A1 软件手册

版本 1.0

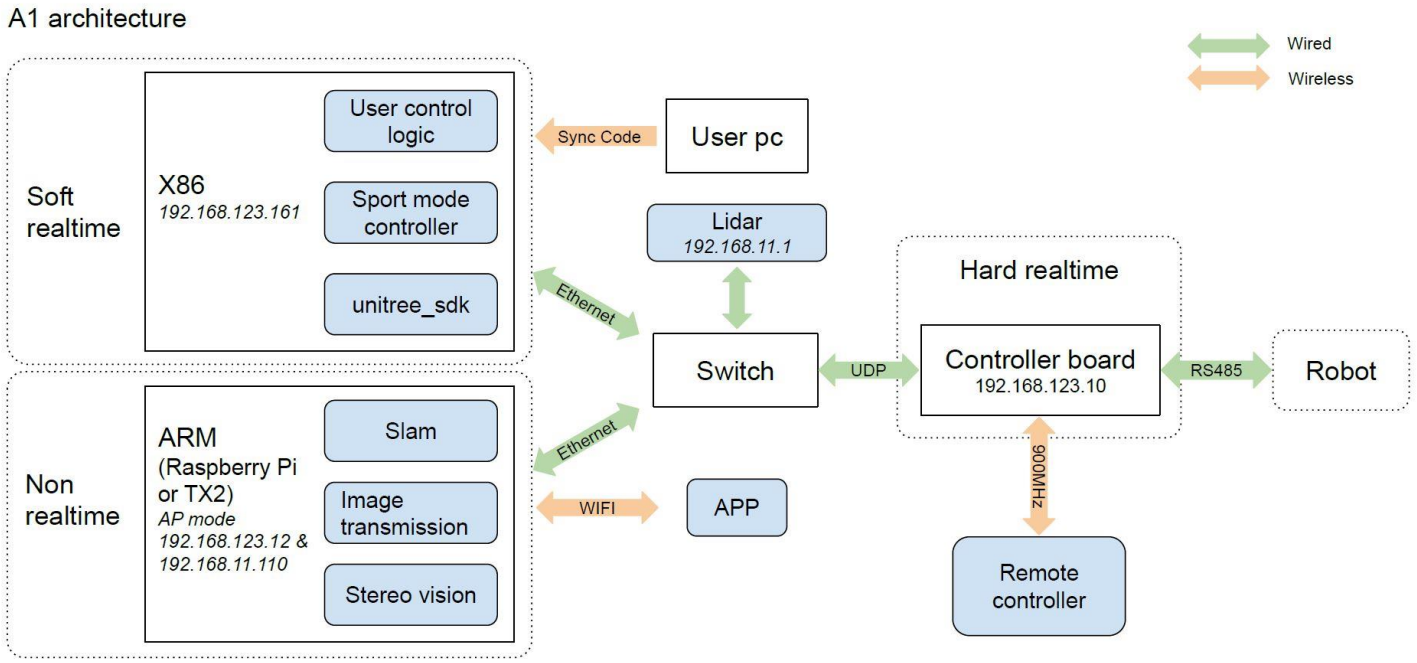
2020.7.28

目录

1 新手入门

1.1 机器人系统架构

模块架构示意图如下：



系统示意图

板载 PC 的操作系统为实时 Linux(Ubuntu)操作系统，实时通讯频率最高 1KHz。如果选用的 X86 平台是 Upboard，并且如果需要使用它的 HDMI 图形界面，需先连接好 HDMI 再开机。

1.2 控制器连接设置

机器人板载 PC 分别是 X86 架构运动控制器和 ARM 架构智能感知控制器。

个人 PC 和 X86 板载 PC 连接需要通过网线插入 X86 网口实现有线连接，第一次连接时，需要使用 HDMI 线，键盘和网线来连接板载 PC，得到相应的 IP 地址。确保自己的 PC 和板载 PC 在同一个局域网中。通过运行"ping" 测试检查连接。

个人 PC 和 X86 板载 PC 需要无线连接时，需要在 X86 板载 PC 上外接 USB 无线网卡（需支持 Linux 系

统)，然后个人 PC 连接外接无线网络。

个人 PC 和 ARM 板载 PC 连接可以通过网线插入 ARM 网口实现有线连接，也可以通过连接机器人自带 wifi（unitree_robotics_xxx,密码是 00000000）实现无线连接，第一次连接时，需要使用 HDMI 线，键盘和网线来连接板载 PC，得到相应的 IP 地址。确保自己的 PC 和板载 PC 在同一个局域网中。通过运行"ping"测试检查连接。

用户代码同步功能依赖于"scp"，远程登录功能依赖于"ssh"，确保电脑已经安装好:

```
sudo apt-get install ssh
ssh-keygen -t rsa
(接下来按三次回车)
scp ~/.ssh/id_rsa.pub unitree@${unitree_1}:~/.ssh/authorized_keys
(远程登录的初始密码是"123"。
如果遇到公钥错误，运行如下:
mv ~/.ssh/known_hosts known_hosts.bak
scp -o StrictHostKeyChecking=no ~/.ssh/id_rsa.pub unitree@${unitree_1}:~/.ssh/authorized_keys
)
```

板载 PC 直接连接到运动控制器，板载有线接口使用默认的 IP 地址和端口。

有线接口与运动控制器直连以控制机器人，无线接口连接到远程 PC，用于数据通信和同步用户控制代码。

注意:

1. *主控板的 IP 是静态的，为 192.168.123.10*
2. *需要通信的两个模块应在同一子网上。*
3. *miniPC 有线网络目前是双子网段。*
4. *有线和无线网络应该在不同的子网上。*

(控制器网络连接方法:
无线连接：使用支持 Ubuntu 系统、免驱的无线网卡接入相对应需要联网的板载 PC 实现网络连接。
有线连接：通过 USB 转网口转接线连接有线网络。
注：因为为保证机器人内部控制器通讯，机器人板载 PC 的网卡都被设置成静态 IP 地址，直接通过板载电脑 PC 的网卡无法连接网络。)

1.3 单位

开发中，未指明的单位统一按照国际标准单位：

长度：米（m）
角度：弧度（rad）
角速度：弧度每秒（rad/s）
力矩：牛米（N.m）
重量单位：千克（kg）
惯性张量单位：($kg \cdot m^2$)

1.4 坐标系、运动学和动力学

1.4.1 关节序号与关节限位

四足机器人像动物一样，身体（Trunk）和四肢（Leg）都是左右对称的。四条腿，按照前后分成两组，这两组除了前后不同，坐标系和关节活动范围等都相同的。但是我们定义的坐标系并不是镜面对称的，见 1.4.2 节。

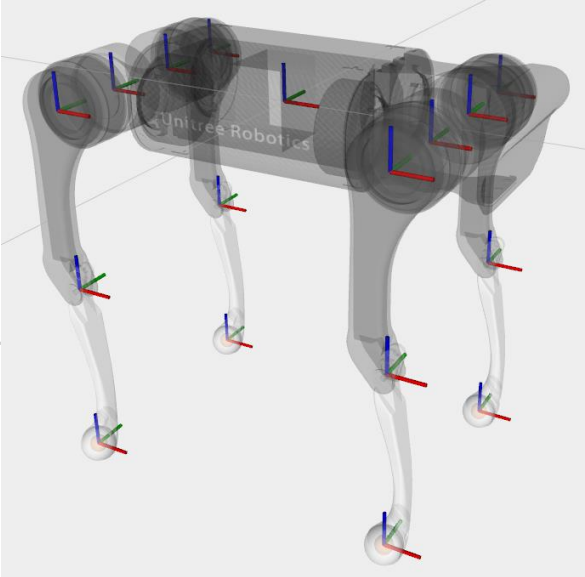
腿和关节的序号：

Leg 0：FR，右前腿
Leg 1：FL，左前腿
Leg 2：RR，右后腿
Leg 3：RL，左后腿
Joint 0：Hip，机身关节
Joint 1：Thigh，大腿关节
Joint 2：Calf，小腿关节
e.g. FR_thigh：右前腿大腿关节

各腿的关节限位是相同的。关节限位：

机身关节：-46°~46°
大腿关节：-60°~240°
小腿关节：-154.5°~-52.5°

1.4.2 坐标系，关节旋转轴与关节零点



ROS 表示的全身各个坐标系

机身关节旋转轴为 x 轴，大腿关节和小腿关节的旋转轴为 y 轴，旋转正方向符合右手定则。

当各个关节均为零度时，各坐标系如上图。红色为 x 轴，绿色为 y 轴，蓝色为 z 轴。小腿关节由于限位，实际到不了这个位置，这里仅为了指明零点位置。可以看出，各个关节坐标系的初始姿态都是一致的，只是位置和旋转轴不同。

1.4.3 运动学参数

运动学参数：

```
Hip link length: 0.0838
Thigh link length: 0.2
Calf link length: 0.2
Trunk length = 0.1805 * 2
Trunk width = 0.047 * 2
```

其他尺寸参数可以通过测量我们提供的三维模型来获得。

1.4.4 动力学参数

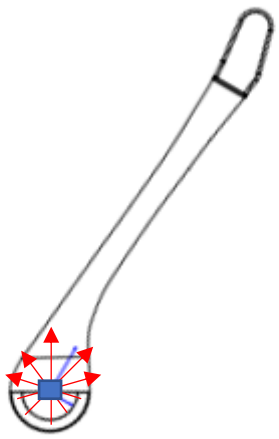
考虑到对称性，我们只提供必要模块的参数。在我们的 3D 模型中可以找到每个模块的参考坐标系。在我们的 ROS 包中同样可以找到这些参数充电。

每个模块包含三个关键要素：质量、质心位置和惯性张量，详细参数请见 GitHub 链接：

https://github.com/unitreerobotics/unitree_ros/tree/master/robots/a1_description。

1.5 足端力传感器

在四个足的末端的相同位置，均安装力传感器，安装位置如图：



力传感器示意图

图中蓝色的部分是传感器，红色箭头代表力传感器的方向，传感器上的力的方向有足端和地面具体的接触的形式决定的，垂直于接触面。如果足端接触是多个点接触的话，力的大小可以等效为这些力的合力，此传感器对于球面任一点的垂线方向都是敏感的。另外，此传感器的漂移比较严重，需要间歇性的标定零点（比如在足端离地的时候进行标定）。

2 API

对机器人的控制分为两种：高层控制和底层控制。开机后，用户只能处于这两种控制方式中的一种，不能进行切换控制方式的操作。

在高层控制下，由于机器人可能运行在普通模式或运动模式，所以需要在初始化 udp 的目标 ip 和 port 时加以区分，即普通模式的 ip:192.168.123.10, port:8007；运动模式的 ip:192.168.123.161, port:8081.

在底层控制下，电机拥有三种控制模式：力矩模式、速度模式和位置模式，详情请查阅第 3.2.3 节。

与那些通过函数调用的 API 不同，我们的 API 是通过收发封装好的结构体实现的，更有利于开发。用户控制的默认工作空间是“~/unitree_legged_sdk”。目前给出的教程与接口，都是基于 C++的。在对机器人

进行命令发送控制与状态接收时，需要使用通讯库 libunitree_legged_sdk.so 及头文件"unitree_legged_sdk.h"。 此库是开发者最常用的库，包含了控制所需的通讯接口。

2.1 高层控制模式

下面包含一些说明，详细代码参考教程中的"comm.h"。

状态（HighState）	意义
levelFlag	高、底层模式标志："0x00"为高层，"0xff"为底层
mode	机器人当前模式："1"为站立模式，"2"为行走模式 切换模式需要 1s 左右的耗时
imu	包括陀螺仪、加速度计、温度，解算出的欧拉角与四元数
forwardSpeed	向前走动的速度
sideSpeed	侧向行走的速度
rotateSpeed	身体旋转速度
bodyHeight	身体当前高度
updownSpeed	站立或蹲下的速度
forwardPosition	身体前方的目标位置
sidePosition	身体的侧向目标位置
footPosition2Body	相对于身体的足部位置
footSpeed2Body	相对于身体的足部速度
footForce	足端受到的力
tick	运动控制器返回从机器人开机开始计时的参考时间
wirelessRemote	无线命令
crc	校验位
指令（HighCmd）	
levelFlag	同上
mode	同上
forwardSpeed	向后/前移动指令，取值范围(-1~1)，对应(-0.7~1 米每秒)的分段线性比例值（取 0 为分界点），前进最大 1m/s 后退最大 0.7m/s
sideSpeed	向右/左移动指令，取值范围(-1~1)，对应(-0.4~0.4 米每秒)的线性比例值
rotateSpeed	向右/左旋转指令，取值范围(-1~1)，对应(-120~120 度每秒)的线性比例值
bodyHeight	调整站立高度指令，取值范围(-1~1)，对应(0.3~0.45 米)的分段线性比例

	值（取 0.41 米为分界点，即默认高度）
yaw	偏航指令，取值范围(-1~1)，对应(-28~28 度)的线性比例值
pitch	俯仰指令，取值范围(-1~1)，对应(-20~20 度)的线性比例值
roll	横滚指令，取值范围(-1~1)，对应(-20~20 度)的线性比例值
led	保留位
wirelessRemote	同上
crc	同上

2.2 底层控制模式

下面包含一些说明，详细代码参考教程中的"comm.h"。

状态（LowState）	
levelFlag	同上
imu	同上
motorState	包括目标电机双编码器的位置、速度、转矩、温度，以及工作模式
footForce	同上
tick	同上
wirelessRemote	同上
crc	同上
指令（LowCmd）	
levelFlag	同上
motorCmd	包括目标电机的位置、速度、转矩、刚度系数、阻尼系数，以及工作模式
led	同上
wirelessRemote	同上
crc	同上

2.3 保护模式

2.3.1 摔倒保护

当处于高层模式，并出现摔倒或跌落状况时（开机状态下将机器人拎起，等同于失重跌落），机器人会自动进入摔倒保护模式：关节切换至纯阻尼模式。

2.3.2 失连保护

失连保护也叫连续丢包保护，是为了应对通讯状况不佳时发生的丢包现象，可能的原因有：网络不稳定、USB 大量数据传输或 GPU 运算造成的 CPU 中断，等等。

如果有丢包但是没有连续 30 毫秒之久，例如丢了一个或几个包，则机器人会按照接收到的最后一条指令继续运行；如果连续 30 毫秒没有收到新的指令，那么机器人会进入失连保护模式，直到新的指令到来。

1. 当处于高层模式并出现失连现象：切换为站立模式，并且之前的高层控制命令失效。
2. 当处于底层模式并出现失连现象：关节切换至电子刹车制动状态。

3 控制教程

3.1 紧急制动

如果出现意外情况，可以通过紧急远程遥控器，按下 OFF 键（1.5 秒），机器人将直接断电，停止一切动作。详细细节请参阅使用手册。

3.2 电机

与电机相关的指令（MotorCmd）与状态(MotorState)，都是经过相对应的减速比之后的，所以对电机的控制可以等价看作为对关节的控制，从而不必考虑减速比。

3.2.1 电机注意事项

电机的使用需要注意发热现象。电机内置了两个温度传感器，用于实时监测电机温度。根据以下两个公式：

1. 根据"焦耳定律": $Q = I^2Rt$ （Q:热量，I:电流，R:电阻，t:时间）
2. 转矩与电流关系: T = K*I （T: 转矩，I:电流，K:转矩系数）

得出，当电机大扭矩输出时，电机发热与输出转矩成平方关系，故大转矩输出时，电机发热会比较大。在短时间内，由于电机本身的比热容，瞬时的大转矩输出不会明显的提高电机的温度。但由于热量传导的速度比较慢（热阻），传感器检测到的温度变化，会有较为明显的滞后。当电机驱动板检测到电机的温度大于 60°C 时，会强制关闭电机，等到电机温度下降到一定程度后才会再次开启。

3.2.2 电机运行模式

底层模式开发是直接控制电机的，第一步需要把设置电机的运行模式：

motorCmd[xx].mode	模式
0x00	电子刹车制动
0x0A	伺服

3.2.3 控制模式：位置、速度与力矩

位置刚度 positionStiffness 的单位是 N.m/rad，速度刚度 velocityStiffness 的单位是 N.m/(rad/s)

电机控制环的位置、速度与转矩模式是通过数值参数设置的。

转矩环模式，输出期望转矩 T，此时需要先禁用位置环与速度环。分为两种禁用方式，第一种需要把期望的位置与速度设为禁用标志值，PosStopF = 2.146E+9f，VelStopF = 16000.0f。第二种需要把位置刚度与速度刚度设为零。第一种禁用方式比第二种稍快一些，二者可以混合使用。这里以左前腿的第二个关节(FL_1)为例：

```
motorCmd[FL_1].position = PosStopF;
motorCmd[FL_1].positionStiffness = 0;
motorCmd[FL_1].velocity = VelStopF;
motorCmd[FL_1].velocityStiffness = 0;
motorCmd[FL_1].torque = T;
```

速度环模式，输出期望速度 V，速度刚度不能为零，此时需要先禁用位置环，并把期望转矩设为零：

```
motorCmd[FL_1].position = PosStopF;
motorCmd[FL_1].positionStiffness = 0;
motorCmd[FL_1].velocity = V;
motorCmd[FL_1].velocityStiffness = 4;    // just for reference
motorCmd[FL_1].torque = 0;
```

位置环模式，输出期望位置 P，位置刚度与速度刚度不能为零，此时需要设置期望速度为零来作为阻尼项，并把期望转矩设为零：

```
motorCmd[FL_1].position = P;
motorCmd[FL_1].positionStiffness = 5;   // just for reference
motorCmd[FL_1].velocity = 0;
motorCmd[FL_1].velocityStiffness = 1;   // just for reference
motorCmd[FL_1].torque = 0;
```

复合控制模式，可以将三种模式按需要组合，例如可以只开两环或三环全开，这里以三环全开为例：

```
motorCmd[FL_1].position = P;
motorCmd[FL_1].positionStiffness = 5;   // just for reference
motorCmd[FL_1].velocity = 0;
motorCmd[FL_1].velocityStiffness = 1;   // just for reference
motorCmd[FL_1].torque = T;
```

3.3 示例

见 https://github.com/unitreerobotics/unitree_legged_sdk/tree/master/examples。注意执行时，因为涉及到系统调用，需要加上 sudo 权限。

其中，example_walk 是高层控制，example_position, example_velocity, example_torque 都是底层控制。

4 A1_ros

见 https://github.com/unitreerobotics/unitree_ros。

我们提供了可以控制虚拟机器人的 ROS 接口。将"unitree_ros"文件夹放入到 catkin 工作空间下（一般为：~/catkin_ws/src/unitree_ros），然后编译：

```
cd ~/catkin_ws
catkin_make
```

我们使用的环境是：Ubuntu16.04 + ROS Kinetic 或 Ubuntu18.04 + ROS Melodic。

4.1 ros 依赖

4.1.1 建立消息 msgs

在 ROS 中，节点本身包含用于通信的许多基本数据类型。需要定义像 C 语言中的结构体一样的，机器人需要的数据集。 我们使用的 msgs 在 unitree_legged_msgs 文件夹下，内容与 A1 API 中使用的通信结构体一致。需要先用 catkin_make 把 unitree_legged_msgs 编译通过，否则可能会产生依赖问题（例如 找不到头文件）。

4.1.2 建立控制器

ROS 自身提供了多种多样的控制器（例如 position_controllers），但是为了实现与机器人良好的集成，我们在这里没有使用它自带的控制器，而是构建了自己的控制器。这个控制器类型继承自“hardware::EffortJointInterface”。

4.2 Rviz 可视化

机器人描述文件使用了比 URDF 格式更简洁的 xacro 格式，包含了机器人关节限位，碰撞空间，运动学和动力学等方面的细节。碰撞空间使用了简单的几何形状来提升物理引擎的性能。 ROS 的 Rviz 只用于可视化，可以通过 joint_state_publisher 来检查连杆和每个关节的运动范围。使用方法见 https://github.com/unitreerobotics/unitree_ros/tree/master/robots/a1_description/launch。

4.3 Gazebo 动力学仿真

见 https://github.com/unitreerobotics/unitree_ros/tree/master/unitree_gazebo。

4.3.1 建立插件 plugin

在 Gazebo 中，如果想得到仿真的数据（比如关节角度、速度、力），需要通过插件来实现。在调用 Gazebo API 获得关键数据并处理后，将数据放入节点进行通讯。还要注意，许多插件需要依赖于连杆，关节或模块，进一步的细节，请参阅“gazebo.xacro”文件。接下来将展示如何构建一个显示足端受力的插件。

1. 获得足端受力的插件

首先，应该得到脚和地面之间的接触力。有关详细信息，请参阅“foot_contact_plugin.cc”。这个插件属于传感器插件。

2. 力可视化插件

在获得力的数值之后，需要将其可视化。有关详细信息，请参阅“draw_force_plugin.cc”。这个插件属于视觉插件。

4.3.2 仿真运行

每一个节点都是一个进程，所以这里至少需要 2 个终端窗口。首先需要确保代码编译成功：

```
cd ~/catkin_ws
catkin_make
```

Terminal-1，通过 roslaunch 运行 Gazebo，它将初始化场景、插件、关节控制器：

```
roslaunch unitree_gazebo normal.launch
(如果不能运行，那么：
cd ~/catkin_ws/src/unitree_ros/unitree_gazebo/launch
roslaunch ./ normal.launch
)
```

Terminal-2，运行示例运动节点：

```
rosrun unitree_legged_gazebo unitree_servo
```

我们还提供了一个施加外部力的节点，来模拟外界的扰动：

```
rosrun unitree_legged_gazebo unitree_external_force
```

用户可添加更多节点以完成目标任务。

4.4 ROS 控制机器人

4.4.1 LCM Server

ROS1 的实时性是没有保障的，在 ROS 的节点里发送 UDP 指令可能引发通讯异常问题。需要注意的是，在和机器人的通讯里，并没有采用 ROS Subscriber/Publisher 的方式来传递数据，而是采用 LCM Channel 实现进程间消息的传递。非实时的 ROS 进程将不会影响到控制机器人的实时进程，以此来保障实时性。

在 LCM Server 里，LCM 的命令将转化成 UDP 命令向下发送，UDP 状态转化成 LCM 状态向上发送。在 ROS 里，（可选）可以建立一个节点单独用来发送 LCM 命令和接收 LCM 状态。

4.4.2 控制示例

这里以底层控制为例，至少需要 3 个终端窗口。

首先把 unitree_legged_real 文件夹拷贝进~/catkin_ws/src，并编译：

```
cd ~/catkin_ws
catkin_make
```

Terminal-1，运行 sdk 中的 lcm server：

```
sudo ~/unitree_legged_sdk/build/sdk_lcm_server_low
```

Terminal-2，运行 ros master：

```
roscore
```

Terminal-3，通过 rosrun 运行用户控制逻辑：

```
rosrun a1_real position_lcm_publisher
```

5 外部传感器

SLAM 功能及相关 API，参考《基于 Mapper 的 2D-SLAM 系统及路径规划功能使用手册》