Programming Nao as an Educational Agent: A Comparison between Choregraphe and Python SDK

Anushka Subedi[1] , Dipesh Pandey[1] and Deepti Mishra[1][0000−0001−5144−3811]

Educational Technology Laboratory,
Interdisciplinary Computing Research (ICR) Group,
Department of Computer Science (IDI),
Norwegian University of Science and Technology, Gjøvik, Norway.
anushkas@stud.ntnu.no, dipeshp@stud.ntnu.no, deepti.mishra@ntnu.no

Abstract. Programming a humanoid robot for educational purpose is a demanding task for a beginner with little experience. Several studies are available in which humanoid robots such as NAO, are used in educational settings to move, recognize objects and hold conversations similar to a human. These studies usually incorporate third party libraries and advanced deep-learning methods making it difficult for a beginner to follow. This paper aims to work as a getting-started guide for someone starting out with programming the NAO robot using Choregraphe and the Python SDK. In this study, NAO robot is used to implement four scenarios based on - dialog, movement, object recognition and obstacle avoidance - using the available components that come with the robot. The paper focuses on comparing the Choregraphe and NAO Python SDK during this process by considering the advantages and limitations of both approaches. The results show that both Choregraphe and the Python SDK have their nuances and their usage depends on the use case. However, for a beginner just starting out, Choregraphe is easier to get things done without writing a single line of code. Python, on the other hand is useful for low-level functionalities and provides rather more flexibility.

Keywords: Humanoid robot · NAO · Educational agent · Programming · Choregraphe · NAO Python SDK

1 Introduction

Humanoid robots have been used in recent years in educational settings to facilitate teaching primary level students with the aim to increase their interest in robotics and programming. The commercial humanoid robots available at present do a good job at looking and acting like a human, but they are still not autonomous enough to navigate and interact freely without rule-based classical programming. The interaction is especially challenging in social robots where interaction usually goes beyond proto-social responses requiring multi-modal aspect of social interaction [1] and needs further research [2].

NAO is a humanoid robot created by SoftBank Robotics designed to work as an assistant to educators. Using several project based learning approaches, NAO is often used to develop problem solving and analytical skills among the students. It can be programmed using Choregraphe which provides a graphical interface with drag-and-drop features as well as using the Python SDK. However, the incomplete and difficult-to-follow documentation presents a number of issues for a beginner, resulting in technological complexity that necessitates spending a lot of time attempting to figure out the best practices rather than actually implementing them. Most of the available research [1][3][4] describes the overall experiment and findings without offering specifics on the technical implementation aspect. Although there are studies [5][6][7][8][9][10][11] already been done with NAO robot using Choregraphe and Python SDK to a large extent producing good results, they are, by no means, easy to explore for a beginner.

With this motivation, this study explores and documents the technical possibilities of NAO in order to implement scenarios to be utilized in a primary educational setting. During this process, two programming approaches are used - Choregraphe and Software Development Kit (SDK) in Python - to implement four scenarios that are centered around dialog, movement, object detection and obstacle detection. The details of these four scenarios are:

- **Dialog**: To create an interactive quiz where NAO asks question and reacts to the answers
- **Movement**: To move around and show different interactive postures
- **Object detection and recognition**: To recognize people and greet them by their name
- **Obstacle detection and avoidance**: To help NAO to navigate in a maze

This paper is organized as follows: firstly related work is described. In the next section, the technical aspects of key functions of NAO are explored using the two programming approaches, along with the comparison of these approaches used to implement the scenarios listed above. The paper finally concludes with suggestions to assist other academics and developers who are starting with programming NAO robot.

2 Related work

Multiple studies have used Choreograph and/or Python SDK with mixed results. Dundar's work [5] on an education robot that teaches the basics of the Italian language includes a comprehensive description of the speech system of NAO and its limitations. Due to performance issues with NAO's in-build speech recognition module, he ended up using Google's speech recognition. In terms of ease of use, however, NAO's speech recognition is a lot easier to operate. Similarly, Liu's implementation of a human following robot [6] is done fully with the Python SDK with the help of some third-party modules. On the other hand, Amberkar's work [7] involves using NAO to recognize hand-signs and is based on programming the robot using Choregraphe.

Mondou et al. [8] created a serious game using NAO robot that allows youth to discover the ethnographic artifacts of a natural history museum in a playful manner. Although the interactions are relatively simple and contains only three questions, they found Choregraphe effective but constraining for their purpose. The possible modifications of the dialogues and, the content or the re-scheduling of the interactions proved to be laborious and hazardous along with another limitation regarding temporality of the experience if they wish to be able to apply a duration period to a particular event in the scenario [8]. Considering these limitations, researchers switched to Python SDK. Zeng et al. [9] implemented a virtual instructor based on a NAO robot using Choregraphe and realized that preprocessing is required for speech enhancement to minimize noise, improve voice clarity and achieve higher speech quality in the real experimental environment. Further, they embedded the open artificial intelligence platform iFLYTEK with NAO for speech recognition and synthesis. Recently, Hsu et al. [10] implemented two scenarios - first, NAO dancing on music using Choregraphe, and second, NAO locating and kicking a ball using Python SDK combined with OpenCV. The results showed that the motion and image recognition of the NAO robot was successful however, the success rate of accurate kicking is not high enough therefore they suggest to combine image recognition with deep learning techniques to increase the success rate.

3 Implementation and Discussion

Four scenarios were implemented in which dialog was used for an interactive quiz, movement was used to enhance the experience using several gestures mimicking real human, object detection was used to recognize the participants and greet them by their name, and obstacle detection to move NAO around a group of students without manual instructions.

Figure 1 shows the architecture diagram employed during the experiment. The exercises were designed partly in Choregraphe and the Python programming environment. The blue arrows in the diagram represent the areas where Choregraphe was used, whereas the black arrows indicate the parts that involved Python as the choice of programming method. The operator, as shown, could only intervene using dialog panel in Choregraphe to enforce the correct input to the robot in case of incorrect speech/object recognition.

3.1 Getting Started

Choregraphe

Getting started with Choregraphe is very easy and takes comparatively less time. One needs to download the Choregraphe Suite Installer from the official SoftBank Robotics Community. After the installation, the second step would be to connect Choregraphe to a robot, which is well documented in the official website.

Python SDK

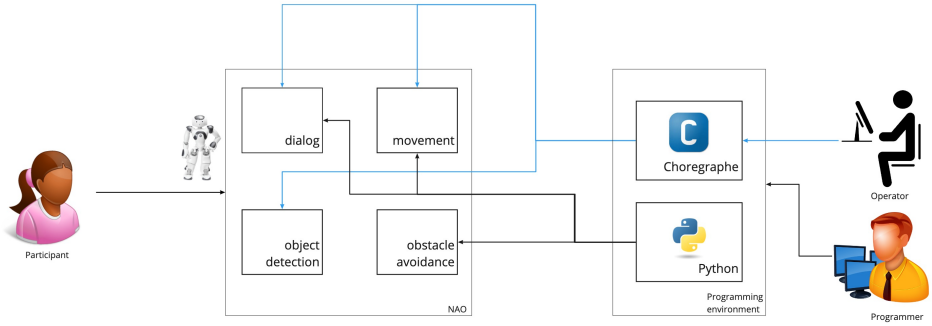

Fig. 1. Architecture diagram

In order to install the SDK, we followed the steps mentioned in this installation guide. Although the documentation available is quite comprehensive, it can be challenging for beginners to get started because it is only available for Python 2.7 while the current standard is 3+. Additionally, getting it to run inside a virtual environment is not possible, so the SDK has to be installed globally, which is not the best practice.

Table 1 presents the comparison of setting up Choregraphe and Python SDK.

Table 1. Comparison of Choregraphe and Python SDK to get started

Items	**Choregraphe**	**Python SDK**
Documentation Complexity	easy	easy
Flexibility	easy	medium
Setup	easy	hard

3.2 Dialog

NAO's dialog system was used to create a simple question-answer system for grade 2-3 students. For this, we experimented with two methods available in the robot:

1. Text To Speech (TTS) + Speech Recognition
2. QiChat with Dialog (top) files

For the TTS approach, a set of words that should be recognized by the speech recognition engine should be provided before starting recognition. However an alternative was required because this approach does not work well as the vocabulary size grows. QiChat seemed to a better option for us because we can feed

script files with the flexibility of moving the control flow between topics. Dialog in NAO has features such as Concepts which can represent a set of common ideas to be reused when required and Proposal rules, which can help trigger a specific robot output without any user input beforehand. An example of the conversation (.top) file with Concepts and Proposal is shown below:

```
\label{Listing}
topic: ~YuckDuck()
language: enu

concept: (correct_answer)  ^rand["Amazing that is correct",
    "Correct answer, I am impressed"]

proposal: %test Let's play a quiz. Which level are you from?
        u1:(level one) Perfect. The book is Yuck Stuck in the
     Muck. Can you tell me who chased the duck?
            u2:(dog) That's correct. $dance=1 ^goto(second)
            u2:("Can we start again?")  ^sameProposal
            ...
proposal: %second  For the next question, do you mind
    touching my head?
          u1:  ([e:FrontTactilTouched e:MiddleTactilTouched e
    :RearTactileTouched]) The second question is, In the
    story, Was chuck able to pull the dog out?
          ...
```

Listing 1.1. An example of a conversation in a .top file

Implementing the Dialog system in SDK requires the top (conversation) file to be uploaded to NAO's hard drive every time it has to be modified. The file needs to be manually uploaded to '/home/nao/.local/share/PackageManager/apps/' directory of the robot. This has not been documented well in the official SDK documentation and a lot of time was spent figuring out the correct location to upload the file for the dialog system to work. While this can be done using GUI FTP programs like Filezilla, WinSCP, etc., when the .top file is modified, we needed a better solution that would automate the upload process. Eventually, we used FTP packages like paramiko, ftplib, etc. to do the transfer before running the program. For anyone who is looking to solve this specific problem, the code is free to access at this repository.

On the other hand, the same thing in Choregraphe requires just a single click on the play button and the .top file gets uploaded to the required location.

There was another challenge related with determining the voice recognition engine's confidence threshold level. The documentation indicates three types of Grammars (BNF, Remote, and SLM) available in NAO's Dialog system, as well as techniques to adjust their confidence levels, but changing the values did not produce the desired outcomes. We couldn't discover any information on how changing the combination of these numbers affects speech recognition, so we had to rely on trial and error to find the workable value. We believe we are yet

to find the right combination. Prior experience with grammar types and how they work might help developers solve this issue.

Choregraphe also offers a Dialog panel that allows an operator to enter human input as text rather than speaking to a real robot. This function comes in handy when the robot is unable to grasp what the speaker is saying (for smooth communication during the experiment). The Dialog panel also include a visualization of human input as understood by the robot along with the level of confidence in the recognition. Due to the lack of these functionalities in the Python SDK, an operator is unable to assist if the dialogue does not progress as planned.

3.3 Movement

In the Python SDK, programming the robot to do basic movements like gestures, hand movements, dancing, etc. require to access each and every position of the body parts and apply mathematical transformations like translation, rotation, etc. on them. This may not ideal in a scenario which requires programming different kinds of movements in a short time. The general aim of social robots is to involve students and expose them to various capabilities of the Robot. For this, Choregraphe comes with ready-to-use movements like dancing, Tai-Chi, gestures, etc. While a drag and drop feature may make Robot do Tai-chi, a simple Arm movement in Python SDK would require a code as follows:

```
# Arms motion
effectorList = ["LArm", "RArm"]
space        = motion.FRAME_ROBOT

pathList     = [
                  [[0.0,   0.08,  0.14, 0.0, 0.0, 0.0], # target 1 for "LArm"]
                  , ...,
                  [[0.0,   0.05, -0.07, 0.0, 0.0, 0.0], # target 1 for "RArm"]
                   ...
                 ]
axisMaskList = [almath.AXIS_MASK_VEL, # for "LArm"
                 almath.AXIS_MASK_VEL] # for "RArm"
coef       = 1.5
timesList  = [ [coef*(i+1) for i in range(5)],  # for "LArm" in seconds
               [coef*(i+1) for i in range(6)] ] # for "RArm" in seconds

isAbsolute   = False
# called Cartesian interpolation
proxy.positionInterpolations(effectorList, space, pathList, axisMaskList, timesList, isAbsolute)
```

Listing 1.2. An example of simple arm movement in Python SDK

The above code does a full movement of the left arm and the right arm of the robot. As mentioned before, before calling the 'positionInterpolations( )' function, we have to specify the pathList of each actuator in the arms in the form of '(x,y,z,wx,wy,wz)' which is an absolute vector of 6D position arrays in meters and radians. This shows that we have to keep track of the position and angles of each and every actuator we want to move and work with them individually.

On the other hand, in order to create new movements, Choregraphe also provides an animation interface that comes with a timeline editor allowing us to edit the behavior of each and every actuator in a timeline without having to deal with angles and degrees.

3.4 Object Detection and Recognition

Using a robot to recognize people, and greet them by their names would help in creating rapport between child and robot. We used the NAO's camera to take photos of people and used them to train the new faces from the Choregraphe interface. The training process is oversimplified and is not so generic in Choregraphe, meaning that recognizing a person's face with different facial expressions would not work so well. We easily obtained workable results with objects that are symmetric, e.g. a red colored ball, however, training to detect non-symmetric objects was a bit non-intuitive because we had to capture every single image from NAO's camera, crop, and label them. Also, we could not find in the documentation which object detection/recognition models are used underneath the robot. In Choregraphe, we tried use cases where NAO is trained to recognize students from pictures, and greets them in real time. Several environments were tried. The result observed is shown in Table 2:

Table 2. Object Recognition test to recognize faces using NAO

Action	**Recognized**	**Not recognized**
Recorded Person A's face and Person A showed his face	✓	
For training, showed Person B's picture, same picture shown	✓	
For Training, showed Person B's picture on a phone, real Person B shows face		✓
For Training, showed Person A's face and then a picture on a phone of Person A was shown to recognize		✓
For training, showed NAO is one posture, for recognition showed NAO in different posture		✓
Same person different expressions		✓

Possible Reasons for this could be:

1. The tests were performed with a limited set of images. So, it is difficult to generalize for every scenarios. The result may be different if there were enough training data.
2. The pictures were taken in a regular room without considering the distance between the camera and the object. Also, better results could have been obtained if proper post processing of images like cropping, rotating etc were performed to reduce background noise.
3. We could not find the documentation explaining ways to modify the resolution using Choregraphe as a result the images taken were of default resolution which is 640 x 480 while the maximum resolution possible is 1280 x 960 according to their website.

This process can be made more end-to-end and explainable using a Python based solution which is as follows:

1. Use Python to capture images from NAO's camera
2. Create a dataset from those images
3. Use a machine learning model to train using the dataset
4. Use the robot's camera to get real time video feeds and using Python SDK, predict the label of the object in the feed

3.5 Obstacle Avoidance

The idea was to show NAO in an autopilot mode trying to find its way on its own and engage students. A simple maze was constructed and the robot was made to avoid collision on the wall. This was done using sonar sensors of NAO. NAO provides obstacle detection based on a two channel sonar system. This enables it to calculate the distance between it and potential impediments in its environment.

The algorithm for path finding was followed to make NAO move from the start to the end. This approach is used to find way through the maze, while choosing how to turn at intersections as follows:

```
for each robot state:
  get sonar_value
  if sonar_value < threshold:
    turn right (-90 degrees)
    get sonar_value
    if sonar_value < threshold:
      turn the opposite direction (180 degrees)
      get sonar_value
      if sonar_value < threshold:
        turn left (+90 degrees)
      else:
        move straight for 10cm
        goto line 2:
    else:
      move straight for 10cm
```

```
16        goto line 2:
17    else:
18      move straight for 10cm
19      goto line 2:
20    if exit found:
21        break
```

Listing 1.3. Path finding algorithm

Implementing this algorithm in Python was comparatively easier because it involves more of core programming concepts like loops and conditions. The similar implementation was tried in Choregraphe which provides boxes for loops and conditions. However, it looked cluttered making it difficult to debug.

4 Conclusion

The results of our experiments show that Choregraphe provides a clear edge over Python SDK when the programmer is looking to get the results quickly and with less effort. We could produce good results in the dialog, movement, and object detection experiments with Choregraphe, while developing an obstacle avoidance program was easier with Python. The fact that we do not have to think about the organization of programming modules, subscribing and unsubscribing them, etc. makes things easier in Choregraphe. On the other hand, one has to be specifically aware of these things while developing the same in Python.

In general, we think that it is not easy to follow the documentation for both Choregraphe and the Python SDK. There are also not enough examples available, especially in Python, forcing us to revert back to Choregraphe in most of the cases. This does not, however, mean that it is extremely difficult to work with the Python SDK provided one has sufficient time to explore and experiment. There is a comprehensive documentation for each of its class modules including all the available methods and variables in their official website however there is a lack of easy to follow examples to use them. This was most evident in our 'dialog' use case where we successfully implemented the QiChat based dialog system with top file in Python. Nevertheless, when there was a need for human intervention during the dialog, we could not do it with Python and had to revert back to Choregraphe, which provides a built in panel for the same.

In addition to documentation, we also analyzed how the two approaches differ when it comes to testing the code and integrating different modules. Testing the code was easier in the Choregraphe environment with all the ready-to-use modules available, whereas testing using the Python SDK was not as straightforward. In terms of integration, however, we felt that Python SDK approach was more flexible. One scenario where it was the most contrasting was during the object detection. While the entire workflow for object detection in Choregraphe felt end-to-end, we felt that adding additional components to the mix, e.g. a deep-learning model to improve detection, would be more difficult. Notwithstanding the fact that we did not implement the same in Python SDK, we believe that because of the flexibility that comes with a well adopted programming language

in Python and availability of lots of libraries, it is easier to integrate different modules and make them work together.

Although we believe that our experience can be a good reference for academicians and practitioners looking to get started with programming NAO, it needs to be stated that it should be taken with scrutiny since the results are based on the experiments performed with specific scenarios in a limited time frame. Therefore, further exploration is necessary in this regard. Ideally, as a next step, with ample time at hand, the Python SDK can be explored in detail and these use cases can be further refined.

References

1. Raquel Ros, Marco Nalin, Rachel Wood, Paul Baxter, Rosemarijn Looije, Yannis Demiris, Tony Belpaeme, Alessio Giusti, and Clara Pozzi. Child-robot interaction in the wild: Advice to the aspiring experimenter. In *Proceedings of the 13th International Conference on Multimodal Interfaces*, ICMI '11, page 335–342, New York, NY, USA, 2011. Association for Computing Machinery.
2. Deepti Mishra, Karen Parish, Ricardo Gregorio Lugo, and Hao Wang. A framework for using humanoid robots in the school learning environment. *Electronics*, 10(6):756, 2021.
3. Dora Matić and Zdenko Kovačić. Nao robot as demonstrator of rehabilitation exercises after fractures of hands. In *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6. IEEE, 2019.
4. Daniel C Tozadore, Adam HM Pinto, Caetano M Ranieri, Murillo R Batista, and Roseli AF Romero. Tablets and humanoid robots as engaging platforms for teaching languages. In *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, pages 1–6. IEEE, 2017.
5. Yigit Can Dündar. A robot system for personalized language education. implementation and evaluation of a language education system built on a robot. *Munin.uit.no*, 2020.
6. Shujun Liu. A human-following nao robot using python programming language. *Theseus.fi*, 2019.
7. Mayuresh Amberkar. Humanoid robot handling hand-signs recognition. *Munin.uit.no*, 2020.
8. Damien Mondou, Armelle Prigent, and Arnaud Revel. A dynamic scenario by remote supervision: a serious game in the museum with a nao robot. In *International Conference on Advances in Computer Entertainment*, pages 103–116. Springer, 2017.
9. Ni Zeng, Shijue Zheng, Jun Zhou, and Qingyu Cai. Design and implementation of virtual instructor based on nao robot. In *International Conference on Mechatronics and Intelligent Robotics*, pages 464–469. Springer, 2017.
10. Roy Chaoming Hsu, Yu-Pi Lin, Chi-Jun Lin, and Li-shun Lai. Humanoid robots for searching and kicking a ball and dancing. In *2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, pages 385–387. IEEE, 2020.
11. Mohd Azfar Miskam, Syamimi Shamsuddin, Hanafiah Yussof, Abdul Rahman Omar, and Muhammad Zaiyad Muda. Programming platform for nao robot in cognitive interaction applications. In *2014 IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, pages 141–146, 2014.