Robotis OP3 quick guide

Filippo M. Smaldone

February 3, 2022

Contents

1 Introduction

The objective of this guide is to provide an easy guide for usage of the Robotis OP3 robot. This manual contains information taken from the official OP3 guide[1] together with specific knowledge acquired by the authors while using the robot. The guide has to be intended as a companion material to that of the GitHub repository FilippoSmald1/Robotis-OP3-MPC-walking , that we will denote as the repository.

Before proceeding, consider that the robot can be accessed and used via a SSH protocol with any operating system, while the OP3 Gazebo simulation can only be run in Ubuntu.

Should you have any question, please contact the authors at:

- smaldone@diag.uniroma1.it
- ...

2 Basic set up

2.1 Turn on the robot

Follow this procedure to turn on the robot:

- Connect the robot to the power supply. Using batteries is not recommended due to the limited operational time (a single battery provides approximately 10 minutes of usage).
- Put the robot in a statically balanced squatting configuration, as in Fig. 1. Note that the robot, in this state, is quite "soft" and tends to sag: there exists only small set of sitting configurations for which the robot is in static equilibrium when turned off.
- Use the switch on the back of the robot to turn it on.
- Wait until the chest led becomes green.

[1] https://emanual.robotis.com/docs/en/platform/op3/introduction/

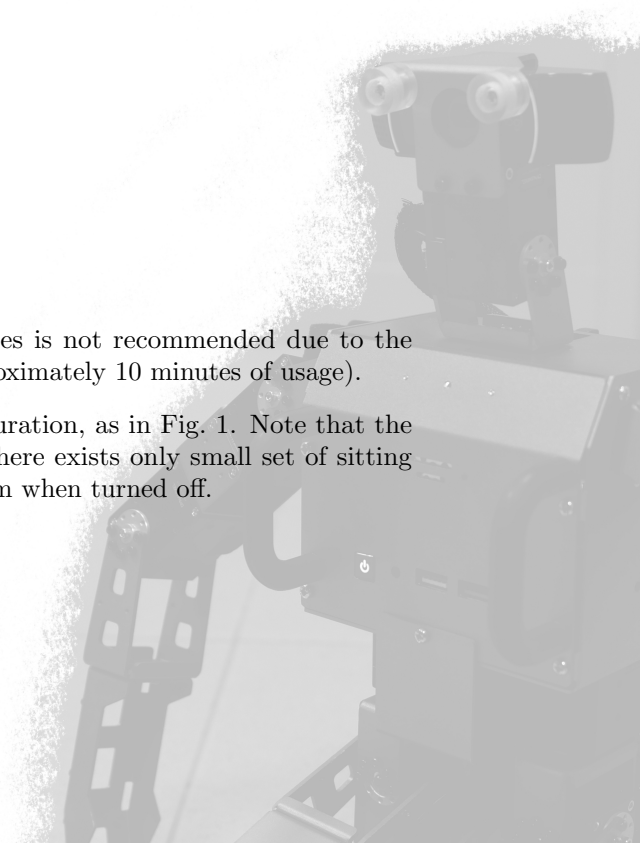


Figure 1: OP3 squatting position.

2.2 Turn off the robot

Follow this procedure to turn off the robot:

- Put the robot in the squatting configuration.
- Press the power button on the robot chest, close to the usb ports. The robot should say: "Bye, Bye!".
- Wait until the blue led of the button becomes red.
- Turn off the robot with using the switch on its back.

3 Remote access to the robot

Follow this procedure to connect to the robot:

- Once the robot is turned on, connect to the WiFi network called "ROBOTIS-OP3-share".
- Open the terminal.
- Use a ssh protocol to connect to OP3, by typing:

  ```
  ssh robotis@10.42.0.1
  ```

  the password is 111111.
- You are inside the robot now! You can check for instance the directories though the terminal by typing the standard command `ls` and change directory with `cd`.
- The robot runs a Linux Mint distribution, all Linux commands can be used! Check: https://ubuntu.com/tutorials/command-line-for-beginners1-overview
- Some further and useful commands are:
  - To edit or create a new file where to implement your software, you can use `nano filename.extension`. For instance, assuming that your directory is an empty folder,

    ```
    nano myScript.cpp
    ```

    will create and open a Cpp file. If instead the folder already contains a file called myScript.cpp, the nano command will just open it.
    The `nano` editor is not intuitive, but it is useful for quick editing, e.g., changing a parameter of your algorithm.
  - To copy and paste a file from your computer to the robot, open a different terminal from you local machine and type:

    ```
    scp <absolute file path in your computer> robotis@10.42.0.1:<target path in the robot>
    ```

    the password is always 111111.
  - To copy a file from the robot to your computer, open a different terminal and type:

    ```
    scp robotis@10.42.0.1:<absolute file path in the robot> <target path in your computer>
    ```

  - Remember that if you want to copy a full directory you have to use the recursive copy, i.e., `scp -r`.

It is also possible to connect a monitor and keyboard to the robot via the available usb ports. To do so, connect the devices to the robot before turning it on.

4 Managers and motion modules

OP3 is natively programmed in C++ and is ROS based[2]. This means two things: the basic code provided by Robotis is in C++, but, being it based on ROS, it is always possible to create and run a ROS node with any supported language.

We focus in this section on the real-time part of the software, responsible for motor commands and joint angle readings. This part, written in C++ for efficiency, has a specific structure that has to be strictly followed.

The sampling time of the system is 8 ms and at each sampling time a certain software routine is executed. Note that the real-time part of the software is written as a ROS package but does not rely on the ROS features to manage the real-time performance. Real-time operations are instead managed by the Robotis Framework, applied to the robot by means of the Robot Manager package. The Robot Manager executes at each time some processes, implemented as Motion Modules.

Wrapping up (and paraphrasing):

- we are considering real-time software, required for the motion of the robot. If you have to do something else just create a ROS package and do it there[3]!
- To implement your real-time software you will need a Robot Manager (an in 99% of the cases it is sufficient to modify an already existing one) and a Motion Module.
- The actual implementation of your software will be in the Motion Module.
- Typical example: if you are implementing a walking algorithm, your implementation (for instance, gait generation plus inverse kinematics) will be organized and written as a Motion Module, which will be called and executed at each sampling by the Robot Manager.
- When you want to run an experiment, you will have to launch[4] the Robot Manager package that contains your Motion Module and that's it!

4.1 Robot Manager

The Robot Manager is a ROS package[5] that applies the Robotis Framework to a robot, whether it is physical or simulated. The following instructions hold equally in both cases.

To inspect the available managers:

- navigate to the directory `catkin_ws/src/ROBOTIS-OP3`.
- Check the packages that have the name manager!
- Suggestion: if you create a new manager, it would be a good idea to use the Manager keyword in its name.

What you typically do with managers:

- Launch a manager with the command:

  ```
  roslaunch package_manager_name launch_file_name.launch
  ```

  for instance to launch the Manager from the repository just type:

  ```
  roslaunch diag_op3_manager diag_op3_manager.launch
  ```

- Modify a manager to include a new Motion Module.

How to modify an existing manager to execute your Motion Module:

- Navigate inside the Manager package that you want to modify.
- Edit the package.xml by adding your Motion Module as dependency

  ```
  <depend>your_motion_module_name</depend>
  ```

[2] http://wiki.ros.org/ff

[3] Example: a planning algorithm that does not need to run with the real-time sampling time.

[4] This word refers to the specific way to execute a ROS package, further instructions will be given later.

[5] See: http://wiki.ros.org/it/ROS/Tutorials/CreatingPackageff

- Edit the CMakeLists.txt by adding the name of your Motion Module inside both the `find_package(...)` and `catkin_package(...)`.
- Navigate inside the `src` directory of the package and open the .cpp file, containing the implementation of the Manager.
- Include your motion module

```
include "your_motion_module_name.h"
```

- ATTENTION: your Motion Module may be in conflict with other Motion Modules not implemented by Robotis (e.g., those that are already there implemented for other experiments). To avoid it, it is recommended to include and use a single custom Motion Module. Commenting out the parts of the code related to other Motion Modules will be sufficient to avoid any issue.
- Add the Motion Module instance to the Robot Controller (a Robotis Framework related thing, do not worry about it) as

```
controller->addMotionModule((MotionModule*) your_motion_module_name::getInstance());
```

  To be sure that you are doing it in the proper location of the script, find similar code lines and add this after them. Do not forget to comment out the lines related to custom Motion Modules (those that you are not including anymore now).

To create a new Robot Manager, you can follow the procedure described in the creating-new-robot-manager section of the Robotis online guide.

4.2 Motion Modules

The Motion Module is where you implement the software that must be executed in real-time, e.g., a balance or walking algorithm. Most likely, all the control related part of your experiment which involves commanding the actuators will be a Motion Module. Each Motion Module will be compiled as a library.

Motion Module structure:

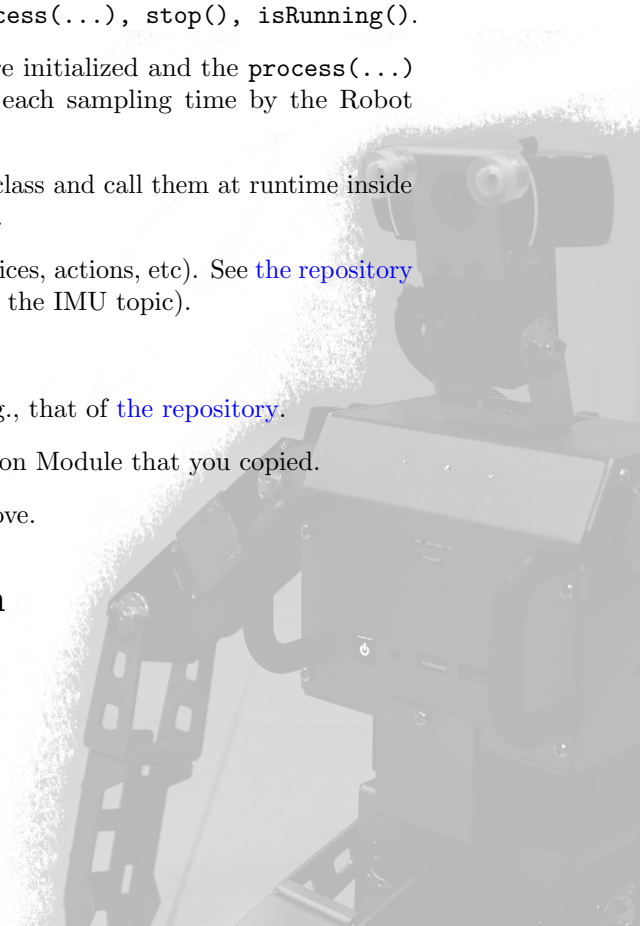
- Each motion module is as a class inheriting from the base Motion Module class, following the singleton pattern. Do not worry about these things, just follow the instructions!
- Beside the constructor and the destructor, there are FOUR methods that characterize the Motion Module: `queueThread()`, `initialize(...)`, `process(...)`, `stop()`, `isRunning()`.
- The `initialize(...)` is typically where the motors are initialized and the `process(...)` contains the actual software routine that is called at each sampling time by the Robot Manager.
- You can create all the methods that you need inside the class and call them at runtime inside the `process` method. See the repository for an example.
- Motion Modules can use all the ROS features (topics, services, actions, etc). See the repository for an example (imu data are accessed by subscribing to the IMU topic).

To create a new Motion Module, my suggestion is to:

- Copy and paste and existing custom Motion Module, e.g., that of the repository.
- Either change all the names or remove the original Motion Module that you copied.
- Remember to update the Robot Manager as written above.

5 Running an experiment/simulation

Follow this procedure to run an experiment/simulation:



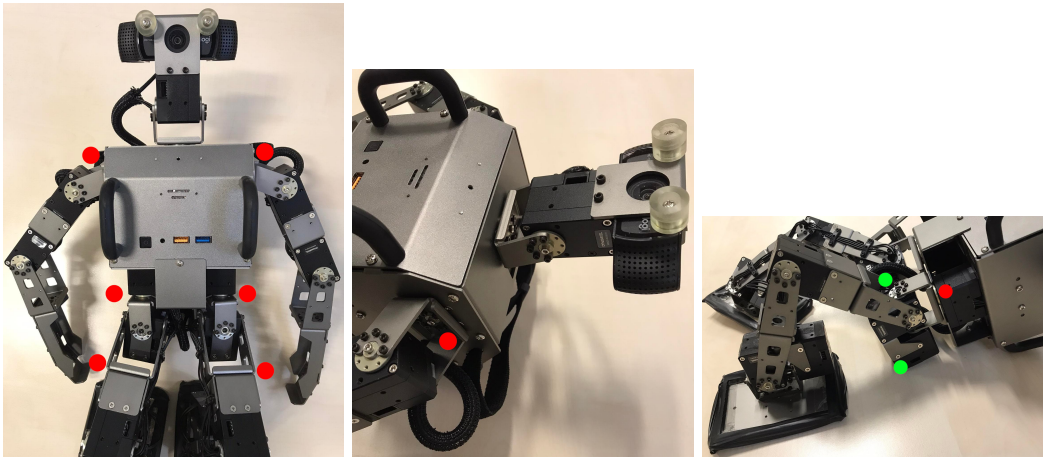


Figure 2: OP3 maintenance tips: critical points (left), arm head detail (center), legs (right).

- ONLY FOR EXPERIMENTS: kill the Manager and some ROS nodes that are activated when booting the robot. In particular, you must kill the active Manager and the demos. We also suggest to kill other nodes that will not be specifically used for the experiment (ball recognition, localization, etc). To check the active nodes

  ```
  rosnode list
  ```

  to kill them

  ```
  rosnode kill /nodename
  ```

- Navigate inside your catkin workspace, typically named as `catkin_ws`.
- Set the environmental variables

  ```
  source devel/setup.bash
  ```

  this things must be done for each terminal you open. Note that you could add it to your .bashrc file (easy tutorials online) for simulations but avoid doing it on the robot.

- From the `catkin_ws` directory, launch the Manager as

  ```
  roslaunch diag_op3_manager diag_op3_manager.launch
  ```

  if instead you want to run a simulation (as in the repository), use

  ```
  roslaunch diag_op3_manager diag_op3_gazebo.launch
  ```

6 Usage and maintenance - READ IT CAREFULLY

OP3 is actuated by position controlled servomotors (different options may be possible but not recommended), implying a certain rigidity during the motions that may cause issues during usage. The Robotis Framework someway accounts for that by providing a safety function that deactivates all the motors in the case of dangerous motions. A further action can be taken by the user, by manually shutting down the motors of the robot with the reset button on the back (there are four buttons in line, the reset is the first starting from the right).

Consider also that OP3 is a low cost open-source platform: you can purchase the single parts from the manufacturer and assemble it by yourself. If this permits you all kind of repairs and upgrades, it inevitably implies less robustness in the overall structure. Some recurring problems are now listed together with their solutions. A criticity level is assigned to each bullet (low, medium, high). Figure 2 reports pictures of the robot with colored dots in correspondence to the joint/links that will be referred to in the following.

- The camera can be displaced if it hurts something (low criticity). Solution: just re-align it so that the sensor is set in correspondence to the head hole. Temporarily loosen the screw on the back of the head may be required.

- The screws tend to melt if high torque is exerted by allen wrenches and/or screwdrivers (high criticity). Solution: NEVER EVER tighten the screws too much. Some screws are already in this situation, but fortunately almost all the operations on the hardware are still possible. To completely solve the issue a drill may be required to remove the ruined screws and place a substitute.

- Arm shoulder joint screw tends to loosen (low criticity). Note that we are referring to the single screw that connects the motor shaft to the link. Solution: slightly tighten them. It is not necessary to dismount the robot arm to do it.

- Leg hip yaw joint screw tends to loosen (high criticity). This is the most important issue of the robot, it happens continuously and there is no permanent solution. Solution: referring to the green points of the center picture of Fig. 2, you must unscrew these joint/link connections to dismount the robot leg and access the hip joint to tighten it (red point). DO NOT TIGHTEN THE SCREW TOO MUCH, BECAUSE IT WOULD INEVITABLY LOOSEN AGAIN DURING THE EXPERIMENTS. HOWEVER, BY TRYING TO TIGHTEN THE SCREW TOO MUCH, THEY CAN GET IRREMEDIABLY DAMAGED AND BECOME IMPOSSIBLE TO BE REMOVED.

  To give you an idea, the operation might be required before every experimental session, as it dramatically reduces the performance of the robot during the walk.

- Hip pitch to knee link can disconnect in its meddle (high criticity). Solution: use some screws to reconnect it. Dismounting the leg at some point may be required.

- The motor emergency shutdown is activated (medium criticity). It happens in case of motor overheating, fall or extreme motions. Solution: do not leave the robot in a standing position if not necessary, as it overheats the motors and can cause a safety motor shutdown. Prepare your experiment environment carefully, trying to reduce the risks of fall. Eventually, avoid programming the robot to execute extreme motions.