Exo-H3 on ROS

The exoskeleton H3 (Exo-H3) is a lower limb rehabilitation robot for patients with limb movement disorders. It is actuated in the sagittal plane with six Degrees of Freedom (DoF): hip, knee, and ankle joints. Besides, the height of the robot is quickly adjusted to different people through sliding extender bars.

Each robot has a Main Controller (MC), which allows a centralized control of the device, and six Joint Controllers (JC), which perform a dedicated control for each joint. Both controllers create a closed-loop control where the gait algorithms are deployed. The algorithms allow, among many other things, to modify the speed and the assistance level of the gait at runtime. The main Exo-H3 characteristics are listed below.

- Security mechanisms as power shut-off button and mechanical stops.
- Wider area for hips attachment
- Wider mobility ranges for a more human like performance
- Stainless Steel and High Resistance Aluminum (7075)
- Subject's height adaptability:
- Minimum 110 cm
- Maximum 210 cm
- Subject's weight limits:
- Minimum 40 Kg
- Maximum 100 Kg
- 6 Actuated DoF in Sagittal Plane

Joint	Flex.	Ext.	RoM
Hip	105°	30°	135°
Knee	105°	5°	110°
Ankle	30°	30°	60°

- Sensors
  - Joints: position, velocity, motor torque, and interaction torque
  - Insoles: pressure sensor
- Closed-loop control in Real Time
  - Position
  - Torque
  - Stiffness
- Communication interfaces
  - External CAN-Bus
  - Bluetooth v3.0 (Class 2)
  - 2.4 GHz Transceiver

Release date: 09/06/21

Version 0.0.1

www.technaid.com – support@technaid.com

- Wi-Fi 2.4 GHz IEEE 802.11 b/g/n

Table of Contents

List of images

List of tables

1. Hardware – General electronic architecture

In the Exo-H3, the MC is the brain of the device. On the one hand, it is the interface in charge of receiving all the sensory information from each joint, and then creating closed-loop controls of position, torque, and stiffness. On the other hand, the MC is also the interface with external devices using the available communication protocols such as CAN bus, Bluetooth, Wireless 2.4 GHz, and Wi-Fi.

Each joint is provided with a JC, responsible for managing the delivered power to the motors, and acquiring data from the joint sensor: angular position, joint torque, joint velocity, motor torque, and foot pressure sensors.

Finally, an Onboard PC running ROS (Robot Operating System) is connected to the MC through the external CAN interface. The Onboard PC runs a ROS node reading the data from the sensors, available in the MC, and publishing them in a network. Besides, the node can be subscribed to a topic, obtaining control commands for H3.

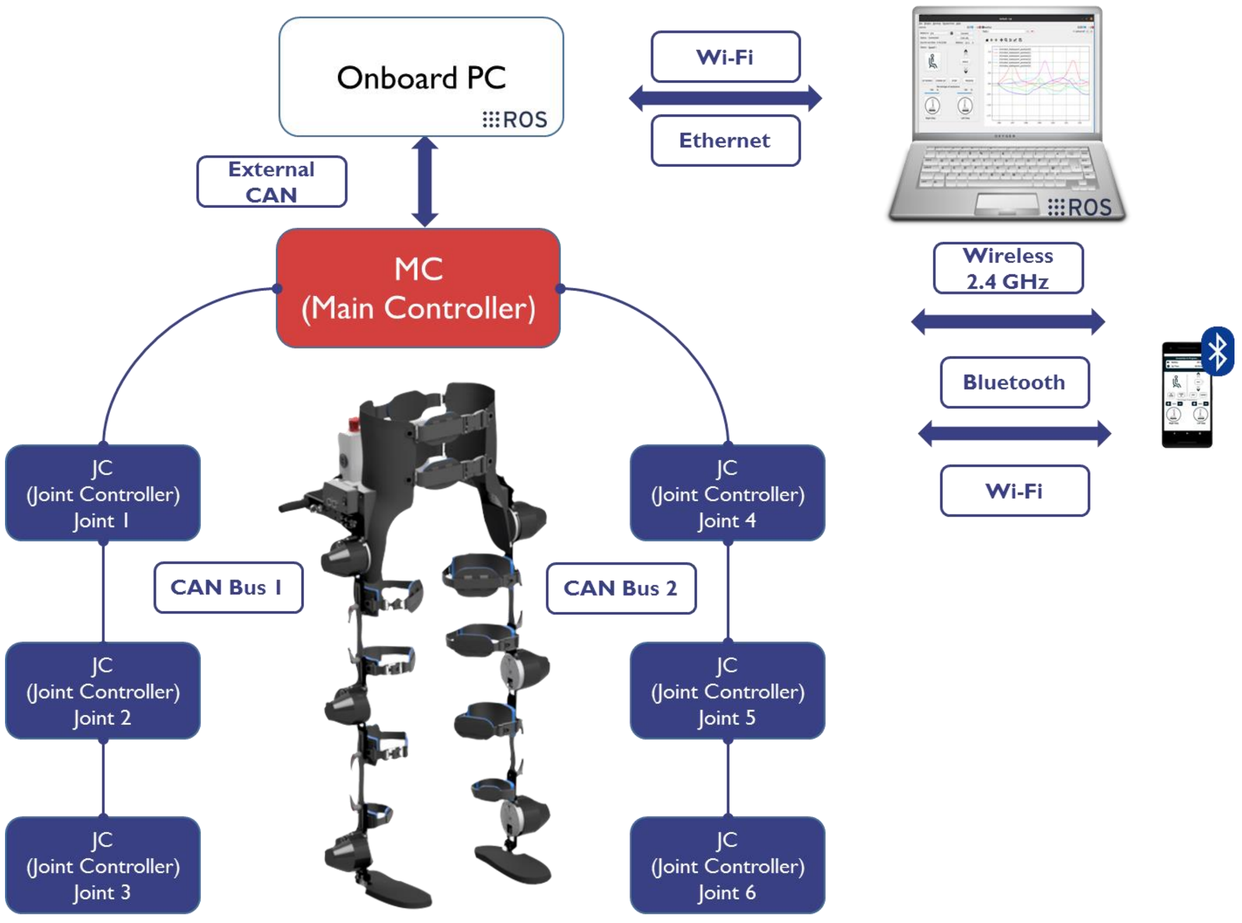

Onboard PC
ROS
Wi-Fi
Ethernet
ROS
External CAN
MC (Main Controller)
Wireless 2.4 GHz
Bluetooth
Wi-Fi
JC (Joint Controller) Joint 1
JC (Joint Controller) Joint 4
CAN Bus 1
CAN Bus 2
JC (Joint Controller) Joint 2
JC (Joint Controller) Joint 5
JC (Joint Controller) Joint 3
JC (Joint Controller) Joint 6

Image 1. General electronic architecture of Exo-H3

2. Data flow diagram

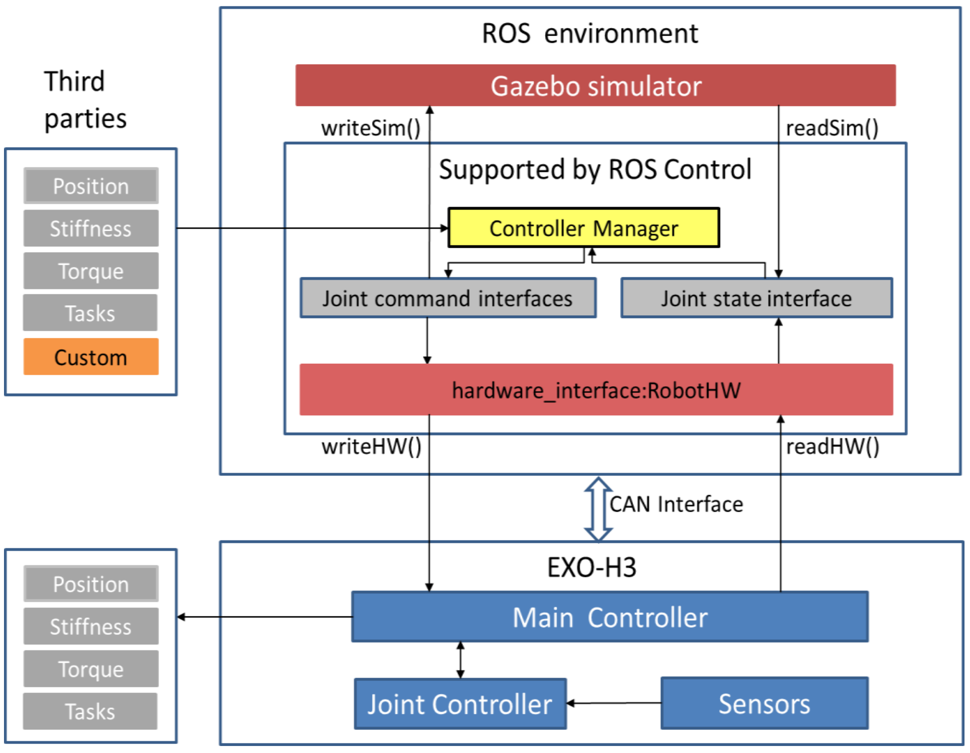


Image 1. Data flow diagram between H3 Main Controller and ROS Controllers.

The MC runs three internal controllers for each joint:

- **Position controller**
- **Stiffness controller**
- **Torque controller**

And a gait algorithm called:

- **Task Controller**

The Hardware Abstraction Layer of the ROS Control framework[1] creates a bridge between the ROS controllers and the controllers running in the MC, allowing command of the joints from the ROS interface. The *controller manager* is the tool that manages the lifecycle controller, providing the mechanisms to load, start, stop, and unload a ROS controller in runtime.

[1] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. Rodríguez Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke and E. Fernandez Perdomo
"ros_control: A generic and simple control framework for ROS"
The Journal of Open Source Software, 2017

3. Hardware interfaces

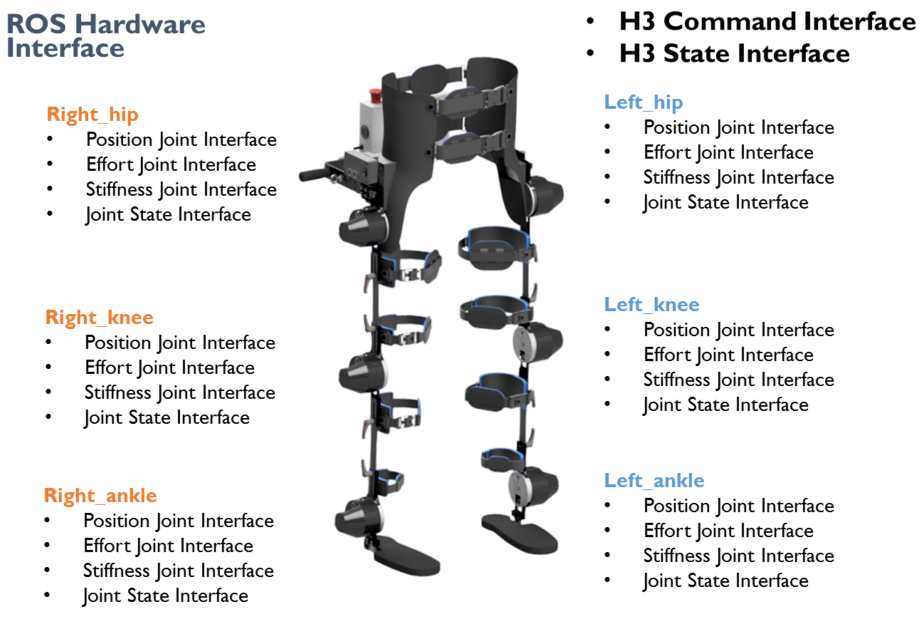


Image 2. ROS hardware interfaces implemented in *h3_hardware_interface* node.

The following interfaces are implemented in the h3_hardware_interface package:

- ***Joint State Interface****:* support reading of joint position (rad), joint velocity (rad/s), and joint motor torque (Nm).
- **Position Joint Interface:** interface with the internal position controller (command$[rad]$).

- **Effort Joint Interface**[1]: interface with the internal torque controller (command$[Nm]$).
- **Stiffness Joint Interface:** interface with internal stiffness controller (command $[rad, \%stiffness]$)
- **H3 State Interface:** support reading of signal defined in the *h3_msgs/State* message
- **H3 Command Interface:** This is a specific Exo-H3 interface that allows ROS controllers to access all Exo-H3 resource; joint state signals, internal controllers command, and Main Controller parameters. The following list shows the command signals exposes for H3 Command Interface to ROS *controller_interface*:
- Minimal angles
- Maximum angles
- Percentage of assistance
- Control type
- Position setpoint
- Stiffness setpoint

[1] ROS interface typically used to command a force/torque joint

- Torque setpoint
- Exo command APP
- Robot task
- Trigger output

4. Controllers

The controller plugins available in ROS repositories compliant with Position Joint Interface, Effort Joint Interface, and State Joint Interface can use the device after selecting the appropriate control type in the Config Controller ROS interface.

Also, you can create your own controller using existent H3 Hardware Interfaces.

The following controllers have been testing with the Exo-H3:

- **Joint State Controller**
- **Joint Effort Controller**
- **Joint Position Controller**
- **Joint Stiffness Controller**: This controller gets a *handle* to the Stiffness Joint Interface and subscribes to *std_msgs/Float64MultiArray* message in a ROS topic.
- **H3 State Controlle**r: This controller gets a *handle* to the H3 State Interface and publishes *h3_msgs/State* message in a ROS topic.
- **H3 Config Controller**: the configuration controller implements six service servers with the following functions:
- Set assistance [*h3_msgs/joints*]
- Set control type [*h3_msgs/ControlType*]
- Set maximum angles [*h3_msgs/Joints*]
- Set minimum angles [*h3_msgs/Joints*]
- Set trigger output [*h3_msgs/TriggerOutput*]
- Set data recording [*h3_msgs/DataRecording*]
- **H3 Task Controller:** This controller gets a *handle* to the H3 Command interface, receives an *h3_msgs/TaskCommand* message from the ROS network, and sends a command to Task Controller running in the Exo-H3 MC.

5. List of ROS packages

The Exo-H3 ROS stack is a group of packages developed in the C++ language. Each package has a library and file set that allows ROS users to interact with the physical device and the virtual device. The main ROS dependency is the *ros_control* framework since the device uses the packages *hardware_interface* and *controller_interface*. The ROS packages are listed below:

- *h3_hardware_interface*
- *h3_urdf*
- *h3_msgs*
- *h3_robot_state_controller*
- *h3_task_controller*
- *h3_config_controller*
- *h3_control_client*
- *stiffness_joint_controller*
- *rqt_h3 plugin*
- *h3_test*

6. Internal Task Controller

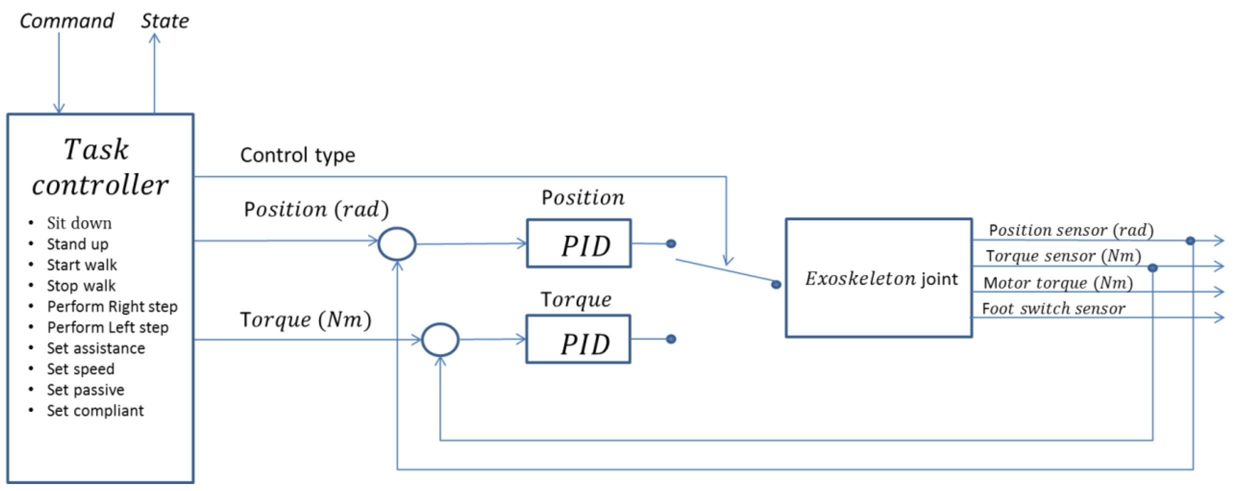


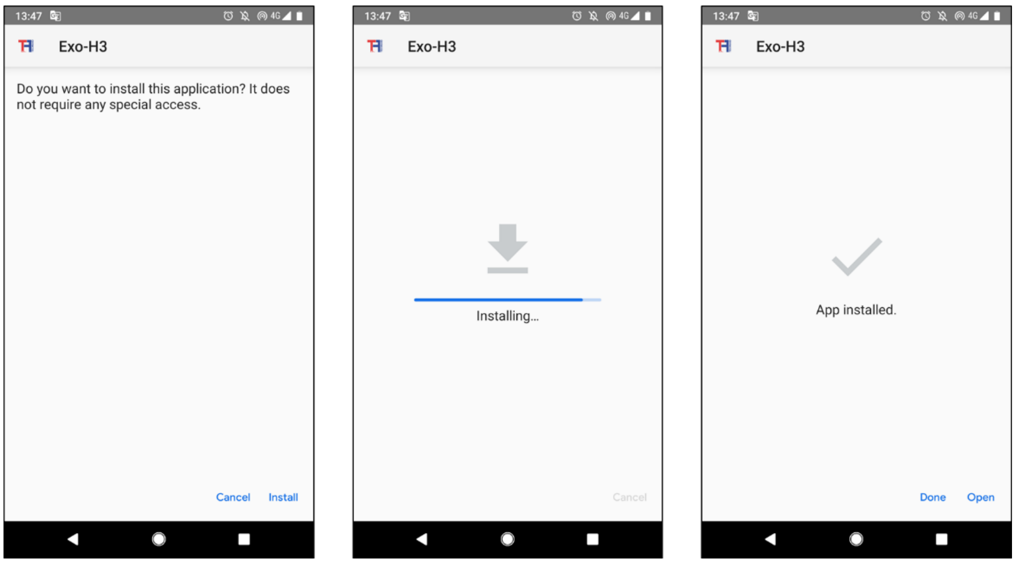
Image 3. Simplified diagram of Task Controller.

The Task Controller is a state machine that uses the position control loop and the torque control loop. The Task controller receives a command from an external system (Android APP or ROS interface) and executes a task according to the machine's present state and the measured signals of the sensors and the internal variables. Also, the Task Controller notifies its current status.

6.1. Sending command to Task Controller from Android APP

6.1.1. Installing the Android APP

1. Download the *Exo_H3.apk* into the internal storage of your cell phone. To do this, consult Android documentation.
2. Before installation, you must allow applications from sources other than the Play Store to be installed in your cell phone. To do this, carry out the following steps:
- Go to Settings > Security.
- Click the option "Unknown Sources".
- Tap Ok on the prompt message.
- Select "Trust".
3. Then with the cell phone file manager, search for the application and tap *Exo_H3.apk* to install it. The displays shown in Image 4. Installing *Exo_H3.apk*.will appear successively.



Do you want to install this application? It does not require any special access.

Image 4. Installing *Exo_H3.apk*.

6.1.2. Using the android APP

The Android APP is the main user interface for high-level control of Exo-H3. Send commands to execute a different task, such as sit down, stand up, start walking, stop walking, etc. After installing the Android APP, pair the MC's Bluetooth interface with the Android device (smartphone or tablet).

Open the Android APP and push the Bluetooth button to start the connection with the MC. A "Connection in Progress" message will appear at the top of the display, and the message will switch to "Exo Passive" as shown in the following image.

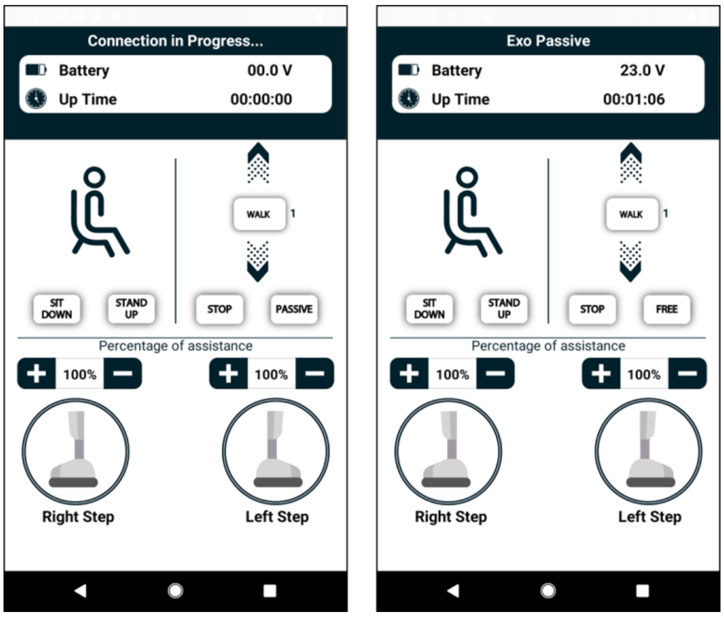
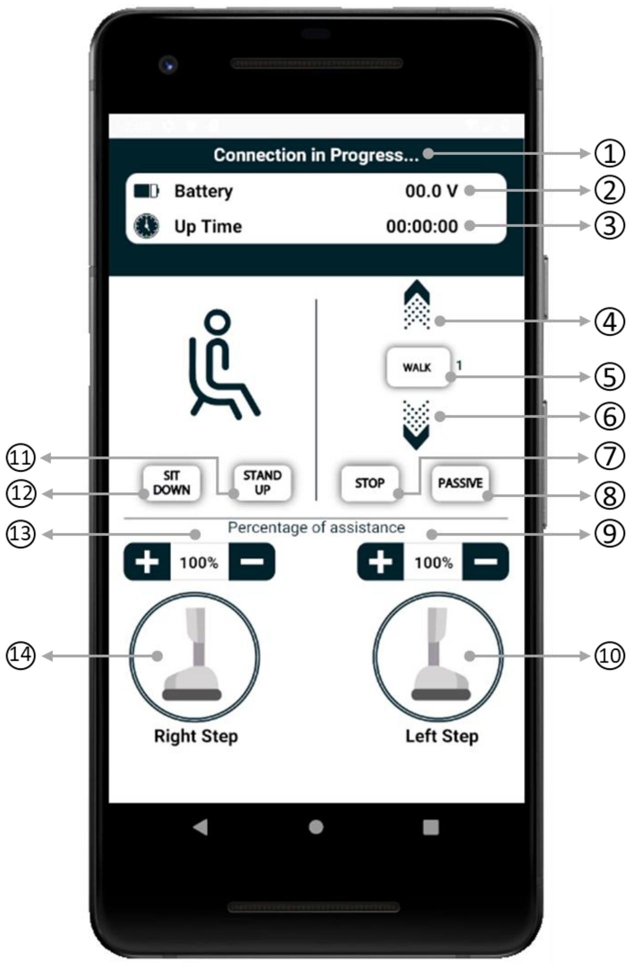


Image 5. Main window of the Exo-H3 application.

A complete description of the Android App is shown in the following table. Now, you can send a command to Exo-H3 by pushing a button.

Distribution of buttons	Operating modes
 Connection in Progress... Battery 00.0 V Up Time 00:00:00 WALK 1 SIT DOWN STAND UP STOP PASSIVE Percentage of assistance 100% 100% Right Step Left Step ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭	**1.** **Process status:** Shows the current status of Exo-H3. **2.** **Battery level:** Shows the voltage level of the Exo-H3 battery. **3.** **Time Up:** Shows the connection time between the mobile device and Exo-H3. **4.** **Up velocity:** Allows the gait speed to be increased by increments of 1. Maximum speed is 10. **5.** **Walking:** The "Walk" button allows walking to begin with a pre-loaded gait pattern (10 speed levels). **6.** **Down velocity:** Allows the gait speed to be reduced by decrements of 1. Minimum speed is 1. **7.** **Stop:** The "Stop" button allows walking to stop. The joints return to their point of origin (zero angle). **8.** **Passive/Compliant:** The "Passive" button deactivates the energy of the motors, and the "Compliant" button allows the joints to try to follow the end user's movements (zero torque mode). **9.** **Left Leg Assistance:** Allows the percentage of assistance for the left leg to be increased (by pressing "+") or decreased (by pressing "-") by 10% intervals, between 10% and 100%. **10.** **Left Step:** Allows a step to be taken with the left leg. Reactivating it requires taking a right step with **Right Step** (14). **11.** **Stand Up:** Allows Exo-H3 to stand up. To initiate the action, the joints must be in the pre-established sitting position. Exo-H3 will help the end user to get up (the assistance of a user is required). A beeping will first be heard. **12.** **Sit Down:** Allows the Exo-H3 to be arranged in the sitting position after having activated the "Stop". Exo-H3 will help the end user to sit down (the assistance of a user is required). A beeping will first be heard. **13.** **Right Leg Assistance:** Allows the percentage of assistance for the right leg to be increased (by pressing "+") or decreased (by pressing "-") by 10% intervals, between 10% and 100%. **14.** **Right Step:** Allows a step to be taken with the right leg. Reactivating it requires taking a left step with **Left Step** (10).

7. Monitoring the Task Controller in ROS

To use the Exo-H3 with ROS, you need a PCAN interface and a computer running Ubuntu operating system with ROS distro installed. Ubuntu 18.04 and ROS Melodic Morenia work fine.

We can use the Exo-H3 in a distributed computing network, typically a LAN (Local Area Network) with different devices such as a motion capture system, a pressure sensor, a virtual reality system, etc. In this scenario, the exoskeleton can execute a task while other devices in the network get the data generated by the exoskeleton's sensors.

As an approximation to the scenario mentioned above, we let's use an Onboard-PC located in the Exo-H3's hip and another PC (we call it, Control-Client PC) located on a desk. The Onboard-PC (On-PC) is connected to a router through WIFI and the Control-Client PC (CC-PC) is connected to a router through Ethernet.

For monitoring the exoskeleton´s sensors in the network that we have created, we need to download and build some packages.

7.1. Installing CAN driver

The *h3_hardware_interface* package uses the PCANBasic library and Peak CAN Driver (Character-based driver). Therefore, is required that you download, compile and install the driver and the PCANBasic library following the below instructions:

1. Download the Peak CAN driver source package for Linux from here.
2. Unzip, Build, and Install the driver following the README file.

Open a new terminal in the driver directory and build the Chardev driver running:

```
sudo make clean all
```

Install the PCAN driver and the PCANBasic library using:

```
sudo make install
```

And,

```
sudo modprobe pcan
```

Connect your pcan interface and check if the driver is installed

```
cat /proc/pcan
```

If you get the error "pop.h not found", please run

```
sudo apt-get install libpopt-dev
```

And try to build and install instructions again.

7.2. Establishing Communication with the Exo-H3 Onboard PC

We will use SSH (Secure Shield) to login into the On-PC from CC-PC. Also, we will follow ROS Network Setup guidelines to configure our ROS network.

The CC-PC will run the ROS Master, while On-PC will be configured to find ROS Master in CC-PC.

Open a new terminal in the CC-PC, replace **CC-PC_IP** address, and run:

```
echo "export ROS_MASTER_URI='http://CC-PC_IP:11311' " >> ~/.bashrc
echo "export ROS_IP= CC-PC_IP " >> ~/.bashrc
source ~/.bashrc
```

For example:

```
echo "export ROS_MASTER_URI='http://192.168.8.111:11311' " >> ~/.bashrc
echo "export ROS_IP=192.168.8.111" >> ~/.bashrc
source ~/.bashrc
```

Open a new terminal in the CC-PC, and log in to On-PC:

```
ssh usuario@hostname
```

For example:

```
ssh exoh3@192.168.8.114
```

Replace the **CC-PC_IP** address and the **On-PC_IP** address, and run:

```
echo "export ROS_MASTER_URI='http://CC-PC_IP:11311' " >> ~/.bashrc
echo "export ROS_IP=On-PC_IP" >> ~/.bashrc
source ~/.bashrc
```

For example:

```
echo "export ROS_MASTER_URI='http://192.168.1.111:11311' " >> ~/.bashrc
echo "export ROS_IP=192.168.8.114" >> ~/.bashrc
source ~/.bashrc
```

7.3. Downloading and building ROS packages

1- Download the H3 ROS package from the Technaid GitHub repository in your ROS workspace.

- If you are using the Onboard PC, only copy the following packages in the Onboard PC ROS workspace: **h3_msgs, h3_hardware_interface, h3_config_controller, h3_robot_state_controller, and h3_task_controller.**
- If your Onboard PC is a Raspberry Pi, copy the package **h3_indicator_rpi** in the Onboard PC ROS workspace.

- If you are using the **Control-Client PC**, only copy the following packages in the Control-Client PC ROS workspace: **h3_msgs, h3_control_client.**

2- Build your workspace.

First, build the *h3_msgs* package. Open a new terminal in your ROS workspace and run:

```
catkin_make --pkg h3_msgs
```

Install the ROS package dependencies using the rosdep tool:

```
rosdep install <package_name>
```

Install the *ros_controllers* packages (only is required in Onboard PC):

```
sudo apt-get install ros-noetic-ros-control ros-noetic-ros-controllers
```

For the h3_hardware_interace package, open a new terminal in your ROS workspace and run:

```
rosdep install h3_hardware_interface
```

Build your ROS workspace running:

```
catkin_make
```

7.4. Launching the ROS nodes

Launch *h3_hardware_interface* node and load configuration in parameters server:

```
roslaunch h3_hardware_interface h3_hardware_interface.launch
```

Get node information using the *rosnode* list command in a new terminal:

```
rosnode list
/h3/h3_hardware_interface
/rosout
```

And,

```
rosnode info /h3/h3_hardware_interface
--------------------------------------------------------------------------------
Node [/h3/h3_hardware_interface]
Publications:
 * /rosout [rosgraph_msgs/Log]
```

```
Subscriptions: None
Services:
 * /h3/controller_manager/list_controller_types
 * /h3/controller_manager/list_controllers
 * /h3/controller_manager/load_controller
 * /h3/controller_manager/reload_controller_libraries
 * /h3/controller_manager/switch_controller
 * /h3/controller_manager/unload_controller
 * /h3/h3_hardware_interface/get_loggers
 * /h3/h3_hardware_interface/set_logger_level
contacting node http://exo:40455/ ...
Pid: 7874
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound (47371 - 192.168.8.114:46920) [12]
    * transport: TCPROS
```

Now, let's go to monitoring the Exo-H3's joint positions using *rqt_plot.*

Log in to On-PC through SSH.

```
ssh exoh3@192.168.1.114
```

Launch the *h3_hardware_interface* node

```
roslaunch h3_hardware_interface h3_hardware_interface.launch
```

Different messages print in the terminal show you if the connection with Exo-H3 is successful.

In CC-PC open new a terminal and run:

```
roslaunch h3_control_client h3_monitoring_controller.launch
```

The *h3_monitoring_controller.launch* loads in *h3_hardware_interface* node a *joint_state_controller* and a *robot_state_controller* that publish */h3/joint_states* and */h3/robot_states* topics.

You can see these topics opening a new terminal, and running:

```
rostopic list
```

```
/h3/joint_states
/h3/robot_states
/rosout
/rosout_agg
```

Now you can run *rqt_plot* and add */h3/joint_states* or */h3/robot_states* topics.

Manually move the exoskeleton right knee and plot the position and torque sensor:

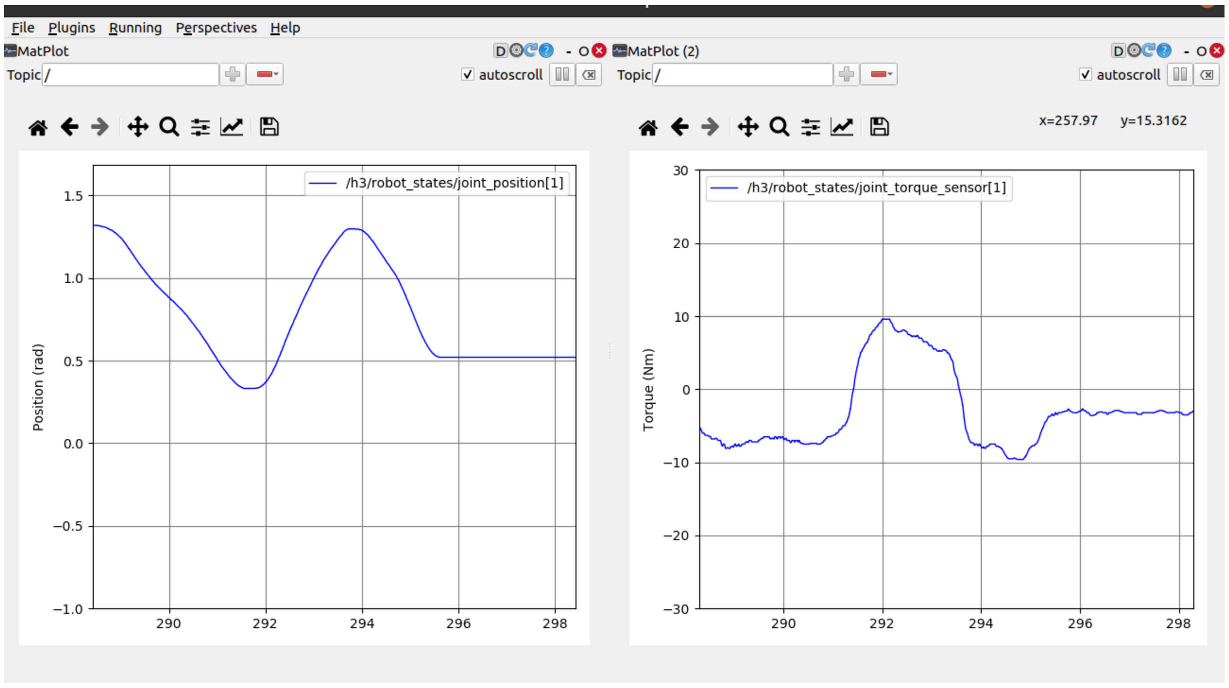


Image 6. Exoskeleton right knee position and torque during a manual movement.

8. Sending commands to Task Controller from ROS

The ROS Task Controller allows the user to sends commands to the Exo-H3 gait patterns algorithm from the ROS network. Our network must have low latency and a stable connection between the Onboard PC and the Control-Client PC to avoid delay or loss of the commands. If the WIFI connection is unstable consider use a wired connection.

Launch *h3_hardware_interface* node

```
roslaunch h3_hardware_interface h3_hardware_interface.launch
```

Load the *h3_task_controller:*

```
roslaunch h3_control_client h3_task_controller.launch
```

List topics:

```
rostopic list
```

```
/h3/joint_states
/h3/robot_states
/h3/task_controller/command
/rosout
/rosout_agg
```

Now, we have a topic called */h3/task_controller/command*. We will use this topic to send a command to Exo-H3.

The command list can be query running:

```
rosmsg show h3_msgs/TaskCommand
uint8 STOP_GAIT=0
uint8 WALK_SPEED_1=1
uint8 WALK_SPEED_2=2
uint8 WALK_SPEED_3=3
uint8 WALK_SPEED_4=4
uint8 WALK_SPEED_5=5
uint8 WALK_SPEED_6=6
uint8 WALK_SPEED_7=7
uint8 WALK_SPEED_8=8
uint8 WALK_SPEED_9=9
uint8 WALK_SPEED_10=10
uint8 JOINTS_PASSIVE=11
uint8 JOINTS_COMPLIANT=12
uint8 STAND_UP=21
uint8 SIT_DOWN=22
uint8 PERFORM_RIGHT_STEP=23
uint8 PERFORM_LEFT_STEP=24
uint8 RIGHT_MAX_ASSITENCE_10=31
uint8 RIGHT_MAX_ASSITENCE_20=32
uint8 RIGHT_MAX_ASSITENCE_30=33
uint8 RIGHT_MAX_ASSITENCE_40=34
uint8 RIGHT_MAX_ASSITENCE_50=35
uint8 RIGHT_MAX_ASSITENCE_60=36
uint8 RIGHT_MAX_ASSITENCE_70=37
```

```
uint8 RIGHT_MAX_ASSITENCE_80=38
uint8 RIGHT_MAX_ASSITENCE_90=39
uint8 RIGHT_MAX_ASSITENCE_100=40
uint8 LEFT_MAX_ASSITENCE_10=41
uint8 LEFT_MAX_ASSITENCE_20=42
uint8 LEFT_MAX_ASSITENCE_30=43
uint8 LEFT_MAX_ASSITENCE_40=44
uint8 LEFT_MAX_ASSITENCE_50=45
uint8 LEFT_MAX_ASSITENCE_60=46
uint8 LEFT_MAX_ASSITENCE_70=47
uint8 LEFT_MAX_ASSITENCE_80=48
uint8 LEFT_MAX_ASSITENCE_90=49
uint8 LEFT_MAX_ASSITENCE_100=50
uint8 REGAIN_COMMAND=51
uint8 GIVE_COMMAND=52
uint8 command
```

The command message can take different values depending on a specific task. When the task controller starts, the Bluetooth interface (Android app) owns the command (you can see this condition *in / h3 / robot_state / command_app*). If command_app is 1, the Bluetooth interface has the Task Controller command, and if *command_app* is 0, the ROS interface has the Task Controller command.

You can see the commad_app indicator running:

```
rostopic echo /h3/robot_states
```

The following tests must be performed without a subject wearing the exoskeleton until you can test the operation of the commands.

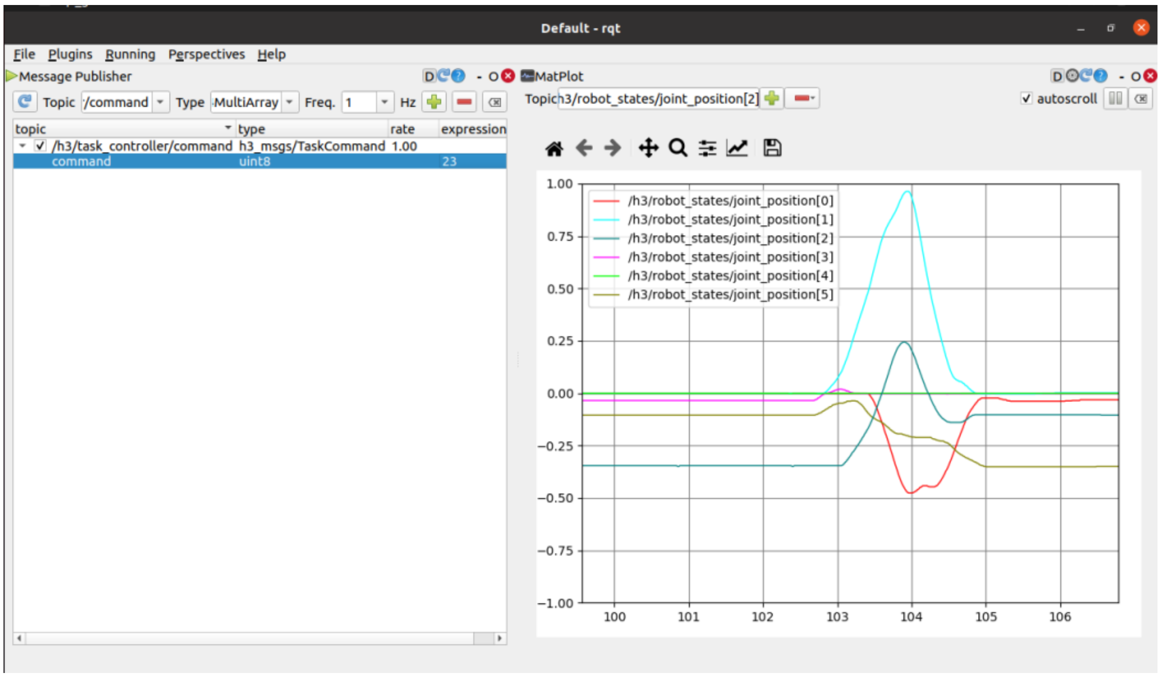
Get the command from ROS Task controller publishing the following message:

```
rostopic pub /h3/task_controller/command h3_msgs/TaskCommand "command: 51
```

And verify that *command_app* change to zero in the topic */h3/robot_states*

Now, you can send a task command to Exo-H3 to perform a right step:

```
rostopic pub /h3/task_controller/command h3_msgs/TaskCommand "command: 23"
```



Default - rqt
File Plugins Running Perspectives Help
Message Publisher
Topic /command Type MultiArray Freq. 1 Hz
MatPlot
Topic h3/robot_states/joint_position[2]
autoscroll

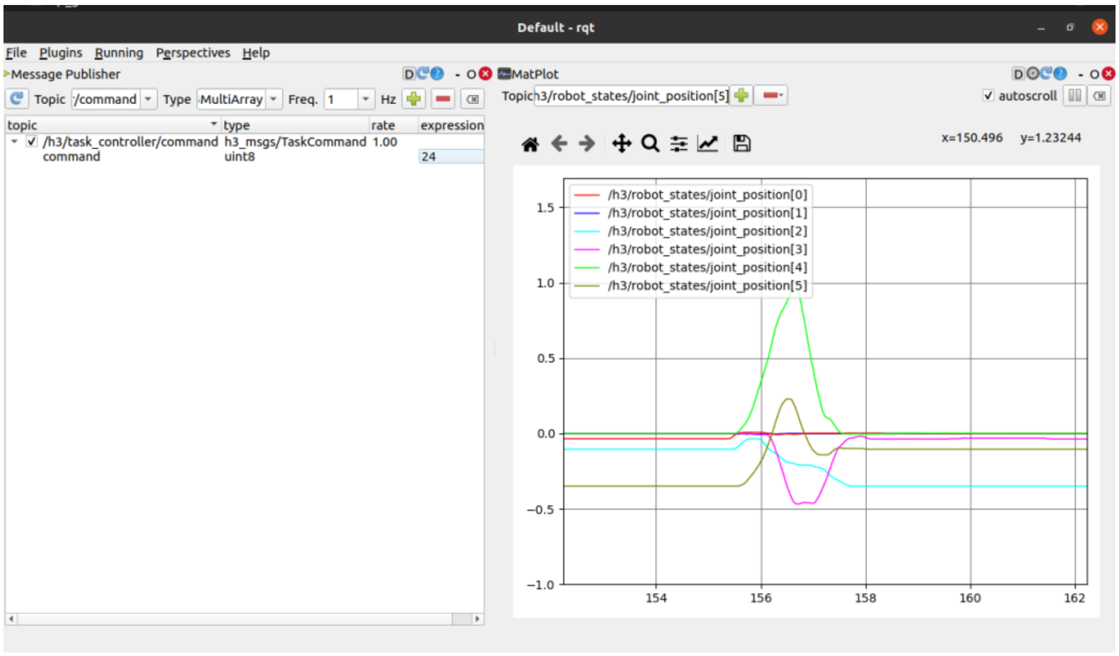
topic	type	rate	expression
/h3/task_controller/command	h3_msgs/TaskCommand	1.00	
command	uint8		23

/h3/robot_states/joint_position[0]
/h3/robot_states/joint_position[1]
/h3/robot_states/joint_position[2]
/h3/robot_states/joint_position[3]
/h3/robot_states/joint_position[4]
/h3/robot_states/joint_position[5]

Image 7. Joint Position while the exoskeleton performs a right step.

And perform a left step:

```
rostopic pub /h3/task_controller/command h3_msgs/TaskCommand "command: 24"
```



Default - rqt
File Plugins Running Perspectives Help
Message Publisher
Topic /command Type MultiArray Freq. 1 Hz
MatPlot
Topic h3/robot_states/joint_position[5]
autoscroll
x=150.496 y=1.23244

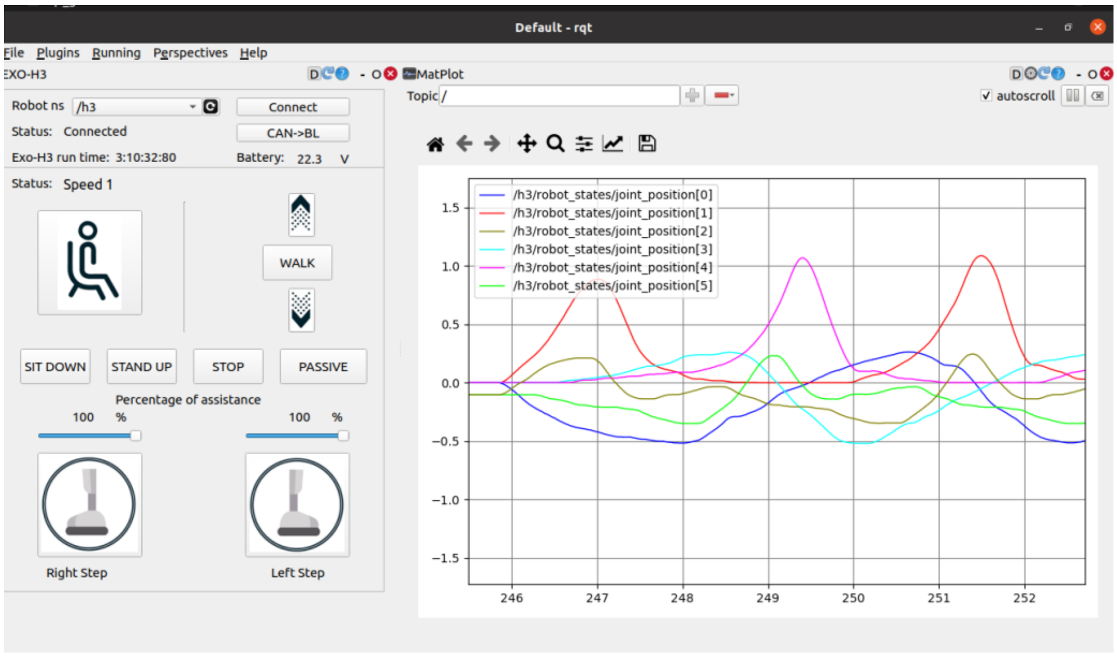
topic	type	rate	expression
/h3/task_controller/command	h3_msgs/TaskCommand	1.00	
command	uint8		24

/h3/robot_states/joint_position[0]
/h3/robot_states/joint_position[1]
/h3/robot_states/joint_position[2]
/h3/robot_states/joint_position[3]
/h3/robot_states/joint_position[4]
/h3/robot_states/joint_position[5]

Image 8. Joint Position while the exoskeleton performs a left step.

The Task Controller receives a command timeout parameter. If while the Exo-H3 is walking it doesn´t receive a command it will stop. You can modify this parameter in *h3_task_controler.yaml* file *in /h3_control_client/config* directory.

An *rqt_gui* plugin in ROS is available to send commands to the Task Controller. Open it in the menu Plugins->Robot Tools->H3 command.



Default - rqt

Robot ns /h3

Status: Connected

Exo-H3 run time: 3:10:32:80

Battery: 22.3 V

Status: Speed 1

Percentage of assistance 100 % 100 %

Topic /

autoscroll

File
Plugins
Running
Perspectives
Help
EXO-H3
MatPlot
Connect
CAN->BL
WALK
SIT DOWN
STAND UP
STOP
PASSIVE
Right Step
Left Step
/h3/robot_states/joint_position[0]
/h3/robot_states/joint_position[1]
/h3/robot_states/joint_position[2]
/h3/robot_states/joint_position[3]
/h3/robot_states/joint_position[4]
/h3/robot_states/joint_position[5]
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
246
247
248
249
250
251
252

Image 9. Joint Position during the exoskeleton walk.

Now, you can send command to the exoskeleton while you plot the sensors data.

9. Using the Configuration Controller

The configuration controller implements six service servers.

```
rosservice list /h3/config_controller/
/h3/config_controller/set_assistance
/h3/config_controller/set_control_type
/h3/config_controller/set_data_recording
/h3/config_controller/set_max_angles
/h3/config_controller/set_min_angles
/h3/config_controller/set_trigger_output
```

The Exo-H3 has different types of control. The type of control must be selected according to the ROS controller to be used (see the parameters of h3_config_controller) . At runtime you can change the control type by calling / h3 / config_controller / set_control_type service.

```
rosservice call /h3/config_controller/set_control_type "{right_hip: 0, right_knee: 0, right_ankle: 0, left_hip: 0, left_knee: 0, left_ankle: 0}"
```

You can see all available options in the ControlType message:

```
rossrv show h3_msgs/ControlType
uint8 NO_CONTROL=0
uint8 POSITION_CONTROL=1
uint8 STIFFNESS_CONTROL=2
uint8 TORQUE_CONTROL=3
uint8 MOTOR_DISABLED=4
uint8 MOTOR_STOPPED=5
uint8 OPEN_LOOP=6
uint8 TASK_CONTROLLER=7
uint8 right_hip
uint8 right_knee
uint8 right_ankle
uint8 left_hip
uint8 left_knee
uint8 left_ankle
---
bool success
```

You should take special care when changing the *control_type* while a controller is running to avoid undesired movements.

The */set_data_recording* service offers different options for recording the Exo-H3 data in a comma-separated values (.CSV) format.

Show recording mode options:

```
rossrv show h3_msgs/DataRecording
uint8 SW_STOP_MODE=0
uint8 SW_START_MODE=1
uint8 TRIGGER_INPUT_PULSE_MODE=2
uint8 TRIGGER_INPUT_HIGH_MODE=3
uint8 TRIGGER_OUTPUT_PULSE_MODE=4
```

```
uint8 TRIGGER_OUTPUT_HIGH_MODE=5
uint8 recording_trigger_mode
string file_base_name
bool timed
float64 duration_time
---
bool success
```

To start recording data you can run:

```
rosservice call /h3/config_controller/set_data_recording "recording_trigger_mode: 1
file_base_name: 'recording_test'
timed: false
duration_time: 0.0"
```

To stop the data recording you can run:

```
rosservice call /h3/config_controller/set_data_recording "recording_trigger_mode: 0
file_base_name: ''
timed: false
duration_time: 0.0"
```

To start the data recording, and stop it ten minutes after, you can run:

```
rosservice call /h3/config_controller/set_data_recording "recording_trigger_mode: 1
file_base_name: "recording_test"
timed: false
duration_time: 600.0"
```

The recording data start/stop can be trigger by input/output digital pin in MC.
You can start/stop data recording by trigger input pulse:

```
rosservice call /h3/config_controller/set_data_recording "recording_trigger_mode: 2
file_base_name: 'trigger_test'
timed: false
```

```
duration_time: 0.0"
```

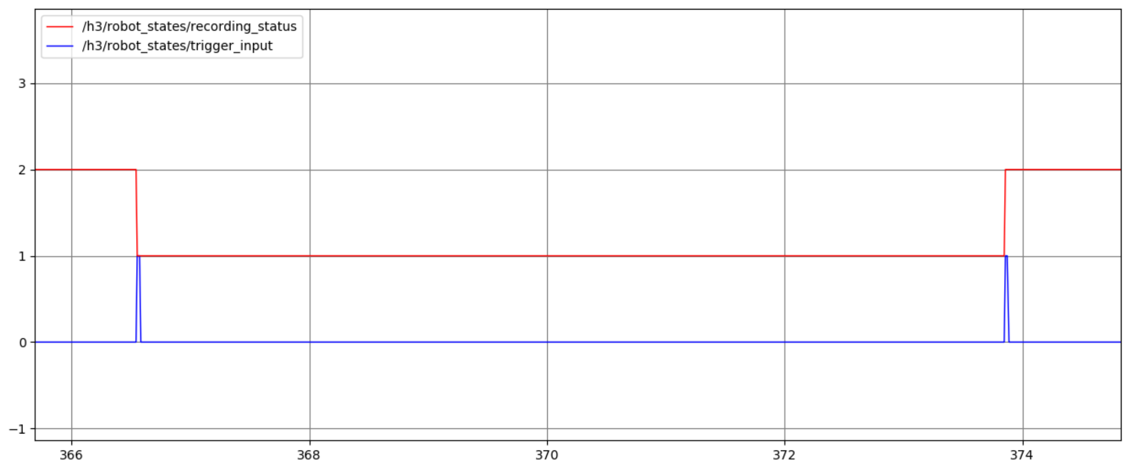


Image 10. Starting/stopping the recording data by a pulse on a trigger input pin.

You can see the *recording_status* indicator in / *h3* / *robot_states* topic, and interpret the values as follows:

0- No data is being recorded.

1- Recording data.

2- Waiting for an input trigger to start data recording.

3- Waiting for an output trigger to start data recording.

The generated file is stored in **~/.ros** directory in On-PC. You can copy the file in CC-PC using the SCP commands.

10. Internal Position Controller

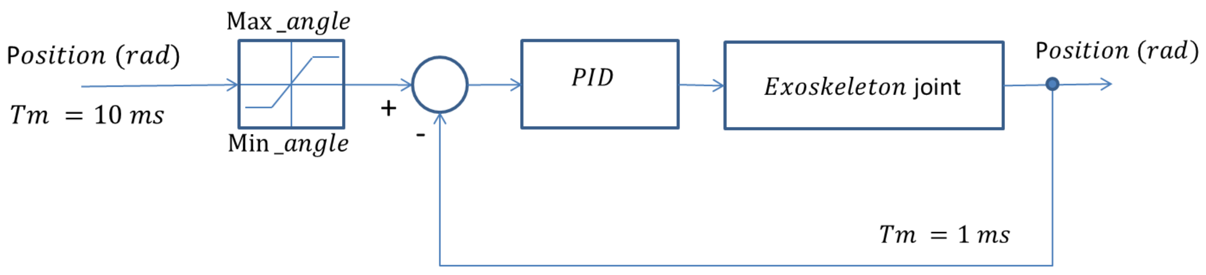


Image 11. Simplified joint torque closed-loop control.

The position controllers are parallel PID with gains *Kp*, *Ki, Kd,* and integral limitation *i_clamp*.

By default, all joints in position control mode have the same PID gains. The PID output control signal is a PWM duty cycle percentage, expressed how a number in the range [-3600, 3600], where 3600 is equivalent to 100% PWM duty cycle.

The Exoskeleton joint is composed of a Joint Controller (JC) that receives PWM duty cycle signal from the MC and applies this signal to motor drivers. The actuation is performed by a brushless motor attached to a reducer.

The Onboard PC sends position setpoint to MC each 10 ms. The MC limits the position setpoint before introducing the reference signal in the closed-loop control.

The position limits: *Max_angles* and *Min_angles* are modified from the Onboard PC. Modify these limits according to exoskeleton mechanical limits for each joint. See

Special care should be taken into account when you work on ROS because the Exo-H3 adopted the system of units and coordinates from ROS, which is different from it used by the MC. While the joint rotation in MC is around the transversal axis, and the position, in degree unit, increment positively on flexion movement, On ROS, the joint rotation is around the y axis (pitch rotation), and the position, in radian unit, increments positively on counter-clockwise movement.

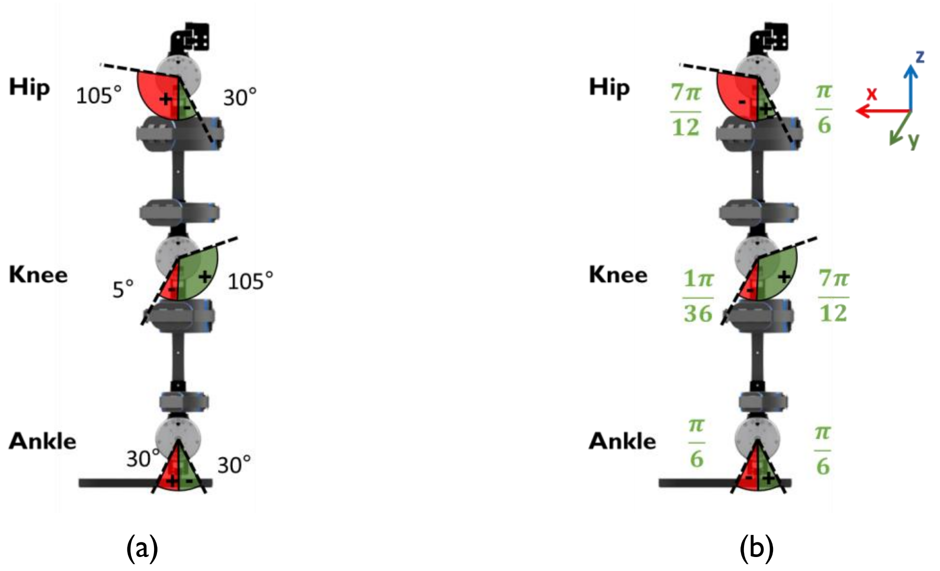


Image 12. System of units and coordinate in the Main Controller (a) and the ROS interface (b).

10.1. Running the ROS Position Controllers

You can load a ROS position_controllers in the *h3_hardware_interface* node. The ROS position controller use *hardware_interface::PositionJointInterface* class instanced in *h3_hardware_interface* node.

Let's go running the following ROS controllers defined in the package h3_control_client:

- position_controllers/JointPositionController
- position_controllers/JointGroupPositionController
- effort_controllers/JointPositionController

Launch h3_hardware_interface node

```
roslaunch h3_hardware_interface h3_hardware_interface.launch
```

Load the position_controllers/JointPositionController in h3_hardware_interface node:

```
roslaunch h3_control_client h3_position_controller.launch
```

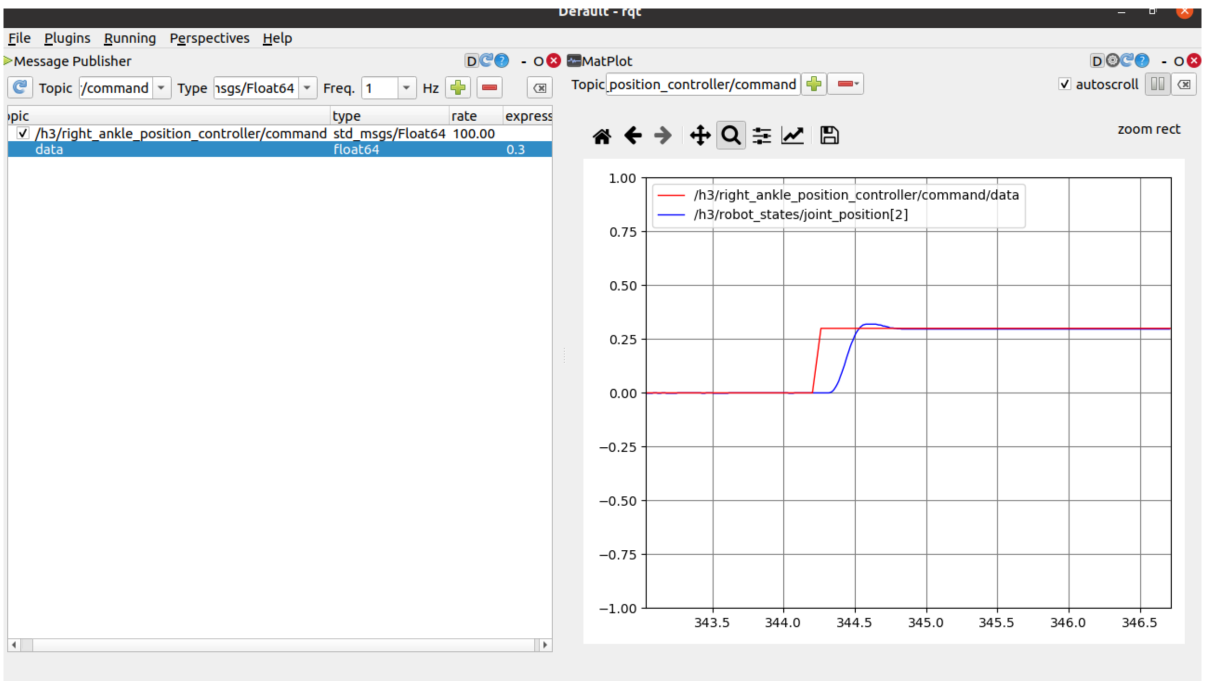
Get the node info:

```
rosnode info /h3/h3_hardware_interface
```

Note that the *h3_hardware_interface* node now publishes topics and subscribes to topics introduced by the controllers. The subscribed topics accept a position command for each joint.

Now, we open a new terminal and publish in the */h3/right_ankle_position_controller/command* topic the position setpoint.

```
rostopic pub /h3/right_knee_position_controller/command std_msgs/Float64 "data: 0.3"
```



Topic /command Type ısgs/Float64 Freq. 1 Hz

Topic position_controller/command

```
ıpic                                       type              rate     express
✓ /h3/right_ankle_position_controller/command  std_msgs/Float64  100.00
    data                                   float64                    0.3
```

Image 13. Temporal response (Step function) of the right ankle position controller.

You should see that the Exo-H3's right ankle move in the plantarflexion direction.

Also, you can use *rqt_publisher* tool and test the joint with a sinusoid signal:

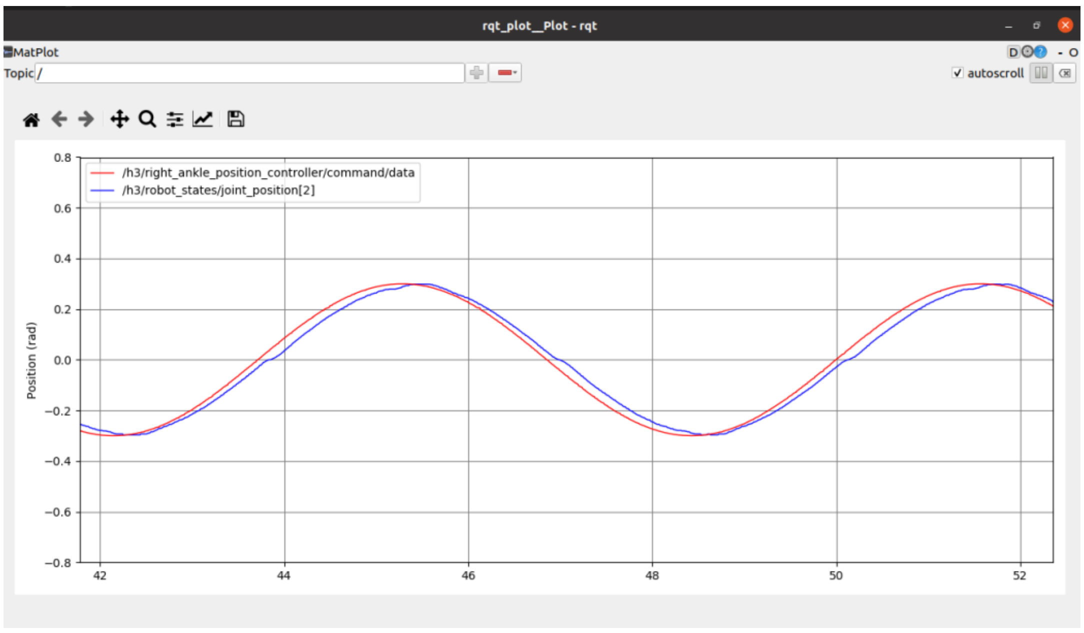


Image 14. Temporal response (sinusoid function) of the right ankle position controller.

You can modify the *h3_position_controllers.yaml* and *h3_position_controller.launch* files to change the controller names and configuration parameters.

Open the *h3_position_controller.yaml* file in */h3_hardware_interface/config* directory and change controllers names and set the assistance to 50:

Note that the subscription topic's names have changed.

After load a controller you can press **Ctrl + c** to unload the controllers.

If we like to set position command to all joints in one topic, we could be interested in load a Joint Group Position Controller.

```
roslaunch h3_control_client h3_group_position_controller.launch
```

Now you can publish in a topic a command with all joint positions:

```
rostopic pub /h3/joint_group_position_controller/command std_msgs/Float64MultiArray
"layout:
 dim:
 - label: ''
   size: 0
   stride: 0
 data_offset: 0
data:
[0, 0, 0, 0, 0, 0]"
```

The before controllers are an interface between internal position controllers (running in the MC) and ROS topics, and they don't have a PID ROS controller. We can use a PID ROS controller loading in *h3_hardware_interface* node an *effort_controllers/JointPositionController* and setting all joints in open-loop.

Launch *h3_hardware_interface* node

```
roslaunch h3_hardware_interface h3_hardware_interface
```

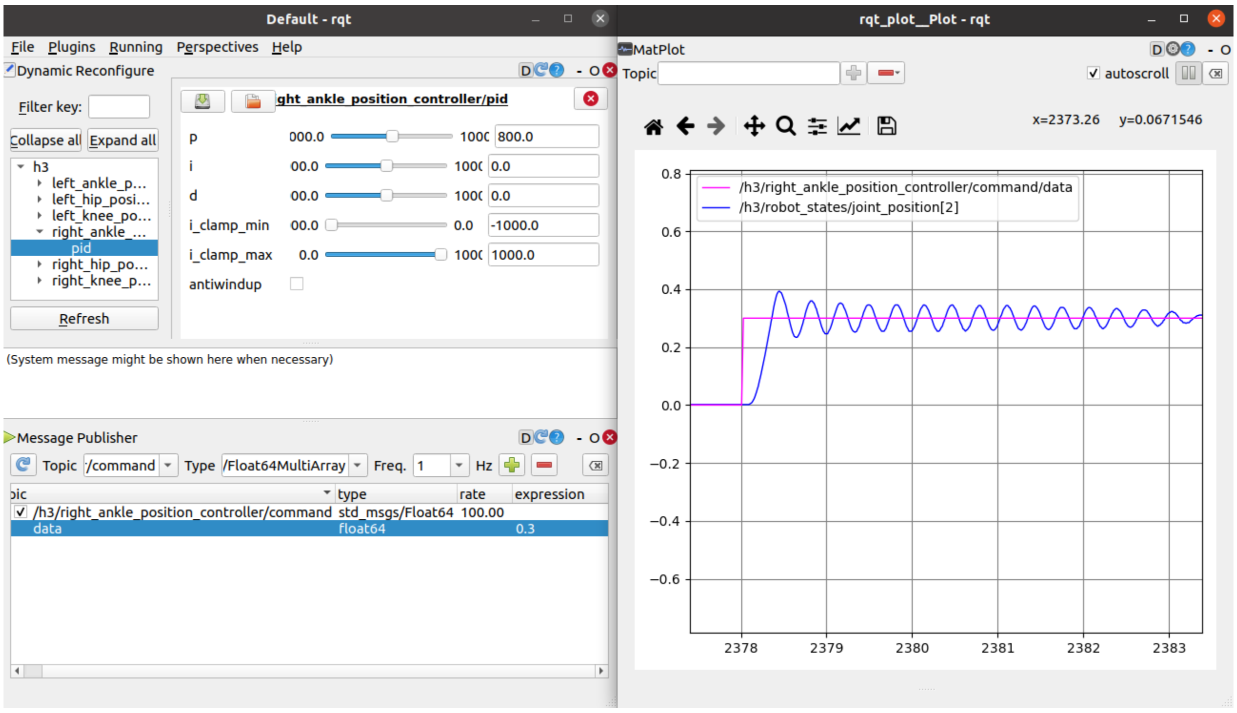
Load the controllers:

```
roslaunch h3_control_client h3_effort_position_control_client.launch
```

And get node info:

```
rosnode info /h3/h3_hardware_interface
```

Now you can tune the PID controllers. For example, using the Ziegler–Nichols method:



ght_ankle_position_controller/pid

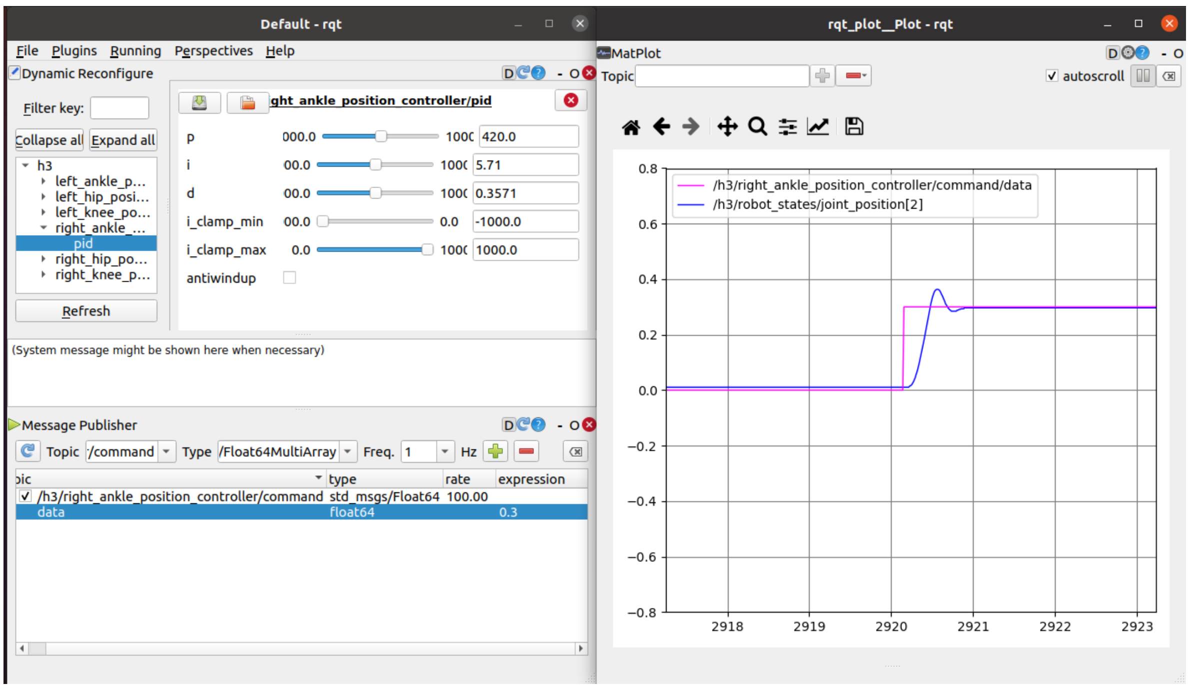
```
p            000.0  1000  800.0
i             00.0  1000  0.0
d             00.0  1000  0.0
i_clamp_min   00.0  0.0   -1000.0
i_clamp_max    0.0  1000  1000.0
antiwindup
```

(System message might be shown here when necessary)

```
Topic /command   Type /Float64MultiArray   Freq. 1   Hz

pic                                        type             rate    expression
✓ /h3/right_ankle_position_controller/command  std_msgs/Float64  100.00
    data                                   float64                  0.3
```

Image 15. Tuning a PID ROS controller, part I.



ght_ankle_position_controller/pid

p	000.0		1000	420.0
i	00.0		1000	5.71
d	00.0		1000	0.3571
i_clamp_min	00.0		0.0	-1000.0
i_clamp_max	0.0		1000	1000.0
antiwindup	☐			

(System message might be shown here when necessary)

Topic /command Type /Float64MultiArray Freq. 1 Hz

pic	type	rate	expression
✓ /h3/right_ankle_position_controller/command	std_msgs/Float64	100.00	
data	float64		0.3

Image 16. Tuning a PID ROS controller, part 2.

11. Internal Torque Controller

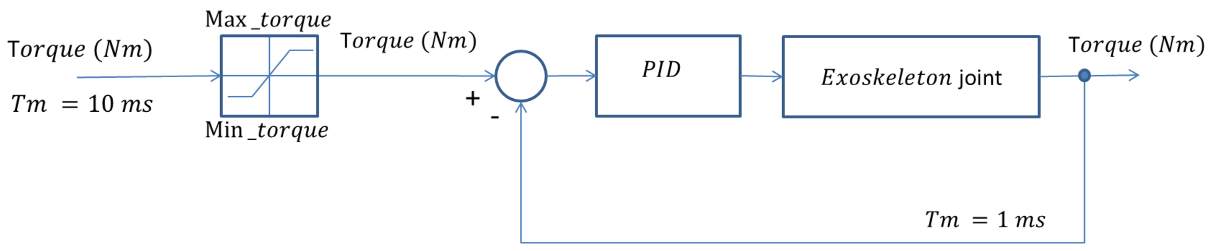


Figure 4. Simplified joint torque closed-loop control.

In the torque closed-loop control, the joint torque setpoint is limited from -50 to 50 Nm. The PID controller takes the error signal (difference between joint torque setpoint and joint torque estimated by strain gauges) and computes the output control signal that is sent to the actuator as a PWM duty cycle in the range [-3600 – 3600].

11.1. Running the ROS Effort Controller

Launch *h3_hardware_interface* node

```
roslaunch h3_hardware_interface h3_hardware_interface.launch
```

Load the *effort_controllers/JointEffortController* in *h3_hardware_interface* node:

```
roslaunch h3_control_client h3_torque_controllers.launch
```

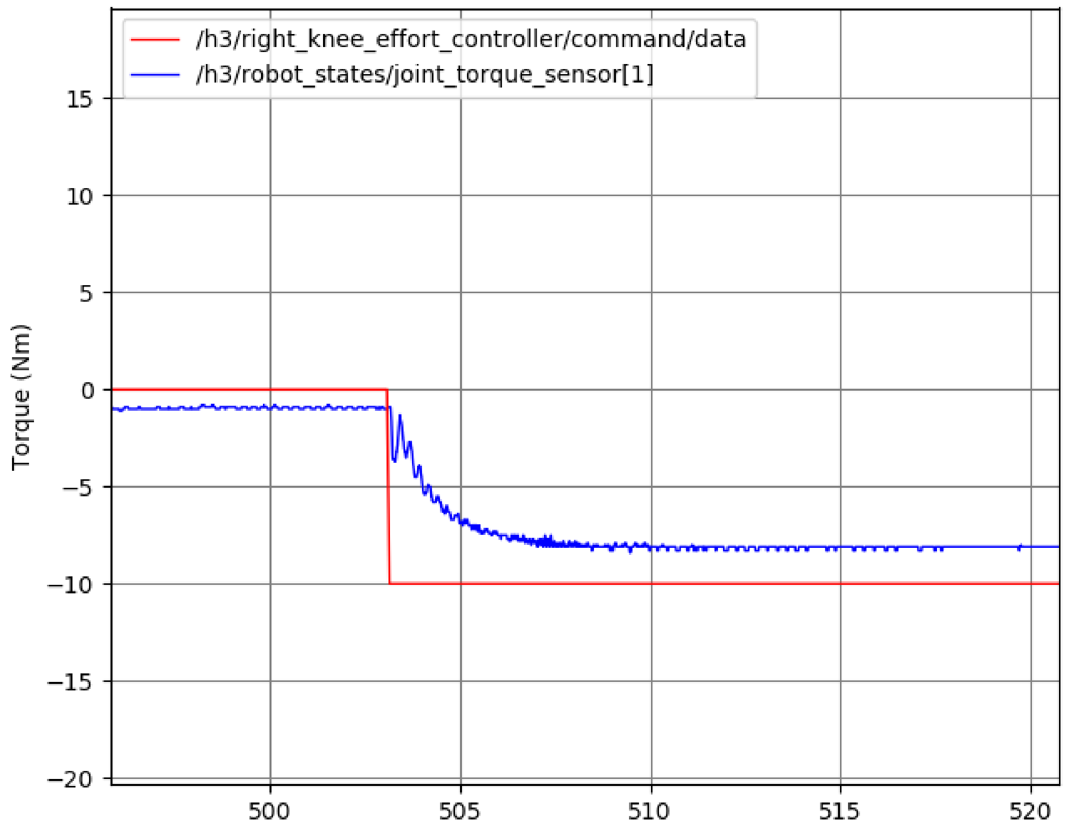
Get node info:

```
rosnode info /h3/h3_hardware_interface
```

Note that the *h3_hardware_interface* node now publishes topics and subscribes to topics introduced by the controllers. The subscribed topics accept an effort (torque) command for each joint.

Now, we open a new terminal and publish in the */h3/right_knee_effort_controller/command* topic the torque setpoint.

```
rostopic pub /h3/right_knee_effort_controller/command std_msgs/Float64 "data: -10"
```



You should see that the Exo-H3's right knee moves in the flexion direction. Take in account that the exoskeleton is suspended into a support struct.

12. Internal Stiffness Controller

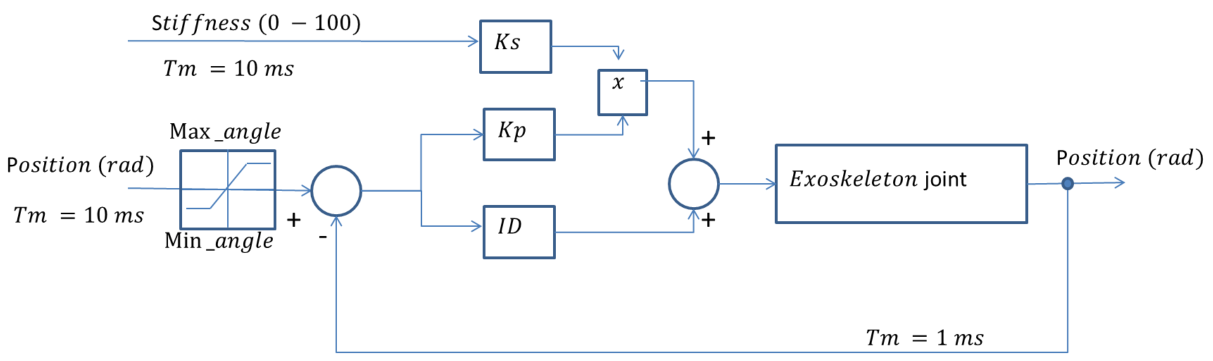


Image 17. Simplified joint stiffness closed-loop control.

The stiffness control loop is a PID position controller with the option to modify the closed-loop control proportional gain as a percentage (0 - 100%).

The Max_angle and Min_angles have the same function that the Position closed-loop control described in the Internal Position Controller chapter.

12.1. Running the ROS Stiffness Controller

Launch h3_hardware_interface node

```
roslaunch h3_hardware_interface h3_hardware_interface.launch
```

Load the position_controllers/StiffnessController in h3_hardware_interface node:

```
roslaunch h3_control_client h3_stiffness_controllers.launch
```

Get the node info:

```
rosnode info /h3/h3_hardware_interface
```

Note that the *h3_hardware_interface* node now publishes topics and subscribes to topics introduced by the controllers. The subscribed topics accept a position and stiffness command for each joint.

Now, we open a new terminal and publish in the */h3/right_knee_stiffness_controller/command* topic the position setpoint.

```
rostopic pub /h3/right_knee_stiffness_controller/command std_msgs/Float64MultiArray "layout:
  dim:
  - label: ''
    size: 0
```

```
   stride: 0
  data_offset: 0
data:
[0.3, 50]"
```

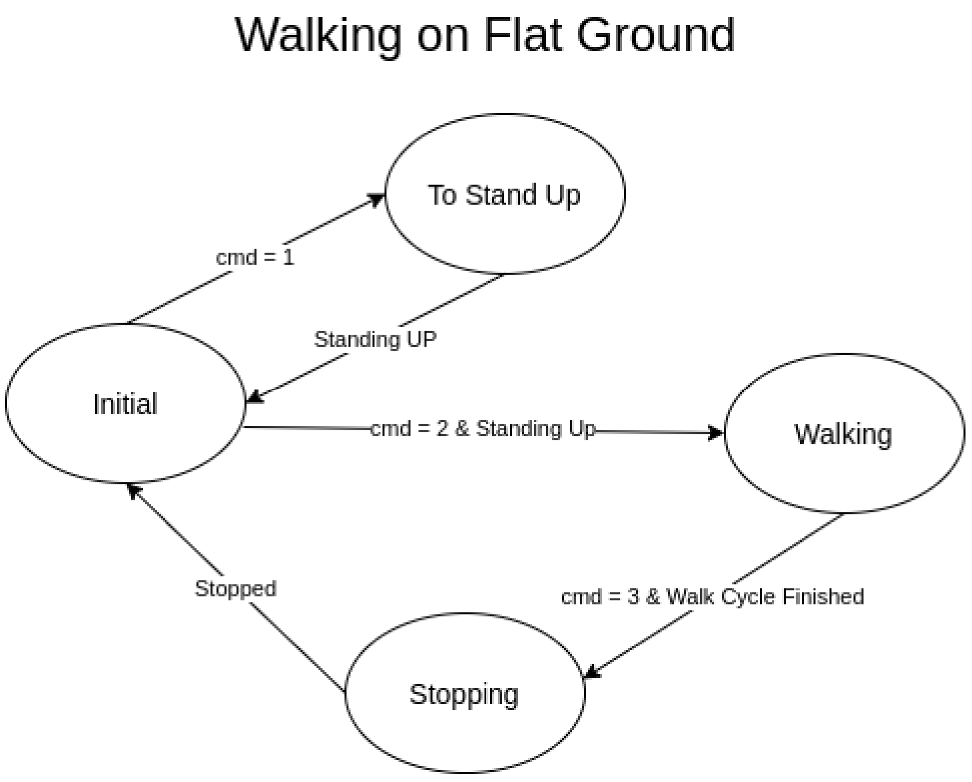
13. Writing and running a ROS custom controller

13.1. Writing a task controller for gait on flat ground

In this example, we go to implement a controller to execute a gait trajectory in the exoskeleton.

The trajectory es composed of two arrays for each joint. The first array specifies the trajectory for the first step, and the second array is the trajectory for the gait cycle. The exoskeleton will start the gait only if it is in the bipedal position.

The controller for gait on the flat ground is according to the following state machine:



States:

Initial: it determines if the exoskeleton is standing up (all joint positions near to zero). If it is the initial state and a command (cmd =1) is received, the exoskeleton moves its joints to the zero position.

Walking: If a command (cmd = 2) is received while the exoskeleton is in a standing position, the exoskeleton begins gait by executing the first step trajectory once and then executing the flat ground gait trajectory periodically.

Stopping: if a command (cmd = 3) is received while the exoskeleton is walking, it ends the walk cycle and then executes the trajectory of the last step. When the trajectory of the last step ends, the exoskeleton returns to its initial state.

We will create a package that implements a ROS node running on the same computer with the *h3_hardware_interface* node. The node will subscribe to the topic */h3 /joint_states* to know the position of the exoskeleton and will publish in the topic */h3/ joint_group_position_controller* the trajectory loaded in the parameter server.

The rqt graph is showed in the following image:

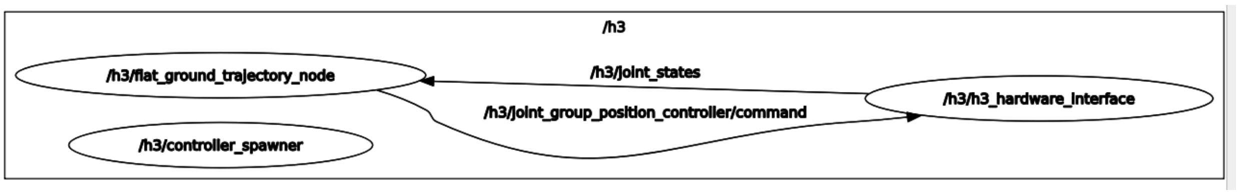

/h3
/h3/flat_ground_trajectory_node
/h3/joint_states
/h3/h3_hardware_interface
/h3/joint_group_position_controller/command
/h3/controller_spawner

Image 18. ROS rqt_graph in the gait trajectory controller example.

13.2. ROS API

13.2.1. Node

- /h3/h3_flat_ground_trajectory_node

13.2.2. Parameters

- sample_time: time between two points of the trajectory.
- first_step/right_hip []
- first_step/right_knee []
- first_step/right_ankle []
- first_step/left_hip []
- first_step/left_knee []
- first_step/left_ankle []
- walking/right_hip []
- walking/right_knee []
- walking/right_ankle []
- walking/left_hip []
- walking/left_knee []
- walking/left_ankle []

13.2.3. Services

13.2.4. Topics

Publications:

- /h3/exo_state [std_msgs/UInt8]
- /h3/joint_group_position_controller/command [std_msgs/Float64MultiArray]

Subscriptions:

- /h3/joint_states [sensor_msgs/JointState]
- /h3/task_command [std_msgs/UInt8]

Testing the node:

Is important that the test be realized in a controlled environment as the exoskeleton joint will move.

Open a new terminal and launch h3_hardware_interface node:

```
roslaunch h3_hardware_interface h3_hardware_interface.launch
```

Open a new terminal and launch:

```
roslaunch h3_control_client h3_group_position_controller.launch
```

Open a new terminal and launch:

```
roslaunch h3_test h3_flat_ground_trajectory.launch
```

Open a new terminal and send a command:

```
rostopic pub /h3/task_command std_msgs/UInt8 "data: 2"
```

Now, you should plot the topics in a *rqt_plot* and see that the exoskeleton executes the gait trajectory. Take into account that the vector order depends on the topic's structures.

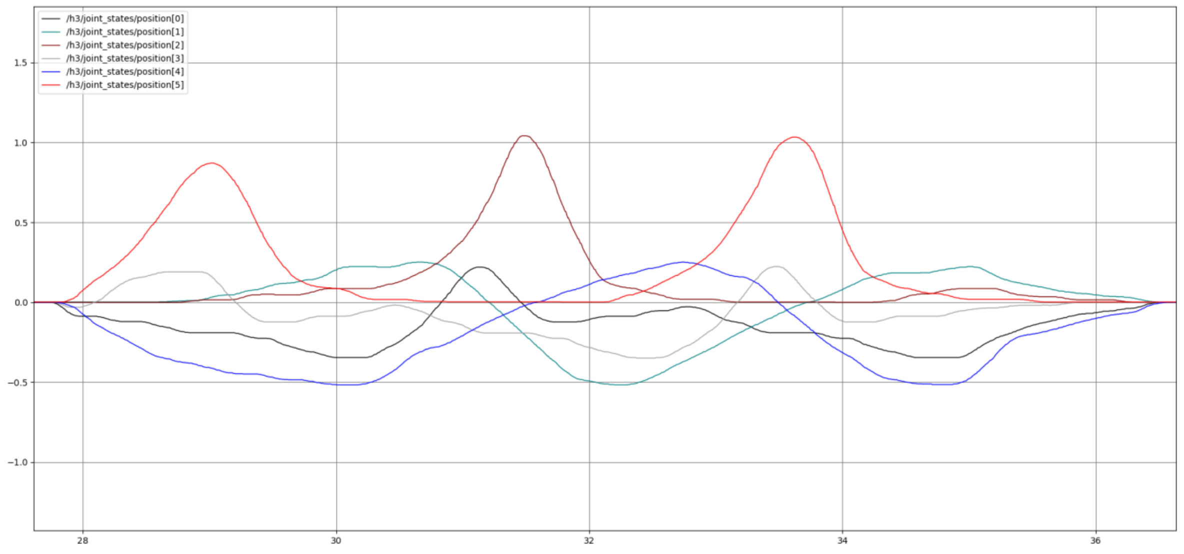


Image 19. Joint position in the flat_ground_trajectory_node execution.

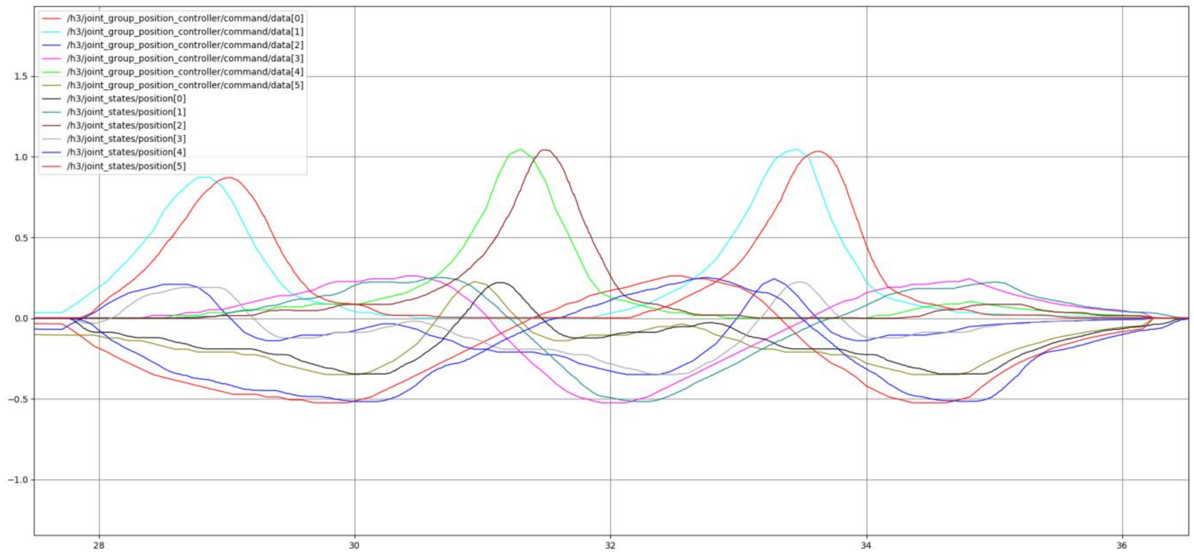


Image 20. Position command and joint position in the flat_ground_trajectory_node execution.

Stop the exoskeleton

```
rostopic pub /h3/task_command std_msgs/UInt8 "data: 3”
```

13.3. The code

You can download the example code from the Technaid Github repository and open it in your favorite code editor.

The node starts getting joints trajectories from the parameter server, and then it executes the machine state logic (transition logic and state logic).

Now let's create a ROS controller that can be loaded by *controller_manager* on node *h3_hardware_interface*. So the state machine will run in the read-update-write loop on node h3_hardware_interface. The flat ground trajectory controller will use h3_command_interface and will have access to all available signals on the Exo-H3. On the other hand, this controller implements the same ROS API as h3_flat_ground_rajectory_node.

Creating the ROS package:

In your ROS *workspace/src* folder create a ROS package:

```
catkin_create_pkg h3_gait_trajectory_controller roscpp pluginlib controller_interface
hardware_interface std_msgs sensor_msgs h3_hardware_interface
```

Using your favorite code editor implements a controller library. Create a *h3_gait_trajectory_controller.h* file in the include directory, and a *h3_gait_trajectory_controller.cpp* file in the *src* directory. You can download this example file from the Technaid Github repository. A class prototype is shown below:

h3_gait_trajectory_controller.h file

```
namespace h3_gait_trajectory_controller

{

class H3GaitTrajectoryController : public
controller_interface::Controller<h3_hardware_interface::H3CommandInterface>

{

public:

H3GaitTrajectoryController() {}

~H3GaitTrajectoryController() {}


/** Controller functions */

virtual bool init(h3_hardware_interface::H3CommandInterface *hw, ros::NodeHandle &root_nh,
ros::NodeHandle &controller_nh);

virtual void starting(const ros::Time &time);

virtual void update(const ros::Time &time, const ros::Duration &period);

virtual void stopping(const ros::Time &time);

//ROS subscriber callback function

void setRobotTask(const std_msgs::UInt8::ConstPtr &cmd);


private:
```

```
std::shared_ptr<realtime_tools::RealtimePublisher<std_msgs::UInt8>> rt_exo_state_pub_;
std::shared_ptr<realtime_tools::RealtimePublisher<std_msgs::Float64MultiArray>>
rt_position_cmd_pub_;
realtime_tools::RealtimeBuffer<TaskCommand> rt_task_command_;
ros::Subscriber sub_;
h3_hardware_interface::H3CommandHandle robot_handle_;
ros::Time last_publish_time_;
ros::Time last_sample_time_;
double publish_rate_;
double sample_time_;
double joint_pos_[6];
double pos_cmd_[6];

/** First Step */
std::vector<double> r_hip_fs_cmd_;
std::vector<double> r_knee_fs_cmd_;
std::vector<double> r_ankle_fs_cmd_;
std::vector<double> l_hip_fs_cmd_;
std::vector<double> l_knee_fs_cmd_;
std::vector<double> l_ankle_fs_cmd_;

/** Walking */
std::vector<double> r_hip_w_cmd_;
std::vector<double> r_knee_w_cmd_;
std::vector<double> r_ankle_w_cmd_;
std::vector<double> l_hip_w_cmd_;
std::vector<double> l_knee_w_cmd_;
std::vector<double> l_ankle_w_cmd_;
int state_ = 0;
int exo_state_ = 0;
int index_w_ = 0;
int index_fs_ = 0;
int index_ls_ = 0;
int index_z_ = 0;

}; // class


// Register plugin

PLUGINLIB_EXPORT_CLASS(h3_gait_trajectory_controller::H3GaitTrajectoryController,
controller_interface::ControllerBase);
} // namespace h3_gait_trajectory_controller
```

Note that our controller inherits from the *controller_interface::Controller* class. This class accepts an *h3_hardware_interface::H3CommandInterface* class that implements the functions to access to read/write in Exo-H3 Main Controller through h3_hardware_interface::H3CommandHandle robot_handle_ object.

Also, take in account that you need register the Controller plugin in *controller_interface::ControllerBase* class. Use the PLUGIN_EXPORT_CLASS macro to do it.

Now, modify the Build section of the CMakeLists.txt file to create a C++ library:

```
## Declare a C++ library

 add_library(${PROJECT_NAME} src/${PROJECT_NAME}/h3_gait_trajectory_controller.cpp)
```

Then, create a *controller_plugin.xml* definition into the project directory:

```
<library path="lib/libh3_gait_trajectory_controller">

<class                                   name="h3_gait_trajectory_controller/H3GaitTrajectoryController"
type="h3_gait_trajectory_controller::H3GaitTrajectoryController"
base_class_type="controller_interface::ControllerBase">

<description>

This is an example of a gait trajectory controller plugin to execute in Exo-H3.

</description>

</class>

</library>
```

Modify the *package.xml file* to include the plugin definition:

```
<!-- The export tag contains other, unspecified, tags -->

<export>

<!-- Other tools can request additional information be placed here -->
```

```
<controller_interface plugin="${prefix}/controller_plugin.xml"/>
</export>
```

Create a *gait_trajectory_controller.yaml file* specifying the controller parameters. Add the *h3_config_controller* to prepare the device for interacts with the internal controller running in Exo-H3 MC.

```
h3:
# Publish all joint states -----------------------------------
joint_state_controller:
type: joint_state_controller/JointStateController
publish_rate: 100 #100
#Configure whole robot
robot_config_controller:
type: h3_config_controller/H3ConfigController
robot_name: exo_H3
control_type: [1, 1, 1, 1, 1, 1]
min_angle: [-1.7453, 0, -0.4363, -1.7453, 0, -0.4363]
max_angle: [0.4363, 1.7453, 0.4363, 0.4363, 1.7453, 0.4363]
assistance: [100, 100, 100, 100, 100, 100]
#Publish all robot data
robot_state_controller:
type: h3_robot_state_controller/H3RobotStateController
robot_name: exo_h3
publish_rate: 100
gait_trajectory_controller:
type: h3_gait_trajectory_controller/H3GaitTrajectoryController
robot_name: exo_h3
publish_rate: 100
sample_time: 0.090
first_step:
walking:
```

Create an *h3_gait_trajectory_controller.launch file* to load controller parameters in the server and launch *controller_spawner* node indicating the controller names:

```
<launch>

<rosparam file="$(find h3_control_client)/config/h3_gait_trajectory_controller.yaml"
command="load"/>

<node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"

output="screen" ns="/h3" args="/h3/joint_state_controller

/h3/joint_gait_trajectory_controller

/h3/robot_config_controller

/h3/robot_state_controller">

</node>

</launch>
```

Launch the h3 gait trajectory controller

First, launch *h3_hardware_interface* node. Open a new terminal and run:

```
roslaunch h3_hardware_interface h3_hardware_interface.launch
```

Next, open a new terminal and launch *controller_spawner* node:

```
roslaunch h3_control_client h3_gait_trajectory_controller.launch
```

Now we can get ROS system information:

```
rosnode info /h3/h3_hardware_interface
--------------------------------------------------------------------------------
Node [/h3/h3_hardware_interface]
Publications:
* /h3/gait_trajectory_controller/exo_state [std_msgs/UInt8]
* /h3/gait_trajectory_controller/position_command [std_msgs/Float64MultiArray]
* /h3/joint_states [sensor_msgs/JointState]
* /h3/robot_states [h3_msgs/State]
* /rosout [rosgraph_msgs/Log]

Subscriptions:
* /h3/gait_trajectory_controller/task_command [unknown type]
```

```
Services:
* /h3/controller_manager/list_controller_types
* /h3/controller_manager/list_controllers
* /h3/controller_manager/load_controller
* /h3/controller_manager/reload_controller_libraries
* /h3/controller_manager/switch_controller
* /h3/controller_manager/unload_controller
* /h3/h3_hardware_interface/get_loggers
* /h3/h3_hardware_interface/set_logger_level
* /h3/robot_config_controller/set_assistance
* /h3/robot_config_controller/set_control_type
* /h3/robot_config_controller/set_data_recording
* /h3/robot_config_controller/set_max_angles
* /h3/robot_config_controller/set_min_angles
* /h3/robot_config_controller/set_trigger_output


contacting node http://technaid-N53SV:38631/ ...
Pid: 3797
Connections:
* topic: /rosout
    * to: /rosout
    * direction: outbound (49105 - 127.0.0.1:43258) [13]
    * transport: TCPROS
```

Note that the gait trajectory controller is executed in the *h3_hardware_interface* node publishing and subscribing to topics to/from a ROS network.

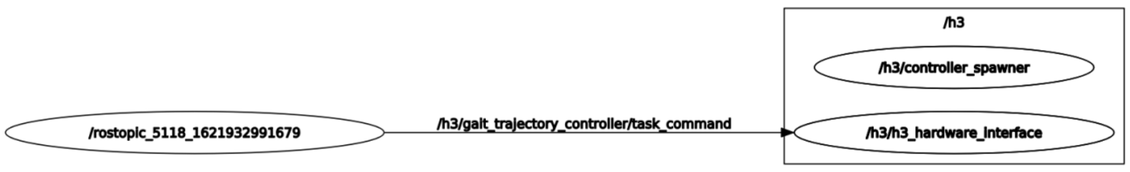

/h3
/h3/controller_spawner
/rostopic_5118_1621932991679
/h3/gait_trajectory_controller/task_command
/h3/h3_hardware_interface

Image 21. ROS rqt_graph in the gait trajectory controller example.

Publish command to start the gait:

```
rostopic pub /h3/gait_trajectory_controller/task_command std_msgs/UInt8 "data: 2"
```

And publish command to stop the gait:

```
rostopic pub /h3/gait_trajectory_controller/task_command std_msgs/UInt8 "data: 3"
```

Plot the position command signal and the joint position to compare to it.

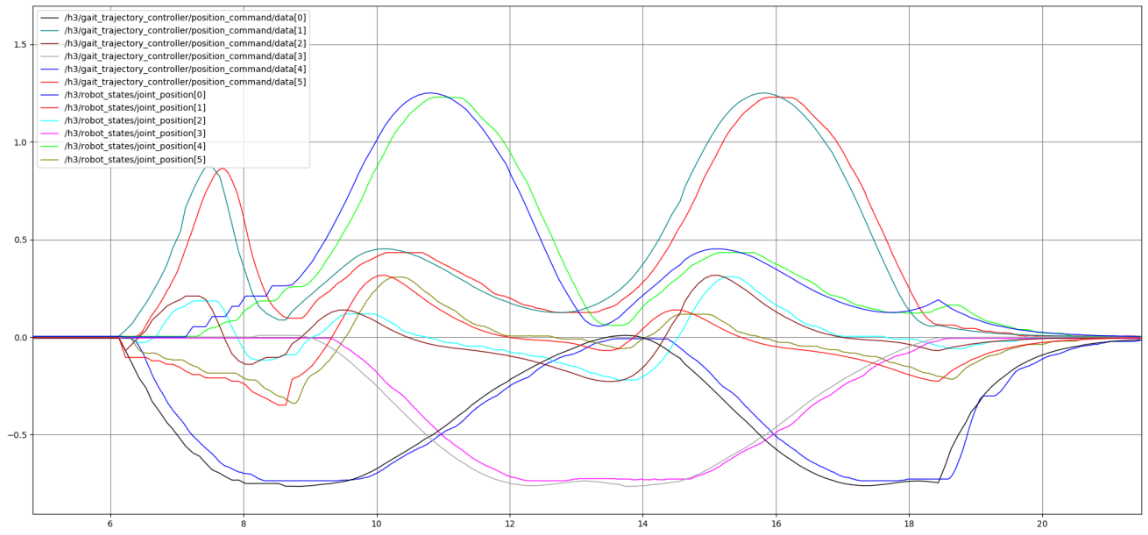


Image 22. Position command and joint position in the gait trajectory controller execution.

14. Technical specifications

Number of degrees of freedom	6 degrees of freedom in the sagittal plane. One for hip, knee, and ankle in right and left legs.		
Type of control	Position control, torque control, and admittance control in real time.		
Reduction gear system	Ratio	160	i
	Repeated peak torque limit	123	T_R [Nm]
	Average torque limit	75	T_A [Nm]
	Torque index at a rate of 2000 rpm	47	T_N [Nm]
	Momentary maximum torque limit	152	T_M [Nm]
	Maximum speed input (Oil Lub)	7500	rpm
	Maximum speed input (Grease Lub)	5600	rpm
	Limit for mean speed input (Oil Lub)	5600	rpm
	Limit for mean speed input (Grease Lub)	3500	rpm
	Moment of inertia	0.282×10^{-4}	[kgm^2]
	Weight	0.24	[kg]
Motor (hips, knees and ankles)	Nominal voltage	18	V
	No load speed	4550	rpm
	No load current	352	mA
	Nominal speed	3930	rpm
	Nominal torque (max. cont. torque)	218	mNm
	Nominal current (max. cont. current)	5.7	A
	Stopping torque	3260	mNm
	Starting current	87	A
	Maximum efficiency	88	%
	Characteristics:		
	Terminal resistance, phase to phase	0.207	Ohm

	Terminal inductance, phase to phase	0.169	mH
	Torque constant	37.5	mNm/A
	Speed constant	255	rpm/V
	Speed/torque gradient	1.41	rpm/mNm
	Mechanical time constant	0.648	ms
	Rotor inertia	44	gcm^2
Final actuators	Net torque	35	Nm
	Peak torque	152	Nm
Communication	External CAN-Bus		
	Wi-Fi 2.4 GHz IEEE 802.11 b/g/n		
	Bluetooth v3.0 (Class 2)		
	2.4 GHz Transceiver		
Power supply (Charger)	100 – 240 V AC / 50-60 Hz (AC power line)		
Battery LiFePO$_4$	Height	14	cm
	Width	10	cm
	Length	16	cm
	Normal capacity type	10.8	Ah
	Normal voltage	19.2	VDC
	Normal energy	230	Wh
	Standard discharge current	2.4(const.)	A
Principal structure production material	Stainless steel and high-strength aluminum (7075).		
Exoskeleton sensors	6x joint position sensors		
	6x torque sensors		
	4x pressure sensors (toe and heel)		
Joint range of motion	Hip (135° - max. 140°)	105°	30° (ext.)

(Flexion/Extension)		(flex.)	
	Knee (110°)	105° (flex.)	5° (ext.)
	Ankle (60°)	30° (flex.)	30° (ext.)
Size variability	Minimum height of subject (*)	110	cm
	Maximum height of subject (*)	210	cm
	Minimum weight of subject	40	kg
	Maximum weight of subject	100	kg
Dimensions	118 cm tall		
	45 cm long (side view)		
	30 cm wide (front view)		
Weight	Approx. 17 kg / 14.2 kg without battery		
*Some size options are only available through optional accessories.			

Table 1. Exo-H3 specifications

Contact

Technaid S.L.
Calle Cabo de la Nao, 2. Nave 12.
28500. Arganda del Rey (Madrid) – Spain.

Telephone: (+34) 918719974
www.technaid.com
support@technaid.com

Release date: 09/06/21

Version 0.0.1