Go1Software Manual

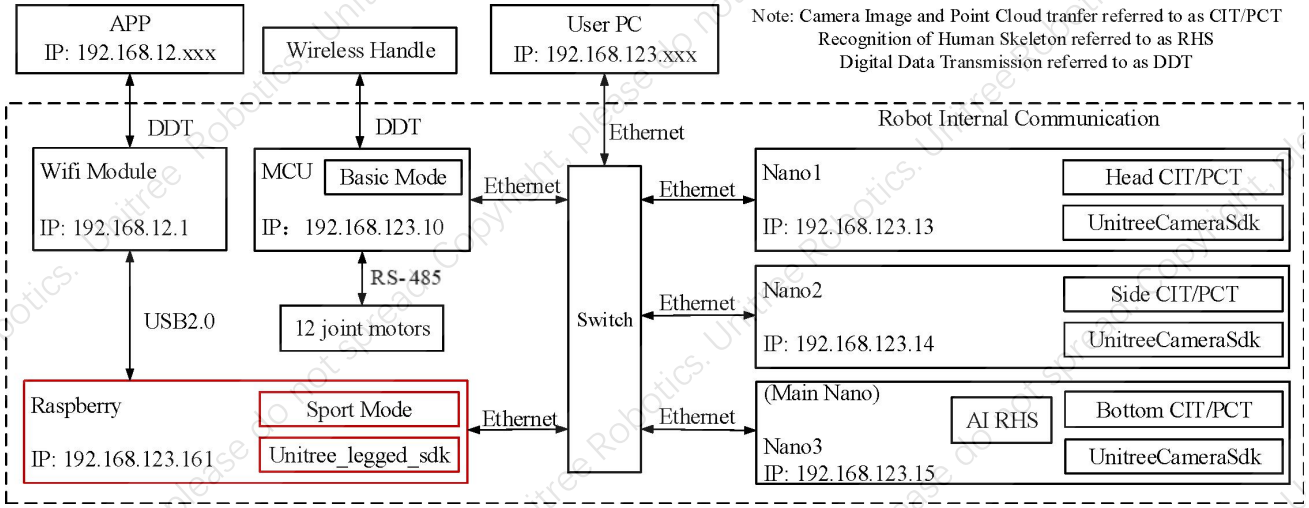
2021.12.1

contents

1 System

1.1 Robot system structure

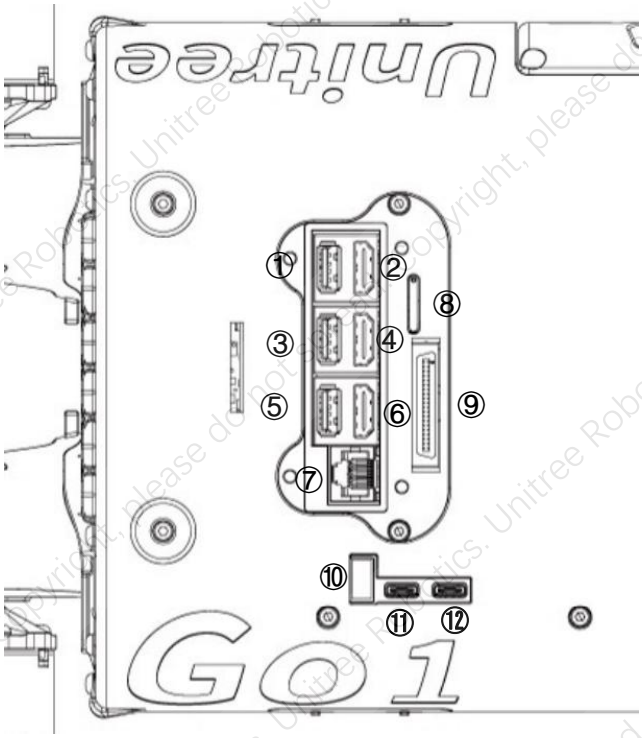
The basic architecture of the system schematic diagram shown below.



System diagram

The operating system of the onboard PC is a real-time Linux (Ubuntu) operating system and an ARM architecture, and the real-time communication frequency is up to 1KHz.

The following is a schematic diagram of the back interface of Go1:



1. Nano2 USB
2. Nano2 HDMI
3. Nano/Nx3 USB
4. Nano/Nx3 HDMI
5. Raspberry USB
6. Raspberry HDMI
7. Gigabit Ethernet port
8. SIM
9. Integratedinterface
10. External power input （24V 30A）
11. Type-C
12. Type-C

In order to meet the different needs of users, the controller with IP 192.168.123.15 is Nano controller or Nx controller

Among them, the head Nano controller 1 does not open the usb and hdmi interfaces. For access, please see the network connection section below

1.2 Network configuration

1.2.1 wifi settings

The robot transmits a wireless network segment of 192.168.12.xxx, so users can connect to this wifi with their PC and remotely connect to the robot's operating system via SSH:

```
ssh pi@192.168.12.1
Password:123
```

After logging in to the Raspberry Pi system, if you need to access other nano controllers, you need to perform a secondary SSH remote through the udp of the internal network segment 192.168.123.xxx. Take nano controller 1 as an example:

```
ssh unitree@192.168.123.13
Password:123
```

1.2.1 wired settings

The back of the robot opens the udp network cable interface, which is directly connected to the internal switch. Therefore, after connecting the user's personal PC and the robot through the network cable, you can access any controller. Take nano controller 1 as an example:

```
ssh unitree@192.168.123.13
Password:123
```

Note: Since the internal wired network of the robot is in the 192.168.123.xxx network segment, when the personal PC and the robot are directly connected through the network cable, the network of the personal PC needs to be set to the static 192.168.123.xxx network segment, which is static The ip should avoid conflict with the ip of the board in the picture above

1.3 Units

In development, unspecified units are unified according to international standard units:

Length unit: meter (m)

Angle: radian (rad)

Angular velocity: radians per second (rad/s)

Torque: Nm (N.m)

Mass unit: kilograms (kg)

Inertial tensor unit: (kg·m2)

1.4 Coordinate, kinematics and dynamics

1.4.1 Joint number and joint limits

The quadruped robot is like an animal, and its trunk and legs are bilaterally symmetrical. The four legs are divided into two groups according to the front and back. The two groups are the same except for the front and rear. The coordinate system and joint motion range of the two groups are the same. But the coordinate system we defined is not mirror-symmetric, see section 1.4.2.

Number of legs and joints:

Leg0 FR = right front leg

Leg1 FL = left front leg

Leg2 RR = right rear leg

Leg3 RL = left rear leg

Joint 0：Hip, Hip joint

Joint 1：Thigh, Thigh joint

Joint 2：Calf, Calf joint

e.g. FR_thigh：right front leg thigh joint

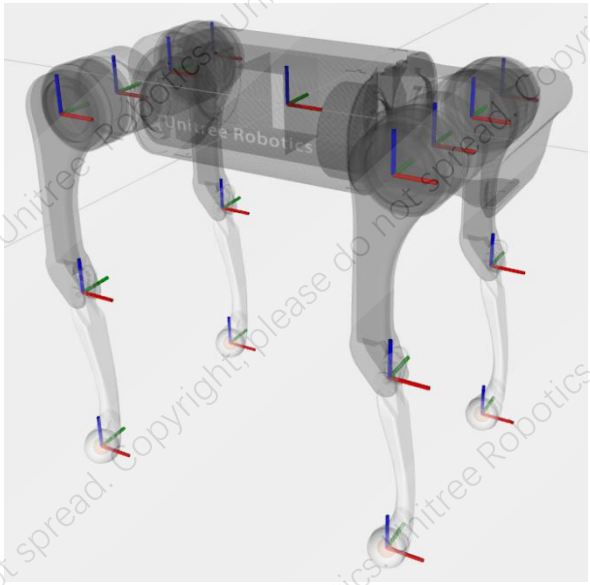
Joint limitations:

Hip joint：-60°~60°

Thigh joint：-38°~170°

Calf joint：-156°~-48°

1.4.2 Coordinate, joint axis and zero point



All coordinates under ROS

The rotation axis of the hip joint is the x-axis, and the rotation axis of the thigh and calf joints is the y-axis, and the positive rotation direction conforms to the right-hand rule.

The zero points of each coordinate are shown above. Red is the x-axis, green is the y-axis, and blue is the z-axis. Due to the limit of the calf joint, this position cannot actually be reached. It can be seen that the initial posture of each joint coordinate system is the same, but the position and rotation axis are different.

1.4.3 Kinematic parameters

Can be measured by the three-dimensional model we provide （unitree_ros/const.xacro at master · unitreerobotics/unitree_ros (github.com)）

1.4.4 Dynamics parameters

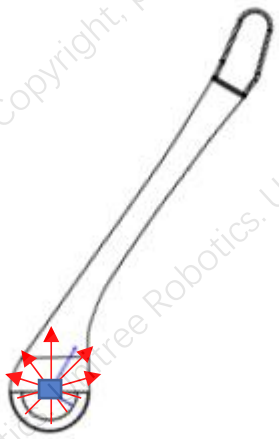
Considering the symmetry, we only provide parameters of necessary modules. You can find the reference coordinate of each module in our 3D models. You can also find those parameters in our ROS package.

Each module contains three key elements: mass, position of the center of mass(CoM) and inertial tensor.

For other details about dynamics, please refer to
unitree_ros/robots/go1_description at master · unitreerobotics/unitree_ros (github.com)

1.5 Foot force sensor

There are four force sensors at the same position of each foot. The position is as follows:



Force sensor

The blue part in the figure is the sensor, and the red arrow represents the direction of the force sensor. The direction of the force on the sensor is determined by the specific contact form of the foot end and the ground, which is perpendicular to the contact surface. If the foot contact is a multiple point contact, the magnitude of the force can be equivalent to the combined force of these forces. This sensor is sensitive to the direction of the vertical line at any point on the spherical surface. In addition, the drift of this sensor is serious, and it needs to calibrate the zero point intermittently (such as when the foot is off the ground).

2 Unitree_legged_sdk

User control to robot is divided into two types: high level control mode and low level control mode. User can only be in one of the two modes at the same time, and cannot switch after running.

Under high-level control, initialize the target ip and port of udp as ip:192.168.123.161, port:8082,

Under the control of the bottom layer, the target ip and port of the initialization udp are ip:192.168.123.10, port:8007.

In low level mode, the motor also has three modes: torque mode, speed mode, and position mode.

Unlike other API through function calls, our API is all through communication interface that packaged into "struct", which is more convenient to development. The default workspace is "~/unitree_legged_sdk". Before control the robot, the library file "libunitree_legged_sdk.so" and header file "unitree_legged_sdk.h" should be included. This library is the most commonly used library by developers and contains the communication interface required for control

For details, please refer to《unitree_legged_sdk manual》

3 Go1_ros

unitreerobotics/unitree_ros (github.com)

Before using it, please note that the ros package only contains the simulation part,If you need to control the robot, you need to compile the routine of **unitree_ros_to_real**and**unitree_ros**in the same workspace..We provide a ROS interface that can control the virtual robot. Put the "unitree_ros" folder into the catkin workspace

(Generally: ~/catkin_ws/src/unitree_ros), then compile:

```
cd ~/catkin_ws
catkin_make
```

Linux system: Ubuntu18.04 + ROS Melodic。

3.1 ros dependency

3.1.1 build message msgs

In ROS, the node itself contains many basic data types for communication. It is necessary to define data sets needed by robots, like structures in C language. The msgs we use are in the go1_msgs folder, and the content is consistent with the communication architecture used in the Go1 API. You need to compile go1_msgs with catkin_make first, otherwise you may have dependency problems (such as that you cannot find the header file).

3.1.2 build controller

ROS itself provides a variety of controllers (such as position_controllers). In order to achieve good integration with the robot, we do not use its own controllers here, but build our own controller. This controller inherits from "hardware::EffortJointInterface".

3.2 Rviz

The robot description file uses the xacro format, which is more concise than the URDF format, and contains details of the robot joint limit, collision space, kinematics and dynamics. The collision space uses simple geometric shapes to improve the performance of the physics engine. Rviz of ROS is only used for visualization. You can use joint_state_publisher to check the link and the range of motion of each joint. See how to use it

unitree_ros/go1_rviz.launch at master · unitreerobotics/unitree_ros (github.com)

3.3 Gazebo

见 unitree_ros/unitree_gazebo at master · unitreerobotics/unitree_ros (github.com)

3.3.1 build plugins

In Gazebo, if you want to get simulated data (such as joint angle, speed, force), you need to achieve this through plugins. After calling the Gazebo API, the data should be put into the node for communication. Also note that many plugins need to rely on links, joints or modules. For further details, refer to "gazebo. xacro" file. The following shows how to build a plugin that can visualize the foot force.

Plugin for obtaining foot force

First, the contact force between the foot and the ground should be obtained. For more information, see "foot_contact_plugin.cc". This plug-in belongs to the sensor plugin.

Plugin for force visualization

After obtaining the force value, it needs to be visualized. For more information, see "draw_force_plugin.cc". This plugin belongs to the visual plugin.

3.3.2 Run the simulation

Each node is a process, so we need two terminals here. Firstly, make sure the compilation:

```
cd ~/catkin_ws
catkin_make
```

Terminal-1 will initiate scene, plugins and controllers by roslaunch:

```
roslaunch unitree_gazebo normal.launch rname:=go1 wname:=stairs
```

Terminal-2 will run all the nodes:

```
rosrun unitree_controller unitree_servo
```

We also provide a node for imposing an external force to simulation the outside disturbance:

```
rosrun unitree_controller unitree_external_force
```

Users can add more nodes to accomplish the target task.

3.4 ROS control robot

3.4.1 LCM Server

The real-time performance of ROS 1 is not guaranteed. Sending UDP instructions in ROS nodes may cause communication abnormalities. It should be noted that in the communication with robot, ROS Subscriber/Publisher is not adopted to transfer data. As an alternative, LCM is used to transfer messages between processes. Non-real-time ROS processes will not affect the real-time process while controlling robot to ensure real-time performance.

In LCM Server, LCM commands will be converted into UDP commands to send down, UDP status will be converted into LCM status to send up. In ROS, a node can be set up to send LCM commands and receive LCM status separately.

3.4.2 control examples

Here take low-level control as an example, and we need three terminals here.

Firstly，copy go1_real to~/catkin_ws/src,and compile it：

```
cd ~/catkin_ws
catkin_make
```

Running lcm server：

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

Running ros master:

```
roscore
```

Run user logic with rosrun:

```
rosrun go1_real position_lcm_publisher
```

4 Go1_camera_SDK

GO1 EDU itself contains 5 sets of cameras. UnitreeCameraSDK is a cross-platform library of the stereo camera. It can provide depth and color video streams, internal calibration information, and depth images where the point cloud is aligned with the color image.

4.1 Control the use of robot cameras

There are three steps to using this software development kit:

·rely

·Compile

·run

4.1.1dependency

OpenCV (version 4.1.1)

CMake(version 2.8 or higher)

OpenGL

GLUT

X11

If you are using the onboard computer of go1, you can skip the step of installing dependencies

4.1.2 compile

Open the terminal and execute the following command.

```
cd UnitreeCameraSDK
mkdir build#&&cd build
```

```
cmake ../
make
```

4.1.3 run

Open the binary files in the "bins" folder in the middle section and run the routines directly when the camera is not in use, such as

```
sudo ./bins/example_getRawFrame
```

Get the internal parameters of the camera：

```
sudo ./bins/example_getCalibaParamsFile
```

output_camCalibParams.yam
Param:

```
Left/RightXi Mei's 模型中:xi
Left/RightIntrinsicMatrix:内参矩阵
Left/RightDistortionCoefficients:畸变矩阵
Left/RightRotation:双目立体视觉矫正矩阵
Translation: 相机间距
```

4.2 System Files

Include：

```
StereoCameraCommon.hpp
Declared the stereo camera algorithm API, such as image capture, image correction, parallax calculation, depth image and point cloud generation functions。
SystemLog.hpp
It declares the API of the logging system
UnitreeCameraSDK.hpp
It declares the unitree camera API, which inherits the API of the StereoCamera class, and updates the camera firmware to the camera
```

Examples：

example_getCalibParamsFile.cc	Get the internal parameters of the camera
example_getDepthFrame.cc	Get camera depth video stream
example_getPointCloud.cc	Get camera point cloud
example_getRawFrame.cc	Get camera RGB binocular video stream
example_getRectFrame.cc	Obtain RGB binocular video stream after camera correction