K1 Instruction Manual-V1.0

Modified October 20

> 中文版本：K1说明书-V1.0

Product Overview

Booster K1 is a humanoid robot development platform for scenarios such as competitions, education, and entertainment, featuring affordable, portable, and durable.

Product Composition

The K1 robot consists of head, torso, arms, and legs, with a total of 22 DoFs, allowing for flexible movement and posture control.

- The head has 2 DoFs, including Yaw Joint and Pitch Joint. It contains a depth camera and microphone array.
- Each arm has 4 DoFs, including Shoulder Pitch Joint, Shoulder Roll Joint, Shoulder Yaw Joint, and Elbow Joint.
- Each leg has 6 DoFs, including Hip Pitch Joint, Hip Roll Joint, Hip Yaw Joint, Knee Joint, and Ankle Up and Down Joint.
- Controller board, speaker and battery are installed in torso.

Product Functions

1. Omnidirectional Walking
   - Supports forward, backward, and lateral walking.
   - Supports rotation and complex walking.
2. Disturbance Resistance while Walking
   - Can walk on uneven surfaces.
   - Can withstand certain impact disturbances while walking.
3. Predefined Actions
   - Waving.
   - Shaking hand.
   - Fall recovery.
4. Safety Protection
   - Automatically enters damping mode in uncontrolled states to prevent damage.
   - Soft emergency stop.

Product Specifications

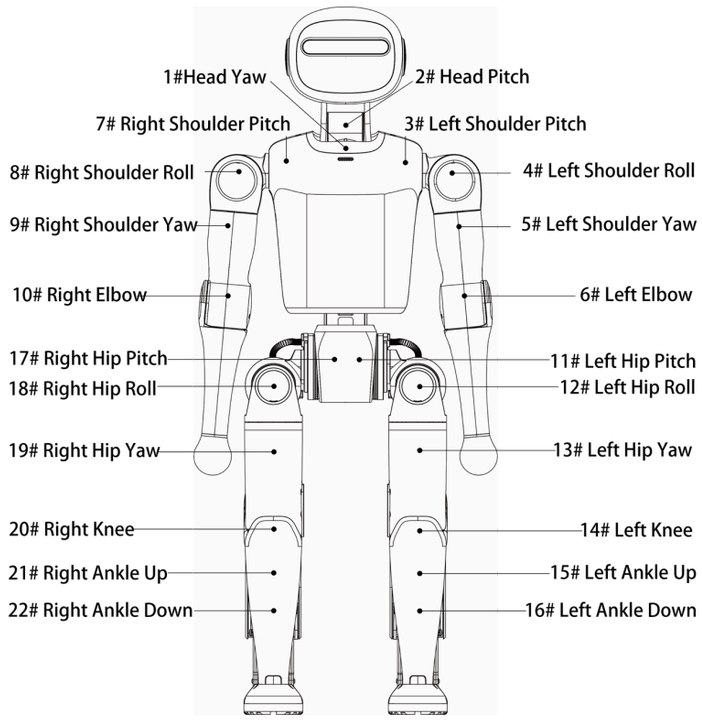
Type	Specification Parameters	Description
Basic Parameters	Height (when standing upright)	0.95m
	Weight	About 19.5kg
DoFs	Total DoFs	22
	Single Leg DoFs	6
	Single Arm DoFs	4
	Head DoFs	2
Operational Parameters	Walking Speed	
	Turning Speed	
Battery Parameters	2Ah Battery Life	50min（0.4m/s）
	5Ah Battery Life	1h20min（0.4m/s）
	Charging Time	≤1h
	Cycle Life	≥500 times
Computing Platform	Processor	Jetson Orin NX 8GB，AI Performance 117TOPS
Sensors	Camera	Depth Camera
	Microphone	Microphone Array
	Speaker	1
Interaction interface	Buttons	interaction button*3
	Indicator Light	RGB LED*1
Safety Fuction	Auditory Alerts	Low Battery Alert, Joint Overheat Alert
Communication Methods	Wired connection	Gigabit Ethernet
	Wireless Connection	WIFI 6
	Bluetooth	Bluetooth 5.2
	Certification Standards	CE/FCC
Noise	Walking Noise	≤70dB
Environmental Adaptability	Environmental Adaptability	-10℃~45℃
	Operating Humidity	5%~90%，without condensation

Main parts

Joint motors.

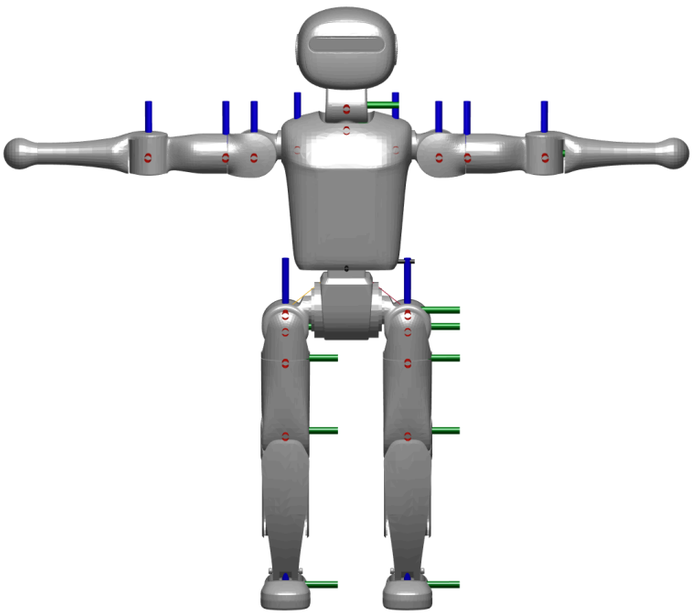
Joint ID and limits.

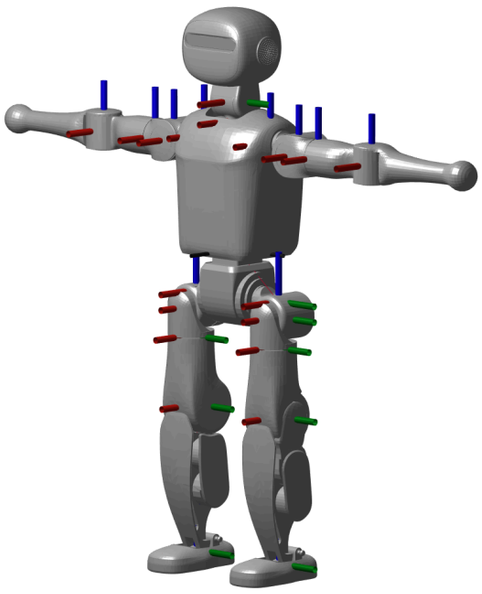
Joint ID	Joint Name	Limit (Degrees)	
		Max	Min
1	Head Yaw Joint	59	-59
2	Head Pitch Joint	43	-17
3	Left Shoulder Pitch Joint	69	-169
4	Left Shoulder Roll Joint	89	-99
5	Left Shoulder Yaw Joint	109	-109
6	Left Elbow Joint	39	-129
7	Right Shoulder Pitch Joint	69	-169
8	Right Shoulder Roll Joint	89	-99
9	Right Shoulder Yaw Joint	109	-109
10	Right Elbow Joint	129	-39
11	Left Hip Pitch Joint	126	-171
12	Left Hip Roll Joint	89	-22
13	Left Hip Yaw Joint	59	-59
14	Left Knee Joint	127	0
15	Left Ankle Up Joint	38	-17
16	Left Ankle Down Joint	41	-16
17	Right Hip Pitch Joint	126	-171
18	Right Hip Roll Joint	22	-89
19	Right Hip Yaw Joint	59	-59
20	Right Knee Joint	127	0
21	Right Ankle Up Joint	38	-17
22	Right Ankle Down Joint	41	-16



Coordinate System

The joint coordinate system with all joints at zero position is shown in the diagram below.





Controller

	Motion control board
Processor	Jetson Orin NX 8GB
Computing performance	6-core Cortex-A78AE CPU@2GHz Tensor Cores GPU@1173MHz AI performance: 117 TOPS
Memory	16GB
Storage	512GB
Wired Network	1000M*1
Wireless Network	WIFI6*1
Audio	Microphone, speaker

Operation Manual

K1 is packed with an out-of-the box motion control program. Follow the instructions below to control K1 remotely.

WARNINGS

1. K1 must first enter PREP mode, and then **put to a stable standing position on the ground**, before switching to WALK mode.
2. **DO NOT lift K1 while under WALK mode.**
3. **Make sure to clear any obstacles on the ground and avoid human injuries while operating K1.**
4. **DO NOT touch any parts of K1 while under WALK mode, except for the handle.**
5. **Make sure to remove ALL zero parts before restarting K1.**

MODES

- K1 operates under **Modes**.
- K1 can perform different **Actions** under different **Modes**.
- **Modes** can switch to one another but **with constraints**. For example, DAMP Mode can only switch to WALK mode by first entering PREP Mode.

DAMP Mode

- Power is on and motion control is working.
- All joints are in a dampening state, i.e, joints will resist position change, but will neither try to hold its position nor actively change its position.
- K1 is NOT able to stand under DAMP mode, therefore support is needed.
- DAMP mode is a safe mode, entering DAMP mode can protect K1 and its operator.
- DAMP mode can be switched to PREP mode, but not directly to WALK mode.

PREP Mode

- Power is on and motion control is working.

- K1 assumes a standing posture and holds it. Joints will strongly resist position change, and try to go back to the standing posture if forcefully moved.
- K1 under PREP mode can stand on its own on the ground. Under PREP mode K1 can be placed to stand on the ground. However, it will NOT try to balance itself.
- K1 can swtich to ALL other modes under PREP mode, including DAMP and WALK.

WALK Mode

- Power is on and motion control is working.
- Under WALK mode, K1 can perform various predefined actions, including omni-walk, rotating, stepping, standing-still and moving head.
- Compared to PREP mode, WALK mode is more resilient, and will try to recover balance if pushed.
- K1 under WALK mode can switch to all other modes, including DAMP and PREP.
- **NOTICE: Make sure K1 is under PREP mode and is already standing firmly on the ground, before switch to WALK mode.**

CUSTOM Mode

- Power is on and motion control is working.
- **K1 gives up control on all joints to developer, who controls K1 through its SDK. Use caution under CUSTOM mode to avoid damage to K1.**
- Only PREP and DAMP modes can switch to CUSTOM mode.
- CUSTOM mode can only switch to PREP or DAMP mode.

> 💡
> - While developing new tricks on K1, it is recommended to use a Hoist at all times under CUSTOM mode.

PROTECT Mode

- PROTECT mode will automatically kickin on errors (i.e, exceeding joint limit or falling).
- Joints under PROTECT Mode behave the same as DAMP mode.
- You can try to reenter DAMP mode under PROTECT mode. (Soft restart)
- PROTECT mode is a safe mode, entering PROTECT mode can protect K1 and its operator.

Powering on

1. Place K1 on its rest.
2. Install the battery pack. Slide the pack into the battery socket, with lights facing outwards.
3. Place K1's hands and legs in a natural posture.
4. Press Power button about 3s（release it after light turns on；press more than 6s, the robot will be powered off），the robot is powered on. Wait for about one minute, the robot will play a prompt tone. Then the robot can be remotely controlled. **NOTE: The initialization of IMU requires that K1 remains stationary during the booting process.**
5. Press **LT+START** on the joystick to enter PREP mode, after which K1 can be placed on the ground, and put in a standing position.
6. Press **LT + RT + A** to enter WALK mode, after which K1 will response to walking commands. **NOTE: DO NOT try to lift K1 while under WALK mode.**

Shutting down

1. Press **LT+START** on the joystick to enter PREP mode, after which K1 can be placed flat on the ground.
2. Press **LT+BACK** to enter DAMP mode, and then press the power button to shut down K1.

> 💡
> - After powering off, you need to wait for 6 seconds before you can power it on again

Joystick control

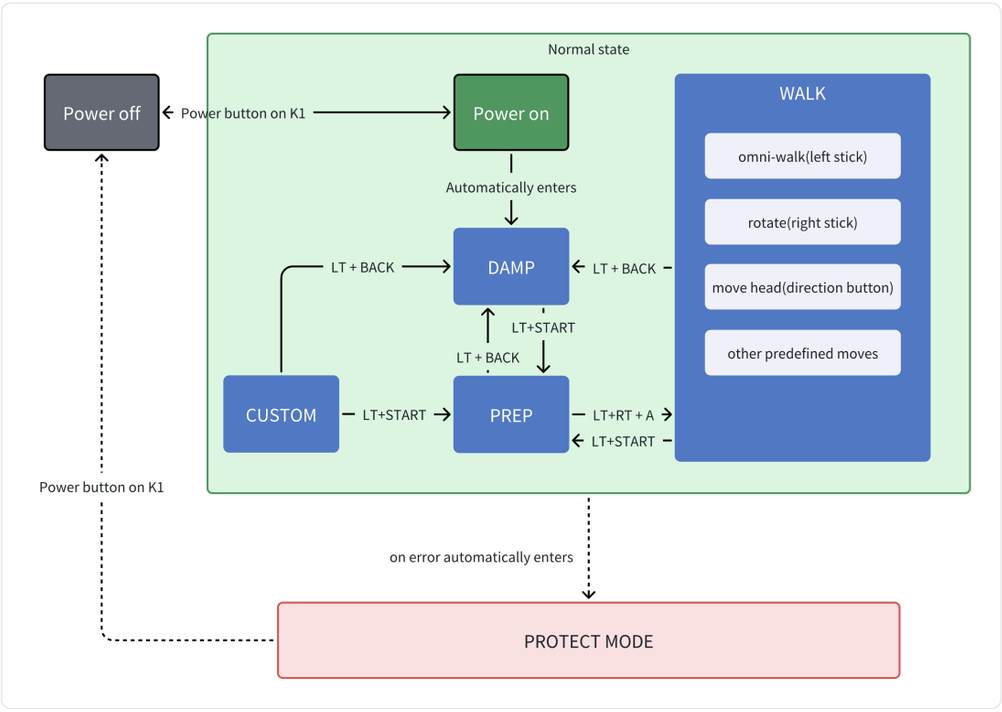
Keymap

K1 is equiped with a XBOX-compatible joystick. **Make sure control mode is Receiver Mode, with 3 LEDs ON.**

Function		Button	Description
Damping Mode	Enter damping mode	LT + BACK	Cannot directly switch from damping mode to any walking mode

Preparation Mode	Enter preparation mode	LT + START	
Walking Mode	Enter walking mode	LT + RT + A	
	Get up and enter walking mode.	RB + UP	It can only be triggered in the lying position in damping or preparation mode. After getting up, it will automatically enter walking mode
	Locomotion control	Left stick	
	Rotation control	Right stick	
	Head rotation	Direction buttons	
	Start/end handshake	A	
	Start/end wave hand	B	

State map



Back Panel Buttons

- In addition to the power button, there are three other buttons on the back of K1, namely: WLAK/STAND/F1.
- The WLAK button is used to enter the walking mode, and the STAND button is used to enter the ready mode.
- The F1 button allows users to customize its function through the configuration file /opt/booster/Gait/configs/K1/task_instruction.yaml. By default, it is used to enter the damping mode.

Connect to Robot

Connect via App

1. Download and install app.

Connect via Terminal

Wired Connection

1. Connect via Ehternet, and configs the wired network in manual mode as following

Code block

```
address: 192.168.10.10
netmask: 255.255.255.0
gateway: 192.168.10.1
```

Static IP Configuration Steps Reference：

- Mac: https://www.macinstruct.com/tutorials/how-to-set-a-static-ip-address-on-a-mac/
- Windows: https://www.trendnet.com/press/resource-library/how-to-set-static-ip-address
- Linux: https://www.freecodecamp.org/news/setting-a-static-ip-in-ubuntu-linux-ip-address-tutorial/

2. Connect the development machine to the robot using an Ethernet cable. The robot's network port is shown in the image.

3. Log in to the robot using SSH

> Open the command - line tool (Terminal) and enter the following command.

Code block

```
# Log in to K1 via Ethernet cable
ssh booster@192.168.10.102
# Initial password: 123456
```

Wireless connection

1. Configure robot's wireless connection via App. Find wireless ip of robot in app, eg `xxx.xxx.xxx.xxx`
2. Log in to the robot using SSH

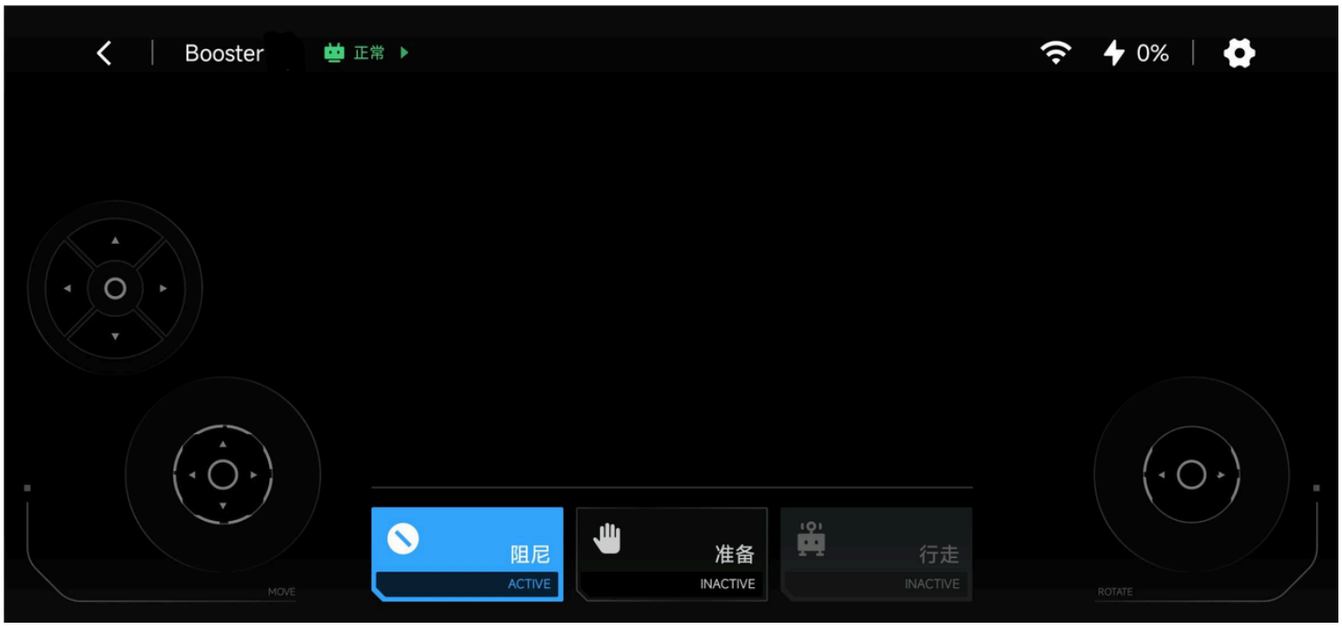
Code block

```
# Log in to K1 via Wi-Fi connection
ssh booster@xxx.xxx.xxx.xxx
# Initial password: 123456
```

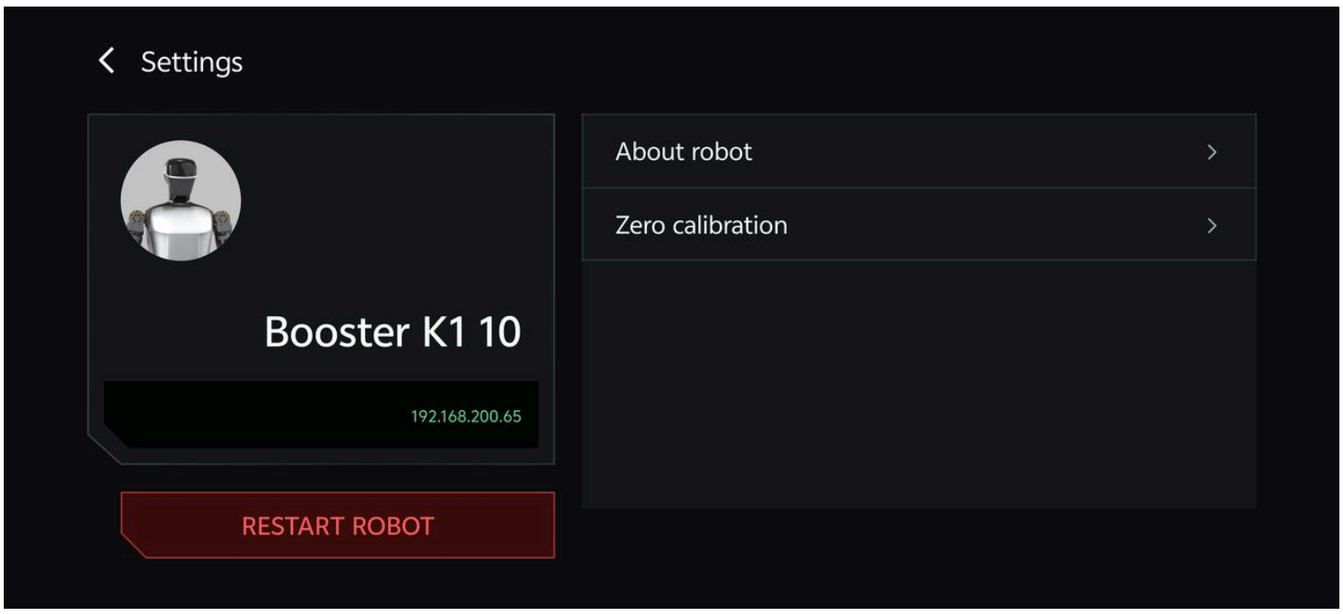
Robot Control Service Start/stop

Via App (Android only)

1. Click the gear icon in the upper right corner on the main page to enter the settings page.



2. Click "Restart the robot" in the lower left corner of the settings page.



Via Terminal

1. Connect to robot
2. Execute comand below

Code block

```
# start robot control service
booster-cli launch -c start

# stop robot control service
booster-cli launch -c stop

# restart robot control service
booster-cli launch -c restart
```

Charging

1. Insert the charging plug according to the indicated direction.

2. The charging status is shown as follows: the green light indicates a full charge, and the red light indicates charging in progress.

Help

Our company will, without violating any applicable laws and regulations or involving any personal privacy information, collect only data related to the operation of the robots for the purposes of fault detection and relevant statistical analysis.

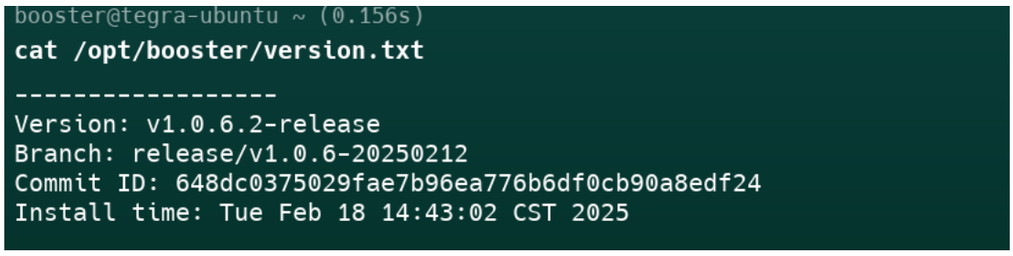
Check Version

After connecting to the robot's board, run the following command to view the current system version of the robot.

Code block

```
cat /opt/booster/version.txt
```

example:



Software Upgrade

> ! When installing a new version of software, the robot's motion control program will be stopped, and the robot's joints will not exert force. **Before upgrading, ensure the robot is in damping mode or motion control is stopped and has good support (e.g., using a stand to support the robot).**

Upgrade with package

> For historical versions, please refer to K1 Version History

Download the latest installation package of the robot software.

Release Date	Software Version	Software Package Download Link	Update Content
2025.08.29	v1.2.1.9	https://obs-cdn.boosterobotics.com/ota_single/v1.2.1.9-release-single-aarch64.run	• optimize gait stability • optimize camera frame rate

Upgrade K1 Software

First, set the file execution permission:

Code block

```
sudo chmod +x v1.0.1.30-release-single-aarch64.run
```

Copy the software upgrade package to the K1 board (can be copied to `/home/booster/Downloads`), and execute:

Code block

```
./v1.0.1.30-release-single-aarch64.run
```

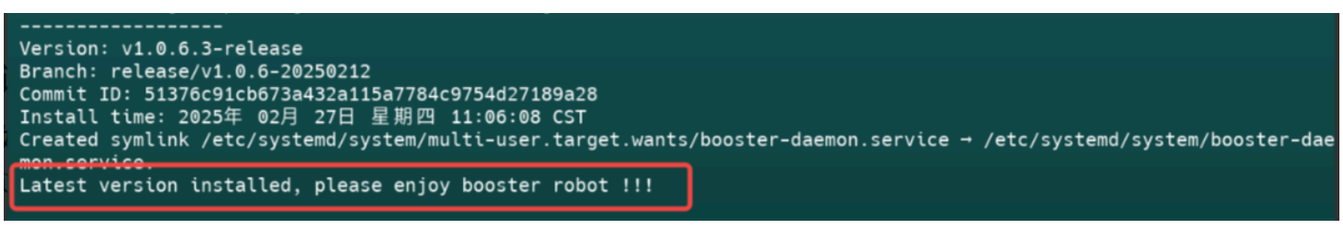
Upgrade with command-line

Run this command in K1 board

Code block

```
booster-cli upgrade
```

The appearance of the following printed content indicates that the system update has been completed



Restore Factory Settings

Introduction

If the robot experiences issues due to configuration changes, try restoring factory settings.

Procedure

- navigate to `/home/booster/Documents/recovery` and run:

```
Code block
1  cd /home/booster/Documents/recovery
2  ./v1.0.1.30-release-single-aarch64.run
```

Log Retrieval

Introduction

When the robot has issues, tech support may need to obtain the robot's operation logs. Here's how to retrieve logs and send them to support.

Procedure

1. Log into the robot via terminal
2. Run the command to get the log compressed file:

> Please note that the robot's default time zone is UTC+8. If you are in a different time zone, you can either modify the time zone in the robot's operating system or use the converted time to access the logs

```
Code block
1  # run this command in terminal
2  # Usage:
3  #   -st --start-time [TIME]: Filter logs starting from a specific time (format:
4  #   -et --end-time [TIME]: Filter logs ending before a specific time (format: YY
5  #   -o, --output [FILE]: Compress logs to a specified output file, default is /h
6  booster-cli log -t YYYYMMDD-HHMMSS -o OUTPUT_PATH
```

Assuming the issue with the robot occurred around 20200808-120810, you can select a time range of approximately ten minutes before and after this point as the start and end times for retrieving the logs, to ensure that the log package covers the time of the issue. Run the following command:

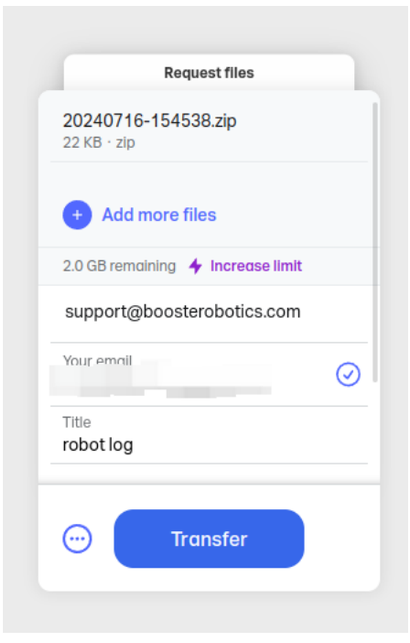
```
Code block
1  booster-cli log -st 20200808-120800 -et 20200808-120820 -o /home/booster/Documer
```

After running, a file like `20200808-120800.zip` will be generated in `/home/booster/Documents`.

3. Send the generated file to technical support via WeChat or Feishu.

```
Code block
1  # To copy the file from the developer host (assuming the log file is in /home/bo
2  scp booster@192.168.10.102:/home/booster/Documents/20200808-120800.zip ~/Downloa
3
4  # Then send the log file to technical support.
```

4. If overseas users are unable to send log files via the chat tool, you can choose to use WeTransfer to send the log files.
   - Upload the log file, then enter support@boosterobotics.com in the 'Email to' field, and click 'Transfer' to send.



Remote Support

Introduction

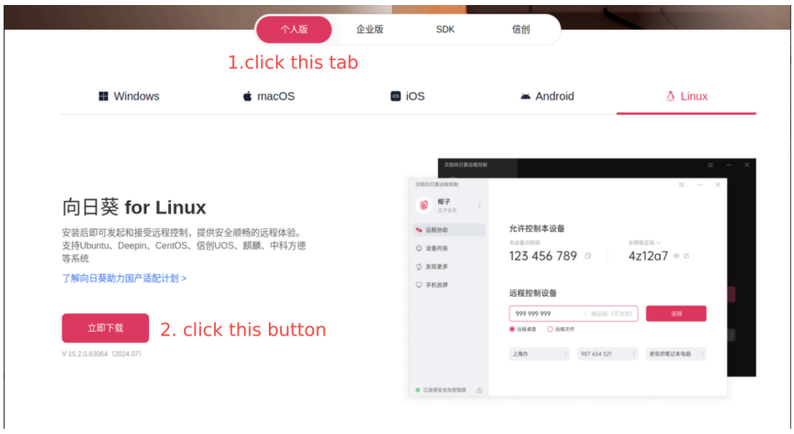
When the robot has issues and you need remote assistance, follow these steps to get support.

Procedure

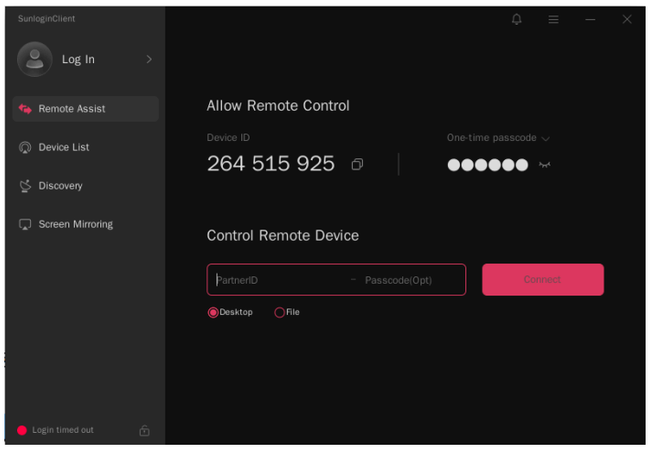
💡 You can choose from the following two remote assistance methods:

1. Connect your personal computer to the robot via SSH; the support tech support can then connect to your computer via remote desktop and subsequently to the robot.
2. Install remote desktop software on the robot; you can use a display and keyboard/mouse to open the remote desktop software, allowing support personnel to connect directly to the robot.

1. Download the remote desktop software: https://sunlogin.oray.com/download
   - Choose the personal edition
   - Select the corresponding operating system based on your device and then click download



2. Install the remote desktop software and provide your Device ID and one-time password to tech support.
   a. After installation, the software interface will appear as follows:



   b. Click the eye icon below 'One-time passcode' to display the one-time password. Then, copy the 'Device ID' and 'One-time passcode' and provide them to our technical support team.
3. Once done, tech support can connect to the robot via remote desktop and begin troubleshooting.

Mobile App

Introduction

The Booster robot can connect to a mobile app for control, status feedback, Bluetooth configuration, and other features.

Download Link (iOS not yet supported):

- Android

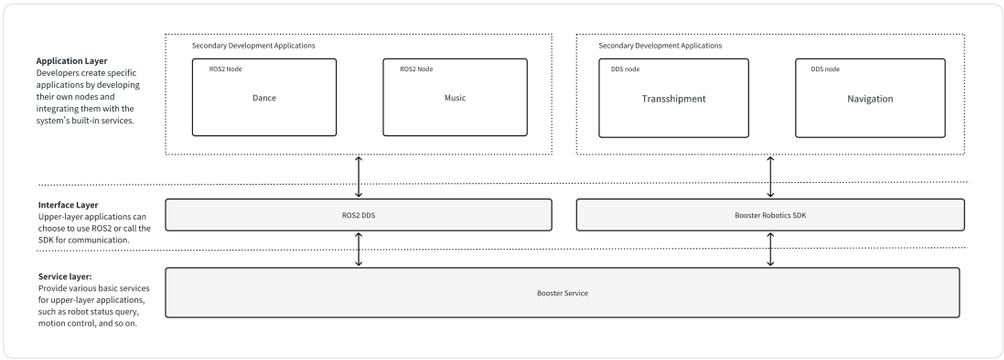
booster_1.0.4.apk
56.46MB

Application Development

SDK

SDK Overview

The Booster Robotics SDK supports developers in secondary development based on Booster robots. The SDK uses Fast-DDS as the message middleware, compatible with the communication mechanism used in ROS2, allowing mutual communication.



Service Introduction

The Booster system exposes two service levels to developers:

- **High-level services**: For controlling high-level robot movements, such as state switching, omnidirectional walking, special actions, and head control. High-level interfaces are called via RPC.
- **Low-level services**: For real-time sensor data acquisition, mainly including motors and IMU, and supporting direct motor control. Low-level interfaces utilize DDS's Pub/Sub model for calls.

High level service interface

> High-level services are available as RPC interfaces

Name	ChangeMode
Effective Version	>= v1.0.0
Definition	int32_t ChangeMode(RobotMode mode)
Description	Set the status of the robot. There are four supported modes: damping mode, preparation mode, walking mode and custom mode.

Parameters	RobotMode {
	kDamping = 0, // Damping mode. This is a safety mode that can make the joints decelerate as soon as possible and has the highest priority.
	kPrepare = 1, // Preparation mode. The robot can enter the walking mode only after entering the preparation mode.
	kWalking = 2, // Walking mode. In walking mode, instructions such as forward, backward, and turning can be sent to the robot.
	kCustom = 3. // Custom mode. The robot accepts user-defined joint instruction movements.
	};

Return Value	If the call is successful, 0 is returned; otherwise, the relevant error code is returned.

Name	GetMode
Effective Version	>= v1.2.0.2
Definition	int32_t GetMode(GetModeResponse &get_mode_response)
Description	Get the mode of the robot

Parameters	RobotMode {
	kDamping = 0, // Damping mode. This is a safety mode that can make the joints decelerate as soon as possible and has the highest priority.
	kPrepare = 1, // Preparation mode. The robot can enter the walking mode only after entering the preparation mode.
	kWalking = 2, // Walking mode. In walking mode, instructions such as forward, backward, and turning can be sent to the robot.
	kCustom = 3. // Custom mode. The robot accepts user-defined joint instruction movements.
	};

Return Value	If the call is successful, 0 is returned; otherwise, the relevant error code is returned.

Name	Move	
Effective Version	>= v1.0.0	
Definition	int32_t Move(float vx, float vy, float vyaw)	
Description	Send a speed command	
Parameters	vx	Forward and backward movement speed. Forward is positive. There are different ranges for different gaits. Adjustments are recommended when |vx| > 0.5, and the unit is m/s.
	vy	Left and right movement speed. Left is positive. There are different ranges for different gaits. Adjustments are recommended when |vy| > 0.5, and the unit is m/s.
	vyaw	Rotational angular velocity. Counterclockwise is positive. There are different ranges for different gaits. Adjustments are recommended when |vyaw| > 1, and the unit is rad/s. When the parameter is out of range, it will continue to run with the boundary value.
Return Value	If the call is successful, 0 is returned; otherwise, the relevant error code is returned.	

Name	RotateHead	
Effective Version	>= v1.0.0	
Definition	int32_t RotateHead(float pitch, float yaw)	
Description	Send a head control instruction. The maximum speed of head rotation is 20 rad/s.	
Parameters	pitch	The movement angle in the up and down direction. Downward is positive. The unit is rad. The range is -0.3 to 1 rad.
	yaw	The movement angle in the left and right direction. Leftward is positive. The unit is rad. The range is -0.785 rad to 0.785 rad.
Return Value	If the call is successful, 0 is returned; otherwise, the relevant error code is returned.	

Name	WaveHand
Effective Version	>= v1.0.2
Definition	int32_t WaveHand(HandAction action);
Description	Used to perform a wave action or stop waving in walk mode.
Parameters	Hand movement parameters support kHandOpen and kHandClose.
Return Value	If the call is successful, 0 is returned; otherwise, the relevant error code is returned.

Name	Handshake	
Effective Version	>= v1.0.6.0	
Definition	int32_t Handshake(HandAction action)	
Description	Used to perform a handshake action or stop handshaking in walk mode.	
Parameters	action	kHandOpen：start handshake action，kHandClose：stop handshake action
Return Value	If the call is successful, 0 is returned; otherwise, the relevant error code is returned	

Name	GetFrameTransform	
Effective Version	>= v1.0.5.0	
Definition	int32_t GetFrameTransform(Frame src, Frame dst, Transform &transform)	
Description	Get the transformation between different frames.	
Parameters	src	Source frame，options: kBody，kHead，kLeftHand，kRightHand，kLeftFoot, kRightFoot
	dst	Destination frame，options: kBody，kHead，kLeftHand，kRightHand，kLeftFoot, kRightFoot
	transform	The computed transformation matrix, which is passed into the function by reference
Return Value	If the call is successful, 0 is returned; otherwise, the relevant error code is returned	

Error code

Error code	Enumeration value	Meaning of error code
100	kRpcStatusCodeTimeout	Request timeout.
400	kRpcStatusCodeBadRequest	Bad request. Usually caused by incorrect request parameters.
500	kRpcStatusCodeInternalServerError	Internal error.
501	kRpcStatusCodeServerRefused	Request rejected.
502	kRpcStatusCodeStateTransitionFailed	Robot state transition error. Usually it is an illegal state transition or the previous state transition is in progress.

Low level service interface.

> The low level service interface is called in a ROS-like Publish/Subscribe manner.
>
> Please note that the low level publish interface will only take effect when the robot is in custom mode. To enter this mode, please refer to the `ChangeMode` interface in the high-level API.

Topic	rt/low_state
Effective version	>= v1.0.0.0
Access method	subscribe
Interface description	Obtain the robot's IMU and joint feedback in real time.

Interface structure	struct LowState {
	  ImuState imu_state; // IMU feedback.
	  sequence<MotorState> motor_state_parallel; // Parallel structure joint feedback.
	  sequence<MotorState> motor_state_serial; // Serial structure joint feedback.
	};
	
	struct ImuState {
	  float rpy[3]; // Euler angle information（0 -> roll ,1 -> pitch ,2 -> yaw）
	  float gyro[3]; // Angular velocity information（0 -> x ,1 -> y ,2 -> z）
	  float acc[3]; // Acceleration information.（0 -> x ,1 -> y ,2 -> z）
	};
	
	struct MotorState {
	  float q; // Joint angle position, unit: rad.
	  float dq; // Joint angular velocity, unit: rad/s.
	  float ddq; // Joint angular acceleration, unit: rad/s².
	  float tau_est; // Joint torque, unit: nm
	};

Remarks	Since some simulation tools (such as Isaac Sim) only support serial structures, while our robot has a parallel structure. Therefore, in the interface, we provide two types of joint feedback results for serial and parallel structures. The system will complete the conversion between serial and parallel data internally. Developers can use one of the feedbacks according to their needs, but they need to ensure that the serial-parallel consistency is also maintained when sending underlying instructions.

Topic	rt/joint_ctrl
Effective version	>= v1.0.0.0
Access method	publish
Interface description	Publish the joint commands of the robot to control the motors.

Interface structure	enum CmdType {
	  PARALLEL, // Parallel type.
	  SERIAL // Serial type.
	};
	
	struct LowCmd {
	  CmdType cmd_type; // Set whether the joint command follows the serial mode or the parallel mode.
	  sequence<MotorCmd> motor_cmd; // Joint command array.
	};
	struct MotorCmd {
	  float q; // Joint angle position, unit: rad.
	  float dq; // Joint angular velocity, unit: rad/s.
	  float tau; // Joint torque, unit: nm
	  float kp; // Proportional coefficient.
	  float kd; // Gain coefficient.
	  float weight; // Weight, range [0, 1], specify the proportion of user set motor cmd is mixed with the original cmd sent by the internal controller, which is usually used for gradually move to a user custom motor state from internal controlled motor state. Weight 0 means fully controlled by internal controller, weight 1 means fully controlled by user sent cmds. This parameter is not working if in custom mode, as in custom mode, internal controller will send no motor cmds.
	};

Remarks	Since some simulation tools (such as Isaac Sim) only support serial structures, while our robot has a parallel structure. Therefore, in the interface, we simultaneously provide two types of joint commands for serial and parallel structures. The system will complete the conversion between serial and parallel data internally. Developers can use one of the feedbacks according to their needs, but they need to ensure that the serial-parallel consistency is also maintained when receiving underlying feedback.

Topic	rt/fall_down
Effective version	>= v1.2.0.2
Access method	subscribe
Interface description	Real-time detection of robot falls

Interface structure	module booster_interface {
	  module msg {
	  enum FallDownStateType {
	  IS_READY, // Not fallen state
	  IS_FALLING, // Currently falling
	  HAS_FALLEN, // Already fallen
	  IS_GETTING_UP, // Currently getting up
	  };
	
	  struct FallDownState {
	  FallDownStateType fall_down_state;
	  boolean is_recovery_available; // Whether recovery (getting up) action is available
	  };
	  };
	};

Remarks	This interface can be used in conjunction with the GetUp (self-righting) interface from the high-level API to enable automatic standing after detecting a fall. Important Notes: 1. The FallDownStateType determination is based on the robot's IMU data. 2. During a fall, sudden collisions may cause the robot to bounce, temporarily elevating the IMU position. - This may trigger 1-2 frames of false IS_GETTING_UP detection 3. Recommended to use in combination with is_recovery_available for accurate state judgment

Topic	rt/button_event
Effective version	>= v1.2.0.2
Access method	subscribe
Interface description	Real-time retrieval of backboard button inputs

Interface structure	module booster_interface {
	  module msg {
	  enum ButtonEventType {
	  PRESS_DOWN,
	  PRESS_UP,
	  SINGLE_CLICK,
	  DOUBLE_CLICK,
	  TRIPLE_CLICK,
	  LONG_PRESS_START,
	  LONG_PRESS_HOLD,
	  LONG_PRESS_END
	  };
	
	  struct ButtonEventMsg {
	  long button;
	  ButtonEventType event;
	  };
	  };
	};

Remarks	This feature can be used in user programs to implement custom backboard button functionality.

Topic	rt/remote_controller_state
Effective version	>= v1.2.0.2
Access method	subscribe
Interface description	Real-time retrieval of remote controller button inputs

Interface structure	module booster_interface {
	  module msg {
	  struct RemoteControllerState {
	  unsigned long event; // refer to remarks
	
	  float lx; // left stick horizontal direction, push left to -1, push right to 1
	  float ly; // left stick vertical direction, push front to -1, push back to 1
	  float rx; // right stick horizontal direction, push left to -1, push right to 1
	  float ry; // right stick vertical direction, push front to -1, push back to 1
	
	  boolean a;
	  boolean b;
	  boolean x;
	  boolean y;
	  boolean lb;
	  boolean rb;
	  boolean lt;
	  boolean rt;
	  boolean ls;
	  boolean rs;
	  boolean back;
	  boolean start;
	
	  boolean hat_c; // Hat centered
	  boolean hat_u; // Hat up
	  boolean hat_d; // Hat down
	  boolean hat_l; // Hat left
	  boolean hat_r; // Hat right
	  boolean hat_lu; // Hat left up
	  boolean hat_ld; // Hat left down
	  boolean hat_ru; // Hat right up
	  boolean hat_rd; // Hat right down
	  uint8 reserved;
	  };
	  };
	};

Remarks	This feature can be used in user programs to implement custom gamepad/controller button functionality.
	event include:
	  NONE = 0, // no event
	  AXIS = 0x600, // axis motion
	  HAT = 0x602, // hat position change
	  BUTTON_DOWN = 0x603, // button pressed
	  BUTTON_UP = 0x604, // button released
	  REMOVE = 0x606 // device has been removed

Topic	rt/tf
Effective version	>= v1.1.0.6
Access method	subscribe
Interface description	Real-time acquisition of coordinate transformations between robot joints
Interface structure	https://docs.ros2.org/foxy/api/tf2_msgs/msg/TFMessage.html
Remarks	1. This interface currently only supports subscription via ROS2 2.If using ros2 topic echo /tf fails to display the message, it may be due to an incomplete installation of the xacro package caused by network issues during a software upgrade. You can manually install it with: **sudo apt install ros-humble-xacro**.Then restart your machine and attempt to subscribe to the topic again.

Visual Interface

- K1 adopts Zed as perception sensor. Zed ROS wrapper node will be launched automatically when robot is powered on. Info related to published topics and services can be found here https://www.stereolabs.com/docs/ros2/zed-node

Getting Started

Getting SDK

- Download Link: https://github.com/BoosterRobotics/booster_robotics_sdk
- Follow the README in the repository to complete the SDK installation on the developer's computer.

Install SDK

Code block

```
# In the booster_robotics_sdk directory
sudo ./install.sh
```

Compile Sample Programs and Install Python SDK

> Assuming the SDK is installed at `/home/booster/Workspace`

navigate to the SDK project path and run

Code block

```
# you should first install dependencies by
# pip3 install pybind11
# pip3 install pybind11-stubgen

cd /home/booster/Workspace/booster_robotics_sdk
mkdir build
cd build
cmake .. -DBUILD_PYTHON_BINDING=on
make
sudo make install
```

Compile Only Sample Programs

> Assuming the SDK is installed at `/home/booster/Workspace`

navigate to the SDK project path and run

Code block

```
cd /home/booster/Workspace/booster_robotics_sdk
mkdir build
cd build
cmake ..
make
```

After compiling, the sample programs will be generated in the build directory, including the `b1_loco_example_client` program as a high-level service interface example.

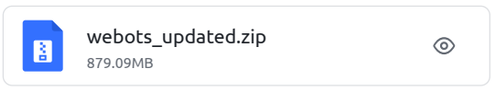
Example: Development Based on Webots Simulation

Preparation Work: Environment Installation

1. System Requirements

Element	Good Spec
OS	Ubuntu 22.04
CPU	Intel Core i7 (7th Generation)
Cores	4
RAM	16 GB

2. Webots Installation



   a. Unzip and copy Webots to the `/usr/local` directory.
   b. sudo cp -r webots/ /usr/local/
   c. Configure the environment variable (the path should match the installation location).

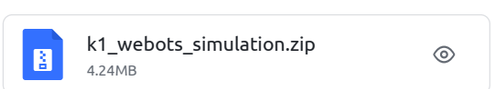
Code block

```
# ~/.bashrc

export WEBOTS_HOME=/usr/local/webots
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$WEBOTS_HOME/lib/controller
```

Development in Webots Simulation Environment

1. Load Webots world files.



   a. Unzip Webots simulation files.
   b. Open in Webots:
      - File -> Open world -> Select `.wbt` file.

2. Install dependency

Code block

```
"cmake"
"ninja-build"
"libgtest-dev"
"libgoogle-glog-dev"
"libboost-dev"
```

```
"libeigen3-dev"
"liblua5.3-dev"
"graphviz"
"libgraphviz-dev"
"python3-pip"
"libcurl4-openssl-dev"
"libsdl2-dev"
"joystick"
"libspdlog-dev"
```

3. Run the simulation control program.

booster-runner-full-webots-k1-0.0.1.run
196.50MB

4. In the Shell, ./booster-runner-full-webots-k1-0.0.x.run
   a. If it doesn't run, use `chmod +x booster-runner-full-webots-k1-0.0.x.run`

ROS2 Dev Guide

> For the secondary development process based on ROS2, please refer to the following manual

Development Guide on ROS2

Resource Download

License

Code block

```
Copyright [2024] [Booster Robotics Technology Co., Ltd ("Booster Robotics")]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

K1 Robot URDF/MJCF for RL training

K1.zip
6.04MB

K1 USD for isaac sim

k1.zip
4.81MB