2018 年 IDT 传感器应用创新设计大赛

作品名称：轮足两用可变形环境感知探测机器人

摘 要

本作品为一种轮式、足式两用环境探测感知机器人。该机器人具有轮式滚动、多足爬行两种形态，能够适应各种复杂地形的同时，具有较强的移动灵活性，可用于井矿、地穴、消防等危险领域的环境感知探测工作。

作品难点在于：1、轮足可变形的机器人结构设计；2、机器人六足爬行形态的精确控制。

作品的创新点在于：1、采用 ZMOD4410 气体传感器，具有较高的气体环境感知检测精度和较强的稳定性，同时根据传感器 ODOR 模式的特性实现了气体浓度突变的自动报警功能，能够第一时间通知探测人员撤离危险区域；2、机器人能够变形，拥有轮式滚动和多足爬行两种运动形态，在满足移动速度的同时能够适应多种复杂地形，具有较强的应用前景。

机器人硬件系统由机器人控制器、ZMOD4410 HiCom 模块、无线图传模块、电机驱动模块四个部分组成。其中，机器人控制器用于控制机器人的整体运动；ZMOD4410 传感器用于测量气体环境参数，包括 TVOC、IAQ、eCO2 以及气体浓度是否发生突变等参数；无线图传模块用于远程传输机器人当前所处环境的图像信息，方便操作人员查看机器人当前所处的地形；电机驱动模块用于驱动机器人轮式形态启动的直流有刷电机，控制电机的转速和转向，进而控制机器人轮式滚动形态的前行速度和方向。

未来，本作品将进行如下功能完善：加入惯性导航控制和激光雷达室内构图功能，使该机器人能够成为具有跨越台阶、门槛等复杂地形场所功能的扫地机器人。

关键词：变形机器人；气体传感器；环境探测

目录

1 作品介绍

1.1 作品功能

如图 1.1 所示，本作品为一种轮式、足式两用环境探测感知机器人。该机器人具有轮式滚动、多足爬行两种形态，能够适应各种复杂地形的同时，具有较强的移动灵活性，可用于井矿、地穴、消防等危险领域的环境感知探测工作。

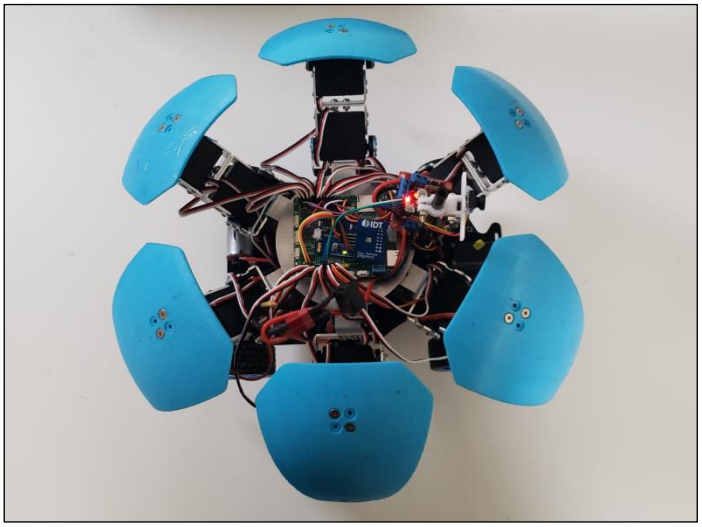


图 1.1　作品实物图片

机器人的工作原理为：环境探测人员通过无线遥控模块操控机器人进入待探测区域，通过搭载在机器人前部的无线图传模块获取当前的地形图像，根据地形的特点启动机器人的轮式滚动或多足爬行功能进入探测区域；同时，安装在机器人内部的气体检测模块将实时检测当前的环境气体参数，并将检测结果通过无线通信模块发送给操作者，操作者根据当前的探测结果判断能否开展实际工作，从而实现未知危险环境的感知探测功能。

作品采用 ZMOD4410 模块检测危险环境区域的气体浓度参数，包括 TVOC、IAQ、eCO_2。同时，针对 ZMOD4410 ODOR 模式能够精确感知气体浓度突变的特性，作品设计了气体浓度突变的自动报警功能，从而在危险降临时能够第一时间通知探测人员撤离危险区域。

1.2 作品难点与创新点

作品难点在于：1、轮足可变形的机器人结构设计；2、机器人六足爬行形态的精确控制。

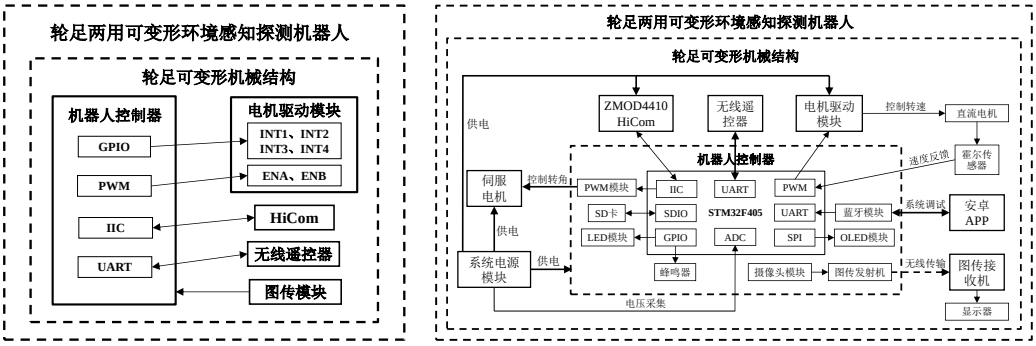
作品的创新点在于：1、采用 ZMOD4410 气体传感器，具有较高的气体环境感知检测精度和较强的稳定性，同时根据传感器 ODOR 模式的特性实现了气体浓度突变的自动报警功能，能够第一时间通知探测人员撤离危险区域；2、机器人能够变形，拥有轮式滚动和多足爬行两种运动形态，在满足移动速度的同时能够适应多种复杂地形，具有较强的应用前景。

2 硬件设计

如图 2.1a 所示，机器人硬件系统由机器人控制器、ZMOD4410 HiCom 模块、无线图传模块、电机驱动模块四个部分组成。其中，机器人控制器用于控制机器人的整体运动；ZMOD4410 传感器用于测量气体环境参数，包括 TVOC、IAQ、eCO2 以及气体浓度是否发生突变等参数；无线图传模块用于远程传输机器人当前所处环境的图像信息，方便操作人员查看机器人当前所处的地形；电机驱动模块用于驱动机器人轮式形态启动的直流有刷电机，控制电机的转速和转向，进而控制机器人轮式滚动形态的前行速度和方向。

如图 2.1b 所示，本次设计的机器人控制器由 PMW 模块、OLED 模块、LED 模块、蜂鸣器模块、蓝牙模块、SD 模块、MCU 模块共七个模块组成。其中，PWM 模块用于控制多足爬行形态下启动的伺服电机的转角；OLED 模块用于显示系统参数信息，方便系统的调试；LED 模块、蜂鸣器模块用于提示系统当前所处的状态信息，方便操作人人员的操作；蓝牙通信模块配合安卓 APP 可完成机器人的短程无线遥控操作，同时能够以数据图表的方式统计气体传感器数据的变化趋势；SD 卡模块用于存储 MCU 的运行日志问题，方便系统故障排查与解决；无线通信模块用于机器人的远程遥控操作。

MCU 模块用于整个机器人系统的运动控制，通过 IIC 通信的方式读取 ZMOD4410 传感器的数据，配合 IDT 官方提供的算法库，能够准确读取 TVOC、IAQ、eCO_2 气体浓度参数；MCU 以 PWM 脉冲的方式驱动电机驱动模块，输出与电压线性相关的电压，从而控制直流有刷电机的转速与转向；远程无线遥控器通过 2.4GHz 频段发送操作人员的控制指令，机器人本地接收器将接收的控制信号以 UART 通信的方式传递至 MCU，最终完成机器人控制指令的执行。系统电源模块用于整个机器人系统的供电。



a 系统硬件组成图　　b 系统硬件连接原理图

图 2.1 系统硬件连接框图

下面对机器人硬件的主要模块设计进行展开说明。

2.1 硬件电路设计

2.1.1 PWM 模块

本文设计的机器人需 19 路 PWM 信号控制伺服电机的转角，同时需两路 PWM 信号控制两个直流无刷电机的转速和转向，故本文设计的机器人系统至少需要 21 路 PWM 脉冲信号。本文选用 NXP 公司的 PCA9685 芯片设计 PWM 模块，该芯片通过 IIC 通信，单颗芯片可同时产生 16 路频率相同的 PWM 脉冲信号。本文选用两颗 PCA9685 芯片产生 32 路 PWM 信号。电路原理图如图 2.2 所示。

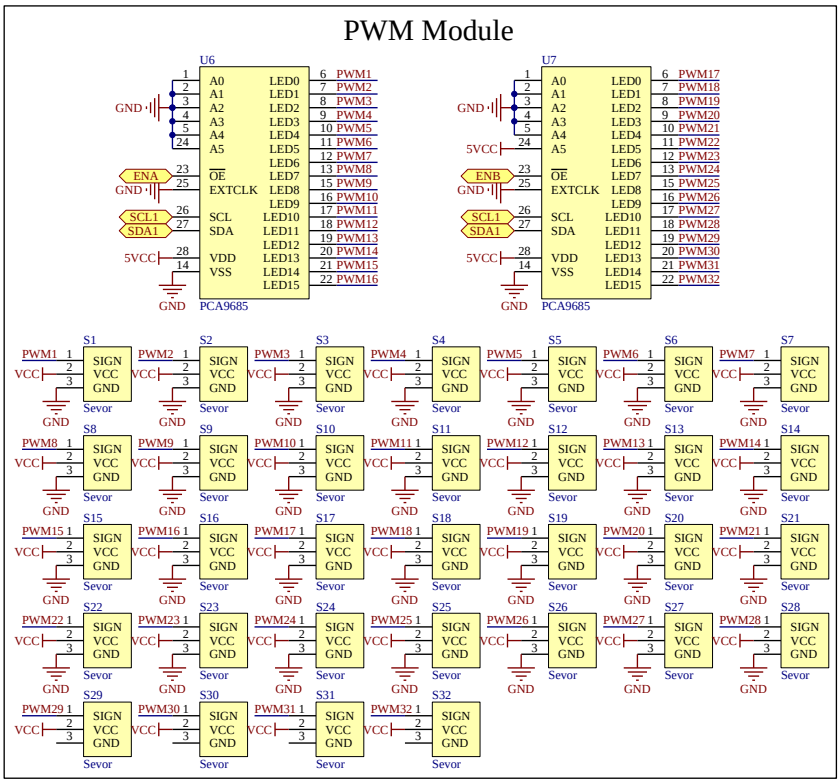


图 2.2 PWM 模块电路原理图

2.1.2 蓝牙模块

蓝牙通信模块用于机器人系统向手机 APP 客户端发送机器人当前状态的数据，方便系统整机调试和人机交互。本文选用信驰达的 RF-BM-S02I 4.0BLE 作为机器人的蓝牙通信模块。该模块可外接增益天线，通信距离较远，可通过 UART 通信接口与 MCU 通信。模块实物图片如图 2.3 所示，安卓 APP 上位机截图如图 2.4 所示。

图 2.3　RF-BM-S02I 蓝牙模块实物图片

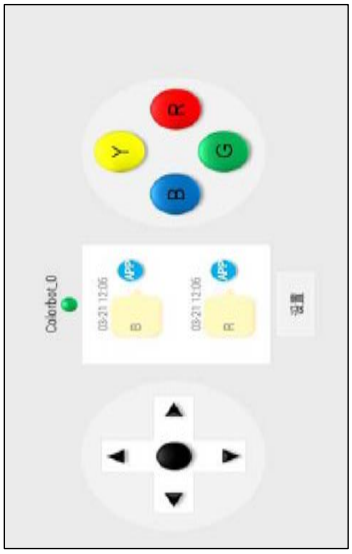


图 2.4　本文使用的安卓 APP

2.1.3　OLED 模块

OLED 显示模块用于显示系统参数数据，方便机器人系统调试。本次设计选用中景电子科技公司的 0.96 寸 OLED 模块，该模块通过 SPI 接口与 MCU 通信，模块实物图片如图 2.5 所示。

2.5　OLED 模块实物图片

图 2.6　HX60D 舵机实物图片

图 2.7　气体传感器模块实物图片

2.1.4　伺服电机

根据前期的结构仿真分析，本文设计的轮足两用机器人整体总量约为 3kg。根据机器人的运动特性可知，机器人多足爬行使用三角步态行走时承受的负载最大，且腿关节最大，需承受约 1kg 的扭矩。根据机器人各关节的扭力负载分析，本文选用 HX60D 并行数字舵机作为机器人的关节驱动电机。该舵机最大扭力可达 18kg/cm，输入电压为 4.8~8.4V，转角范围为 0~270 度，舵机实物图片如图 2.6 所示。

2.1.5　气体传感器模块

本文选用 ZMOD4410 HiCom 模块作为机器人的气体传感器模块，模块实物图片如图 2.7

所示。ZMOD4410 传感器的组成如图 2.8 所示。通过 IDT 关于 TVOC、eCO_2 和 CO_2 的数据统计可知，如图 2.9 所示，三者之间存在极强的线性关系，故可直接通过 TVOC 的数据计算出 eCO_2 的数据，进而估计出实际空气中 CO_2 的浓度大小。根据 IDT 官方提供的算法库文件，可直接通过调用 API 的方式计算出 TVOC、IAQ、eCO_2 三个参数的大小，能够监控气体浓度是否在短时间内发生突变，极大地简化了气体浓度检测的硬件设计和软件设计，同时具有极强的稳定性和准确性。本文通过连接 HiCom 的 IIC 接口与 MCU 通信，未使用 USB 功能。

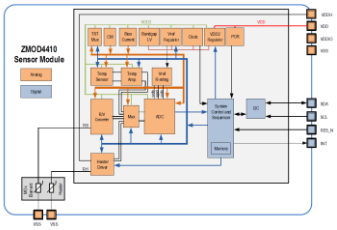


图 2.8　ZMOD4410 传感器组成

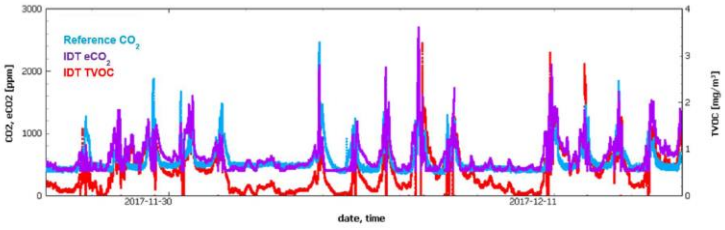
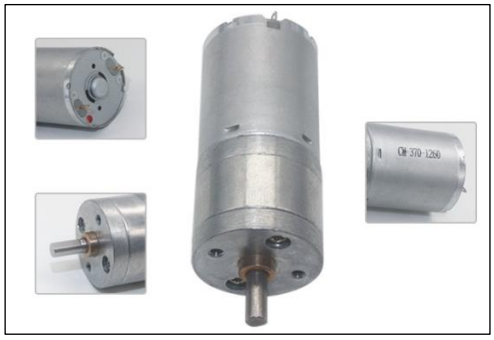


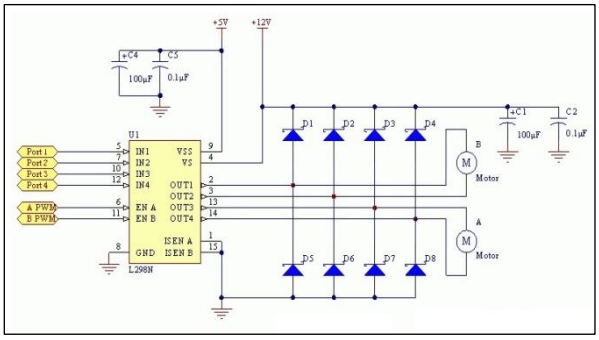
图 2.9　气体传感器模块实物图片

2.1.6　电机驱动模块

根据机器人轮式滚动形态需保持较快转速快速前行的需求，且轮式滚动形态为两路直流电机后驱动 1 路伺服电机前转向的特性，本文选用两路图 2.10(a) JGA25-370 直流减速电机作为机器人的轮式驱动电机，该电机的额定工作电压为 12V，额定工作电流为 0.5A，额定转速为 215RPM。根据该电机的工作特性，本文选用 L298N 模块作为电机的驱动模块。该模块可控制两路电机的转向与转速，每路输出电压范围为 0.6~12V，每路最大输出电流为 2.0A。该模块的电流原理图如图 2.10(b)所示，其中：IN1~IN4 与 MCU 的 GPIO 相连接，用于控制两路电机的转向，ENA、ENB 与 PCA9685 的 PWM OUT 端相连，通过 PWM 占空比控制两路电机的转速。



a　JGA25-370 直流减速电机

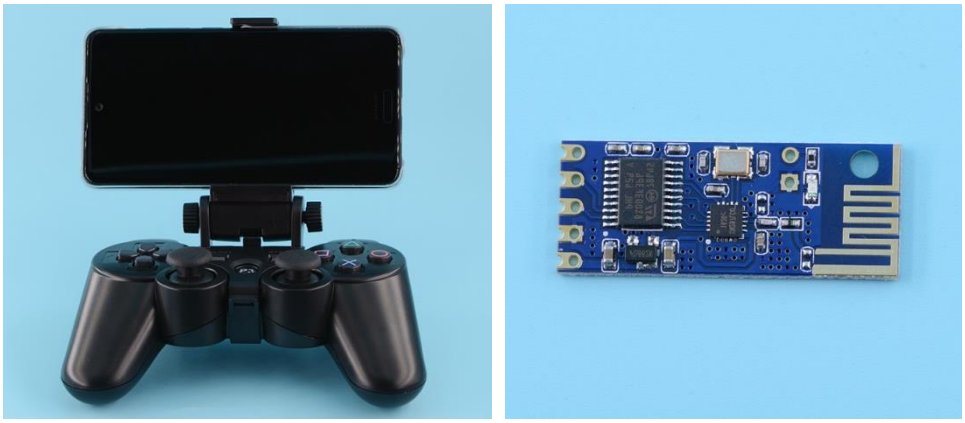


b　L298N 电机驱动原理图

图 2.10　气体传感器模块实物图片

2.1.7 无线通信模块

为了实现机器人的无线远程遥控，本文选用常用的 433MHz 无线通信模块，模块通信距离可达 1000m。如图 2.11 所示，该模块由发射机和接收机两个部分组成：其中，发射机由 14 个按键和两个全方向摇杆组成，可发送 48 种控制信号；接收机通过 UART 通信接口与 MCU 相连。



a 发射机实物图片　　b 接收机实物图片

图 2.11 无线遥控器模块

2.1.8 无线图传模块

为了保证机器人的远程无线图像传输，本文选用图 2.12(a)的 FPV 摄像头、图 2.12(b)的 TS5828 图传发射机模块、图 2.12(c)的 RC832 图传接收机模块作为本文设计的机器人的无线图像传输模块。

a fpv 摄像头　　b TS5828 图传发射机　　c RC832 图传接收机

图 2.12 无线图传模块

2.1.9 系统电源模块

根据前述内容分析可知，本文设计的机器人硬件系统需 10A 以上的电流供给，同时需产生 5V、3.3V 电压。为此，本文选用图 2.13 所示的 2200mAH 2S 锂电池作为系统的唯一供

电电源。该电池为航模电池，能够稳定 22A 的电流，输出电压为 7.4~8.4V。锂电池的输出电压直接为伺服电机、直流减速电机供电，5V、3.3V 的电压通过图 2.14 的线性稳压电路产生。其中 5V 电压选用 L7805 三端集成稳压芯片产生，最大能够输出 1.5A 的电流。3.3V 电压通过 AMS1117-3.3 稳压芯片产生。为了提升系统的稳定性，使用 XC6206P332 单独产出 3.3V 的电压为 MCU 供电。

图 2.13　2S 锂电池

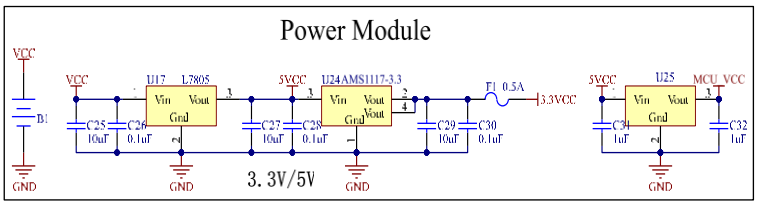


图 2.14　3.3V 和 5V 线性稳压电路原理图

2.1.10　MCU 模块

根据上述硬件模块选型及机器人系统浮点型数据计算的需求分析，本次设计选用 ST 公司的 STM32F405RET6 单片机作为机器人系统的控制单元。该控制器以 ARM Cortex-M4 为内核，CPU 主频为 168MHz，内存容量为 512KB，具有多个定时器、USART 接口、IIC 接口、SPI 接口、ADC 通道、DAC 通道、DMA 通道等外设资源。此外，该单片机具有的 FPU 浮点型数据计算单元能够精确快速求解机器人步态行走时所需的运动数据，完全能够满足本文设计的需要。该芯片最小系统电路原理图如图 2.15 所示。

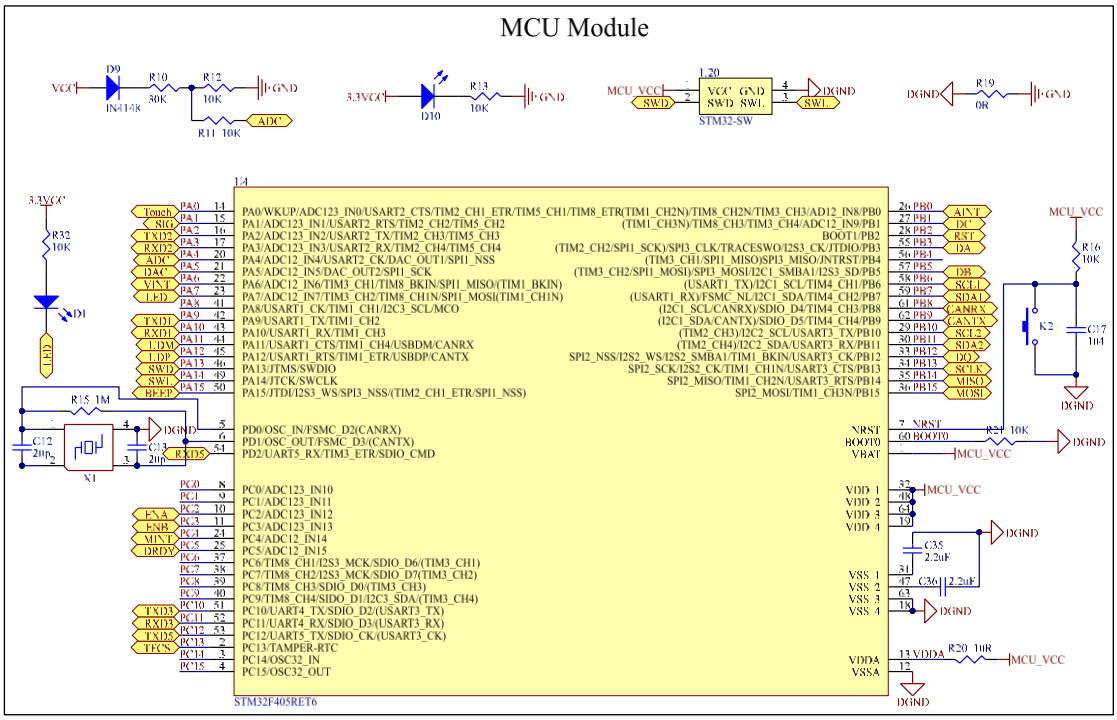


图 2.15　本次选用的单片机最小系统电路原理图

2.2 硬件实物

机器人控制器的实物图片如图 2.16 所示，系统总体硬件连接实物图片如图 2.17 所示。

图 2.16 本文设计的机器人控制器实物图片

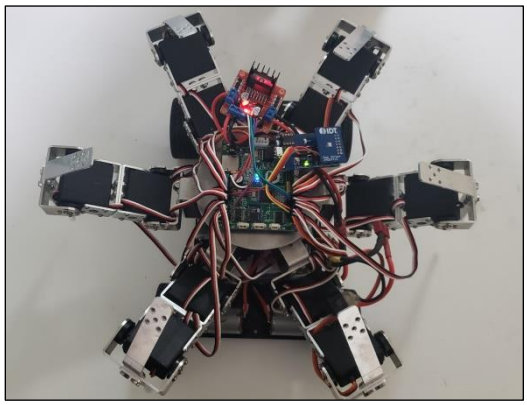


图 2.17 系统总体硬件连接实物图片

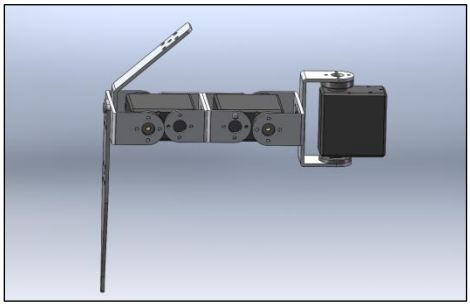
3 结构设计

如何实现机器人轮式滚动、多足爬行两种形态的任意切换是本文设计的难点与重点。为此，本文的结构设计采用模块化组装的方法，机器人总体结构由轮式滚动结构、多足爬行结构两个模块组成。其中，轮式滚动结构为后驱前转弯四轮结构，由一个伺服电机驱动转向和两个直流电机提供前行动力；多足爬行结构为六足爬行结构，由六条腿组成，每条腿有三个自由度，总计 18 个自由度，由 18 个伺服电机驱动。

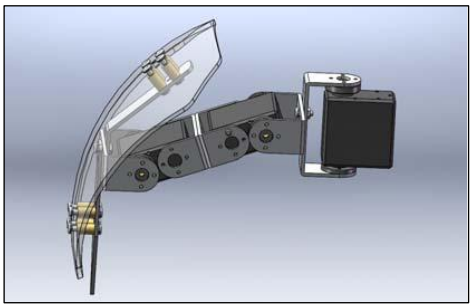
机器人变形的原理为：当机器人处于轮式滚动形态时，上半部分的多足爬行结构将保持抬腿收缩形态，保持六条腿完全脱离地面，此时机器人将通过后轮驱动前轮转向向前滚动；当机器人收到变形指令时，机器人后轮停止直流电机转动，同时，上半部分的多足爬行结构将缓慢向下伸腿蹬地，直到完全将下半部分的轮式滚动结构撑离地面，机器人开始多足爬行。

前期，本文使用三维造型软件 SolidWorks 对机器人的结构设计进行了充分的仿真验证，从原理上证明了设计方案的可行性。下面开始描述机器人的结构设计过程，如图 3.1 所示。

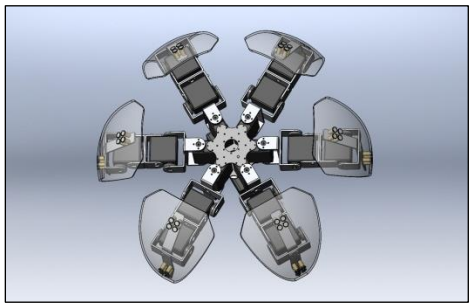
(a) 舵机装配图



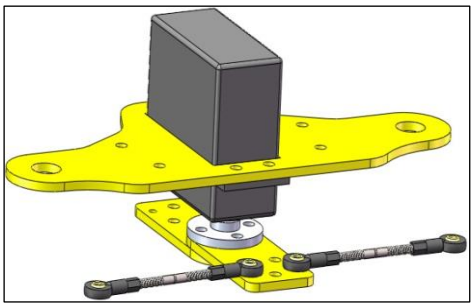
(b) 多足爬行结构腿部装配图



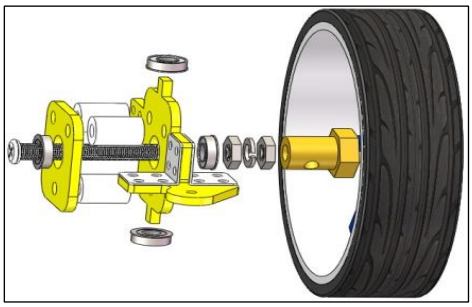
(c) 多足爬行结构单腿装配图



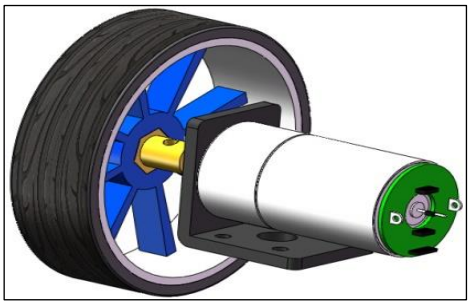
(d) 多足爬行结构整体转配图



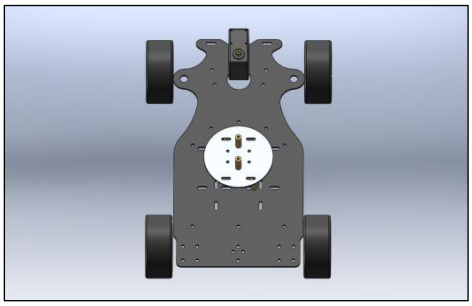
(e) 轮式滚动结构转向结构装配图



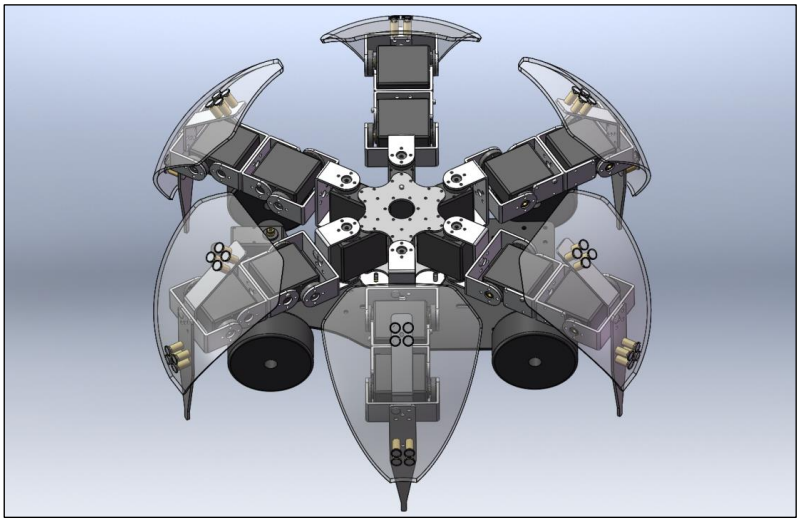
(f) 轮式滚动结构前轮装配图



(g) 轮式滚动结构后轮装配图



(h) 轮式滚动结构车模装配图



(i) 机器人总体装配效果图

图 3.1 基于 SolidWorks 的机器人结构装配流程图

4 软件设计

4.1 运动原理分析

4.1.1 原理分析

本机器人具有球形滚动和六足爬行两种运行方式。正常情况下，机器人呈现球行状态，通过机器人腿部的张开与收缩动作改变球面重心的位置驱动球面滚动。当机器人遇到无法翻越的障碍物时，机器人将变形成六足爬行机器人爬行越过障碍物。

下面介绍本机器人的六只爬行运动原理。机器人采用六足机器人中应用广泛的一种步态——三角步态进行移动。如图 4.1 所示，机器人左侧前足 1、左侧后足 3 与右侧中足 5 构成一组三角移动足，机器人右侧前足 4、右侧后足 6 与左侧中足 2 构成另一组三角移动足，两种移动足交替支撑，交替摆动，完成机器人的爬行运动。

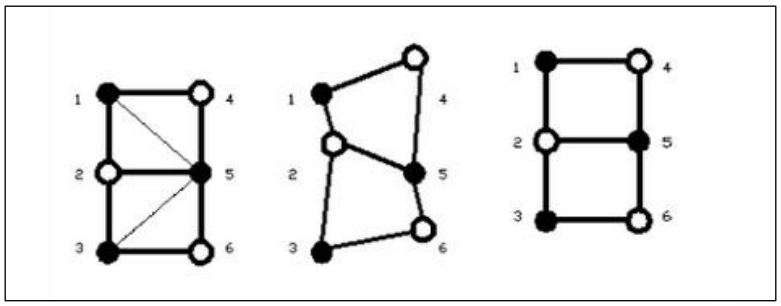


图 4.1 三角步态前进运动示意图

4.1.2 运动算法

4.1.2.1 D-H 算法简介

仿生六足机器人的运动数学建模采用的是 Denavit 和 Hartenberg 提出来的标准方法，称为 D‐H 模型，D‐H 建模方法适用于任何复杂的机器人建模。

通常机器人都是有旋转关节、滑动关节等组成的，而连杆的长度则是根据需求来制定的即是长度是可以任意的。在构建机器人的数学模型时，首先要做的就是固连一个参考的坐标系，然后就是选择旳简便的变换顺序，把坐标从第一个的关节变换到旳后的关节。

若一个机器人的结构是由很多个的关节与连杆组成，且依次连接的三个关节能在坐标系中通过变换互相转换；可将第一个关节为关节 n,依次类推为关节 n+1、关节 n+2,同时称在 n 与 n+1 之间的关节为连杆 n，依照顺序为连杆 n+1、连杆 n+2。如图 4.2 所示。

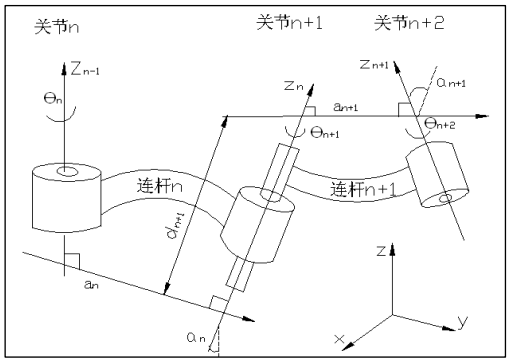


图 4.2 D-H 参数法关节标注

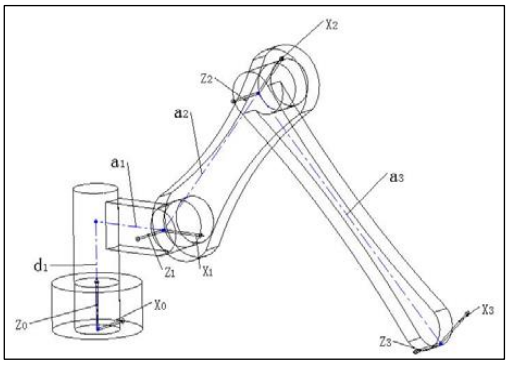


图 4.3 腿部简化模型结构简图

4.1.2.2 运动计算

仿生六足机器人的腿部简化模型有三个关节，确定其旋转的正方向。如图 4.3 所示，在确定关节的旋转方向之后，以每个关节的旋转轴心为 Z 轴，并以右手旋转法则确定 Z 轴的方向。在关节 1 处的 Z 轴为 Z_0，依次类推。在确立了 Z 轴后，继续建立 X 轴。X_0 的方向可以自由设定，但 X_1 的方向由 Z_0 指向 Z_1 的垂线所确立，X_2 也类似。X_3 - Z_3 坐标系固连在简化模型尾端，由于尾端没有接其他关节，故 X_3 - Z_3 坐标系与 X_2 - Z_2 坐标系坐标系一样，即从 X_2 - Z_2 坐标系平移至 X_2 - Z_2 坐标系，其姿态不变。

根据六足机器人正运动学和逆运动学的分析可得图 4.4 公式。其中，θ_n 为 X_n - 1 绕 Z_n - 1 旋转至与 X_n 平行的旋转角度；α_n 为 Z_n - 1 绕 X_n - 1 旋转至与 Z_n 平行的旋转角度；d_n 为 Z_n - 1 与在 Z_n 轴上分别做的两条公垂线的距离；a_n 为 Z_n - 1 与 Z_n 的垂线距离。

该公式的意义在于：已知仿生六足机器人的腿部简化模型的尾端位置（即预期运动终点坐标）的前提下，可以通过逆运动学的计算推导出仿生六足机器人的腿部三个关节各个电机的旋转角度。

$$\theta_3 = \arccos \frac{(C_1 p_x + S_1 p_y - a_1)^2 + (p_z - d_1)^2 - a_3^2 - a_2^2}{2 \times a_2 a_3}$$

$$\theta_3 = \arccos \frac{p_x^2 + p_y^2 + p_z^2 - 2 \times a_1 \times \sqrt{p_x^2 + p_y^2} - 2 \times d_1 \times p_z + d_1^2 + a_1^2 - a_2^2 - a_3^2}{2 \times a_2 a_3}$$

$$\theta_3' = -\theta_3$$

$$\theta_2 = \arccos \frac{(C_1 p_x + S_1 p_y - a_1)(a_3 C_3 + a_2) + a_3 S_3 (p_z - d_1)}{(a_3 C_3 + a_2)^2 + a_3^2 S_3^2} \qquad \theta_2' = -\theta_2$$

$$S_2 = \frac{(C_3 a_3 + a_2)(p_z - d_1) - (C_1 p_x + S_1 p_y - a_1) S_3 a_3}{(a_3 C_3 + a_2)^2 + a_3^2 S_3^2}$$

$$\theta_2 = \arctan \frac{(C_3 a_3 + a_2)(p_z - d_1) - (C_1 p_x + S_1 p_y - a_1) S_3 a_3}{(C_1 p_x + S_1 p_y - a_1)(a_3 C_3 + a_2) + a_3 S_3 (p_z - d_1)}$$

图 4.4 六足机器人腿部关节转动角度公式

4.1.3 步态设计

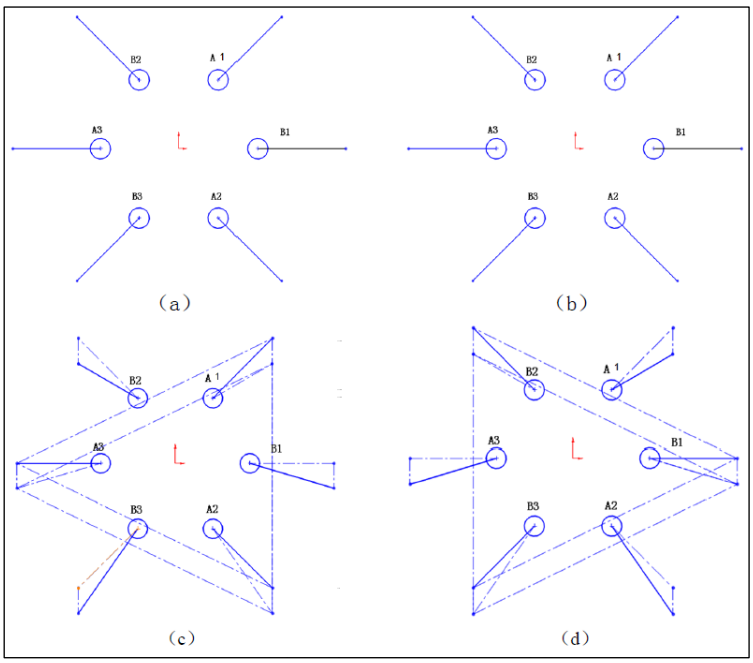
4.1.3.1 行走步态设计

仿生六足机器人的运动是要协调多条腿来共同完成的，为了维持其在运动时的稳定性，采用“三角算法”来设计其行走步态。所谓的“三角算法”即是仿生六足机器人在行走的时候是有三条腿作为支撑点的，仿生六足机器人的重心稳定在三条腿组成的三角形中，另三条腿处于摆动状态中。在仿生六足机器人执行行走、转弯时，每条腿都是分成两部分动作来完成的，第一部分是抬腿摆动相，第二部分是支撑摩擦相。仿生六足机器人在行走步态时有三个重要参数：t_1 抬腿摆动时间参数、t_2 支撑摩擦时间参数、t_3 中继时间参数。其中中继时间参数是包含在在摆动相与支撑摩擦相的时间参数内。当中继时间参数为 0 时，表达的意义是完成抬腿摆动相后才进行支撑摩擦相；当中继时间参数大于 0 时，表达的意义是在抬腿摩擦相运行至 (t_1-t_3)时，支撑摩擦相就开始进行；当中继时间参数小于 0 时，与中继时间运动步骤参数大于 0 相反。通常 t_3 时间都设为 0，其用途是增加支撑腿的摩擦力，或是减少仿生六足机器人行走时完成一个步态的时间，在此只分析 t_3 为 0 时的情况。如图 2.5 是仿生六足机器人的运动的简图，在这里将每个腿都看出是一个整体，其关节变化不做分析。将腿分为 A 组与 B 组。

图 4.5(a)是仿生六足机器人的站未进入行走步态时的站立图,此时的重心位于坐标中心点.当仿生六足机器人处于行走步态时，其的腿的摆动顺序可由 A→B，或者是由 B→A。这里的腿的摆动顺序并不会影响仿生六足机器人的步行的效果，故在这里只分析 A→B 的摆动顺序，B→A 的分析也与之类似。图 4.5 (b)是仿生六足机器人从站立姿态向行走步态转换是的第一个步态，A 组腿此时都处于抬腿摆动相中，B 组腿为支撑腿。从图中可以看出仿生六足机器人此时的重心位于由 B 组腿即是支撑腿所组成的三角形中。图 2.5 (c)是当 A 组腿的抬腿摆动相完成，支撑摩擦相开始执行时，B 组腿的抬腿摆动相也相应开始。在图中我们可以看出随着 A 组腿的支撑摩擦相的变化，仿生六足机器人的重心都是稳定在由三条腿组成的三角形内，在仿生六足机器人的原点 Y 轴上变化，其变化幅度与其一个步态的行走距离相关。图 2.5(d)是与图 2.5(c)的运动效果相似，只是抬支撑摩擦相变为由 B 组执行，A 组进行抬腿摆动相。即是此时又变换为了图 2.5 所示的运动效果，这样不断的往复循环，构成了仿生六足机器人的行走步态。从图2.5(d)中可以看出仿生六足机器人的重心的变换跟图4.5(c)是一样的。

当已知行走步态一个步态的行走距离与仿生六足机器人的初始的时候六条腿相对腿的

三维坐标值时，由以上的分析可以得出抬腿摆动相结束后相对于腿部坐标轴的坐标值，由此可以由六次项规划对轨迹进行求解，再有腿部的数学模型求解出关节的旋转角度。



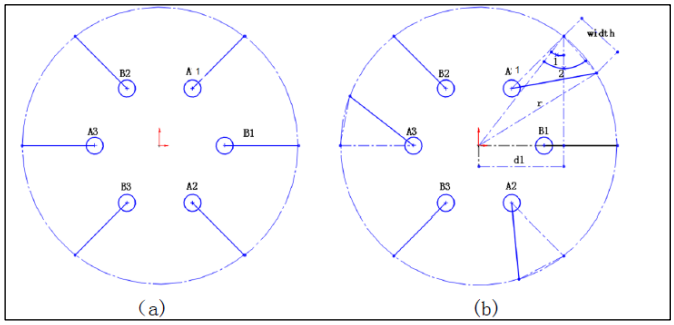
(a) 站立姿态 (b) A 组腿运动 (c) B 组腿运动 （d） A 组腿运动

图 4.5 仿生六足机器人行走时腿部姿态变换

4.1.3.2 转弯步态设计

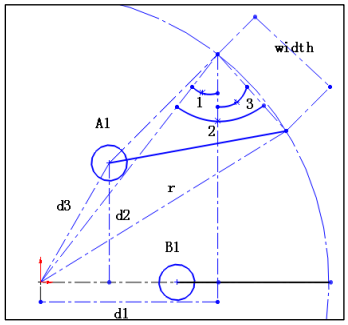
所谓的原点转弯，即是仿生六足机器人绕着其自身的中点自转，其与之前的行走步态相类似，都是由抬腿摆动相、支撑摩擦相组成，但是其轨迹的求解比行走步态较为复杂。在上一节，已经指出仿生六足机器人的行走、转弯都是两部分组成的，即抬腿摆动相，支撑摩擦相。当仿生六足机器人由站立姿态向原点转弯步态变换的时候，若先是 A 组腿先执行抬腿摆动相，其绕原点的旋转的顺序与优先由 B 组腿执行的顺序刚好相反。

如图 4.6（a)，为仿生六足机器人的站立姿态，从图中可以看出：在仿生六足机器人的初始设置中，其六条腿的站立点都是在以仿生六足机器人的中心为原点的圆中。当仿生六足机器人原点转弯时，其每条腿都是在这个圆上运动的。由于仿生六足机器人是以点来支撑的，其每个步态的轨迹并不需要如图中为一段圆弧，在这里拆分成长度为 width 的直线一断断来逼近圆弧，这样可以减少运算量，在下面的章节分析中，也做类似的处理。



（a） 站立姿态　　(b)　A 组腿抬腿

图 4.6 仿生六足机器人原点转弯时腿部姿态变换图



4.7　A1 腿角度分析

图 4.6（b) 为仿生六足机器人由站立姿态向绕原点转弯步变化时的第一步。从中可以看出其与行走步态时相类似，不过其轨迹的投影是与 Y 轴成一定角度的为∠3，通常在绕原点转弯时，其步态的行走距离是已知的。只要求解出∠3，在已知起始点的位置坐标，就可以求解出仿生六足机器人的腿部在抬腿摆动相结束时所处的位置坐标。因仿生六足机器人的每条腿在原点转弯时，其运动类似，在这里只分析 A_1 腿 α的求解。图 4.7 所示， d_1 是 A_1 腿部末端原点坐标轴上 Y 轴的投影；d_2 是 A_1 腿部在 Y 轴的投影；d_3 是 A_1 腿部中点到原点的距离；width 是绕原点转弯时一个步态的行走距离。这些参数都是在机构设计与初始化参数设置中已知的。设 A_1 腿末端的初始点的相对于腿部的坐标轴为（因高度在初始与结束后不变，在此忽略)。由坐标变换可得机器人腿部末端起点坐标（x_0, y_0）相对于原点的坐标为（$x+d_3$, $y+d_2$)。r 是原点到 A_1 腿部末端的距离，也是仿生六足机器人在原点转弯步态中腿部末端所在点的圆的半径。依图与几何知识可得：

$$\theta_1 = \tan^{-1}\left(\frac{x_0+d_3}{y_0+d_2}\right),\ \theta_2 = \frac{width}{2r} = \frac{width}{2*\sqrt{(x_0+d_3)^2+(y_0+d_2)^2}},\ \theta_3 = \theta_2 - \theta_1。$$

因而可得仿生六足机器人的 A_1 腿的抬腿摆动相结束后相对于 A_1 腿部坐标轴的坐标为：（$x_0 + width * \sin\angle 3,\ y_0 - width * \cos\angle 3$)。由腿部的数学模型可求解出关节的旋转角度。

4.2　系统软件流程图

本文设计的轮足两用机器人系统软件代码均已上传至 GitHub，具体网址下载链接为：https://github.com/liren197968/SweepRobot/tree/master/Stm32F4/KeilMdk5Project/SweepRobot 。

如图 4.8 所示，系统上电后，MCU 首先完成外设的初始化，其次开始初始化各个硬件模块，具体 LED 模块、蓝牙模块、气体传感器模块、蜂鸣器模块、PWM 模块、直流电机初始化、运动形态的初始化。之后进入系统的循环主流程。

机器人系统首将先检测电池当前的电量，若电池电量较低，系统将停止运行，并通过蜂鸣器播放充电提示音；若电池电量充足，系统开始接收遥控器的控制信号。若系统收到气体

测量控制指令，系统将开启 ZMOD4410 的气体检测功能，否则将停止气体检测功能。若系统收到形态切换指令，机器人将进行形态切换（系统默认的机器人形态为轮式滚动形态），否则系统将保持当前的运动形态。系统软件流程图如图 4.8 所示。

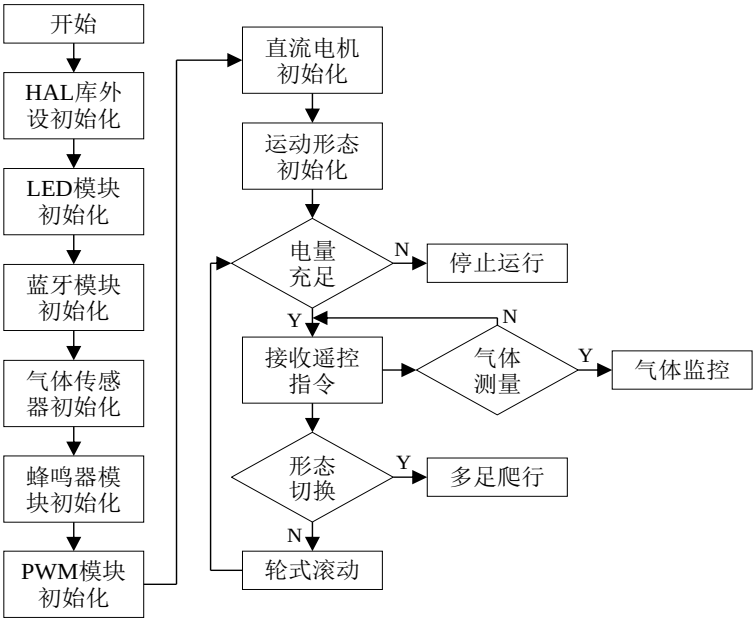


图 4.8　系统软件流程图

4.2.1　气体检测软件流程图

气体检测的软件流程图如图 4.9 所示，当系统收到气体测量控制指令时，MCU 将首先通过 IIC 总线读取 ZMOD4410 MOX 电阻的数据，其次运行 IDT 的算法库依次读取 TVOC、IAQ、eCO_2、ODOR state 的数据。若此时气体浓度超标（通过 eCO_2 或 IAQ 等级与设定阈值进行比较），系统将发出报警信号；若气体浓度在极端的时间内发生突变，此时 ODOR state 将置位，此时系统也将发出报警信号。气体浓度数据读取的软件代码如图 4.10 所示。

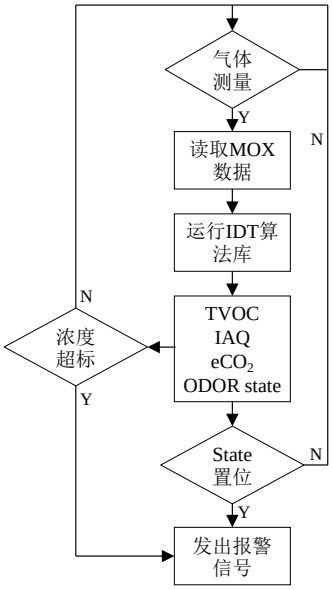


图 4.9　气体检测软件流程图

```c
/* evaluate and show measurement results */
ret = zmod44xx_read_rmox(&g_dev, &r_mox);
if(ret) {
    ROC_LOGE("zmod44xx_read_rmox error %d, exiting program!\r\n", ret);
    goto exit;
}

/* To work with the algorithms target specific libraries needs to be
 * downloaded from IDT webpage and included into the project */
ROC_LOGI("ZMOD4410 [%lld] measurement result is:", count_time);
ROC_LOGI("Rmox  = %5.0f kOhm", (r_mox / 1000.0f));

/* calculate clean dry air resistor */
r_cda = r_cda_tracker(r_mox);
ROC_LOGI("CDA %.3f", r_cda);

/* calculate result to TVOC value */
tvoc = calc_tvoc(r_mox, r_cda, &tvoc_par);
ROC_LOGI("TVOC %f mg/m^3", tvoc);

/* calculate IAQ index */
iaq = calc_iaq(r_mox, r_cda, &tvoc_par);
ROC_LOGI("IAQ %d", iaq);

/* calculate estimated CO2 */
eco2 = calc_eco2(tvoc, 3, &eco2_par);
ROC_LOGI("eCO2 %f", eco2);

/* get odor control signal */
cs_state = calc_odor(r_mox, &odor_par);
ROC_LOGI("odor control state %d \r\n", cs_state);

RocZmod4410SensorStatusSet((uint32_t)cs_state);

/* INSTEAD OF POLLING THE INTERRUPT CAN BE USED FOR OTHER HW */
/* waiting for sensor ready */
while (FIRST_SEQ_STEP != (zmod44xx_status & 0x07)) {
    g_dev.delay_ms(50);
    ret = zmod44xx_read_status(&g_dev, &zmod44xx_status);
    if(ret) {
        ROC_LOGE("Error %d, exiting program!\r\n", ret);
        goto exit;
    }
}
```

图 4.10　气体浓度数据读取代码

4.2.2　轮式滚动形态软件流程图

轮式滚动形态的软件流程图比较简单，如图 4.11 所示。本文采用的轮式滚动方式为后轮直流电机驱前轮伺服电机转向的方式，当机器人处于轮式滚动形态收到转向控制指令时，系统软件将调整 PWM 数值使得伺服电机转角发生改变，从而驱动机器人轮式形态的转向角度；当机器人收到提速指令时，系统软件将调整 PWM 的数值提升电机的输入电压，调整电机的转速，从而改变机器人的轮式形态的滚动速度。

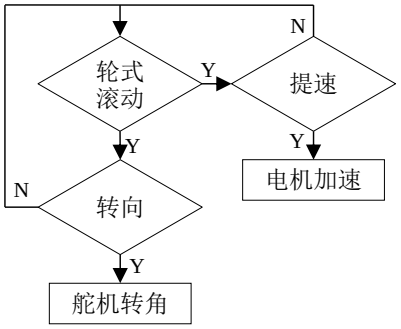


图 4.11　轮式滚动流程图

4.2.3　多足爬行形态软件流程图

下面以机器人六足形态下的前进动作为例对机器人多足爬行形态的运动控制流程进行说明，其它动作与此类似。

如图 4.12 所示，根据 4.1.2.1 节机器人行走步态的分析可知，当机器人控制器收到遥控器发送的前进指令后，机器人将按如下四个步骤完成前进动作的执行：第一步，抬起 A 组腿（第一组腿），同时让 B 组（第二组腿）向后滑动；第二步： A 组腿落地，同时让 B 组

腿继续向后滑动；第三步： A 组腿开始向后滑动，同时抬起 B 组腿；第四步：A 组腿继续向后滑动，同时 B 组腿落地。执行每次步骤前，系统软件将根据机器人每步前行的步长数据使用 D-H 算法计算出六条腿 18 个关节伺服电机的旋转角度（具体计算过程见 4.1.2.2 节内容，计算代码见图 4.14）。

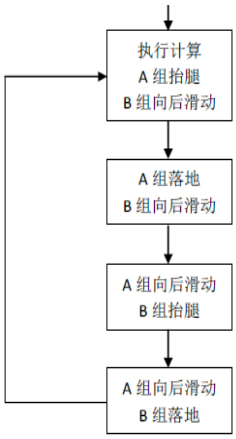


图 4.12 机器人步态执行流程

```
case 1:
        RocRobotOpenLoopWalkCalculate(ROC_ROBOT_DEFAULT_LEG_STEP, ROC_ROBOT_DEFAULT_FEET_LIFT, g_RobotForwardPwmVal);

        RocRobotRigForLegCtrl(g_RobotForwardPwmVal[0], g_RobotForwardPwmVal[1], g_RobotForwardPwmVal[2]);
        RocRobotLefMidLegCtrl(g_RobotForwardPwmVal[12], g_RobotForwardPwmVal[13], g_RobotForwardPwmVal[14]);
        RocRobotRigBakLegCtrl(g_RobotForwardPwmVal[6], g_RobotForwardPwmVal[7], g_RobotForwardPwmVal[8]);

        RocRobotLefForLegCtrl(g_RobotStandPwmVal[9], g_RobotStandPwmVal[10], g_RobotStandPwmVal[11]);
        RocRobotRigMidLegCtrl(g_RobotStandPwmVal[3], g_RobotStandPwmVal[4], g_RobotStandPwmVal[5]);
        RocRobotLefBakLegCtrl(g_RobotStandPwmVal[15], g_RobotStandPwmVal[16], g_RobotStandPwmVal[17]);
        break;

case 2:
        RocRobotOpenLoopWalkCalculate(ROC_ROBOT_DEFAULT_LEG_STEP, 0, g_RobotForwardPwmVal);

        RocRobotRigForLegCtrl(g_RobotForwardPwmVal[0], g_RobotForwardPwmVal[1], g_RobotForwardPwmVal[2]);
        RocRobotLefMidLegCtrl(g_RobotForwardPwmVal[12], g_RobotForwardPwmVal[13], g_RobotForwardPwmVal[14]);
        RocRobotRigBakLegCtrl(g_RobotForwardPwmVal[6], g_RobotForwardPwmVal[7], g_RobotForwardPwmVal[8]);

        RocRobotLefForLegCtrl(g_RobotForwardPwmVal[27], g_RobotForwardPwmVal[28], g_RobotForwardPwmVal[29]);
        RocRobotRigMidLegCtrl(g_RobotForwardPwmVal[21], g_RobotForwardPwmVal[22], g_RobotForwardPwmVal[23]);
        RocRobotLefBakLegCtrl(g_RobotForwardPwmVal[33], g_RobotForwardPwmVal[34], g_RobotForwardPwmVal[35]);
        break;

case 3:
        RocRobotOpenLoopWalkCalculate(ROC_ROBOT_DEFAULT_LEG_STEP, ROC_ROBOT_DEFAULT_FEET_LIFT, g_RobotForwardPwmVal);

        RocRobotRigForLegCtrl(g_RobotStandPwmVal[0], g_RobotStandPwmVal[1], g_RobotStandPwmVal[2]);
        RocRobotLefMidLegCtrl(g_RobotStandPwmVal[12], g_RobotStandPwmVal[13], g_RobotStandPwmVal[14]);
        RocRobotRigBakLegCtrl(g_RobotStandPwmVal[6], g_RobotStandPwmVal[7], g_RobotStandPwmVal[8]);

        RocRobotLefForLegCtrl(g_RobotForwardPwmVal[9], g_RobotForwardPwmVal[10], g_RobotForwardPwmVal[11]);
        RocRobotRigMidLegCtrl(g_RobotForwardPwmVal[3], g_RobotForwardPwmVal[4], g_RobotForwardPwmVal[5]);
        RocRobotLefBakLegCtrl(g_RobotForwardPwmVal[15], g_RobotForwardPwmVal[16], g_RobotForwardPwmVal[17]);
        break;

case 4:
        RocRobotOpenLoopWalkCalculate(ROC_ROBOT_DEFAULT_LEG_STEP, 0, g_RobotForwardPwmVal);

        RocRobotRigForLegCtrl(g_RobotForwardPwmVal[18], g_RobotForwardPwmVal[19], g_RobotForwardPwmVal[20]);
        RocRobotLefMidLegCtrl(g_RobotForwardPwmVal[30], g_RobotForwardPwmVal[31], g_RobotForwardPwmVal[32]);
        RocRobotRigBakLegCtrl(g_RobotForwardPwmVal[24], g_RobotForwardPwmVal[25], g_RobotForwardPwmVal[26]);

        RocRobotLefForLegCtrl(g_RobotForwardPwmVal[9], g_RobotForwardPwmVal[10], g_RobotForwardPwmVal[11]);
        RocRobotRigMidLegCtrl(g_RobotForwardPwmVal[3], g_RobotForwardPwmVal[4], g_RobotForwardPwmVal[5]);
        RocRobotLefBakLegCtrl(g_RobotForwardPwmVal[15], g_RobotForwardPwmVal[16], g_RobotForwardPwmVal[17]);
        break;
```

图 4.13 运动计算部分代码

```
static void RocDhAlgorithmReverse(double x, double y, double z)
{
    double          a = 0;
    double          e = 0;
    double          f = 0;
    double          h = 0;
    double          j = 0;

    g_DhAngleBuffer[0] = (atan(y / x)) * 180 / ROC_ROBOT_MATH_CONST_PI;

    a = x * cos( (g_DhAngleBuffer[0] * ROC_ROBOT_MATH_CONST_PI) / 180) + y * sin( (g_DhAngleBuffer[0] * ROC_ROBOT_MATH_CONST_PI) / 180) - ROC_ROBOT_DH_CONST_A1;

    g_DhAngleBuffer[2] = ( (acos( ( pow( a, 2 ) + pow( ( z - ROC_ROBOT_DH_CONST_D1 ), 2) - pow(ROC_ROBOT_DH_CONST_A3, 2)
                              - pow(ROC_ROBOT_DH_CONST_A2, 2) ) / ( 2 * ROC_ROBOT_DH_CONST_A2 * ROC_ROBOT_DH_CONST_A3 ) ) ) ) * 180 / ROC_ROBOT_MATH_CONST_PI;

    if( (g_DhAngleBuffer[2] > 0) || (g_DhAngleBuffer[2] < -180) )    // limit the anlge in the range of [0~(-180)]
    {
        if( (g_DhAngleBuffer[2] > 0) && (g_DhAngleBuffer[2] <= 180) )
        {
            g_DhAngleBuffer[2] = -g_DhAngleBuffer[2];
        }

        if( (g_DhAngleBuffer[2] > 180) && (g_DhAngleBuffer[2] <= 360) )
        {
            g_DhAngleBuffer[2] = g_DhAngleBuffer[2] - 360;
        }

        if( (g_DhAngleBuffer[2] < -180) && (g_DhAngleBuffer[2] >= -360) )
        {
            g_DhAngleBuffer[2] = -(g_DhAngleBuffer[2] + 360);
        }
    }

    e = cos(g_DhAngleBuffer[2] * ROC_ROBOT_ANGLE_TO_RADIAN);
    f = sin(g_DhAngleBuffer[2] * ROC_ROBOT_ANGLE_TO_RADIAN);
    h = (e * ROC_ROBOT_DH_CONST_A3 + ROC_ROBOT_DH_CONST_A2 ) * (z - ROC_ROBOT_DH_CONST_D1) - a * f * ROC_ROBOT_DH_CONST_A3;
    j = a * (ROC_ROBOT_DH_CONST_A3 * e + ROC_ROBOT_DH_CONST_A2) + ROC_ROBOT_DH_CONST_A3 * f * (z - ROC_ROBOT_DH_CONST_D1);

    g_DhAngleBuffer[1] = (atan2(h, j) ) * 180 / ROC_ROBOT_MATH_CONST_PI;
}
```

图 4.14 D-H 算法公式代码

5 效果演示

本文设计的机器人实物图片如图 5.1 所示，机器人可变形，具有轮式滚动、多足爬行两种运动形态，可通过无线远程遥控器、手机 APP 进行遥控。机器人的轮式滚动形态为四轮滚动结构，通过后驱前转向的方式实现，具有前滚、后滚、转向等滚动功能；机器人多足爬行形态为六足爬行结构，能够完成前进、后退、逆时针转圈、顺时针转圈等功能；此外，机

器人还具有无线图像传输功能。

本文设计的机器人通过 IDT 的 ZMOD4410 传感器测量气体浓度数据，测量结果如图 5.2 所示。当环境的气体浓度超标或气体浓度在极短的时间内发生浓度突变时，系统将发出报警信号，从而在第一时间内通知探测人员测量危险现场。

机器人的具体演示功能请参照作品演示视频——“作品功能演示视频.mp4”文件。

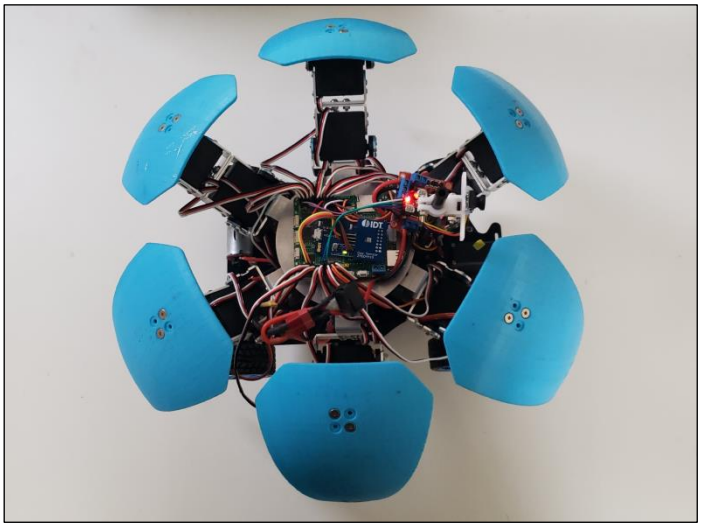


图 5.1　机器人实物图片

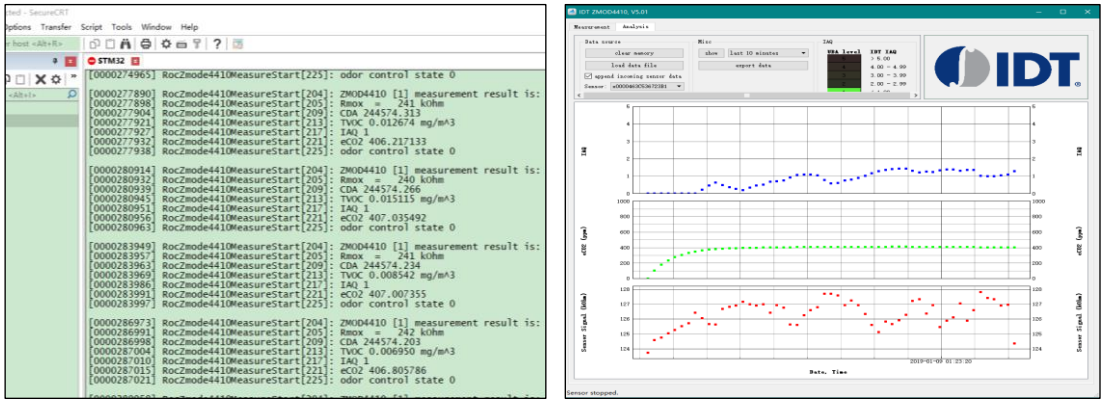


图 5.2　ZMOD4410 传感器气体浓度数据测量结果

6　作品总结

6.1　作品特点

本作品的技术优势在于：机器人多足爬行形态采用 D-H 运动控制算法，且机器人爬行速度、步长可调节，具有较高的控制精度和控制准确性。

本作品的先进性在于：1、采用 ZMOD4410 气体传感器，具有较高的气体环境感知检测精度和较强的稳定性，同时根据传感器 ODOR 模式的特性实现了气体浓度突变的自动报警

功能，能够第一时间通知探测人员撤离危险区域；2、机器人能够变形，拥有轮式滚动和多足爬行两种运动形态，在满足移动速度的同时能够适应多种复杂地形。

6.2 作品价值

本作品的市场推广前景在于：机器人可变形，保证运行速度的同时能够适用于多种复杂地形，可用于井矿、地穴、消防等其他危险领域的环境感知探测工作。

未来，本作品将进行如下功能完善：加入惯性导航控制和激光雷达室内构图功能，使该机器人能够成为具有跨越台阶、门槛等复杂地形场所功能的扫地机器人。