HRI Documentation*

Emanuele Frasca

April 18, 2025

Contents

1 Nursing Pepper Documentation

The software toolkit provided by the professor is a pre-configured environment that contains all the necessary software and dependencies to run the project. The toolkit is designed to be used with the Pepper robot and includes the following components:

- NAOqi SDK
- Choregraphe
- Pepper Tools
- ROS Melodic
- qiBullet

1.1 Docker Container

The Docker container defined in the `hri_software/docker` folder is a pre-configured environment that contains all the necessary software and dependencies to run the project.

The folder structure of the Docker container is as follows:

*The Pandoc Markdown source of this document is attached to this PDF file.

```
hri_software/
   docker/
       README.md
       Dockerfile
       build.bash
       run.bash
       dc_[X11|nvidia|vnc].yml
```

- `README.md`: This file contains the instructions for the installation of the Docker container.
- `Dockerfile`: This file contains the instructions to build the Docker image. It defines all the necessary dependencies and sets up the environment for the project, but is not capable of setting runtime parameters like output display and others.
- `build.bash`: This script is used to build the Docker image using the Dockerfile. It installs all the necessary dependencies and sets up the environment for the project.
- `dc_[X11|nvidia|vnc].yml`: These files are docker-compose files that define the services, networks, and volumes for the Docker container. They are used to run the container with different configurations, such as using X11 for display or using NVIDIA GPU support.
- `run.bash`: This script is used to run the Docker container with the specified configuration. It takes one argument, which can be either X11, nvidia, or vnc, depending on the desired configuration and it starts the container calling the associated docker-compose file.

1.1.1 Installation

Install Docker

```
sudo apt install docker.io
```

Install Docker Compose

```
sudo wget -O /usr/local/bin/docker-compose
 ↪ https://github.com/docker/compose/releases/download/1.28.5/docker-compose-Linux-x86_64
sudo chmod +x /usr/local/bin/docker-compose
```

Clone all the repositories and set up the environment

```
# Clone main HRI software
git clone https://bitbucket.org/iocchi/hri_software.git

# Set up Pepper workspace and clone tools
mkdir -p $HOME/src/Pepper && cd $_
```

```
git clone https://bitbucket.org/mtlazaro/pepper_tools.git
git clone https://bitbucket.org/mtlazaro/modim.git

# Create playground directory
mkdir -p $HOME/playground
```

Finally build the Docker image using the `build.bash` script:

```
cd ~/hri_software/docker
./build.bash
```

This will create a Docker image with all the necessary dependencies and configurations for the project. The image will be named `pepperhri` by default.

The installation process can be aso referred to the `docker/README.md` original file from the professor.

1.1.2 Usage

Run the following command to start the Docker container:

```
 cd ~/hri_software/docker
./run.bash [X11|nvidia|vnc]
```

In another terminal, to enter the container's shell, run:

```
docker exec -it pepperhri tmux a
```

1.2 NAOqi SDK

The NAOqi SDK is a powerful library that allows developers to control various Aldebaran/SoftBank robots, including the Pepper robot. The APIs that we are interested in are the Python ones, which are used to control the robot's movements, speech, and other functionalities. The NaoQI SDK provides a set of APIs that allow developers to interact with the robot's hardware and software components.

We can use NAOqi both as a library and as a server. The library is used to write Python scripts that control the robot, while the server is used to run the scripts and communicate with the (virtual) robot.

The core middleware library used to write the NAOqi SDK is called libqi.

1.2.1 Usage

NAOqi follows a client-server architecture, where the NAOqi server typically runs directly on the robot (like Pepper). In this case, however, since we're working in a simulated envi-

ronment on our local machine, we need to manually start the NAOqi server to simulate the robot's behavior.

To start the NaoQi server, you need to run the following command in the docker container terminal:

```
cd /opt/Aldebaran/naoqi-sdk-2.5.7.1-linux64
./naoqi
```

If you want, you can create an alias to make it easier to start the NaoQi server writing only `naoqi` in the terminal. To do this, run the following command in the docker container terminal:

```
alias naoqi='cd /opt/Aldebaran/naoqi-sdk-2.5.7.1-linux64 && ./naoqi'
source ~/.bashrc
```

The problem is that, since this is a container, the alias will be lost when you close the container. So, if you want to keep it, you need to add it to the `~/.bashrc` file in the Dockerfile. Sad.

> NOTE: The NaoQi server is composed of several services, each of which is responsible for a specific functionality. For example, the `ALMotion` service is responsible for controlling the robot's movements, while the `ALTextToSpeech` service is responsible for generating speech. Since we are working in a simulated environment, not all the services are available.

You can also run Choregraphe (Section below), which also starts the NaoQi virtual robot server automatically but in a different port from the default one (9559). This means that you need to specify the port when connecting to the NaoQi server from your Python code, so maybe it's easier to run the NaoQi server manually on the default port.

A simple example (from `src/pepper_tools/say.py`) of how to use the NaoQI SDK to control the Pepper robot is shown below:

```
import qi

# Connect to the robot
app = qi.Application(["SayExample", "--qi-url=tcp://127.0.0.1:9559"])
app.start()
session = app.session

# Get the text-to-speech service
tts = session.service("ALTextToSpeech")

# Say something
tts.say("Hello, world!")
```

Instead of using directly the NaoQI sdk, we can use the `pepper_cmd` library contained in the `src/pepper_tools` folder. This library provides a set of commands to control the robot's movements, speech, and other functionalities in a more user-friendly way. The library is built on top of the NaoQI SDK and provides a higher-level interface for controlling the robot. More on this library can be found in the Pepper Tools section.

1.3 Choregraphe

Choregraphe is a graphical programming environment that allows developers to create and simulate robot behaviors using a visual interface. It provides a drag-and-drop interface for creating and connecting different components, such as sensors, actuators, and behaviors. This is only to simuate behaviors, so it does not take into account the physics of the robot.

To run the Choregraphe application, you need to run the following command in the docker container terminal:

```
cd /opt/Aldebaran/choregraphe-suite-2.5.10.7-linux64
./choregraphe
```

if asked, enter the license key `654e-4564-153c-6518-2f44-7562-206e-4c60-5f47-5f45` to activate the software.

This starts automatically a NaoQi server, but usually not in the standard port (9559). This mean that maybe it's easier to run the NaoQi server manually, as described in the previous section. You can choose to connect directly to the NAOqi server running in the docker container or to the one running in the Choregraphe application.

Since we do not need all the features of Choregraphe, from the `view` menu you can disable all the non-needed features, such as the `Project Objects` and others.

Note: If using the WLS (WLSg 2), audio will not work out of the box, to make it work you will have to modify the Docker Compose file to choose the correct Pulseaudio server socket used by WLSg. This means that you have to add, under the `environment` section of the `dx_x11.yml` file, the following lines:

```
  - PULSE_SERVER=unix:/mnt/wslg/PulseServer
```

Remove the old container

```
docker rm -f pepperhri
```

Rebuild the container

```
cd ~/hri_software/docker
./build.bash
```

Then run the container again using the `run.bash` script. This will allow you to use the audio features of Choregraphe in the WLSg environment (I did not have time to test it, but it should work, I don't even know if Choregraphe has audio features, but in this way, audio from Firefox and other applications should work too).

1.4 Pepper Tools

The `src/Pepper/pepper_tools` folder contains the Pepper Tools library, which is a set of tools and utilities for working with the Pepper robot.

The list of folders in the `src/pepper_tools` folder is as follows:

```
animation, asr, audio, bags, behaviors, cmd_server, demo, faces, grab_image, html, laser, leds,
↪  memory, motion, say, setjointangle, setposture, setstate, slu4p, sonar, tablet, tf, touch
```

1.4.1 say

`python say.py --sentence "Hello World"` does make the robot say "Hello World" using the NAOqi SDK.

1.4.2 cmd_server

The `cmd_server` folder contains a command server that allows you to send commands to the Pepper robot using a simple command-line interface. The command server is built on top of the NaoQI SDK and provides a higher-level interface for controlling the robot.

`pepper_cmd` needs to be used as a library, so it is imported in your script to control the robot and then passed to the `pepper_cmd_server` to be executed using the `pepper_send_program.py` script.

So, the workflow is something like this:

1. Start the NaoQi server in the docker container.

```
cd /opt/Aldebaran/naoqi-sdk-2.5.7.1-linux64
./naoqi
```

2. Run the command server in the docker container:

```
python pepper_cmd_server.py
```

3. In another terminal, send the code of your script to the command server:

```
python pepper_send_program.py --file <your_script.py>
```

4. In the command server you will see the received code and the output of the script. You can also use the inputs defined in your code to simulate and test the robot's behavior.

So, complete workflow is:

```
Your Script using `pepper_cmd` → pepper_send_program.py --program
→ pepper_cmd_server.py → qi session → NAOqi (Pepper)
```

1.5 MODIM

To use MODIM you will have to start two different containers, one for the Pepper robot and one for the NGINX server.

1. Start the NaoQi server in the pepper container:

```
cd /opt/Aldebaran/naoqi-sdk-2.5.7.1-linux64
./naoqi
```

2. Start the MODIM server in the pepper container:

```
cd ~/src/modim/src/GUI
python ws_server.py -robot pepper
```

3. Start the NGINX server container (from your host machine, so not in the pepper container):

```
cd hri_software/docker
./run_nginx.bash $HOME/src/Pepper/modim/demo/sample
```

This will create a docker container with the NGINX server running on port 80. You can access the server by going to `http://localhost:80` in your web browser.

4. Run a modim demo in the pepper container:

```
cd ~/src/modim/demo/sample/scripts
python demo1.py
```

This will run a demo that uses the MODIM server to control the Pepper robot. The demo will use the NGINX server to serve the HTML files and the JavaScript code that is used to control the robot from the python script runned in the previous step.

1.5.1 MODIM in our project

To start the project you will have simply to copy the `$/src/Pepper/modim/demo/sample` folder to your working directory, in this case we will call it `nursing_app`.

```
cp -r ~/src/modim/demo/sample ~/playground/nursing_app
```

Since we are working in the `playground/nursing_app` folder, to start the interaction with the robot, we need to run the following commands, replacing step 3 and 4 with the following:

1. Start the NGINX server container in the `playground/nursing_app` folder (from your host machine, so not in the pepper container):

```
cd  ~/playground/nursing_app
./run_nginx.bash $HOME/playground/nursing_app
```

This will create a docker container with the NGINX server running on port 80. You can access the server by going to `http://localhost:80` in your web browser.

1. Run a modim demo in the pepper container:

```
cd ~/playground/nursing_app/scripts
python start_interaction.py
```

This will run our interaction demo.

1.6 qiBullet

qiBullet is a physics engine that is used to simulate the Pepper robot's movements and interactions with its environment. It is built on top of the PyBullet library, which is a Python wrapper for the Bullet physics engine.

This simulator seem to not use the NAOqi SDK, but it is a standalone library that can be used to simulate the robot's movements and interactions with its environment.

Probably is useful only in combination with its ROS interface, which is probably not needed for our project.

1.6.1 Pybullet Data

If you need some objects to make a phisical simulation, you can find some importing the `pybullet_data` package. The objects are stored as URDF files, which are a standard format for representing 3D models in robotics.

A simple example of how to use the `pybullet_data` package to load a URDF file is shown below:

```
import pybullet as p
import pybullet_data

# Set up the folder that contains the URDF files
p.setAdditionalSearchPath(pybullet_data.getDataPath())

# Load the URDF file
p.loadURDF("plane.urdf")
p.loadURDF("cube.urdf", basePosition=[0, 0, 1])
```

This code sets up the search path for the URDF files and loads a plane and a cube into the simulation environment. The `basePosition` parameter specifies the position of the cube in the simulation world.