Pepper Workshop using Python

Note: the code for the exercises can be found on GitLab:

https://gitlab.enterpriselab.ch/RobLab /exercises

Contents

1. Talk and Move

Note: This tutorial does not require a physical robot.

Goal: - Get familiar with the motion of Pepper and the text-to-speech API.
- Know how to execute sequential and parallel calls.

Task: Define a route Pepper should walk and try to get back to the initial position.

1. Make sure you have installed python-naoqi and the python naoqi wrapper. Look at the Python file `PepperConfiguration.py` that can be found in the wrapper. It contains the IP addresses, ports and passwords of the three Pepper robots. If you do not have a real robot at hand, you can use a simulated robot. To do so, open Choregraphe and connect to a virtual robot (*Connection > Connect to virtual robot*). You can steer the virtual robot with your python code. Open the file `talk_and_move.py` and create a configuration object for your robot with the port for the simulated robot. You can find the port in *Choregraphe > Edit > Preferences > Virtual Robot* Tab.

```
port = 54643
robot = Robot(PepperConfiguration("simulated_robot", "localhost", port))
```

   The Robot class automatically connects to the robot and provides proxies to the robot APIs that are all in the folder `naoqi_proxy_pyhton_classes` of the wrapper. Using these proxy classes makes programming easier. They allow you to work with auto completion and docstrings in your IDE.

2. Now you can access the robot's text-to-speech and motion APIs:

```
# create proxy for text to speech and motion api
tts = robot.ALTextToSpeech
motion = robot.ALMotion

# let the robot talk and move in sequence
tts.say("i cannot move while i'm talking")
motion.moveTo(1, 0, 0)
tts.say("now i'm done")
```

   Check if your virtual robot moves in Choregraphe (*Robot view*) and talks (*Dialog window*). Observe that both the text-to-speech and the motion are blocking, which means that the robot is not talking and moving at the same time.

3. How can you make Pepper speak and talk at the same time? `Qi.Async` provides such functionalities:

```
# let the robot talk and move in parallel
qi.async(tts.say, "I'm starting to move now. Can you see how I can move and
    talk at the same time?")
future = qi.async(motion.moveTo, -1, 0, 0)
future.wait()
tts.say("ok, I'm done")
```

4. Run the program and check in Choregraphe if the virtual robot moves and talks as expected.

5. Create now your own path for pepper. Try to turn around and come back to the original position. Test it on a real Pepper, but be careful! Make sure Pepper doesn't collide with obstacles.

2. Modulate Pepper’s voice

Goal: - Be able to modulate Pepper’s voice.
- Learn how to organize peppers functionalities in your own classes.

Task: - Let Pepper say something normally, slowly or loudly.

1. In the previous exercise, we scripted our behavior in one python file. For larger projects, it is recommended to organize your code in a better way. In this exercise, we will create a new class Dialog that implements certain functions. Open the file `dialog.py` with the following class definition:

```
class Dialog:

    def __init__(self, robot):
        self.__al_tts = robot.ALTextToSpeech

    def say(self, text):
        self.__al_tts.say(text)
```

2. We can use the Dialog class in our `ex2_voice.py` file:

```
dialog = Dialog(robot)
dialog.say("how are you today?")
```

3. So far, it is not clear why we created our own Dialog class. However, suppose you want say something slowly. We therefore need to change some speech parameters, say something, and restore the original speech parameters. This can be nicely done in our Dialog class:

```
def say_slowly(self, text):
    original_speed = self.__al_tts.getParameter("speed")
    self.__al_tts.setParameter("speed", 50)
    self.__al_tts.say(text)
    self.__al_tts.setParameter("speed", original_speed)
```

Implement this method and test it with the `ex2_voice.py` to say “I am tired”.

4. Now imagine you want Pepper to shout something. To increase the volume, you cannot use the ALTextToSpeech API, you need to use the ALAudioDevice API. This is not very intuitive, but we can hide this inconvenience from the `ex2_voice.py` script by implementing the method `shout` in the `Dialog` class.

First, we need to include the ALAudioDevice in the Dialog class. Add

```
self.__al_audio = robot.ALAudioDevice
```

in the constructor (`__init__`). The function `shout` could look like this:

```
def shout(self, text):
    original_volume = self.__al_audio.getOutputVolume()
    self.__al_audio.setOutputVolume(min(original_volume + 20, 100))
    original_pitch = self.__al_tts.getParameter("pitch")
    pitch = 80
    self.__al_tts.setParameter("pitch", pitch)
    self.__al_tts.say(text)
    self.__al_tts.setParameter("pitch", original_pitch)
    self.__al_audio.setOutputVolume(original_volume)
```

5. When we use

```
dialog.shout("what are you doing here?")
```

in the `ex2_voice.py` script, we don't have to care about different APIs nor restoring the original speech settings. Implement the `shout` method and test it by executing the `ex2_voice.py` script. Do you have other ideas for modulating Pepper's voice? Implement your own function in the dialog class!

3. Dialog

Goal: Get familiar with the basic functions of the QiChat engine.

Task: Create a simple application that responds to a predefined user input.

In this exercise, we will extend our Dialog class. There's an ALDialog API provided by Softbank. With this API, you can create interactive conversations using the QiChat engine.

First, we will create a simple human-robot dialog where the robot will react to a user's input. The qi syntax is:

```
u:(hello) hello human, pleased to meet you
```

If the user says "`hello`", the robot will respond "`hello human, pleased to meet you`". We can define this rule in a topic called introduction. The language in this example is set to English (enu). Make sure that the robot's language is set to English as well.

```
topic: ~introduction ()
language: enu
u:(hello) hello human, pleased to meet you
```

We can either create a text file called `introduction.top` or implement this topic in python. Here we will use the latter approach.

1. In the `Dialog` class, define the following function:

```
def add_simple_reaction(self, topic_name, user_input, robot_output):

    topic_content = ('topic: ~' + topic_name + '()\n'
          'language: enu\n'
          'u:(' + user_input + ') ' + robot_output + '\n')
```

2. We have to load the topic content to the ALDialog API and return the name of the topic:

```
self.__al_dialog.loadTopicContent(topic_content)
```

3. Since there is no proxy class for the ALDialog API, we have to access the API directly. Define the local variable `__al_dialog` in the constructor and open a session with an arbitrary id:

```
self.__al_dialog = robot.session.service("ALDialog")
self.__my_id = 123
self.__al_dialog.openSession(self.__my_id)
```

4. We have to remember the name of the topic, since it is needed to activate, deactivate and finally unload the topic content. In the Dialog class, create a method `start_topic(topic_name)` that activates the topic, as well as a method `stop_topic(topic_name)`, that deactivates the topic and unloads it (see the ALDialog API manual for details). In the `start_topic` method, you should

call the ALDialog.subscribe() method and set the focus on your `topic_name`. If Pepper shows its blue eyes, it is listening.

5. Test your code by executing the file `ex3_simple_interaction.py`.

4. Yes or No Question

Goal: Learn how Pepper can ask a yes or no question and react to the user's answer.

Task: Let the user chose if he/she wants to start playing a game.

We will implement this with the ALDialog API.

If Pepper should start the conversation, a 'proposal' can be used:

```
topic: ~introduction ()
language: enu
proposal: hello human, are you ready to play a game?
```

Using a user subrule, we can react differently if the users says yes or no:

```
    u1: (no) what a pity
    u1: (yes) great, let's start the game
```

1. Create a method `load_yes_no_question(topic_name, question, reaction_yes, reaction_no)` in the Dialog class that loads the topic and returns its name.

2. To ask the question, you have to activate the topic, set the focus to the topic and trigger the proposal by forcing an output:

```
  def ask_yes_no_question(self, topic):
    self.__al_dialog.activateTopic(topic)
    self.__al_dialog.subscribe('myself')
    self.__al_dialog.setFocus(topic)
    self.__al_dialog.forceOutput()  # start proposal sentence
    time.sleep(5)
    self.__al_dialog.deactivateTopic(topic)
```

   Try your code with the `ex4_yes_no_question.py` file. Does it work?

3. It can be useful to save the user's choice as a variable. This can be done by setting a variable `agree` in the subrule using the following syntax:

```
u1: (no) what a pity $agree=0
u1: (yes) great, let's start the game $agree=1
```

   You can fetch the variable `agree` in the `ask_yes_no_question` method in the Dialog class:

```
  do_agree = self.__al_dialog.getUserData("agree", self.__my_id)
```

   Return this value and print the result in the `ex4_yes_no_question.py` file.

More resources:

The syntax of the QiChat is explained in more detail here:
http://doc.aldebaran.com/2-5/naoqi/interaction/dialog/dialog-syntax_full.html#dialog-rules
http://doc.aldebaran.com/2-5/naoqi/interaction/dialog/aldialog_syntax_cheat_sheet.html

http://doc.aldebaran.com/2-5/naoqi/interaction/dialog/aldialog-api.html

5. Speech Recognition

Goal: Learn how you can make Pepper react to certain keywords.

Task: Create a behavior that can be started and stopped by saying “let’s start” and “stop”.

Consider you want Pepper to react to certain words throughout your behavior. For instance, it may be useful that your behavior stops if you say “Stop”. In this exercise, we will make Pepper listen to two keywords, that allow us to start and stop our behavior.

We will add a vocabulary to the ALSpeechRecognition API. Whenever a word is recognized, the event “`WordRecognized`” is triggered in the ALMemory API. We can connect a callback method to this signal and react according to the word that was recognized.

To begin with, we create a class `SpeechRecognition` that takes care of setting the vocabulary and connecting the speech recognition with a callback function:

```
class SpeechRecognition(object):

    def __init__(self, robot, vocabulary, callback):
        memory = robot.session.service("ALMemory")
        self.subscriber = memory.subscriber("WordRecognized")
        self.subscriber.signal.connect(callback)
        self.__speech_recognition =robot.session.service("ALSpeechRecognition")
        # need to pause speech recognition to set parameters
        self.__speech_recognition.pause(True)
        self.__speech_recognition.setLanguage("English")
        self.__speech_recognition.setVocabulary(vocabulary, False)
        self.__speech_recognition.pause(False)
        self.__speech_recognition.subscribe("SpeechDetection")
        print('Speech recognition engine started')
```

The `subscribe()` method starts the speech recognition engine and causes the module to start writing information to the ALMemory, specifically to the event “`WordRecognized`”. We should also add an unsubscribe method to stop the speech recognition engine:

```
def unsubscribe(self):
    self.__speech_recognition.unsubscribe("SpeechDetection")
    print('Speech recognition engine stopped')
```

Look at the file `ex5_speech_recognition.py`. It contains an application class with a constructor that uses the `SpeechRecognition` class defined above with the vocabulary “let’s start” and “stop”. We will create the speech callback function now. It receives a variable called `value` that contains the word that was recognized and its estimated accuracy:

```
def __speech_callback(self, value):
    print("recognized the following word:" + value[0] + " with accuracy: "
                    + str(value[1]))
    if value[0] == "let's start":
```

```
                if value[1] > 0.35:
                    print "received start signal"
                    self.__start = True
            if value[0] == "stop":
                if value[1] > 0.35:
                    print "received stop signal"
                    self.__stop = True
```

Now it is your turn to create the `run` method. Make use of the `__start` and `__stop` variables that are set by the callback function to start your behavior. Here, your behavior can be simulated by making Pepper say something. Make sure to stop the speech recognition when stopping your program.

6. Take a Picture

Goal: - Learn how to take a picture with Pepper's camera.
- Learn how to transfer files between Pepper and your computer.

Task: Use one of Pepper's cameras to take a picture and transfer it to your computer.

1. NaoQi provides an ALPhotoCapture API to take pictures with the different cameras. So first, we create new class `Camera` in the file `camera.py`. We also add method to set the camera id, the resolution and the picture format with help of the vision_definitions:

```
import vision_definitions

class Camera:

    __camera_id = vision_definitions.kTopCamera
    __resolution = vision_definitions.kVGA # 640x480px
    __picture_format = "jpg"

    def __init__(self, robot):
        self.__al_photo = robot.ALPhotoCapture
        self.configure_camera(self.__camera_id, self.__resolution,
self.__picture_format)

    def configure_camera(self, camera_id, resolution, format):
        self.__al_photo.setCameraID(camera_id)
        self.__al_photo.setResolution(resolution)
        self.__al_photo.setPictureFormat(format)
```

2. To take a picture, we simply add the method `take_picture`:

```
def take_picture(self, path, file_name):
    self.__al_photo.takePicture(path, file_name)
```

3. Now we can test our code. To do that, we execute the file `ex6_take_picture.py`. If you are connected to a real robot, `my_picture.jpg` should have been created in the remote folder path on Pepper. Try to find the file on Pepper using ssh or Putty. How can you use another camera of Pepper? How do you use a different image resolution?

4. Due to the limited computing power of Pepper, it may be useful to transfer a picture to your local computer for image processing. It is possible to create a SSH client in Python using Paramiko. In PyCharm, go to *File > Settings > Project Interpreter* and click on the plus icon. Search for Paramiko and add the package.

5. To implement the file transfer, we create a new class called `FileTransfer` in `file_transfer.py`:

```
import paramiko

class FileTransfer():

    def __init__(self, robot):
        self.ssh = paramiko.SSHClient()
        self.ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        self.ssh.connect(robot.configuration.Ip, username=robot.configuration.Username,
password=robot.configuration.Password)
```

   We add the function `close()`

```
  def close(self):
      self.ssh.close()
```

   And add the function `get(remote, local):`

```
  def get(self, remote, local):
      sftp = self.ssh.open_sftp()
      sftp.get(remote, local)
      sftp.remove(remote)
      sftp.close()
```

6. This class allows us to transfer the picture easily. The image is saved in the folder `/home/nao/recordings/cameras/`, and the target path is in this example `C:\\work\` in Windows. In the file `ex6_take_picture.py`, transfer the image to your local computer.

   Can you find the file on your computer?

7. Show a Picture on the Tablet

Goal: Learn how to display a picture on Pepper's tablet.

Task: Use one of Pepper's cameras to take a picture and show it on its tablet.

1. Taking an image and displaying it on the tablet is not as easy as one could think. One has to put the image in a folder on Pepper's hard disk that can be accessed by the tablet. The IP address of Pepper from the tablet is 198.18.0.1. Via this IP address, one can access the folder `/opt/Aldebaran/var/www/apps/`. One can create a folder there to place the image that was taken. To make it even more complicated, the method provided by ALTablet showImage automatically searches for the file in the subfolder html of the path that is given as an argument. Therefore, we have to put the image in the folder `/apps/my_app/html/` and call the method `showImage` with the path /`apps/my_app/`.

   To start, we take a picture and save it in the standard location on Pepper (`/recordings/cameras/`) with the `Camera` class we defined above.

2. To hide all the complicated details from `ex7_show_picture_on_tablet.py`, we create a new file `tablet.py` with the following scaffold:

```
import qi
import os
import shutil


class Tablet:

    __pepper_path = "/opt/aldebaran/var/www/apps/"
    __path_from_tablet_to_pepper = "http://198.18.0.1/apps/"

    def __init__(self, robot, app_name = "my_app"):
        self.__al_tablet = robot.session.service("ALTabletService")
        self.__al_tablet.loadApplication(app_name)
        self.__app_name = app_name
        self.__create_directories()

    def __create_directories(self):
        if not os.path.exists(self.__pepper_path+self.__app_name):
            os.mkdir(self.__pepper_path+self.__app_name)
        if not os.path.exists(self.__pepper_path + self.__app_name+ "/html"):
            os.mkdir(self.__pepper_path+self.__app_name + "/html")

    def __remove_directories(self):
        shutil.rmtree(self.__pepper_path+self.__app_name)

    def close(self):
        self.__remove_directories()
```

3. Now it's your turn to implement the methods `show_image(self, path, file)` and `hide_image(self)`. I recommend to use `showImageNoCache` from the tablet API. As described above, you need to copy the image from `path` to `__pepper_path + self.__app_name + "/html/"`.

4. You need to execute the script `ex7_show_pictrue_on_tablet.py` locally on Pepper. Copy the files to a temporary folder on Pepper (eg: `/home/nao/tmp/`) with `scp` and execute it locally with `ssh` and running

```
> python ex7_show_picture_on_tablet.py
```

   Did it work as expected?