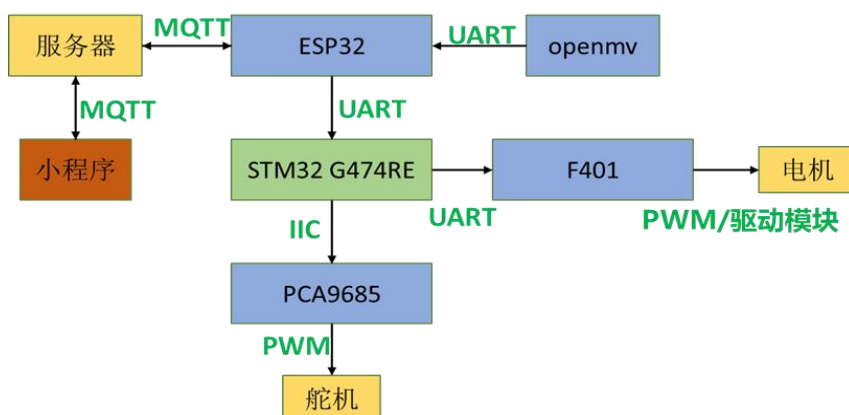
2.1 整体介绍

本项目用到的模块如下：



机器人主控是 STM32 G474RE 开发板，在主控上进行控制及运动解算。G4 通过 IIC 与两个 PCA9685 机型通讯，进而控制 24 个舵机；G4 通过 UART 与一个 F401 核心板进行通讯，进而控制 4 个电机驱动轮足转动；此外，G4 还可以通过 IIC 获取 MPU6050 的加速度数据，进行姿态解算，以及通过 ESP32 接收来自微信小程序用户端的指令。

系统整体框图如下：



2.2 各模块介绍

2.2.1 主控

主控 G4 用于接收来自用户的指令，完成对应的运动功能。主控中完成的功能如下：

1) 机器人规范定义：

定义了机器人的物理参数和运动参数。物理参数包括六足每个关节的对应的腿长、机器人主体的长宽、六个足端的平衡坐标、六条腿的夹角等；运动参数包括机器人的运动模式、运动速度、跨步的步长、抬腿高度以及机器人在个方向的姿态角。

2) 足端轨迹定义

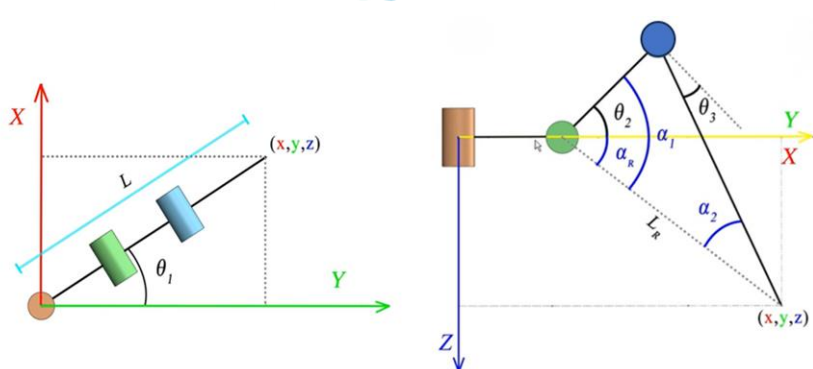
定义了机器人运动时六条腿分别对应的足端轨迹坐标。一段特定的轨迹坐标通过插值实现细化。足端轨迹六足直行、六足转弯、六足与轮足转换、六足姿态改变、六足抬腿等运动对应的轨迹。其中足端轨迹采用改进的摆线轨迹，使落地时的将速度降为 0，降冲击力降至最低。摆线轨迹如下：

$$\begin{cases} x = S \left[\frac{t}{T_m} - \frac{1}{2\pi} \sin \left(\frac{2\pi t}{T_m} \right) \right] \\ y = H \left[\operatorname{sgn} \left(\frac{T_m}{2} - t \right) (2f_E(t) - 1) + 1 \right] \end{cases}$$

$$f_E(t) = \left[\frac{t}{T_m} - \frac{1}{2\pi} \sin \left(\frac{2\pi t}{T_m} \right) \right]$$

3) 运动学求逆

通过轨迹规划得到足端坐标值之后，通过运动学逆解得到一条腿上三个关节的角度。



已知足端相对于第一关节的坐标 (x, y, z) 之后，便可以通过公式可以求出三个关节的转动角度：

$$\begin{aligned}\theta_1 &= \text{atan2}(x, y) \\ L &= \sqrt{x^2 + y^2} \\ \alpha_R &= \text{atan2}(z, L - l_1) \\ L_R &= \sqrt{z^2 + (L - l_1)^2} \\ \alpha_1 &= \arccos((L^2 + l_2^2 - l_3^2)/(2 * l_2 * L)) \\ \alpha_2 &= \arccos((L^2 - l_2^2 + l_3^2)/(2 * l_3 * L)) \\ \theta_2 &= -\alpha_R + \alpha_1 \\ \theta_3 &= -90 + (\alpha_1 + \alpha_2)\end{aligned}$$

4) 舵机驱动

G4 主控与 PCA9685 通过 IIC 进行通讯，通过 I2C 总线向 PCA9685 寄存器写入数据，以配置舵机通道、PWM 频率和初始位置等参数。

G4 使用 I2C 总线向 PCA9685 的对应寄存器写入 PWM 脉冲宽度数据，以控制舵机的角度。PCA9685 支持 16 个通道，每个通道都有两个寄存器来控制脉冲宽度，通过设置这两个寄存器的值来控制舵机的角度。

5) 运动组合

G4 实现了各种状态下的运动组合，包括转弯，直行、变换姿态等。

6) 姿态获取及自稳定

通过 IIC 通讯获取 MPU6050 的寄存器值、得到角速度和加速度，通过卡尔曼滤波结合定时器分析出当前的姿态角。

7) 指令处理主循环

G4 主控定义和实现了各种指令下的触发操作。

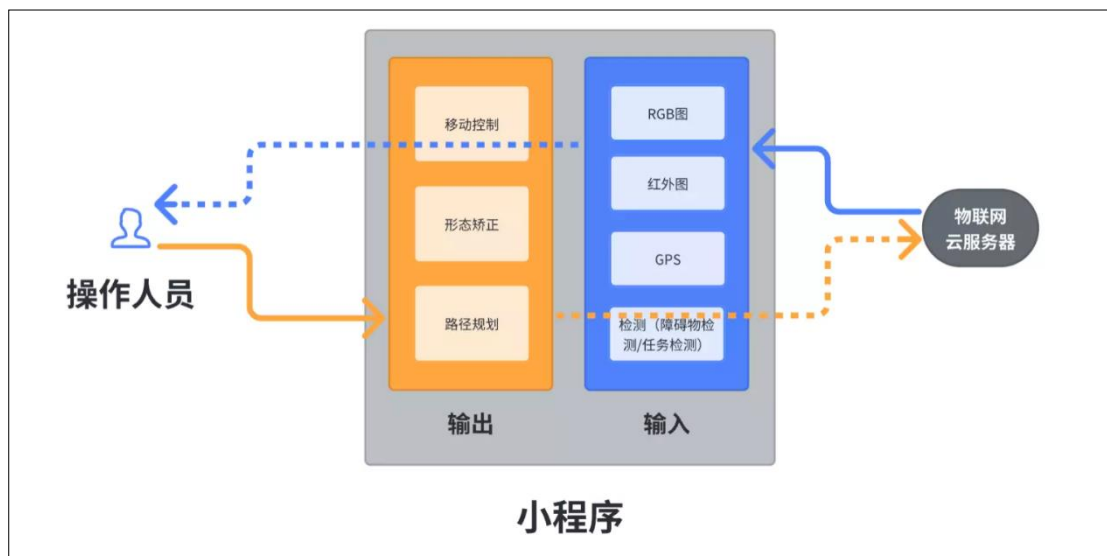
2.2.2 F401 扩展板

由于主控板的 GPIO 口数量有限，而驱动 4 个轮子相关电机进行运动需要耗费一定数量的 GPIO 口，这里选择使用一个 F401 扩展板专门用于电机控制。

F401 通过上升沿、下经验触发检测轮足的转速，通过 PID 调节轮子的转速。

2.2.3 小程序模块

1) 整体框图



我们的小程序基于微信开发者平台开发，主要用于远程接受小车摄像头和传感器信息，控制小车运动，必要时进行矫正。

2) 基本页面布局

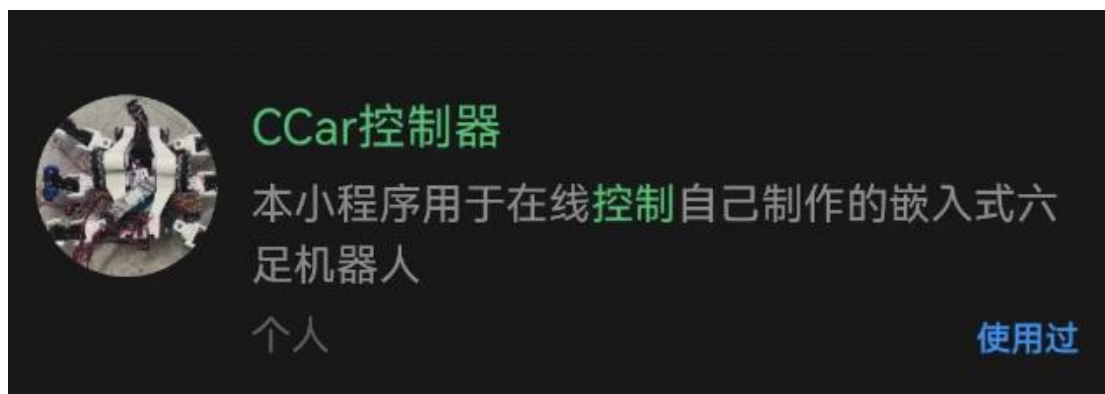


上半部分页面是作为接受和显示视频的窗口，以及显示 GPS，目标检测状态等信息栏。下半部分分为三个控制页面，分别是移动控制、形态控制和导航云台。

小程序可以接受用户的操控，并将信息传送给云服务器，再由云服务器将信息传递给嵌入式机器人；同时小程序可以通过订阅主题的方式，从云服务器中获取到嵌入式机器人传来的信息，并显示给用户。

3) 小程序上线

我们的小程序如今已经通过审核并发布和上线，可以搜索“CCar 控制器”找到我们的控制器小程序。



2.2.4 云服务器

1) MQTT 简介

MQTT 是一种轻量级的发布/订阅消息传递协议，旨在在受限设备和低带宽、高延迟或不可靠网络中使用。

MQTT 协议基于 TCP/IP 协议栈，支持多种编程语言和平台，具有灵活、可靠、轻量级等特点。MQTT 中的三个重要概念是发布者（publisher）、订阅者（subscriber）和主题（topic）。发布者将消息发布到一个主题，而订阅者则订阅感兴趣的主题，从而接收相应的消息。

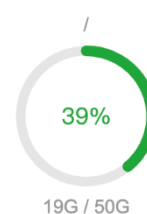
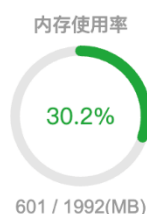
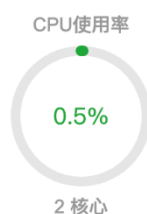
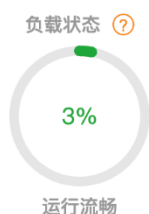
我们微信小程序和机器人之间的通信依据 MQTT 协议来完成。

2) 基本信息

我们依托于云服务器搭建我们 MQTT 的服务器，用以统一管理消息的收发。我们的云服务器基于腾讯云搭建，其基本信息如下：

 系统: Ubuntu 18.04.4 LTS x86_64(Py3.7.8)

状态



由于微信小程序的使用要求，同时确保网站的安全，我们申请了域名，进行了域名备案，并申请了 ssl 证书，添加了对应的解析。可以通过 www.mqttt.xyz 来访问我们的服务器网站。

3) EMQX 搭建

网站搭建完成之后，需要搭建 EMQX 从而使得云主机上可以运行 MQTT 服务器程序。

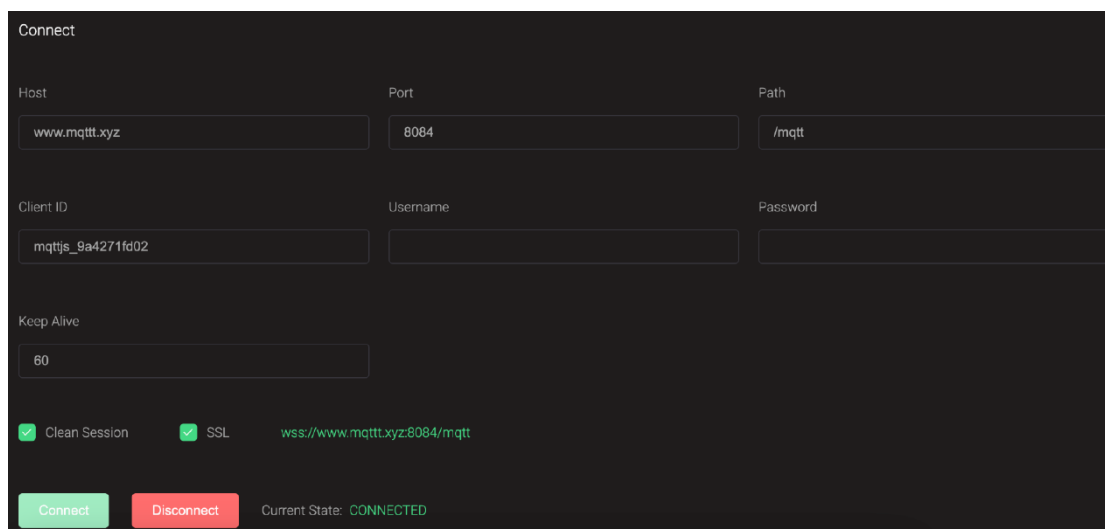
我们的端口配置如下：

端口	说明
1883	MQTT/TCP 协议端口，一般用于客户端端口
11883	MQTT/TCP 协议内部端口，仅用于本机客户端连接
8883	MQTT/SSL 协议端口
8083	MQTT/WX 协议端口
8084	MQTT/WSS 协议端口
18083	EMQX 后台管理端口

通过解压、安装 emqx 开源工具，并运行 emqx，它就会默认在以上放通的端口上工作。我们微信小程序连接的是 8084 端口，python 端连接的是 1883 端口。

4) 测试

使用 websocket 进行连接测试（带 ssl 验证）：



The screenshot shows the EMQX Connect dashboard with the following fields and values:

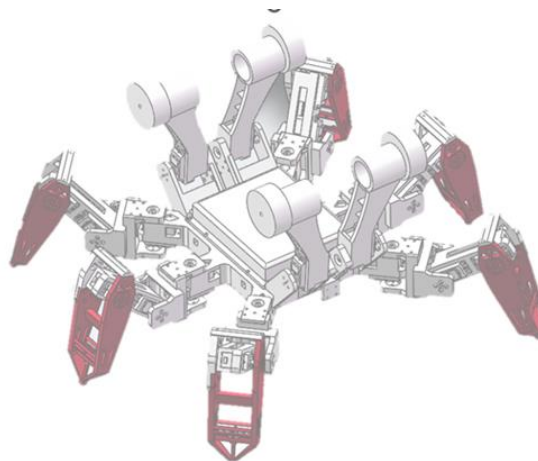
- Host:** www.mqtt.xyz
- Port:** 8084
- Path:** /mqtt
- Client ID:** mqttjs_9a4271fd02
- Username:** (empty)
- Password:** (empty)
- Keep Alive:** 60
- Checkboxes:** Clean Session (checked), SSL (checked)
- URL:** wss://www.mqtt.xyz:8084/mqtt
- Buttons:** Connect (green), Disconnect (red)
- Current State:** CONNECTED

在 dashboard 上，可以看到已经正确连接上云主机，并可以进一步根据 MQTT 协议来收、发消息。

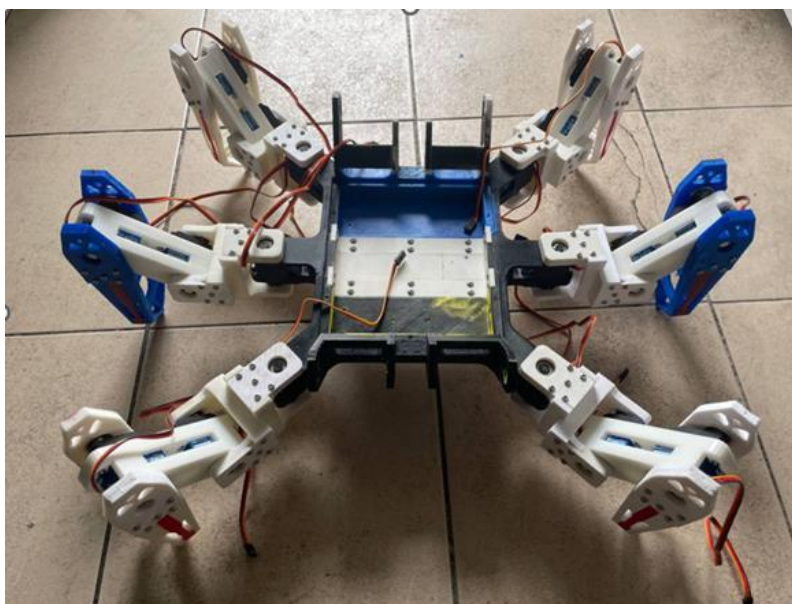
第三部分 完成情况 & 性能参数

1) 机械设计方面：

通过 solidworks 进行机器人的结构设计，实现六足和轮足转换。机械设计如图所示：



通过 3D 打印得到最终的（部分）机器人外壳，如图所示：



2) 运动实现方面：

实现了六足机器人的直行（任意角度、跨度、高度）、转弯（任意方向、曲率半径、跨度）及各种转换步态，由于机器人可以自行进行运动学逆解，因此可以实现多种姿态变换。同时将姿态变换与 mpu6050 结合起来，可以实现机器人的自稳定。六足形态下，在跨步步长为 80mm，速度为第三档的前提下，直行速度可以达到 0.2m/s。此外，

3) 通讯方面：

在腾讯云主机上搭建 MQTT 服务器，基于 MQTT 协议，实现手机端微信小程序和机器人的远程通信。手机端可以基于 MQTT 协议，发布主题，向机器人发出控制、矫正、形态调节和路径规划等命令，同时可以依据 MQTT 协议，订阅机器人发布的信息主题，接收机器人传送的视频、红外图、GPS 和状态（是否检测到人/障碍物，工作是否正常）等信息。通信维持 Qos2 服务质量，保证消息接收和发送不重不漏，并严格控制主题数量和消息长度，保证消息接受和发送的效率。

4) 视觉方面:

实现了 11 层 cnn 的部署, 可以实现识别摄像头中是否有人体出现

第四部分 总结

4.1 可扩展之处

4.1.1 SLAM

六足机器人相对于其他机器人来说, 一个巨大的优势就是它的稳定性, 这代表了它在野外探索等方面具有巨大的优势。六足机器人结合 SLAM 技术可以通过感知周围环境并构建地图, 实现自主导航和避障, SLAM 技术还可以提供高精度的定位信息, 使机器人在复杂的环境中定位更加准确。

结合 SLAM 技术的六足机器人可以更好地完成多种任务, 例如自主巡逻、搜索和救援、仓库管理等。通过准确的定位和地图信息, 机器人可以规划有效的路径, 执行复杂的任务并在实时中进行调整。

4.1.2 多智能体协同

在户外环境中, 六足机器人面临着地形不平坦、复杂障碍物、不确定的天气条件等挑战。单个六足机器人可能会遇到困难, 无法有效地完成任务。因此, 多智能体协同成为解决这一问题的有效方法之一。

多个六足机器人可以通过分工合作来完成任务。例如, 其中一些机器人可以负责探索前方路线, 另一些机器人可以负责携带货物或进行数据收集。通过合理的任务分配和协同工作, 可以提高任务完成效率。在户外环境中, 六足机器人需要找到可行的路径来跨越复杂地形和障碍物。多个六足机器人可以通过协同路径规划来共同制定行动方案。它们可以通过集体决策算法, 通过评估和比较不同路径的优劣, 选择最合适的路径来达到目标。

需要指出的是, 户外六足机器人的多智能体协同依赖于高度智能的算法和控制系统。这些算法和控制系统需要能够实时处理和分析各个机器人的数据和情境信息, 做出相应的决策和调整。因此, 开发出高效可靠的多智能体协同算法和控制系统是户外六足机器人研究的重要方向。

4.2 心得体会

通过这个项目的设计, 我深刻认识到了机器人设计需要综合考虑多个因素, 如功能需求、传感器选择、通信与控制等。同时, 多功能设计、多传感器融合的思想对于提升机器人的性能、智能和适应性至关重要。在合作的过程中我们也发现了信息充分共享对于项目开发的促进作用

在工作过程中, 我们意识到在进行机器人设计时首先做好仿真的重要性, 本次项目我们只进行了步态仿真, 却没有考虑到实际舵机扭矩测试等物理仿真, 导

致最终测试阶段出现了一些问题。

我相信这些经验和体会将对未来的机器人设计和开发工作产生积极的影响。

第五部分 参考文献

- [1] SAKAKIBARA Y, KAN K, HOSODA Y, et al. Foot trajectory for a quadruped walking machine[C] // Proceedings IROS' 90. IEEE International Workshop on, July 3-6, 1990, Ibaraki, Japan. New York, NY, USA: IEEE, 1990: 315-322.
- [2] 何冬青, 马培荪. 四足机器人动态步行仿真及步行稳定性分析[J]. 计算机仿真, 2005(2):146-149. HE Dongqing, MA Peisun. Simulation of dynamic walking of quadruped robot and analysis of walking stability[J]. Computer Simulation, 2005(2): 146-149.
- [3] 李贻斌, 李彬, 荣学文, 等. 液压驱动四足仿生机器人的结构设计和步态规划[J]. 山东大学学报, 2011(5):32-36, 45. LI Yibin, LI Bin, RONG Xuewen, et al. Mechanical design and gait planning of a hydraulically actuated quadruped bionic robot[J]. Journal of Shandong University, 2011(5):32-36, 45.
- [4] Hunkeler, U, Truong, C., &Stanford-Clark, A. (2008). "MQTT—a publish/subscribe protocol for Wireless Sensor Networks." Proceedings of the 3rd IEEE International Conference on Communication Systems Software and Middleware, 791-798.

第六部分 附录

```
/*主循环, 接收指令的对应逻辑*/
void mainloop(struct hexapod* hexa_p)
{
    while(1) {
        if((*hexa_p).start!=1) {
            //没有启动时等待开机信号, 等到开机信号则启动
            if(data.command==c_start_stop && fabs(data.data-1.0)<1e-5) {
                //开机时首先从下蹲到站立状态, 开机时默认进入六足状态
                stand(hexa_p, 80);
                //修改状态值
                (*hexa_p).start=1;
                (*hexa_p).mode=1;
                (*hexa_p).mode_6lg=2;
            }
        }
        else{
            switch(data.command) {
                case(c_move_strait):
```

```

/*直行命令，如果在六足状态，首先需要调整为默认状态*/
if((*hexa_p).mode==1){
    //六足模式下首先应该恢复默认状态
    to_state(hexa_p,0.0,0.0,0.0,(*hexa_p).tz);
    straight_6lg(hexa_p);
}
else if((*hexa_p).mode==2)
    ;
else if((*hexa_p).mode==3)
    straight_wheel(hexa_p); //轮足状态下直行
break;
case(c_move_turn):
    /*如果在六足状态，首先需要调整为默认状态*/
    if((*hexa_p).mode==1){
        //六足模式下首先应该恢复默认状态
        to_state(hexa_p,0.0,0.0,0.0,(*hexa_p).tz);
        turn_6lg(hexa_p);
    }
    else if((*hexa_p).mode==2)
        ;
    else if((*hexa_p).mode==3)
        turn_wheel(hexa_p); //轮足状态下直行
    break;
case(c_switch_mode):
    /*转换步态*/
    if(fabs(data.data-1.0)<1e-5){
        if((*hexa_p).mode==3)
            wheel2six(hexa_p);
    }
    else if(fabs(data.data-3.0)<1e-5){
        if((*hexa_p).mode==1){
            to_state(hexa_p,0.0,0.0,0.0,(*hexa_p).tz);
            six2wheel(hexa_p);
        }
    }
    break;
case(c_switch_velocity):
    (*hexa_p).speed=data.data;
    break;
case(c_switch_span):
    (*hexa_p).span=data.data;
    break;
case(c_roll):

```

```

to_state(hexa_p,data.data,(*hexa_p).pitch,(*hexa_p).yaw,(*hexa_p).tz);
    break;
case(c_pitch):

to_state(hexa_p,(*hexa_p).roll,data.data,(*hexa_p).yaw,(*hexa_p).tz);
    break;
case(c_yaw):

to_state(hexa_p,(*hexa_p).roll,(*hexa_p).pitch,data.data,(*hexa_p).tz);
    break;
case(c_hieght):
    printf("%f\n",(*hexa_p).tz);

to_state(hexa_p,(*hexa_p).roll,(*hexa_p).pitch,(*hexa_p).yaw,-data.data);
    break;
case(c_platform_h):
    SERVO_SPT(11,-data.data);
    HAL_Delay(100);
    break;
case(c_platform_v):
    SERVO_SPT(23,-data.data);
    HAL_Delay(100);
    break;
case(c_path_planning):
    pathplan(hexa_p);
    break;
case(c_self_stable):
    to_state(hexa_p,0.0,0.0,0.0,-(*hexa_p).height*1.2);
    self_stable(hexa_p);
    break;
case(c_servo_start):
    servo_correct(hexa_p);
    break;
case(c_start_stop):
    if(fabs(data.data-0.0)<1e-5){
        if((*hexa_p).mode==3)
            wheel2six(hexa_p);
        to_state(hexa_p,0.0,0.0,0.0,-(*hexa_p).height);
        outer(hexa_p);
        crouch(hexa_p,80);
        (*hexa_p).start=0;
    }
    break;
default:

```

```

        break;
    }
}
}
}

/*定义各种命令对应的指令*/
#define c_move_strait 1           //直行
#define c_move_turn 2            //转弯
#define c_switch_mode 3          //转换步态, 1六足, 3轮足
#define c_switch_velocity 4      //改变速度 1 2 3档
#define c_switch_span 5          //步长跨度
#define c_roll 6                  //改变roll角          #弧度
#define c_pitch 7                 //改变pitch角         #弧度
#define c_yaw 8                   //改变yaw角          #弧度
#define c_hieght 9                //改变行走高度
#define c_platform_h 10           //改变舵机云台的水平角度
#define c_platform_v 11           //改变舵机云台的竖直角度
#define c_path_planning 12        //路径规划
#define c_stop 13                 //停止
#define c_self_stable 14          //自稳定
#define c_self_stable_quit 13     //退出自稳定, 注意和stop相同
#define c_servo_start 15          //开始舵机矫正, 进入程序
#define c_servo_quit 13           //退出舵机矫正
#define c_servo_crooect 16        //舵机矫正参数
#define c_start_stop 17           //1开机, 0关机

```