Python - Setup Guide for Pepper Robots

Dennis C. Zhuang
Sebastian R. Mason

Contents

Documentation: http://doc.aldebaran.com/2-4/naoqi/index.html

Aldebaran Python-SDK

This tutorial is meant to help you set up the Python SDK from Aldebaran. This will allow you to directly connect to the Pepper Robots through the NAOqi API using Python.

Installing the API requires you to download python 2.7. This has to be done as Pepper does not support any version later than 2.7.

This tutorial is a slightly re-written version of Softbank Robotics' Python SDK - Installation Guide. If any problems occur Softbank Robotics also provide a trouble-shooting page.

Windows Setup

Step 1 Make sure to download Python 2.7 - **32 bits** (or 64-bit depending on your OS). The direct page link to the installer can be found here.

(a) Make sure that you know in which directory you install Python 2.7. I prefer to have my python versions installed in `.../AppData/Local/Programs/Python/`.

(b) Navigate to your `Python27` folder and edit the name of the `python.exe` file to whatever you wish to have it called when you call it in your terminal. This could eg. be `python2.exe`.

(c) Add your newly installed Python 2.7 to your PATH. This should include:
`.../Programs/Python/Python27`
`.../Programs/Python/Python27/Scripts`
`.../Programs/Python/Python27/Lib/site-packages/pip`

Check if everything works by opening up a terminal and writing `python2`. If everything works it should show you a Python 2.7.x terminal. If you named your Python 2.7 executable `python2.exe` you run Python 2.7 in the command prompt using `python2`. If you run a specific file, you follow the convention `python2 file.py`.

Step 2 Now install NAOqi.

(a) Use the following hyperlink to install the Pepper SDK 2.8.6 - Python 2.7-2.8 SDK for Windows and extract the `.zip` file.

(b) For the sake of niceties, we will rename the extracted file as `python-sdk`. Take the extracted file and paste it into,
`.../Programs/Python/Python27/Lib/site-packages/`

(c) Now we wish to configure our System Environment Variables again. Create a new environment variable called `PYTHONPATH`, and the variable value should be the `path/to/python-sdk/lib`. Following the above method, it should look something like this,
`.../Programs/Python/Python27/Lib/site-packages/python-sdk/lib`

Step 3 Download and install the Microsoft Visual C++ 2010 Redistributable Package (x86).

Step 4 Open your Python 2.7.x terminal and run

- `import naoqi`
- `import qi`

If no errors are displayed, then congratulations, you completed the installation and are now ready to work with the Pepper Robots.

Mac Setup

Step 1 Make sure to download Python 2.7 - **32 bits**. The direct page link to the installer can be found here.

Make sure to use Python from `/usr/local/bin/python`, not `/usr/bin/python`.

Before continuing to step 2, make sure your Python 2.7 installation works and is callable.

Step 2 Now to install NAOqi.

(a) Use the following hyperlink to install the Pepper SDK 2.5.10 - Python 2.7 SDK for Mac and extract the file (clicking the link will download a `.tar` file).

(b) For the sake of niceties, we will rename the extracted file as `python-sdk`. Take the extracted file and paste it into, `.../Python27/Lib/site-packages/`

(c) Now we wish to add our newly install NAOqi to our system variables. Run the following two commands,

```
$ export PYTHONPATH=${PYTHONPATH}:/path/to/python-sdk/lib/python2.7/site-
    packages
```

Step 3 Use your newly installed Python 2.7 from `/usr/local/bin/python`, and try and import the following two libraries

- `naoqi`
- `qi`

If no errors are displayed, then congratulations, you completed the installation and are now ready to work with the Pepper Robots.

Step 4 To spare ourselves from calling (c) in **Step 2** every time we open a new terminal, we can ensure that every terminal instance can run the pythonsdk upon opening a new terminal. This is done by adding the *'export PYTHONPATH=...'* command as a new line in your `.zshrc` file (if you use `zsh`). You can use any text editor to edit the `.zshrc` file (we used vim), which is typically located in the user home directory (i.e. `/Users/user/.zshrc`). Here is an example of the `.zshrc` file after adding the path (see line 15):

```
1  # >>> conda initialize >>>
2  # !! Contents within this block are managed by 'conda init' !!
3  __conda_setup="$('/Users/sebastianmason/miniconda3/bin/conda' 'shell.zsh' 'hook' 2>
       /dev/null)"
4  if [ $? -eq 0 ]; then
5      eval "$__conda_setup"
6  else
7      if [ -f "/Users/sebastianmason/miniconda3/etc/profile.d/conda.sh" ]; then
8          . "/Users/sebastianmason/miniconda3/etc/profile.d/conda.sh"
9      else
10         export PATH="/Users/sebastianmason/miniconda3/bin:$PATH"
11     fi
12 fi
13 unset __conda_setup
14 # <<< conda initialize <<<
15 export PYTHONPATH=${PYTHONPATH}:/Users/sebastianmason/Downloads/pythonsdk/lib/
       python2.7/site-packages
```

Linux Setup

Step 1 Download Python 2.7 - **32 bits**. There are many distributions shipped with Python3 by default, so you should make sure to install the python2 package.

Before continuing to step 2, make sure your Python 2.7 installation works and is callable.

Try to run: `sudo apt-get install python2.7-dev`

Otherwise, contact a TA.

Step 2 Now to install NAOqi.

(a) Use the following hyperlink to install the Pepper SDK 2.5.10 - Python 2.7 SDK for Linux and extract the file.

(b) For the sake of niceties, we will rename the extracted file as `python-sdk`.

(c) Now we wish to add our newly install NAOqi to our system variables. Set the environment variable `PYTHONPATH` to, `/path/to/python-sdk/lib/python2.7/site-packages`

By doing for example:

```
$ export PYTHONPATH=${PYTHONPATH}:/path/to/python-sdk/lib/python2.7/site-
    packages
```

Step 3 Use your newly installed Python 2.7 and simply try and import the following two libraries,

- `naoqi`
- `qi`

If no errors are displayed, then congratulations, you completed the installation and are now ready to work with the Pepper Robots.

Server, client and more

The MAvis assignment is created in `Python3`, so we offer a server and a client that convert commands from `Python3` to `Python2` for execution using the NAOqi API.

This setup includes two scripts: `robot_server.py` and `robot_interface.py`. Both contain a similar robot class with various functions. To understand these functions, examine the scripts individually, but reviewing `robot_interface.py` should provide a general idea. In summary, it's essential to know the following:

- `robot_server.py` needs to be run in `Python2` and waits for commands from `robot_interface.py`. It then executes these commands on the robot using the API.
- `robot_interface.py` runs in `Python3` and simply sends instructions directly to `robot_server.py`.

Both the server and client rely on additional packages. These can be found in `robot_server.py` and `robot_interface.py`. For more information look at both script's comment at the top.

Additionally, we include `robot_config.json` to keep track of each robots' credentials and the connection ports for the server (run locally) and the vision (run on the robot). In short, the `robot_config.json` contains the username and password for each robot, and assigns two port numbers; `port` for the server (to communicate `Python3` to `Python2`) and `vision_port` for the camera stream (this will updated each time you run the server, and just makes sure that our improved camera stream is being fetched correctly).

Also, make sure that you are using the Pepper WiFi connection:

Network

[WiFi Name] : Pepper
[Password] : 60169283

Server

The `robot_server.py` file is located in the *root* folder (with the `server.jar`, `README.md`, and the newly added `robot_config.json`). To let the server know which robot to connect to, you need to provide your robot's IP address. Since robots can have multiple connections, avoid connecting to robots from other groups. If you're unsure of your robot's IP, press the button on Pepper's stomach, which is behind the tablet. Pepper will then announce the IP - be ready to write it down! To start the server, open a terminal in the root folder and run the following command (shown with an example IP):

```
> python2 robot_server.py 192.168.1.100
```

The server will then connect to the robot with the given IP and will wait for commands.

Oh no, my robot is missing!

If we are unlucky, the robots can alter their IP addresses. When this happens, the `robot_config.json` is outdated, and `robot_server.py` will raise an exception: *Robot's IP not in configuration file, please update the configuration file with the correct robot IP*. Consequently, we have to reset the robots IP in the configuration. However, we keep the username and password for each robot on their back - written on a small sticky note! You can use that to identify your robot and update the config file.

My server is bugging out!

In rare cases, or if you run `robot_server.py` concurrently, you will get the following exception: *socket.error: [Errno 48] Address already in use*. Most of the time this is fixed by (1) making sure that every instance of `robot_server.py` is closed and then (2) waiting a few seconds running the server again. If the problem persist, you can always manually change the `port` value in the config.

Client

The client is contained in the *searchclient* folder and is called `robot_interface.py`. This should be imported into your `robot.py` that is executed at the end of your `searchclient.py`. You can test if the `server` and `client` is working by running the main script of `robot_interface.py` in a terminal from the root folder (**remember** *the server has to be running!*):

```
python3 searchclient/robot_interface.py 192.168.1.100
```

If the robot introduced itself, you are now ready to implement your robot agent type in the search client!

Whisper Speech Recognition

We'll use OpenAI's Whisper for speech recognition because it works well in noisy situations. Whisper requires Python 3.8-3.10 (3.11 is not supported). You can get any suitable version from the official site: https://www.python.org/downloads/

To set up Whisper, follow the guide here (read carefully): https://github.com/openai/whisper

Although Whisper handles noise well, stand near your robot when talking. The first time you use Whisper, loading the base model may take a while. This only happens once per session. Here's a code example from the demo that transcribes a temporary `test.wav` file in the `tmp` folder (this is where the `robot.listen()` function from `robot_interface.py` will save to):

```python
from robot_client import *
import whisper
import os

# Load the model
model = whisper.load_model('base')

# Get transcription
text = model.transcribe(str(os.getcwd())+"/tmp/test.wav")['text'].lower()

# Print resulting text
print(text)
```

Listing 1: Python example

Localization issues

You may observe that the robot's navigational accuracy as slightly lacking. It's also very sensitive to its starting position within the cells. These inaccuracies can accumulate and potentially cause frustration. By utilizing the `VisionStreamThread` class found in `robot_interface.py‘` you can obtain data on the nearest apriltag visible to the robot (specifically via the camera located below the mouth). This information can be used to implement a basic controller by completing the `localization_controller()`function in `robot_interface.py`.

Remember that `localization_controller()` must utilize an active `VisionStreamThread` as a parameter, which you can assign by using the `instantiate_vision_processes` function (refer to the $\_\_main\_\_$ script in `robot_interface.py` for further details).

A solution might be to develop a controller that begins by aligning the robot to the closest apriltag, followed by ensuring the correct orientation through a continual loop (you may need to specify an epsilon for both centering and orientation to help determine completion). Immediately after every n actions in the action plan, you can call the controller to help mitigate the cumulative error. One thing you may discover is that certain actions cause more substantial errors than others. This could be an important factor to consider when deciding the point to activate the controller during the execution of a plan.

Adding more functions

If you'd like to try new things and add more features to the server/client, go ahead. You can find the complete NAOqi API proxies here. You can also include other features not in the API. To do this easily, follow the general procedure in `robot_server.py` and `robot_interface.py`. If you have interesting ideas but are unsure if they can work, ask the Robot TA for help.