**四足机器人上位机设计**

王海宁

2019.08

南京集萃智能制造技术研究所有限公司研制的高越障仿生四足机器人，面向山地作战与巡逻、城市巷战的爬壁侦察与突袭、野外复杂山地环境巡逻、崎岖地形通行、陡峭岩体攀爬、纵深沟壑跳跃等任务的需求，以复杂环境下高性能自适应越障机器人性能优化为项目目标。作为四足机器人研制项目的一部分，开发四足机器人上位机，是为了解决如下问题：

1. 为四足机器人提供实时的、图形界面式的人机交互方式（包括控制、调参等）；
2. 监测四足机器人运动状态，存储运动数据以便后续分析；
3. 与下位机相互独立，可基于外环控制，方便地测试和拓展更高层的功能。

本文描述四足机器人的上位机设计，包含以下三个方面的内容：

1. 系统概览
2. 任务需求
3. 程序说明

# 一、系统概览

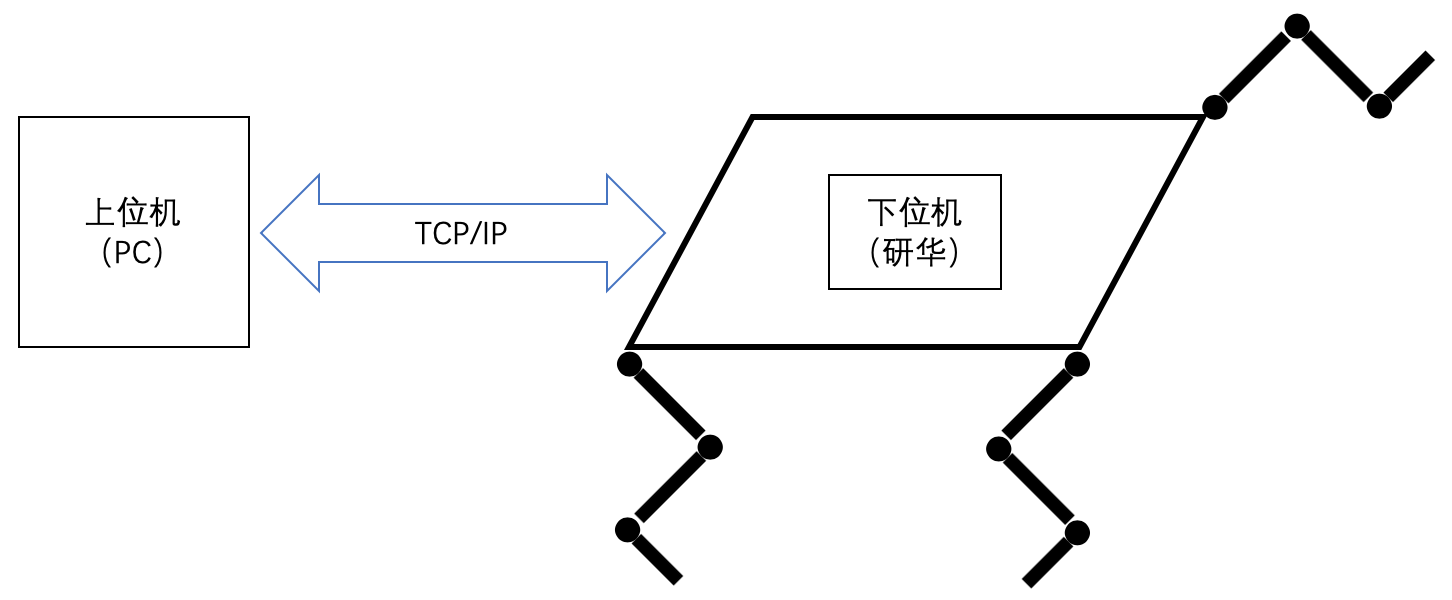


图1 四足机器人上下位机示意

四足机器人上下位机整体系统如图1所示。下位机安装于机体上，负责伺服机构的驱动、传感器数据采集以及运行内环控制算法等；上位机运行于独立的PC，负责传感器数据的实时显示和存储备份、发送外环控制指令、调试参数等，并为以上任务提供易于操作的用户界面。上下位机通过有线或无线的方式相连，上位机向下位机发送控制指令、姿态参数等数据，下位机向上位机发送传感器数据、运行状态等数据。

本任务中，上位机采用python编写；用户界面采用Qt设计，用QtDesigner搭建界面，用PyUIC将QtDesigner生成的.ui文件转为python源码，界面实现依赖python的第三方库PyQt5；上下位机基于网口通讯协议（TCP/IP）进行数据交互，采用python的socket库实现。

# 二、任务需求

* 上位机启动后，显示图形界面；
* 上位机以一定频率检测连接状态，对于连接状态的改变作出响应，具体的，当连接建立时开始响应收发请求、自动定时接收数据、创建新的通信日志；当连接断开时停止收发动作并释放相应的内存资源；
* 通信建立后，上位机以一定频率接收下位机上发的数据，并对接收的数据进行解包和处理，具体的，对于传感器数据，进行动态图像显示和后台存储；对于运行状态的改变，进行屏幕标签显示；对于需要应答的数据，根据具体需要进行应答；
* 上位机通过按钮开启/关闭通信连接、解锁/锁定机器人、控制步态等；
* 上位机通过单独窗口设置姿态参数和PID参数，并将设置好的参数保存和下发；
* 可拓展其他的外环控制算法。

# 三、程序说明

程序围绕主界面设计整体逻辑，采用类（class）封装不同的模块。程序包含两个界面模块和四个功能模块。

## 3.1 界面模块

界面模块用于显示上位机的图形界面，本程序的两个界面模块分别为：

* 主窗口（interface\_Main.py）：默认启动窗口，包含数据监视页和基本操作按钮，如图2所示；
* 调参窗口（interface\_PoseParam.py）：从主窗口通过按钮调用，用于显示和设置参数，如图3所示。

界面使用QtDesigner以图形拖拽的方式设计，设计结果保存为.ui文件。利用PyUIC工具，从.ui文件自动生成python源码（.py文件），不需要额外改动。生成的源码将界面封装为python类（class），该类依赖于PyQt5库。外部调用时，一般在另一个源文件中创建新的类并继承界面类，从而可以在新的类中编写代码以实现界面部件的逻辑联系（即Qt的信号和槽机制）。



图2 主窗口



图3 调参窗口

## 3.2 功能模块

功能模块实现了上位机的主要功能，四个功能模块分别为：

* 通信协议模块（protocol.py）：包含协议数据结构和解码/编码等方法；
* 网口通讯模块（communication.py）：包含基于socket实现的服务端和客户端；
* 绘图模块（plot.py）：包含基于pyqtgraph的图像控件；
* 界面逻辑模块（interface.py）：利用Qt的信号/槽机制实现窗口各部件功能。

其中communication.py和protocol.py除了用于上位机，也可用于下位机。从通信层面而言，下位机被视为服务端，上位机被视为客户端，二者的代码均包含在communication.py中；从数据层面而言，二者基于统一的协议对收发的数据进行编/解码，因此都可以用protocol.py中所定义好的协议。

**3.2.1 通信协议模块**

为了在统一的框架下实现数据交互，以及实现模块封装和代码重用，需对通信中可能出现的数据约定统一的打包方式，即制定数据协议。本文拟定的数据协议如附录一所示，所有数据的打包统一遵循“版本号（VER）-应答标记（ACK）-帧类型（TYP）-帧数据（DATA）”的形式，帧类型（TYP）指定了数据的种类，在本协议中共有四个种类，分别是：

* 传感器数据（SENS）；
* 运行状态（STAT）；
* 控制指令（COMD）；
* 参数设置（PARA）。

根据自身的类型，帧数据（DATA）有不同数据排布方式，详见附录一的帧数据说明。

通信协议模块实质上就是该协议的程序实现，对于每一种类型的帧数据，该模块定义了一个类（class）来封装数据段中出现的所有变量，并提供了对这些变量的基本操作方法。其中最核心的方法是变量的编码（encode）和解码（decode），解码负责解析其接收到的字节流（datastring）并为变量赋值，编码负责将变量组装为字节流以用于发送。

对于传感器数据类（class SensorPackage，即帧类型0x01所对应的帧数据），还提供了一些特殊的属性。因为程序中不是仅存储一帧传感器数据，而是存储一个连续时间段内所有的传感器数据，以用于屏幕显示和文件写入。因此本类中除了定义了一个包含一帧数据所有变量的字典（data），还定义了一个包含历史数据的缓冲区（data\_buf），以及与缓冲区相关的一系列操作方法，包括将帧数据加入缓冲区（bufferIn）、从缓冲区写出到文件（bufferOut）、将缓冲区队列整体后移一帧（bufferShift）、获取缓冲区的有效段（bufferGet）等。由于缓冲区长度是固定的，而缓冲区中填充的帧数是变化的，因此用一个指示标记（buflen）来标注当前缓冲区中的有效帧数。本类还提供了一个process方法，该方法整合了一组常规操作，即每解码一帧传感器数据，就将这一帧数据添加到缓冲区，若缓冲区已满则将缓冲区的内容写出到文件，并清空缓冲区。

对于参数设置类（class ParameterPackage，即帧类型0x04所对应的帧数据），提供了保存文件（save）和加载文件的（load）方法。创建该类的实例时默认加载“para\_default.txt”文件，而设置了新的参数后会保存新的参数文件。用文件记录参数的好处是在程序未运行的时候也能方便地查看和修改参数。

除了处理这四种帧数据的类之外，本模块还提供了Protocol类（class Protocol）来总管这四个类。每创建一个Protocol类的实例，就自动创建了其他四个类的实例。该类可以视作本模块对外的接口，它的两个主要功能是收集（collect）和分发（distrib）。每当接收到一帧字节流，先在Protocol类中进行第一层解码，获取版本号、应答标记、帧类型这三个信息，再根据帧类型，分发（distrib）到相应的类中进行第二层解码。当需要发送某种数据时，先在帧数据所对应的类中设置好变量值，再调用Protocol的收集（collect）方法，根据帧类型来打包所需发送的数据。

**3.2.2 网口通讯模块**

网络上的两个程序通过一个双向的通信连接实现数据的交换，这个连接的一端称为一个“套接字（socket）”，它实质上是IP地址与端口的组合, 应用程序可以通过它发送或接收数据。对socket的操作类似于文件的打开、读写、关闭等操作。

python中实现网口通讯依赖于“socket”库。每建立一个连接，两端的进程中需各自创建一个socket。惯常的做法是（参考官方例程：https://docs.python.org/3/library/socket.html）：

1. 需要相互连接的两个进程被区分为服务端和客户端；
2. 服务端首先创建一个socket并与端口绑定，该socket只用于监听和接收新的连接请求，但不用于实际数据收发，将该socket称为主socket；
3. 客户端创建一个socket，并向服务端请求连接；
4. 服务端的主socket收到连接请求后，创建一个新的socket与客户端连接。

至此，便建立了一对连接，视实际需要可以增加新的连接。由于主socket的存在，服务端socket总数比客户端socket总数多1。

当客户端或服务端中某一方掉线或程序卡死，另一方需要即时知晓、切断连接、释放内存资源，并根据需要选择是否重新连接。这个功能通过“心跳机制”来实现。即连接一旦启动，双方便以某一频率持续向对方发送特殊的“心跳信号”，若一段时间内未收到对方的心跳信号或其他任何信息，则表示对方已掉线。心跳机制有以下特征：

* 在未发生主动通信时，双发也要以固定频率互发心跳信号；
* 在发生主动通信时，心跳信号掺杂在数据信息中；
* 心跳机制在后台不间断运行，不应影响主线程的正常工作；
* 心跳信号不应过长，以免造成通信压力，也不应过于简单，以免与数据信息无法区分。

网口通信还面临着“粘包”、“分包”问题，即一次调用所发送的一帧数据可能会被拆分为两段字节流分别发送，也可能与下次发送的数据合并为一段字节流。除此之外，数据丢包、误码的问题，虽然发生的概率不大，但也应纳入考虑。本程序采用“帧头标识-帧长度-帧内容-帧尾校验”的局部协议来解决这些问题。以帧头标识来标记帧的起始，帧头标识后紧跟一个整数描述帧内容的字节长度，随后是真正需要发送的帧内容，最后是对帧内容的校验以确认传输过程中未出现误码。注意“局部协议”的含义是，它只存在于收发过程中，其作用范围仅在communication.py内，每次收发都依照这个局部协议对字节流进行编/解码，而上节所述的通信协议在这里被视为帧内容。

依照以上程序设计思路，本模块包含两个类：服务端类（class Server）和客户端类（class Client）。服务端在下位机运行，客户端在上位机运行。

客户端包含收消息列表（I\_socket）、发消息列表（O\_socket）、字节流缓存（I\_stream）、最后接收时间（last\_time）。服务端的结构和客户端类似，不同之处在于，服务端支持多个客户端同时连接，即需要同时管理多个socket，因此服务端用字典来存储每个socket与其消息列表等的对应关系。

服务端一经创建，便自动开始监听端口，随时接收新连入的客户端，并且在整个进程的生命周期内不会停止。而客户端创建后需调用open方法手动开启连接，这是为了使得上位机能够控制连接的通/断；客户端的连接操作在一个单独的线程中进行，这是为了在断线自动重连时不堵塞主线程；客户端还具有一个关闭连接（close）的方法，在任何时候都可以调用该方法以终止连接（即使另一个线程正在尝试请求连接）。

客户端和服务端都具有收发的方法。以客户端为例，每次接收（read）到的字节流被加入字节流缓存（I\_stream）中，并对I\_stream进行解码（decode），去除其中的帧头（framehead）、心跳信号（heartbeat）等干扰信息，得到完整的帧内容存入收消息列表（I\_socket）中。解码的算法描述如下：

1. 从头开始对整个I\_stream进行扫描；
2. 若整个I\_stream不包含帧头和心跳信号，则退出等待下次读取（可能在末尾存在被截断的帧头，虽然概率不高）；
3. 若扫描到心跳信号，则进行相应处理（客户端与服务端处理方式不同）；
4. 若扫描到帧头，则读取其后跟随的帧长度（len\_dt）；
5. 根据帧长度，若当前I\_stream未容纳完整一帧（包括帧尾校验位），则退出等待下次读取；
6. 若容纳了完整一帧，则找到帧尾，根据帧尾（checksum）对帧内容进行校验；
7. 若校验通过，保存此帧内容，跳过帧尾继续向后扫描；
8. 若校验失败，可能是由于误码或假帧头，则跳过帧头继续向后扫描；
9. 已扫描过的部分将被丢弃（有用的信息已经被保存到收消息列表中了），以保证字节流缓存不会无限增长。

编码（encode）和发送（write）操作相对简单，因为不涉及粘包、分包的问题，直接在所需发送的帧内容前面添加帧头、帧长度，后面添加帧尾，然后调用socket自带的sendall方法即可。

服务端或客户端一旦连接成功，便会在一个单独的线程里开启一个循环定时器，每隔1秒运行一次检测（detect），检测会尝试一次消息接收（read），若收到新消息则记录接收时间，若超过一定时间未接收到任何消息则表明对方掉线，此时服务端直接丢弃掉线客户端对应的socket并释放对应的资源，而客户端则重启socket并尝试重新连接。心跳机制也是在检测（detect）中完成的，一个心跳信号（heartbeat）的完整历程是：从服务端产生-被客户端接收-被客户端回传-被服务端接收-被服务端销毁。因此，服务端在每次运行检测（detect）时发送心跳信号，客户端在解码（decode）中识别出心跳信号后将其加入发消息列表（O\_socket）以待发送，而服务端收到心跳信号后无需任何处理，直接将其丢弃即可，下一次心跳信号会在服务端检测时自动产生。

**3.2.3 绘图模块**

本模块依赖pyqtgraph库进行动态图片绘制。每一种图片都用一个类（class）来封装，该类继承了pyqtgraph的GraphicsLayoutWidget类，其中包含了基本的绘图方法，而本模块定义的类就是调用这些基本绘图方法来实现具体图片的绘制。关于绘图库的使用可参考官方网站：<http://www.pyqtgraph.org/documentation/>以及官方提供的例程（pyqtgraph.examples）。要使用本模块定义的类，需要用QtDesigner在界面文件（.ui文件）中创建用于显示图片的控件（widget），并将控件提升为本模块所定义的类（详见pyqt相关书籍）。

每个图片类在初始化时设置图片的坐标轴、坐标范围、标题等基本特征，若有固定不变的背景标线也一并绘制好，从而在更新图片时只改动需要更新的部分，减少更新图片的工作量以提高更新效率。每个图片类提供了update方法，调用update时需提供用于显示的数据，以固定频率调用update即可实现动态更新。

本模块目前提供了两种图片，一种是用于显示一组传感器数据随时间变化的曲线图，一种是用于显示瞬时机体姿态的仪表图。曲线图目前设定x坐标范围是现有数据中最新的5秒，y坐标范围是0～1，应根据实际需要进行调整。仪表图中用红色箭头表示机头朝向，黑色箭头表示机身倾斜方向和倾斜程度，背景的虚线分别标注了倾斜30、60、90度的位置。仪表显示机身倾斜的原理是：假想一根垂直于机身向上的单位矢量，当机身倾斜时该矢量在水平面上的投影即为黑色箭头。

显示一组（12张）图片有两种方法：一种是创建12个图片控件（widget），每个控件上显示一张图片；另一种是创建一个图片控件，该控件上显示12个子图。经测试，第二种方法运行效率较高，即pyqtgraph对“少控件，多子图”的支持更好。除了pyqtgraph之外，matplotlib也是python中常用的绘图库，但它面向的是生成出版质量的高清图片，因此运行较慢。总体来说，绘图模块是本程序中耗时最多的部分，其实际运行效率与硬件有关，降低绘图频率有助于程序整体效率的提高。

**3.2.4 界面逻辑模块**

该模块包含了主窗口类（class MainWindow）和调参窗口类（class DialogPose），围绕主窗口进行界面逻辑设计，主要是利用Qt的信号/槽机制来实现各控件的功能。

整个程序首先默认启动主窗口，主窗口类的的初始化过程中生成了客户端类（class Client）、调参窗口（class DialogPose）、协议类（class Protocol）的实例，实例名称分别为client, dialpose, prot；同时还创建了一个传感器数据类（class SensorPackage）的实例datashow，用来存储绘图所需的数据；另外还创建了两个计时器。

计时器0（timer0）在初始化中便启动，每隔1秒执行一次检查连接状态的操作（checkConnection），若连接状态为“断开”，则不会产生任何通信，也就不存在任何数据的编/解码，从而也不会绘制任何新的图像。若检测到连接状态由“断开”变为“连接”，则会激活计时器1（timer1），该计时器每隔40毫秒执行一次听取（hear）操作。

听取（hear）操作中，调用client的接收（recv）方法接收一帧数据，并交由prot进行分发（distrib）。注意client的recv方法实质上是：调用一次client的read方法，然后返回收消息列表（I\_socket）中最新的一帧数据，实际上这次read未必能读到新数据，所以recv返回的数据只能保证是现有数据中最新的一帧，但未必就是本次读到的数据。尽管如此，在正常的通信状况下，recv返回的数据很大概率就是本次读到的数据，程序只是为了增强鲁棒性而预留了容错的空间。client的recv方法中，若收消息列表（I\_socket）中包含多帧数据，只有最新一帧会被返回，其他帧都被丢弃了，若不希望丢弃，可换用client的interact方法，该方法需要接收一个处理字节流的函数（例如prot.distrib或prot.process）作为参数，从而可以循环调用该函数处理所有帧。实际上，在良好的通信状态下，并不会频繁地丢帧，此时容许偶尔丢帧是更高效的选择。

听取（hear）操作中，执行完数据分发（distrib）后，根据prot成功处理的有效帧数（prot.cnt），来决定是否执行某项操作，即“每n帧执行一次m”。本程序中设定的是：每隔5帧更新一次绘图数据（update\_figdata），每隔5帧更新一次仪表图片（update\_figure\_2），每隔15帧更新一次曲线图片（update\_figure\_1）。因为曲线图片更新是最耗时的，所以令其更新频率最低。应根据具体需求和计算机计算能力来合理配置更新频率。

绘图数据存储在传感器数据类（class SensorPackage）的实例datashow中，因为它的缓冲区（data\_buf）可以方便地存储绘图所需的所有传感器数据。更新绘图数据（update\_figdata）的操作中，将接收到的传感器数据（sens.data\_buf）的最新5帧取平均后拷贝给绘图数据（datashow.data），从而实现数据的过滤以抹除噪音。

# 附录一：通信协议

