A1 Software Developer Guide

Version 1.0
07.28.2020

Catalogue

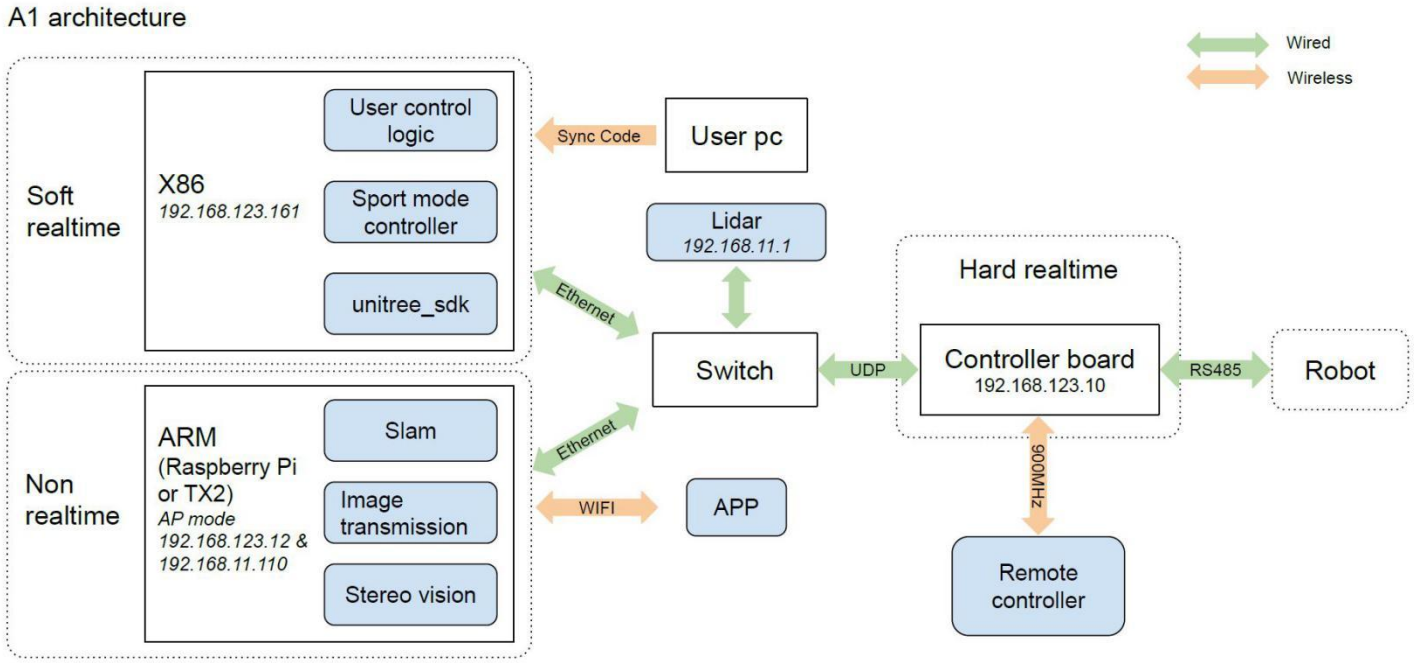
A1 Software Developer Guide 1
1 Getting started 4
1.1 Robot system structure 4
1.2 Setting up the network 4
1.3 Units 5
1.4 Coordinate, kinematics and dynamics 5
1.4.1 Joint number and joint limits 5
1.4.2 Coordinate, joint axis and zero point 6
1.4.3 Kinematic parameters 7
1.4.4 Dynamics parameters 7
1.5 Foot force sensor 7
2 API 8
2.1 High level control mode 8
2.2 Low level control mode 10
2.3 Protection mode 10
2.3.1 Fall protection 10
2.3.2 Disconnection protection 10
3 Control tutorial 11
3.1 Emergency braking 11
3.2 Motor 11
3.2.1 Cautions for motor 11
3.2.2 Motor operation mode 12
3.2.3 Control Mode: Position, Speed and Torque 12
3.3 Examples 13
4 A1_ros 13
4.1 ros dependency 13
4.1.1 build message msgs 13
4.1.2 build controller 14
4.2 Rviz visualization 14
4.3 Gazebo dynamic simulation 14
4.3.1 build plugins 14

4.3.2 Run the simulation.....14
4.4 ROS control robot.....15
4.4.1 LCM Server.....15
4.4.2 control examples.....15
5 External Sensors.....16

1 Getting started

1.1 Robot system structure

The basic architecture of the system schematic diagram shown below.



Robot system schematic

The operation system of Onboard PC is real time Linux(Ubuntu), which maximum real-time communication bandwidth is 1000Hz. If the X86 platform is Upboard, and if you need to use its HDMI graphical interface, you need to connect HDMI before turning on.

1.2 Setting up the network

When using robot for the first time, you need to use an HDMI cable and a keyboard to connect the Onboard PC to get its IP address. Once you have obtained the IP address (for example: 192.168.1.100), you can remove the HDMI cable and keyboard and connect it wirelessly.

Check the connection by running the "ping" test.

The user code synchronization depends on "scp", and the remote login depends on "ssh". Please ensure that the computer is installed:

```
sudo apt-get install ssh
ssh-keygen -t rsa
(Press "Enter" for three times)
```

scp ~/.ssh/id_rsa.pub unitree@${unitree_1}:~/.ssh/authorized_keys

(The initial password for remote login is "123".

If you encounter a public key error, run as follows:

mv ~/.ssh/known_hosts known_hosts.bak

scp -o StrictHostKeyChecking=no ~/.ssh/id_rsa.pub unitree@${unitree_1}:~/.ssh/authorized_keys

)

The On board PC is directly connected to the motion controller without any intermediate switches to decrease delay, jitter or package loss. The LAN port is initialized with default IP address and Port.

note:

1. The IP of the main control board is static, which is 192.168.123.10
2. The two modules that need to communicate should be on the same subnet.
3. The miniPC wired network is currently a dual subnet segment.
4. Wired and wireless networks should be on different subnets.

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:

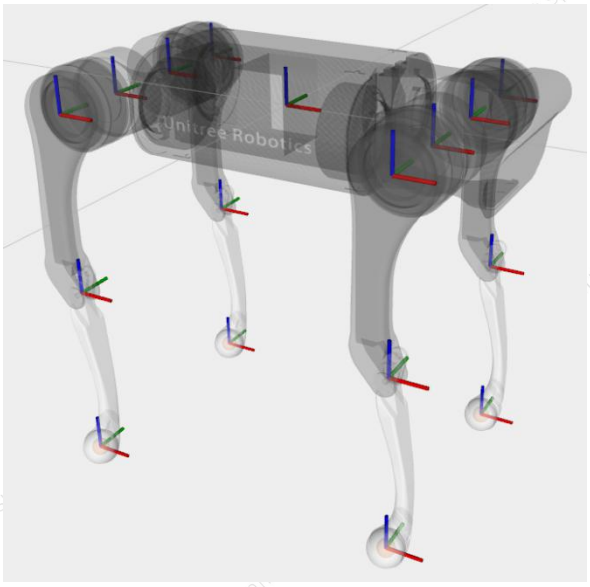
Leg0 FR = right front leg
Leg1 FL = left front leg
Leg2 RR = right rear leg
Leg3 RL = left rear leg

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: -46°~46°
Thigh joint: -60°~240°
Calf joint: -154.5°~-52.5°

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

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

Other dimensional parameters can be obtained by measuring the 3D model we provide.

1.4.4 Dynamics parameters

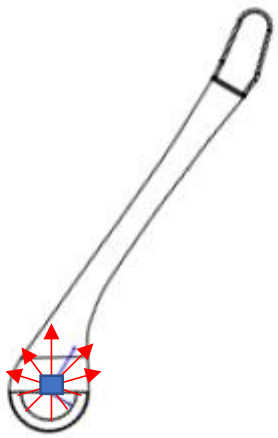
Considering the symmetry, we only provide parameters of necessary modules. You can find the reference coordinate of each module in our 3D models. You can also find those parameters in our ROS package.

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 https://github.com/unitreerobotics/a1_ros/tree/master/a1_description

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 API

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, since the robot may run in normal mode or motion mode, it needs to be distinguished when initializing the udp target ip and port, that is, normal mode ip: 192.168.123.10, port: 8007; motion mode ip: 192.168 .123.161, port:8081.

In low level mode, the motor also has three modes: torque mode, speed mode, and position mode. For details, refer to Part 3.2.3.

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". Currently, the tutorials and interfaces given are based on C++. 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.

2.1 High level control mode

Some instructions, specifically refer to the "comm.h" in the tutorial codes.

HighStatus	Significance
levelFlag	the flag of control level, high level:0x00, low level:0xff
mode	the running mode: standing:1, walking:2 . It takes about a second to switch modes.
imu	include gyroscope, accelerometer, thermometer and solved euler angle and quaternion
forwardSpeed	the speed of forward walking about the body

sideSpeed	the speed of sideward walking about the body
rotateSpeed	the speed of self rotating about the body
bodyHeight	the current height about the body
updownSpeed	the speed of standing or squating
forwardPosition	the forward position from odometry
sidePosition	the sideward position from odometry
footPosition2Body	the foot position about the body
footSpeed2Body	the foot speed about the body
footForce	the foot force
tick	reference time since robot boot
crc	check code
HighCmd	
levelFlag	ditto
mode	ditto
forwardSpeed	Move backward/frontward command, value range (-1~1), corresponding to the piecewise linear proportional value of (-0.7~1m/s) (0 is taken as the dividing point), the maximum forward speed is 1m/s, and the maximum backward speed is 0.7m/s
sideSpeed	Move rightward/leftward command, value range (-1~1), corresponding to the linear proportional value of (-0.4~0.4 m/s)
rotateSpeed	Turn right/left command, value range (-1~1), corresponds to a linear proportional value of (-120 ~120 degrees per second)
bodyHeight	Adjust body height command, value range (-1~1), corresponding to the piecewise linear proportional value of (0.3~0.45m) (take 0.41m as the dividing point, as the default height)
yaw	Yaw command, value range (-1~1), corresponding to the linear proportional value of (-28~28 degrees)
pitch	Pitch command, value range (-1~1), corresponding to the linear proportional value of (-20~20 degrees)
roll	Roll command, value range (-1~1), corresponding to the linear proportional value of (-20~20 degrees)
led	reserved
crc	ditto

2.2 Low level control mode

Some instructions, specifically refer to the “comm.h” in the tutorial codes.

LowState	
levelFlag	ditto
IMU	ditto
motorState	include the position, velocity, torque, temperature and working mode about the target motor
footForce	ditto
tick	ditto
wirelessRemote	ditto
crc	ditto
LowCmd	
levelFlag	ditto
motorCmd	include the torque, position, velocity, stiffness of position and stiffness of velocity about the target motor, and also the working mode
led	ditto
wirelessRemote	ditto
crc	ditto

2.3 Protection mode

2.3.1 Fall protection

When the robot is under high level control and encounters falling down, or hung up while the robot is standing or walking, it will switch to protection mode: All the joints switch to pure damping mode, and light up red led.

2.3.2 Disconnection protection

Disconnection protection is also called consecutive lose package protection, as a response to poor communication. Possible reasons are: unstable network, USB mass data transfer, interruptions by GPU, and so on.

If packet is lost but does not last for 30 milliseconds, such as one or more packets, the robot will continue to run according to the last received command. If no commands are received for 30 milliseconds, then it will switch to disconnection protection mode, until new command arrives.

1. High level disconnection: Robot will switch to stand mode and all of the high level command lose efficacy
2. Low level disconnection: Joints will switch to electronic braking mode.

3 Control tutorial

3.1 Emergency braking

If any accident occurs, pressing the OFF key (1.5 seconds) through the emergency remote control , then the robot will stop all actions and power off. For more details, please refer to the user manual.

3.2 Motor

The commands (MotorCmd) and state (MotorState) related to the motor are after the corresponding reducer, so the control of the motor can be equivalently regarded as the control of the joint. So that the reduction ratio does not have to be considered.

3.2.1 Cautions for motor

User should pay attention to the phenomenon of heating while using motor. Two thermal sensors are built into the motor to monitor the temperature in real time. According to the following two formulas:

1. According to Joule Law: Q = I2Rt (Q: heat, I: current, R: resistance, t: time)
2. Torque versus current: T = K*I (T: torque, I: current, K: torque coefficient)

It is concluded that when the motor has a square relationship with the output torque, so when output a bigger torque, the heat of the motor will be larger. In a short time, due to the specific heat capacity of the motor itself, the instantaneous high torque output will not significantly increase the temperature of the motor. However, due to the slow heat conduction speed (thermal resistance), the temperature changes detected by the sensor will have a more obvious lag. When the temperature of any thermal sensor detected by the motor drive board is over 60°C, the motor will be forced to shut down until the temperature drops to a certain extent before opening it again.

3.2.2 Motor operation mode

The low level control can operate the motor directly. The first step is to set up the operation mode of the motor:

motorCmd[xx].mode	Explaination
0x00	Electronic brake mode
0x0A	Servo (PMSM) mode

3.2.3 Control Mode: Position, Speed and Torque

The unit of position stiffness is Nm/rad, and the unit of velocity stiffness is Nm/(rad/s).

The position, speed and torque modes of the motor control are set by numerical parameters.

Torque mode: output the desired torque T, which need to disable the position and speed control loop. There are two ways, the first is to set the desired position and speed as the forbidden flag value, PosStopF = 2.146E+9f, VelStopF = 16000.0f; the second is to set the position stiffness and velocity stiffness to zero. The first is a little faster than the second, and they can be used compound:

```
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;
```

Speed mode: output expected speed V, which speed stiffness can not be set to zero and the position loop should be set to forbidden flag. The expected torque should be set to zero too:

```
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;
```

Position mode: output desired position P, which position and velocity stiffness can not be set to zero. At this time, the expected velocity should be set to zero as the pure damping term, and the expected torque should be set to zero:

```
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;
```

Compound mode: three modes can be combined as needed that two or three loops can run at the same time. For example, position loop and torque loop are running together:

```
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 Examples

See https://github.com/unitreerobotics/unitree_legged_sdk/tree/master/examples, Please watch out that when executing, it involves system calls, you need to add **sudo** permissions.

Among these examples, example_walk is a high level control, and example_position, example_velocity, and example_torque are all low level controls.

4 A1_ros

Detailed at https://github.com/unitreerobotics/a1_ros

We provide ROS interface to control virtual robots. Place the "A1_ros" folder under catkin workspace(normally path: ~/catkin_ws/src/unitree_ros), then compile:

```
cd ~/catkin_ws
catkin_make
```

The environment we use: Ubuntu16.04 + ROS Kinetic or Ubuntu18.04 + ROS Melodic.

4.1 ros dependency

4.1.1 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 a1_msgs folder, and the content is consistent with the communication architecture used in the A1 API. You need to compile a1_msgs with catkin_make first, otherwise you may have dependency problems (such as that you cannot find the header file).

4.1.2 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”.

4.2 Rviz visualization

The robot description file is Xacro format, which is simpler than URDF format. It contains details of robot joint limitation, collision space, kinematics and dynamics parameters. The collision space uses simple geometry to improve performance of the physical engine. Rviz is only used for visualization, and the range of motion of each joint can be checked by joint_state_publisher. Usage method: https://github.com/unitreerobotics/a1_ros/a1_rviz

4.3 Gazebo dynamic simulation

Detailed in https://github.com/unitreerobotics/a1_ros

4.3.1 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.

1. 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.

2. 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.

4.3.2 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_legged_gazebo a1_normal.launch
(if can't launch, run:
cd ~/catkin_ws/src/unitree_legged_ros/unitree_legged_gazebo/launch/
roslaunch ./a1_normal.launch
)
```

Terminal-2 will run all the nodes:

```
rosrun unitree_legged_gazebo a1_servo
```

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

```
rosrun unitree_legged_gazebo a1_external_force
```

Users can add more nodes to accomplish the target task.

4.4 ROS control robot

4.4.1 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.

4.4.2 control examples

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

Firstly, copy a1_real to ~/catkin_ws/src, and compile it:

```
cd ~/catkin_ws
catkin_make
```

Terminal-1, run 'lcm server' in sdk:

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

Terminal-2, run ros master node:

```
roscore
```

Terminal-3, run user logic with rosrun:

```
rosrun a1_real position_lcm_publisher
```

5 External Sensors

The SLAM function and related API are detailed in “The Manual about 2D-SLAM and Path Planning System Based Mapper”.