Universal Journal of Electrical and Electronic Engineering 6(4): 214-222, 2019 http://www.hrpub.org
DOI: 10.13189/ujeee.2019.060404

Intermediate Programming Methodologies for Manipulating Modern Humanoid Robots

Wright J. R., Jr.[1,*], Ginter E. S.[2], David B. G.[1], Kilbourne B. J.[3], Wells J. R.[4]

[1]College of Science & Technology, Millersville University, United States
[2]Hexcel Corporation, United States
[3]Andritz Fabric & Rolls, United States
[4]Progressive Engineering Group, United States

Received May 26, 2019; Revised August 04, 2019; Accepted August 15, 2019

Abstract The future for robotics in today's world is clear. We must automate our manufacturing processes to remain competitive in the global marketplace. Modern robotics is an important part of that automation. Humanoids, a more recent development in robotics, have a host of capabilities that may reveal potential new uses in industry and modern engineering education. This paper describes how one may advance from a beginning user of an NAO robot to that of an intermediate user of the Choregraphe development software. The authors seek to bridge the gap between the available documentation for the novice and the advanced users/programmers of the NAO humanoid platform within the Choregraphe user environment. The NAO platform was utilized due to its high impact of use by more than 13,000 educators and researchers from more than 70 countries worldwide [1].

Keywords Humanoid, Programming, Methodologies

1. Introduction

Robotics technology has experienced tremendous advancement since initial industry employment in the 1960s. While simple routine robotic tasks such as spot welding and casting extraction were once considered "high tech," twenty-first century industrial robotics demands more human-like, if not superhuman, capability. Such robots will not only have greater data processing capability, articulation, and mobility, but likely will replicate more human traits including those that have social and emotional aspects. According to Waytz & Norton, robots will soon be assuming such tasks as taxi drivers, real-estate agents, library assistants, sports referees, and health-care assistants [2]. Robots that take on the shape of humans or animals, as demonstrated by Boston Dynamic's Atlas humanoid robot, or the canine-like AlphaDog robot, show the rapid progress currently underway in this arena.

"Robots have primarily been used in automotive plants, but this is changing as other industries adopt them. For example, food-grade robots that meet United States Department of Agriculture (USDA) standards have been introduced, making robots practical in these applications. Robots can be applied in new projects and in existing manufacturing plants. Robots provide an advantage for improving existing plant efficiencies since they can easily be added to improve operations without redesign of machines and production lines. Robots can perform repetitive manufacturing functions with accuracy and precision every time. Robots can also be used in place of humans in areas where there are safety and health hazards. Since robots can be programmed to do multiple tasks, they are good for achieving flexible manufacturing requirements. Robots are not just for large manufacturing companies; I have had discussions with owners of companies with fewer than 150 employees that are using robots to improve productivity and be competitive" [3].

With rapid technological advancements and broad applications of robotics, the demand for qualified personnel such as robotics technicians, technologists, and engineers is already great and is expected to persist well into the future. According to Lombardi, a hiring demand database, there were 45,000 jobs for robotics-skilled professionals advertised online during 2013 [4]. Of those, 16,000 positions were for robotics professionals needed in the healthcare industry. Furthermore, according to the Robotics Industries Association, it could be learned that "only about 10% of the U.S. companies that could benefit from robots have installed any so far" [5]. A search of the jobs database indeed turned up numerous opening for individuals with skills that match those of the proposed new major. The manufacturing sector is again growing in the United States, with automated and robotic systems at the forefront of "advanced manufacturing."

Students studying robotics need the academic preparation necessary for a twenty-first century industry in which multiple technologies, including more human-like traits, is utilized in the design and implementation of robots requiring advanced programming capability. An online resource of the Occupational Information Network (O•Net) identifies computers, electronics, design, machines/tools, mathematics, physics, and production and processing among the more important areas of knowledge for robotics personnel. However, dominating those knowledge areas, according to O•Net information, is overall expertise in "practical application of engineering science and technology. This includes applying principles, techniques, procedures, and equipment to the design and production of various goods and services" [6]. In other words, in addition to discipline-specific training, higher-level cognitive skills (including critical thinking, analysis, and problem-solving) are critical skills for robotics professionals.

Figure 1. The NAO Robot [7]

The future of robotics is bright, with emerging technology and innovative applications poised to make it one of the more pervasive instruments of our technological society. "North American robot orders to non-automotive companies surged to record highs through the first nine months of 2018, according to Robotic Industries Association (RIA), the industry's trade group" [8]. The demand for a trained workforce (from Robotics Technicians to Robotics Engineers) will continue to increase. Educational institutions must develop or advance programs to ensure students are prepared to meet workforce demand with the advanced skill sets necessary for successful employment and professional growth. One way they can do this is to embrace the use of humanoid robotic platforms such as the Softbank Robotics NAO robot (see Figure 1.) in addition to industrial robots in the engineering classrooms and laboratories.

Beginning users of the NAO typically utilize standard functions or blocks of code called "boxes". These boxes are objects that can be strung together to control the robot. Choregraphe is essentially an object-oriented programming software. Intermediate and advanced users can access and modify the root Python code for these objects by double clicking on the boxes allowing them to modify the functions.

There is, however, a step between these two programming levels; another level of programming within Choregraphe. This step involves the creation of custom objects using one of three methods: (1) Diagram, (2) Timeline, or (3) Script. Each of these methods allows the user to create custom code for the NAO robot without the need of the user to be an expert at Python. This paper seeks to assist those users in making the transition from a beginner-level to advanced user with its tutorial-like descriptive sections, and user-friendly screen shots. Programming the NAO humanoid in this fashion will open opportunities for those learning to program a humanoid robot, while gaining the full understanding and power of using an object-oriented programming language.

2. Materials and Methods

The methods used in the development of this paper are classified as applied qualitative research. The authors present an authoritative explanation on how to program one of the most popular humanoid robotic platforms in the world, the Softbank Robotics NAO humanoid robot, in order to assist those interested in working with or performing research on modern humanoid robotics. The NAO robot has 25 degrees of freedom (DoF), seven touch sensors, four directional microphones, two speakers, speech recognition capability (20 different languages), four sonars, two 2D cameras with facial and recognition abilities, and much more. The robot is programmed using object-oriented software which can be customized using high level languages such as C++ and Python. Figure 2 illustrates many of the features of the NAO robot. This work presents three programming methodologies or techniques: (1) Diagram, (2) Timeline, and (3) Script that are typically utilized by intermediate humanoid users. These techniques fall squarely between drag and drop use of ready-made objects (beginners), and open source high level languages with machine learning or artificial intelligence techniques (advanced users).

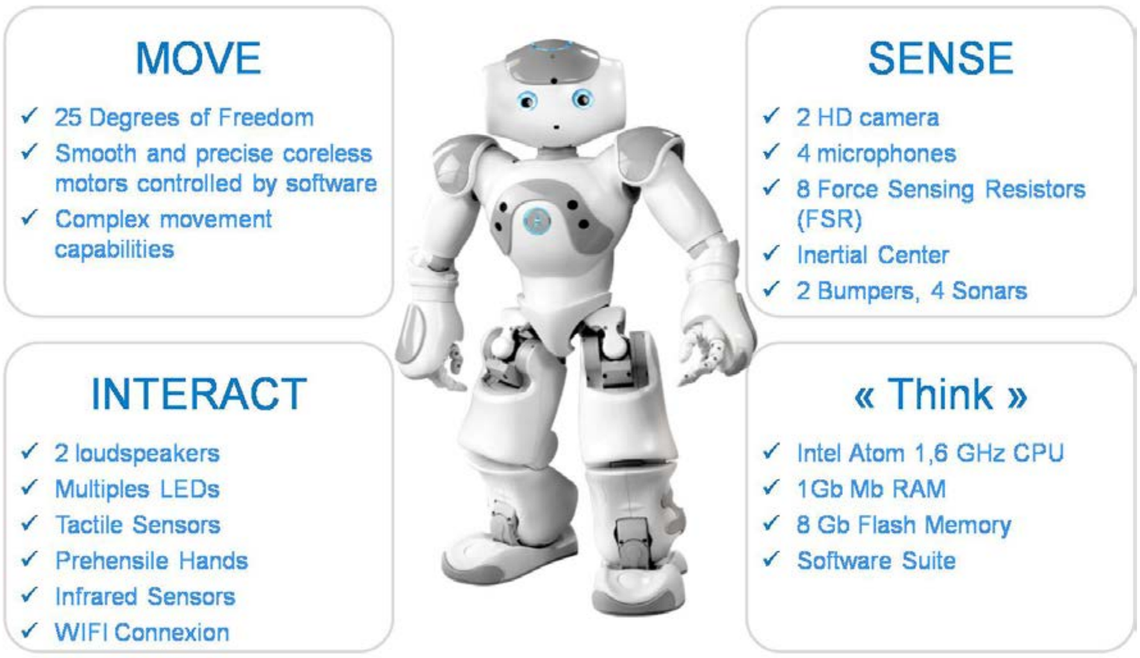


Figure 2. Features of the NAO Platform [9]

2.1. Diagram

The simplest method for creating an object of custom code is to create a box of code via the diagram function or method. Within the Standard Box Library tab, one simply needs to select Diagram and drag it into the root (programming) window as shown in Figure 3. Diagram is the term used for creating a custom box (object) of code via the combination of other existing boxes. Once the Diagram Box is in place, double click on it to expose the inside of the box. Click and drag the boxes you want inside the Diagram box and then string them together as you would normally do with the traditional entry-level box programming technique (see Figure 4).

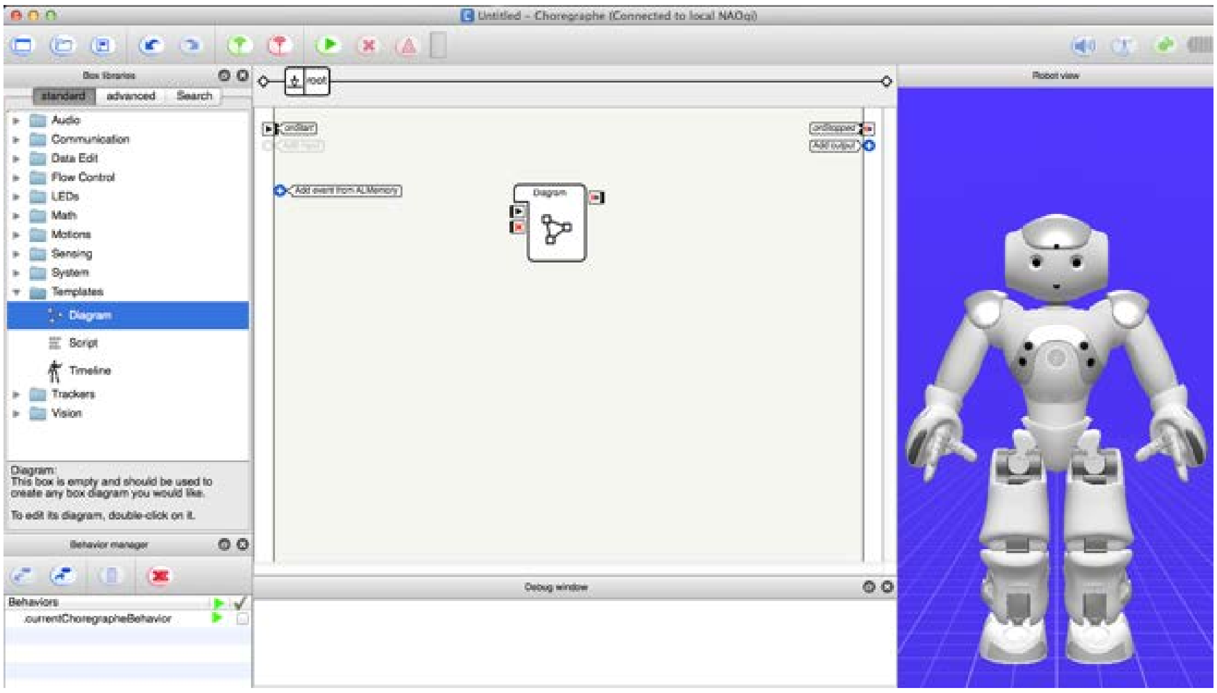
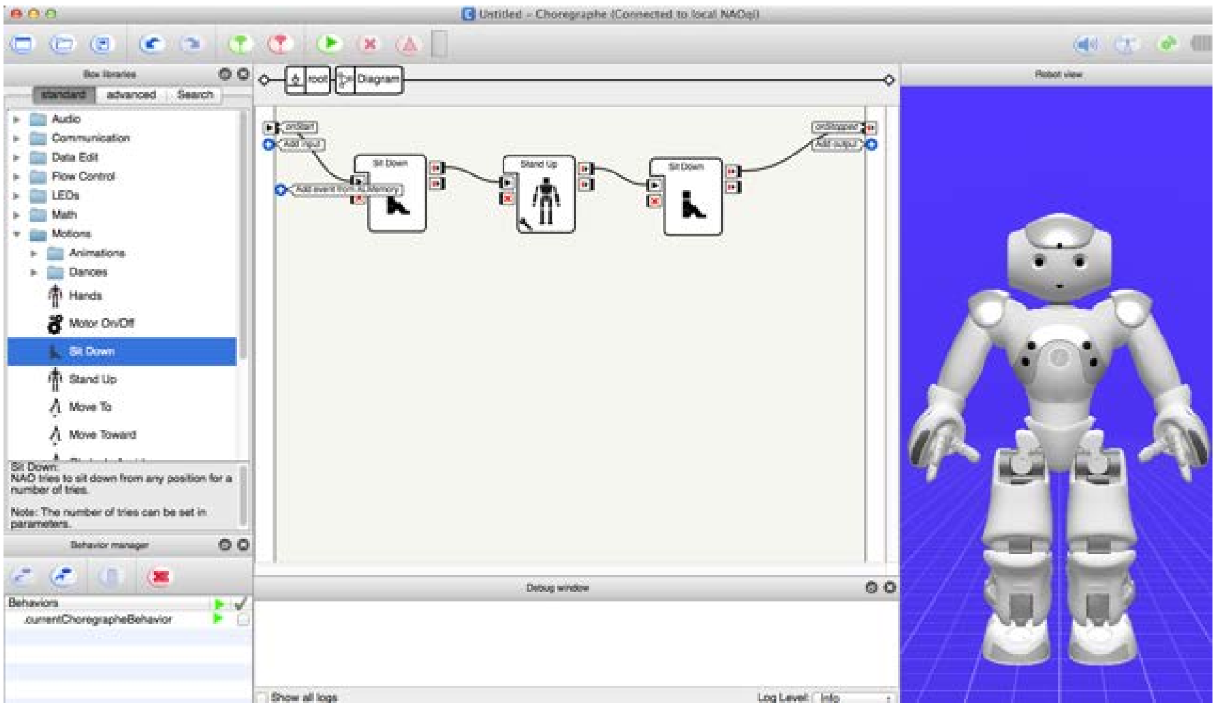


Diagram:
This box is empty and should be used to create any box diagram you would like.
To edit its diagram, double-click on it.

Figure 3. Creating a New Custom Box via Diagram

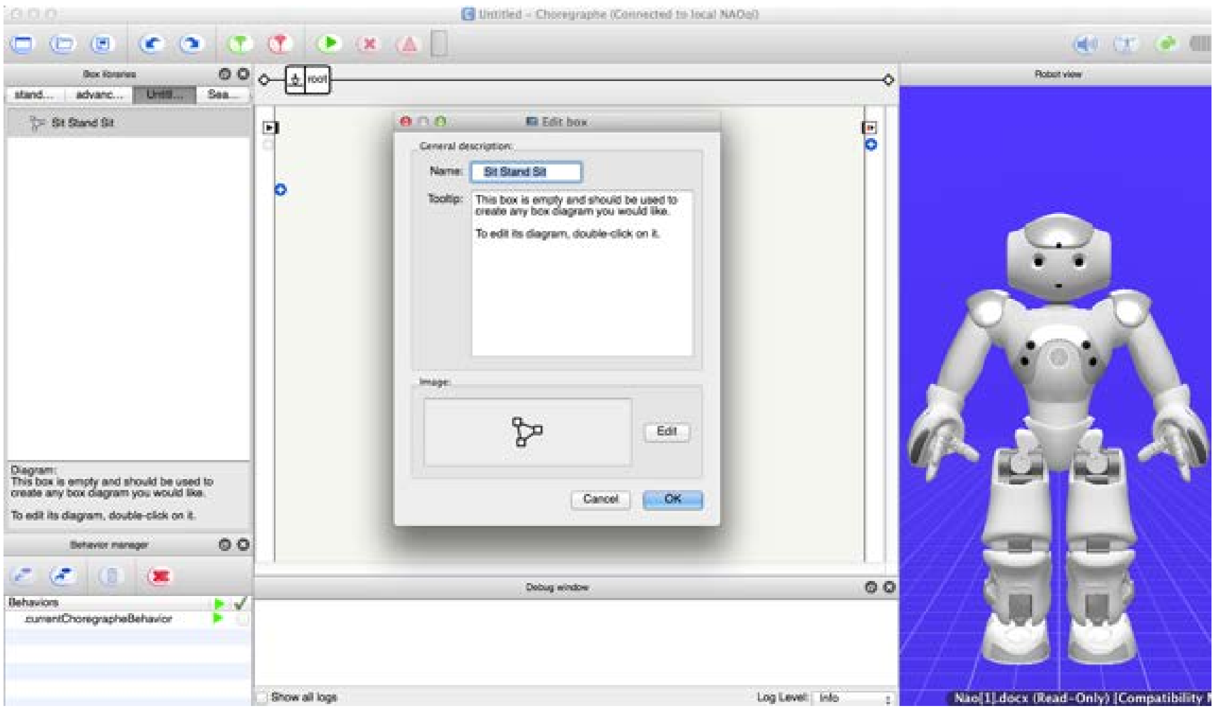


Sit Down:
NAO tries to sit down from any position for a number of tries.

Note: The number of tries can be set in parameters.

Figure 4. Linking the Selected Boxes within the Diagram

The boxes selected are dependent on what one desires the NAO robot to do. Double clicking on the root box located in the top left-hand corner (above your code) next to the diagram box will take the user out to the view of the new custom box that was just developed. To save this new object, select "Create New Box Library" from the box library pull-down menu and drag your new diagram box over to the left. In order to rename the new function, simply double click on the name of the new box (see Figure 5), re-label it, and select "OK." The box library may then be saved using the "Save As" dialog for future use.



Name: Sit Stand Sit

Tooltip: This box is empty and should be used to create any box diagram you would like.

To edit its diagram, double-click on it.

Diagram:
This box is empty and should be used to create any box diagram you would like.

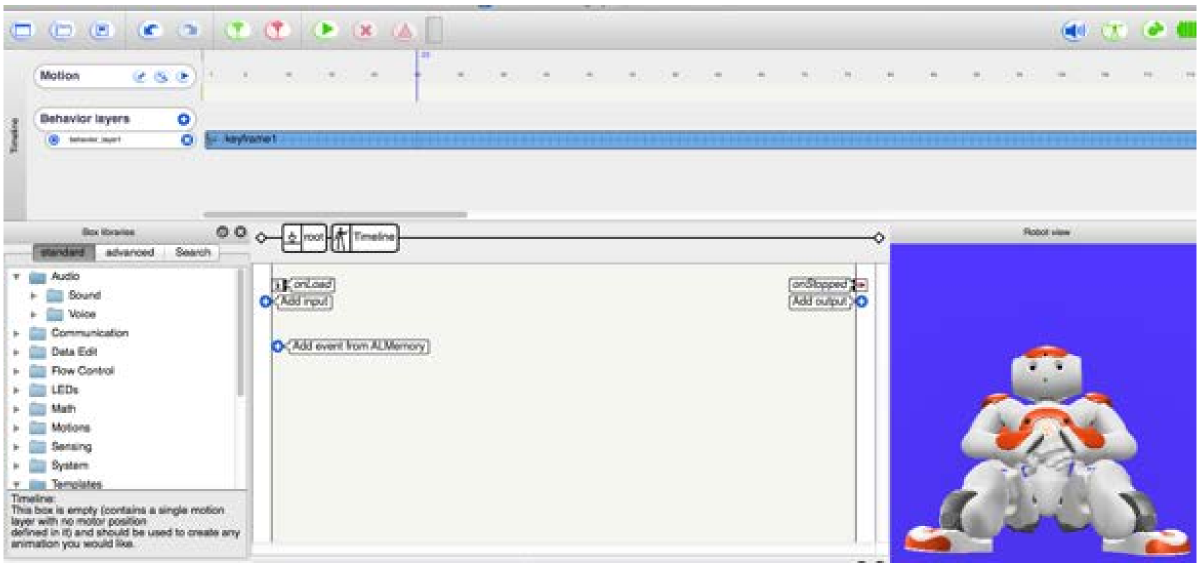
To edit its diagram, double-click on it.

Figure 5. Renaming Your New Custom Diagram

2.2. Timeline

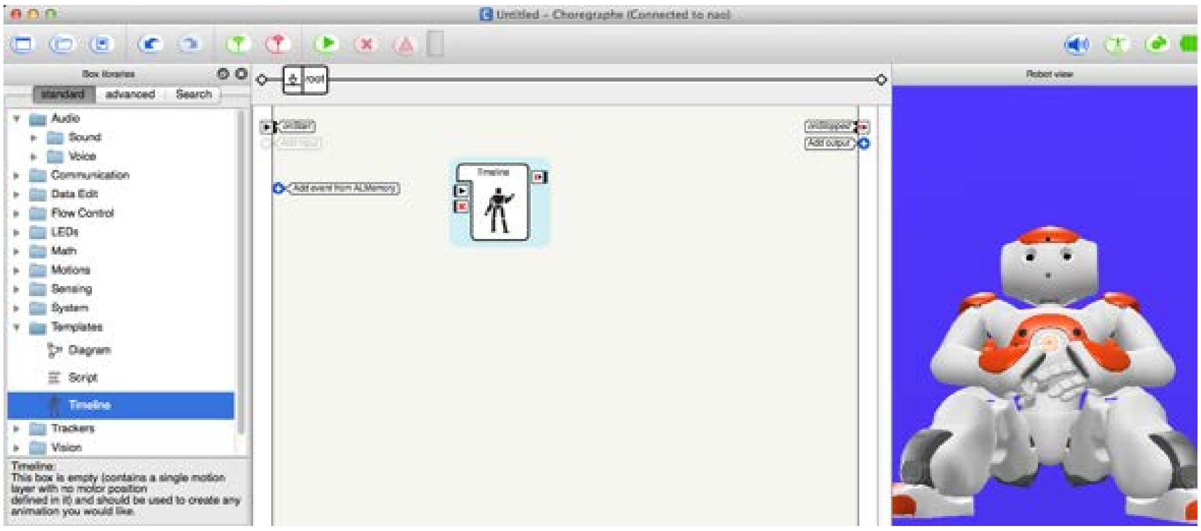
The next level of complexity for creating a custom box of code involves the use of the timeline feature within Choregraphe. Figure 6 shows the Timeline environment where custom motion can be taught in layers of behaviors (recorded layers of motion). In order to use this feature, a NAO robot must be connected to the computer. The robot should be placed into a safe position (e.g., sitting position) as one may turn on and off different motors of the robot during the teaching process. To ensure a safe initial experience with using the Timeline process, it is recommended that one try initially to manipulate NAO's head by teaching him to nod. This behavior (the nod) will be taught in the behavioral layer 1 by disengaging the robot's motor(s) and recording the joint or axis of motion while the programmer physically moves the robot. To develop one's first custom motion box of code, select the Timeline box from within the Standard Box Library tab, and drag it into the root (programming) window as shown in Figure 7. Double-click on the block to open the Timeline function. This is where one may select the joints of the robot that the programmer desires to physically manipulate and record.

One can also select the recording resolution from 1-50 FPS (frames per second). The default setting is 25 FPS. The programmer can also change the maximum duration of the timeline layer here as well. To access this menu, simply click/select the Timeline Properties icon located next to the Motion Menu bar. Accept the default Timeline Properties and then proceed to open the Timeline Editor by selecting the first icon in the Motion toolbar. Next deselect all actuators except the head. Double click on the head actuator and select the HeadPitch axis as shown in Figure 8. Within this screen, note the icons across the top. The fifth icon allows the programmer to switch into recording mode. Select the icon and your menu screen should look like Figure 9.



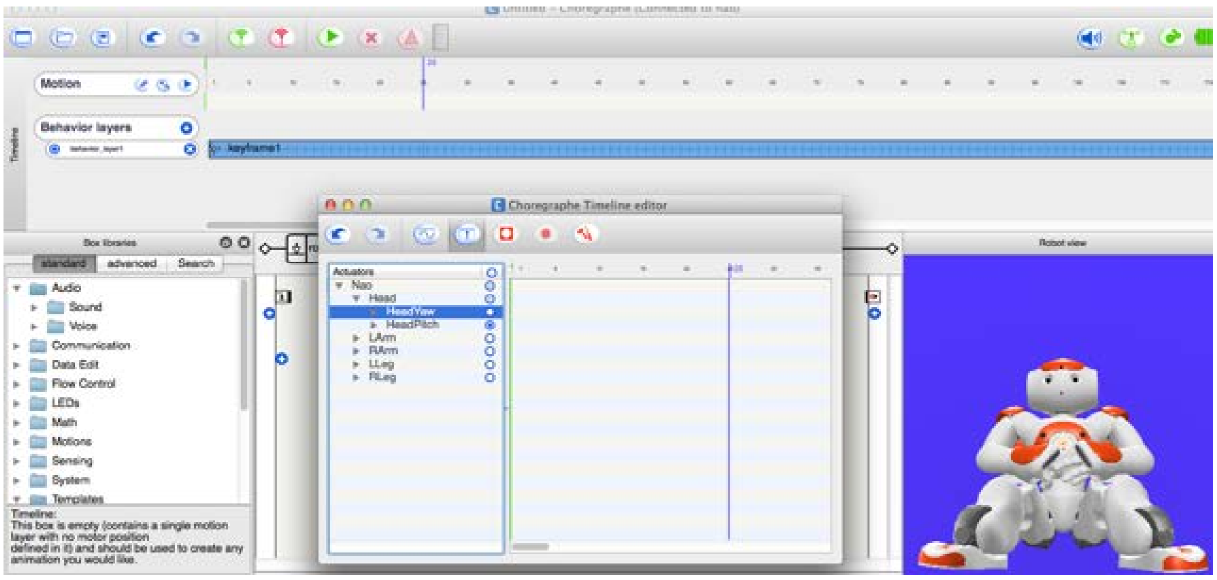
Timeline:
This box is empty (contains a single motion layer with no motor position defined in it) and should be used to create any animation you would like.

Figure 6. Timeline Initial Screen



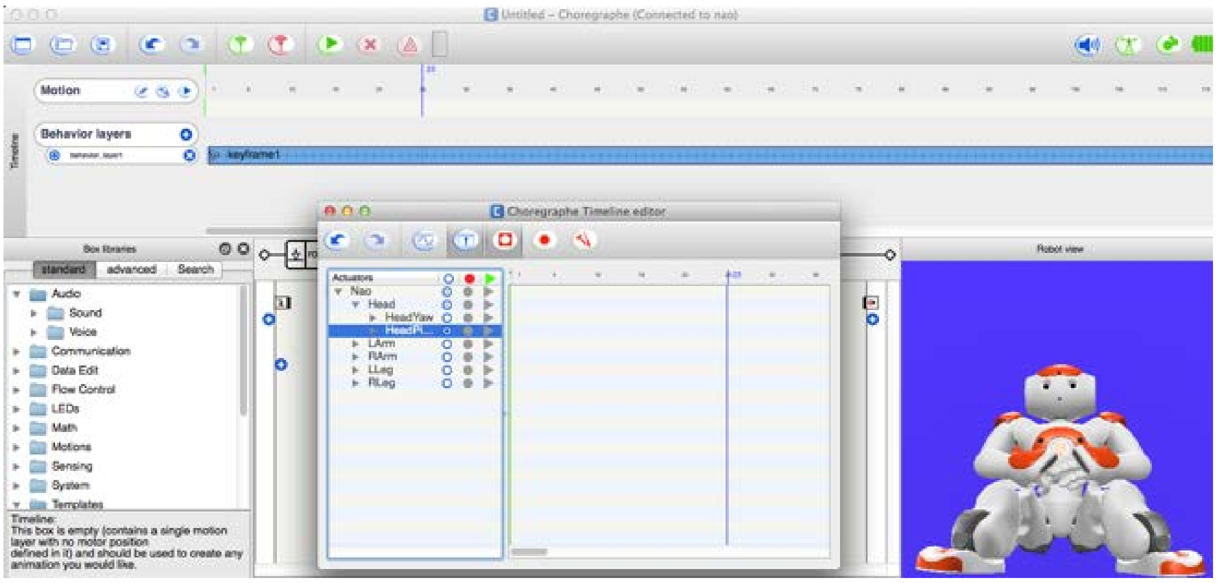
Timeline:
This box is empty (contains a single motion layer with no motor position defined in it) and should be used to create any animation you would like.

Figure 7. Starting a Timeline Custom Motion Code Box



Timeline:
This box is empty (contains a single motion layer with no motor position defined in it) and should be used to create any animation you would like.

Figure 8. Timeline Editor Screen



Timeline:
This box is empty (contains a single motion layer with no motor position defined in it) and should be used to create any animation you would like.

Figure 9. Switching to Recording Mode

One should now be able to see the axis selected, in this case the HeadPitch, along with a red dot and green arrow for each axis. To begin to record a movement, select the red dot associated with the axis selected. This prepares the robot to record that axis of motion. After the joint(s) are selected, one must click on the Start/Stop icon (sixth from the left in the top menu) and then physically move the robot with your hand.

When complete, select the Start/Stop icon. To test the recording, select the green arrow associated with the axis you are programming and then use the Start/Stop icon to see your recorded movements. Notice that as one selects the Start/Stop red icon, it changes in shape (center of icon). A circle indicates it is ready to start, and a square indicates the robot is ready to stop. Select the root menu, drag the new motion behavior box into one's custom library, and rename it.

2.3. Script

There are two ways to create a custom script box in Choregraphe. The first method involves finding the "Script" option in the Box Libraries toolbar located on the left side of the screen. This option can be found under the "Templates" section. Click and drag the "Script" text into the workspace to add the box to the project.

One can also create a custom box by right clicking in the workspace and selecting the "Add a new box" option. An option window will pop up that allows the user to edit specific properties of the script box. In order to develop a custom script box, it is important to take a moment to discuss the parameters involved and the features available to programmers using Choregraphe.

2.3.1. Parameterizing

Choregraphe/NAO boxes require inputs, outputs, and parameters. While this section of this paper is absolutely key to creating custom script boxes, one can apply this information to help customize the other custom boxes already discussed. *Inputs* signal the box to begin some behavior or perform some task, *Outputs* are the results of the operation, and *Parameters* are used to configure the box for repeated or otherwise similar tasks without having to edit the code of the script. These properties are divided up into two tabs, “General” and “Advanced”. The “General” tab includes the following:

- **Name:** Set the name of the currently selected box. This does not need to be unique, but a descriptive name will make it easier to work with in the future.
- **Tooltip:** The description of the box. This should include any specific notes regarding the box. As with the name, a more descriptive tooltip will make the box much easier to work with.
- **Image:** The icon associated with the box. One can choose from a number of available images provided by Softbank Robotics or import a custom image. This is purely for organization and will have no bearing on the performance of the script.
- **Inputs/Output/Parameters:** This is where all of the inputs, outputs, and parameters for the box are displayed. By default, a newly created box will have two inputs, “onStart” and “onStop”, and 1 output, “onStopped”.
- **Box Type:** This displays current box type. By default, this should be “Script”, but one may also select “Diagram” or “Timeline” from this menu.

Editing an *Input* or *Output* brings up a small dialog that allows the user to edit the name, tooltip, type, and nature. “Name” and “Tooltip” are similar to those one may encounter when creating a new box, and are for organizational purposes. “Type” represents the object type that the input accepts or that the output returns. For instance, one would expect a box called “Sum” to accept numbers, and return a number that holds the summed values of all that was passed to it. If passing or expecting a number, set the type to “Number”. In order to pass or receive a word or sentence, set it to “String”, and if one is sending a signal to start, or a box has just finished, then select “Bang”. Select “Dynamic” if unsure of the input or return, or if its type does not fall into one of the earlier categories. Advanced users can pass objects, lists, and other data structures using this Dynamic type.

“Nature” refers to the type of input or output that is being specified. Many boxes have more than one input to do more than one task. For instance, several boxes have an “onStop” to stop the box if something goes wrong. If the input is to run only when some event occurs, the Nature should be specified as “onEvent”. If the input is to run when it receives some signal or input value, specify “onStart”. If the input is an interrupt or a cancelling operation, specify “onStop”.

Outputs are very similar, but with fewer options. If the output is to signal when the process is finished, select “on Stopped”. If the output is to signal before the process is complete (for instance, if something went wrong) or if the output is to signal more than once (like a timer or something that must be looped), one should select “punctual”. It is important to note that inputs and outputs must match one-to-one with methods in code. For instance, if an input exists called “onStartMe” that takes a Number value, the code to use this input will also reflect this in both its name and parameter list.

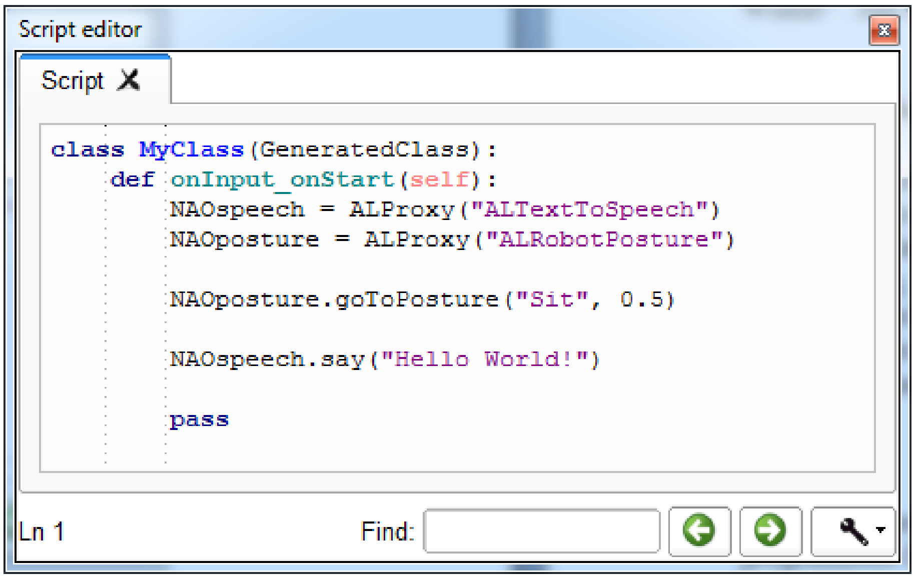
2.3.2. Scripts with Python

Intermediate users need to understand some fundamental concepts of Python in order to modify or design custom scripts. Choregraphe boxes create Python scripts automatically, and the most documentation available is for programming NAO in Python.

Python is an interpreted language, which can save you considerable time during program development because no compilation and linking are necessary. The interpreter can be used interactively, which makes it easy to experiment with features of the language, to write throw-away programs, or to test functions during bottom-up program development. Python enables programs to be written compactly and readably. Programs written in Python are typically much shorter than equivalent C, C++, or Java programs, for several reasons: (1) The high-level data types allow you to express complex operations in a single statement; (2) statement grouping is done by indentation instead of beginning and ending brackets; and (3) no variable or argument declarations are necessary.

By the way, the language is named after the BBC show “Monty Python’s Flying Circus” and has nothing to do with reptiles. Making references to Monty Python skits in documentation is not only allowed, it is encouraged [10]! Most programmers like to space and indent their code in order to make it more readable. Python actually forces indentation in order to determine scope and boundaries. The indented code is treated as part of the loop body and will be executed inside the loop. The line that remains not indented will be executed after the loop is finished.

So what happens when our functions or loops don’t do anything, or if there are no useful statements that are to be placed between them? There is a special keyword in Python for exactly that purpose: pass. Pass indicates that where there would ordinarily be an indented block of code, the block is empty, and that the interpreter should act as though the pass statement is the “end” of the loop, block, or function. For blocks with statements present, pass is not necessary, but will not cause adverse behavior if used similar to a return or break statement. Alternatively, if one desires to separate lines of code with a pass, the lines before will be treated as though they are inside the block or loop, and those after will be treated as though they are not.



```
class MyClass(GeneratedClass):
    def onInput_onStart(self):
        NAOspeech = ALProxy("ALTextToSpeech")
        NAOposture = ALProxy("ALRobotPosture")

        NAOposture.goToPosture("Sit", 0.5)

        NAOspeech.say("Hello World!")

        pass
```

Figure 10. Sample Python Code for Motion and Speech

For those familiar with Object-oriented programming, the use of the "self" keyword may appear similar to "this" or "with" constructs from their favorite language of choice. This is actually the case, as Python makes use of the "self" keyword to mean the current instance of some object. The "self" syntax can be taken more intuitively to mean "assign or add to me", so "self.memberA" means "Add to me a variable called memberA". Because all statements after the class declaration are indented, they are all part of the class's scope, and all functions defined and all variables declared are parts of the class.

Class-level variables exist only in one place, their class declaration, but can be accessed by any instance of their class. Instance-level variables can be thought to be copies of the same name, and must be referenced by using the 'self' keyword. These variables are instance-specific and do not change everywhere when they are altered. It may help to think in terms of class-level referring to semi-global, while instance-level is local in usage. To access the Script editor, double-click/select the new script box that one just created earlier in this paper. This window allows the programmer to input custom Python code in order to send commands to NAO. Figure 10 illustrates an example code that will cause NAO to perform two basic operations: movement and speech.

The first line of the code in Figure 10 creates the class, which is automatically generated by Choregraphe. The next line contains the following:

def onInput_onStart(self):

This line indicates that the code is looking for a signal from the "onStart" input. When a signal is received, all of the code up until the "pass" command will be executed. The next two lines are for creating proxies:

NAOspeech = ALProxy ("ALTextToSpeech")
NAOposture = ALProxy ("ALRobotPosture")

A proxy is an object that grants access to some of NAO's built-in functionality. The two proxies utilized in this example are "ALTextToSpeech" and "ALRobotPosture". Upon being initialized, speech and posture can be controlled by the two variables "NAOspeech" and "NAOposture". A detailed listing of proxies and their functions can be found on the Softbank Robotics website. The next line will issue a command to NAO to change his current posture:

postureProxy.goToPosture("Sit", 0.5)

Utilizing the variable "postureProxy", the command "goToPosture" is sent with the parameters "Sit" and "0.5". The first parameter, "Sit", causes NAO to assume a sitting posture. The second parameter, "0.5", indicates the speed at which the sitting posture will be assumed. This can be anywhere between 0.01 and 1, with 0.01 being the slowest possible speed and 1 being the fastest possible speed. When NAO has finished moving to the designated posture, the next line of code will be executed:

speechProxy.say("Hello World!")

The command "say" will be sent utilizing the variable "speechProxy", with "Hello World!" as a parameter. This will simply cause NAO to speak the parameter aloud, which in this case is "Hello World!". The last line, "pass", will simply indicate that all of the code for this input has been completed.

3. Conclusions

The future for robotics in today's world is clear. We must automate our manufacturing processes to remain competitive in the global marketplace. Robotics is an important part of that automation and is speculated to play an even larger role in all types of industries including medical, manufacturing, service, construction, etc. Humanoids have a host of capabilities that may reveal potential new uses in industry. Increasing capabilities of these new robotic devices continue to develop and amaze us all while becoming more and more affordable [11].

This paper was developed to assist students, academics, and practitioners involved with programming robotic humanoids. While the paper focused exclusively on intermediate programming techniques for the Softbank Robotics NAO humanoid platform, many of the concepts presented may be transferred to other platforms in the marketplace.

The future of robotics is bright, with emerging technology and innovative applications poised to make it one of the more pervasive instruments of our technological society. The demand for a trained workforce from robotics technicians to engineers will continue to increase. Educational institutions must develop or advance programs to ensure students are prepared to meet workforce demand with the advanced skill sets necessary for successful employment and professional growth. The authors hope that this applied engineering paper will assist those who seek to progress their skills with humanoid robotics in academia as well as the industrial sector.

REFERENCES

[1] Softbank Robotics Press Kit (n.d.). NAO6 – the versatile humanoid robot. Retrieved May 23, 2019, from http://www.softbankrobotics.com/emea/sites/default/files/press-kit/NAO-press-kit-EN.pdf

[2] Waytz, A., & Norton, M. (2014, June 1). How to make robots seem less creepy. The Wall Street Journal.

[3] Lydon, B. (2013, March/April). Perspectives from the Editor: The robots are coming! InTech, 60 (2), p. 7.

[4] Lombardi, A. (2014, January 22). Move Over Tech and Engineering, Healthcare Leads Robotics Hiring. Retrieved July 30, 2019 from Robotics Tomorrow: https://www.roboticstomorrow.com/news/2014/01/22/move-over-tech-and-engineering-healthcare-leads-robotics-hiring-/22971

[5] North American Robotic Orders Record Second-Highest Quarter Ever in First Quarter 2014. (2014, May 1). Robotics Online.

[6] O*Net. (2019). Summary Report for 17-2199.08-Robotics Engineers. O*Net. Retrieved May 23, 2019 from O*Net: https://www.onetonline.org/link/summary/17-2199.08

[7] Hornyak, T. (2010, August 14). Tech Culture. Retrieved May 23, 2019, from CNET: www.cnet.com/news/humanoid-robot-nao-gets-emotion-chip/

[8] North American Robotics Orders to Non-Automotive Companies Surge to New Records. (2018, November 29). Robotics Online.

[9] Inside NAO (2011, December 26). Retrieved July 30, 2019 from https://robosavvy.com/forum/viewtopic.php?t=7670

[10] The Python Tutorial: Whetting Your Appetite (2014). Retrieved May 23, 2019, from The Python Software Foundation: https://docs.python.org/3.3/tutorial/index.html

[11] NAO Released to the General Public at Half the Price (2014, March 18). Retrieved May 23, 2019, from Bot Scene: http://botscene.net/2014/03/18/nao-released-to-general-public-at-half-the-price/