

MIKE BEITER

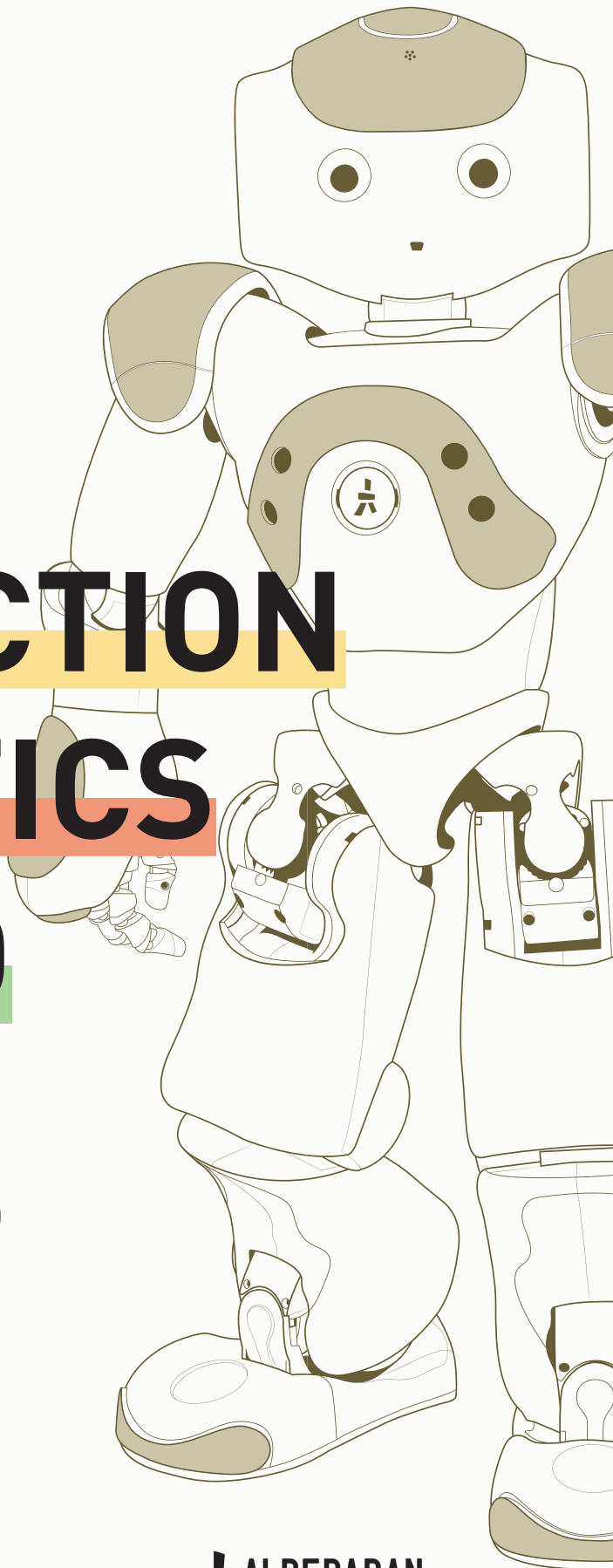
BRIAN COLTIN

SOMCHAYA LIEMHETCHARAT

AN INTRODUCTION TO ROBOTICS WITH NAO

A STEM INTEGRATED,
PROJECT BASED APPROACH
TO LEARNING ROBOTICS AND
COMPUTER SCIENCE

Aligned to the Common Core
State Standards Initiative



Welcome to NAO.

I hope that you will find working with the NAO robot platform as interesting as my students have. Many of you may have worked with robots in the past, Robots such as Lego Mindstorms™ or FIRST™ Robotics. I think you will find this a completely different experience.

NAO is humanoid, two arms, two legs, eyes, ears, he can walk and talk. Notice how I said “He”, if you are like my students you will find yourself personifying your NAO immediately. We named ours “Pablito”, and for us as we created the artificial intelligence, and developed behaviors for our robot he took on the personality of his programmers. I believe it is this “humanness” that makes working with NAO so fascinating, because it can do so much that you can do, the possibilities of what you can do with the robot are limitless.

After only a few short weeks of working with this curriculum you will have your robot, walking, talking, listening, and interacting with the environment around it. Once you do that I am sure that you will come up with hundreds of uses that we could never have dreamed. Your creativity and imagination are the only things that limit what you can do with NAO, from a service robot to help children, or the elderly, the police, or fireman, to an entertainer dancing, singing and chatting with its audience.

Post your videos on YouTube® so you can share your experiences with the NAO community. We would love to see your videos!

Mike Beiter

Computer Science Instructor
Central Career and Technical School
Erie, PA
Mbeiter@eriesd.org



TABLE OF CONTENT

- **INTRODUCTION FOR TEACHERS**
- **HOW TO USE THIS CURRICULUM**
- **OVERALL OBJECTIVES**

- **Module 1 – Hello World!**

- * Objectives
- * Lesson plan
- * Exercises

- **Module 2 – Walk it Out**

- * Objectives
- * Lesson plan
- * Exercises

- **Module 3 – Hearing Things**

- * Objectives
- * Lesson plan
- * Exercises

- **Module 4 – Let's Dance**

- * Objectives
- * Lesson plan
- * Exercises

- **Module 5 – Sense and Act**

- * Objectives
- * Lesson plan
- * Exercises

- **Module 6 – Do the Robot**

- * Objectives
- * Lesson plan
- * Exercises

- **Module 7 – Face Off**

- * Objectives
- * Lesson plan
- * Exercises

- **Module 8 – Object Recognition**

- * Objectives
- * Lesson plan
- * Exercises

- **Module 9 – Human-robot Interaction**

- * Objectives
- * Lesson plan
- * Exercises

- **Module 10 – Finding Your Way**

- * Objectives
- * Lesson plan
- * Exercises

- **MODULE QUESTION SOLUTION**

INTRODUCTION

FOR TEACHERS

Welcome to NAO.

This curriculum has been designed to allow you to develop interesting, challenging and fun projects with your robot. I have been teaching computer science for the last 20 years. I began in the days of the first PCs and I currently teach computer science to Grades 9 and 11 at a Comprehensive Career and Technical School, and as an adjunct professor of Computer Science at two Universities. Over the years I have taught with many types of technologies: Robotic arms, Lego Mindstorms, PLC's and all different types of computers. I believe this is the most exciting curriculum I have ever been involved with.

When our school district first purchased the NAO robots I was unsure about the cost versus benefit of these platforms. What I have found is that the humanoid robot generates and unparalleled interest from students. My traditional computer science students are driven to program the robot to do everything from dance to fold laundry. But it is not the traditional students that really surprised me so much as the overwhelming response to the robot from non-traditional students: I had students from our nursing and carpentry programs beating down my door for an opportunity to work with the NAO. These students were captivated by the humanoid robot in a way that traditional robotics platforms and computer software simply could not duplicate.

This curriculum has been developed with a number of goals. First and foremost it is engaging for the students. As you look at the modules I do not believe you will find a single thing that will cause students to roll their eyes and say "Do we have to do that?". Second, and just as important, is that it is project-based and aligned to the Core STEM standards as laid out in the Common Core Standards. Each module covers a set of objectives specific to learning robotics, but also includes objectives, standards, and lesson plans that cover a wide variety of academic core standards in Math and English. As a general rule you should start with Module 1 and work foreword, but other than the first module you could really pick and choose modules to fit your needs.

I hope that you will find this curriculum to be an exciting and useful addition to your Computer science or robotics classroom. I am confident that your students will find that this is one of the most enjoyable and interesting ways they have ever learned.

HOW TO USE THIS CURRICULUM

As a general rule each module is independent. In each module you will find a set of robotics/computer science objectives, as well as related academic STEM objectives.

Both sets of objectives will identify the common core standards addressed in that module.

**YOU ARE ALLOWED TO REPRODUCE THE
CONTENT OF THIS BOOK AND TO SHARE IT
WITH YOUR CLASSROOM ONLY.**

Aldebaran Robotics does not warrant the accuracy of the provided content which shall be used at your own risk and under your control. Aldebaran Robotics disclaims all liability related to the use as well as the content.

All rights not specifically granted herein are reserved to Aldebaran Robotics. Aldebaran Robotics and/or its licensor shall retain all rights, title and interest and ownership in and to the book and its content.

This curriculum has been done with the 1.12.0 version of Choregraphe, our programming software. The screenshot of the software included in this curriculum may be different depending the version of Choregraphe you have.

SUGGESTED TEACHING PRACTICES

1/ HAVE STUDENTS PRE-READ THE MODULE. YOU MAY WANT TO USE THE KWL READING STRATEGY*

→ Prior to reading

- * have students prepare a list of what they already Know about the subject
- * Then create a series (1-3) questions of what they want to know.

→ After reading

- * Have student list what they Learned from the reading

2/ PLAN PLAN PLAN

→ have students present a short algorithm or step by step instruction set for each module

→ ask them to include safety and best practices for keeping the NAO safe

3/ COMPLETE THE MODULE QUESTIONS AT THE END OF EACH MODULE

4/ COMPLETE THE MODULE

→ Have students complete the module with the NAO and demonstrate the completed behaviors

→ You may consider having students do a Lab Report of the module

A. Title

The title states what the module covered

B. Introduction / Purpose

A paragraph that explains the objectives and purpose of the module

C. Materials

List everything needed to complete the module

D. Methods

Very similar to your prior algorithm a list of steps required in order to complete the module.

E. Data / Observations

What actually happened while you complete module. (record both expected and unexpected results)

F. Results

A conclusion paragraph that states what you learned from the module

* Valmont, W | 2003 | *Technology for literacy teaching and learning* | New York: Houghton Mifflin Company.
Allington, R. and Cunningham, P | 2003 | *Classrooms that work* | Boston: Allyn and Bacon.
Padak, N. and Rasinski, T | 2004 | *Effective reading strategies: teaching children who find reading difficult* | New Jersey: Pearson Education, Inc.
Buehl, D | 2006 | *Classroom strategies for interactive learning* | Delaware: International Reading Association.

1 HELLO WORLD

LEARNING

In this module, students will learn:

→ **How to switch on the NAO humanoid robot**

→ **How to connect to the NAO with Choregraphe on a computer**

→ **How to make the NAO speak with Choregraphe**

* RST.9-10.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks, attending to special cases or exceptions defined in the text.

→ **How to vary the pitch and speed of the NAO's voice**

* RST.9-10.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks, attending to special cases or exceptions defined in the text.

→ **How to program the NAO to speak with Python**

* RST.11-12.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.

* RST.11-12.8. Evaluate the hypotheses, data, analysis, and conclusions in a science or technical text, verifying the data when possible and corroborating or challenging conclusions with other sources of information.

* RST.11-12.9. Synthesize information from a range of sources (e.g., texts, experiments, simulations) into a coherent understanding of a process, phenomenon, or concept, resolving conflicting information when possible.

* RST.11-12.10. By the end of grade 12, read and comprehend science/technical texts in the grades 11–12 text complexity band independently and proficiently.

* Reference COMMON CORE Stem standards

CONTENTS

01/ Preparing to Use the NAO

02/ Basic Task: Hello World!

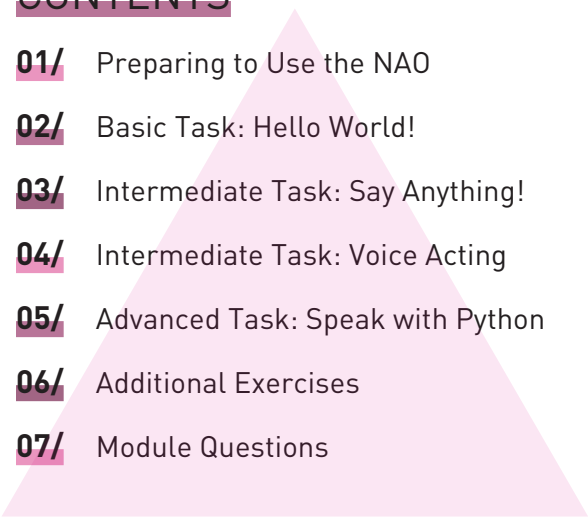
03/ Intermediate Task: Say Anything!

04/ Intermediate Task: Voice Acting

05/ Advanced Task: Speak with Python

06/ Additional Exercises

07/ Module Questions

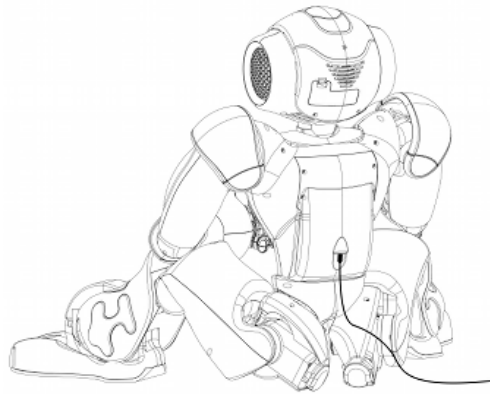


PREPARING TO USE THE NAO

Before we begin to use the NAO, we will learn how to charge, set up, and handle the robot safely. In our figures, red arrows mean single-click, blue arrows mean click-and-drag, and green arrows mean double-click.

01/

When the NAO is not moving, plug the charging cable into the robot's back (see picture to right). This will ensure that the robot remains charged for your continued use. If the robot is charging, the charger's light will turn red.

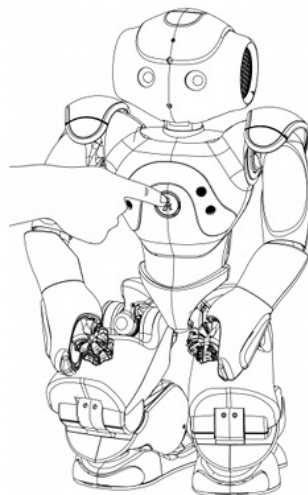


02/

Before turning on the NAO, make sure it is in a stable position. The pose shown in the image below, with the back upright and both feet flat on the floor, is recommended. Do not place the NAO on a table where it can fall and damage itself.

03/

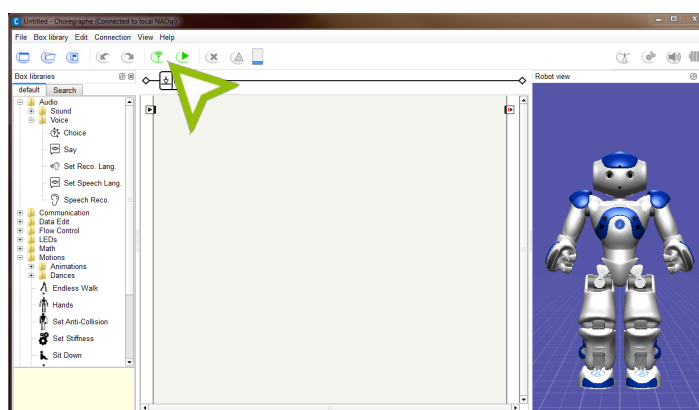
Turn on the NAO by pressing the chest button. The NAO's lights will turn on. The NAO can take up to several minutes to boot, and it will make a sound when finished.





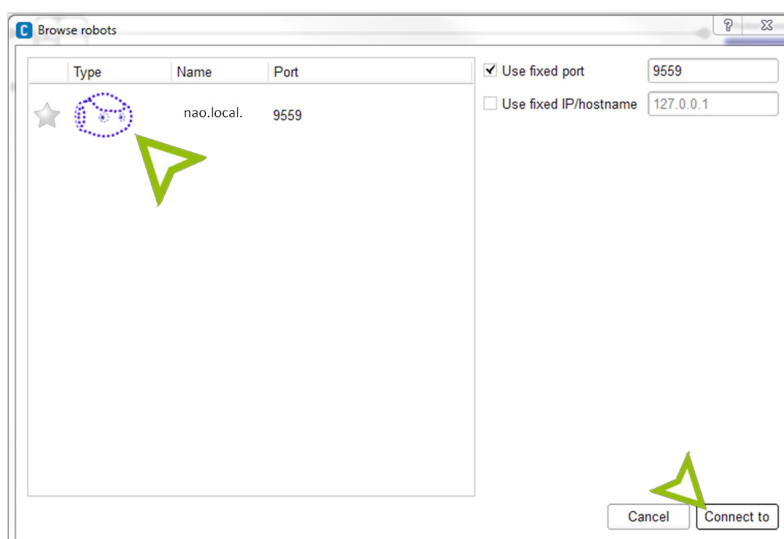
04/

Now, you will connect to the NAO using Choregraphe. First, start Choregraphe on your computer. The following window will appear. Click the indicated “Connect” button, which is green and looks like a wireless signal, to connect to the NAO.



05/

The window below will appear, with your robot listed on the table to the left. Select your robot in the list, and click the “Connect To” button.



06/

Now you should be connected to the NAO. Try moving some of the NAO’s body parts by hand. The window to the right in Choregraphe will update to reflect the changes in the NAO’s joint positions.

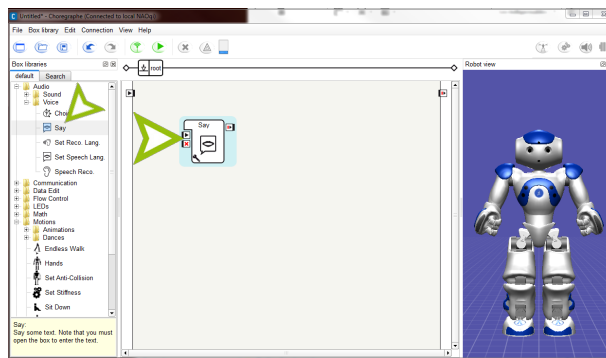
BASIC TASK

HELLO WORLD!

NEXT, WE WILL PROGRAM THE NAO TO SAY “HELLO.” THE NAO USES A TEXT-TO-SPEECH ENGINE TO CONVERT TEXT INTO SOUND, WHICH IS OUTPUT THROUGH THE SPEAKERS.

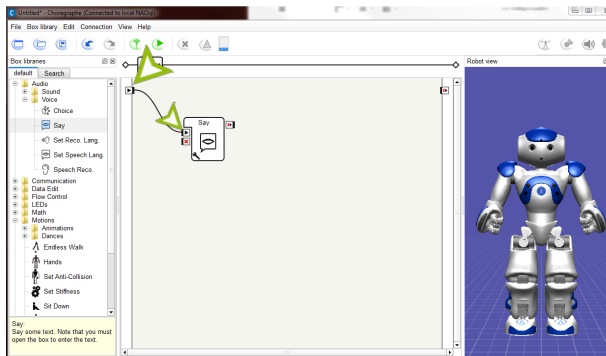
01/

In Choregraphe, look at the “Box List” to the left. Navigate to Audio → Voice, then drag and drop a “Say” box to the central area. This central area is called the workspace, and contains the commands that the robot executes.



02/

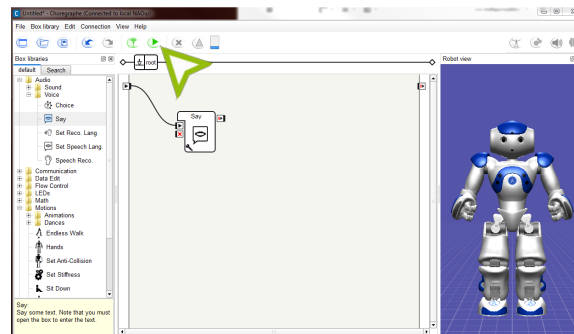
To execute the “Say” box, we must connect it into the program “flow”. Click on the small arrow to the left of the workspace, and drag a line to the arrow on the left of the “Say” box. The boxes will be executed sequentially in the order that they are connected. The last box should connect to global stop [x] on the right hand side of the work area.





03/

Finally, click the green “Play” button next to the “Connect” button. The NAO should say “Hello.”



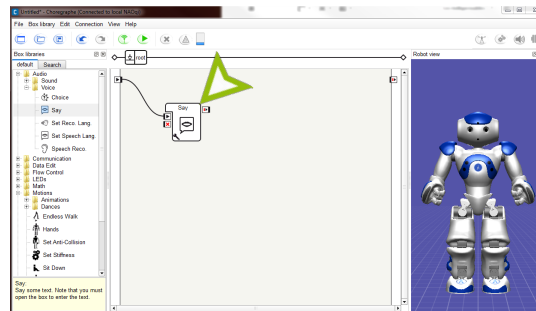
INTERMEDIATE TASK

SAY ANYTHING!

WE WILL LEARN HOW TO CHANGE THE WORDS THE NAO SAYS TO “HELLO WORLD.”

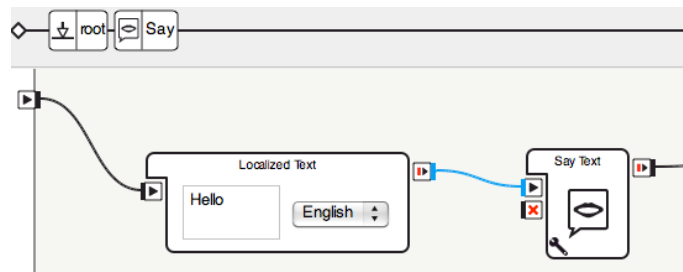
01/

Beginning with the results of the previous task, double-click on the “Say” box.



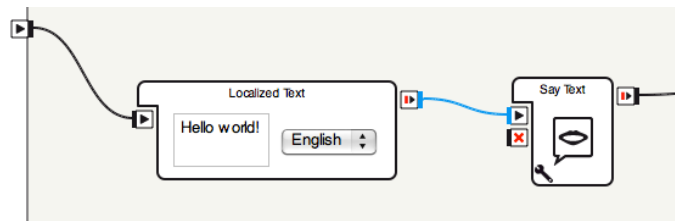
02/

A “Localized Text» will appear connected to a “Say Text” box. To return to the previous screen, click “root” on the upper box hierarchy “breadcrumb trail”.



03/

Replace the “Hello” in the text box with “Hello World!”.



04/

Click the play button, and listen to what the NAO says.

05/

Try experimenting with different words and phrases.

INTERMEDIATE TASK

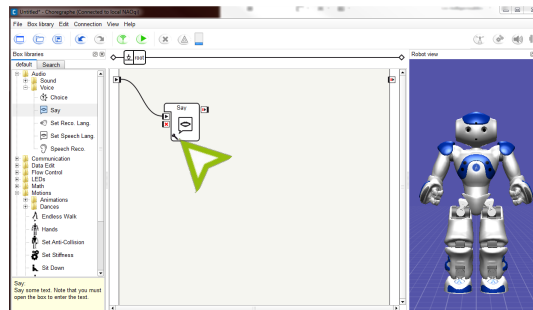
VOICE ACTING



THE NAO CAN SPEAK FASTER, SLOWER, LOWER OR HIGHER, DEPENDING ON TWO PARAMETERS, “VOICE SHAPING” AND “SPEED”.

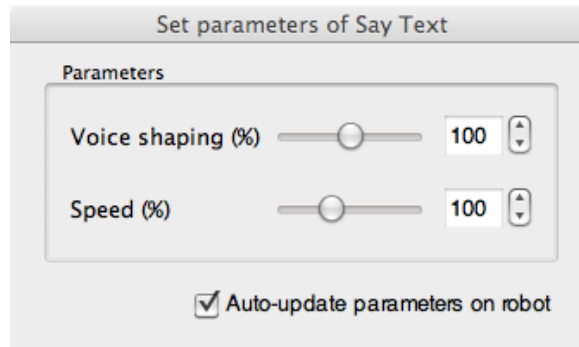
01/

Click on the wrench in the bottom left corner of the “Say” box.



02/

The window below will appear:



03/

Move the two sliders to a different position and click OK. Press the play button for the NAO to speak again.

04/

What changed? Try experimenting with different values for each of the sliders. What do the two parameters control?

ADVANCED TASK

SPEAK WITH PYTHON

IN ADDITION TO DRAGGING AND DROPPING BOXES IN CHOREGRAPHE, THE NAO CAN BE PROGRAMMED USING SEVERAL PROGRAMMING LANGUAGES INCLUDING C++ AND PYTHON. WE WILL BE USING PYTHON FOR OUR EXERCISES, BUT DOCUMENTATION ABOUT PROGRAMMING IN C++ IS AVAILABLE ONLINE.

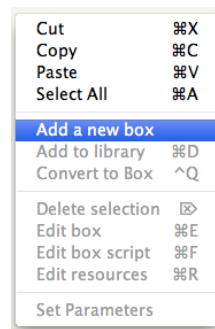
We will begin with a simple programming exercise in Python, where we have the robot say “Hello World” exactly as we did before with the Choregraphe Say box. We will do this by creating our own box in Choregraphe that executes our python code.

01/

First, open an empty workspace in Choregraphe by creating a new project.

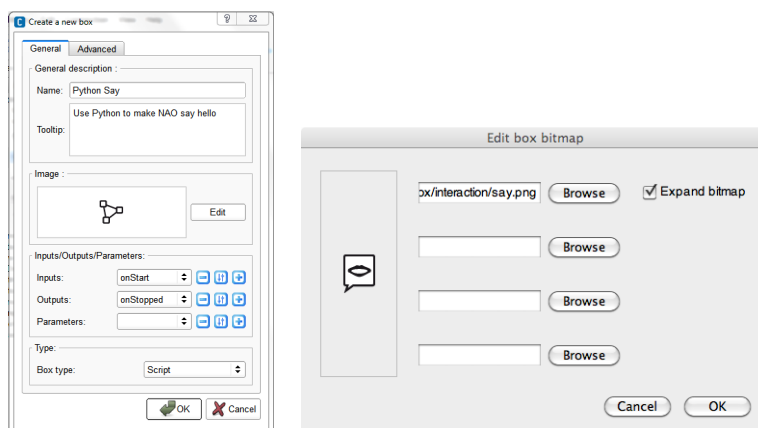
02/

Right click on the workspace, and choose “Add a new box” from the menu.



03/

A dialog will appear. In the “Name” box, enter “Python Say”, and in the tooltip box, enter a description of what your box will do. Do not change any of the other options. In particular, make sure the “Box Type” type is “Script”.





04/

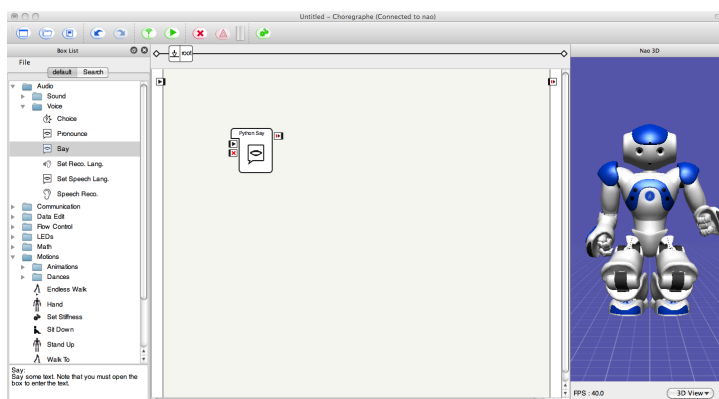
Click the “Edit” button in the “Image” section. A new dialog box will appear.

05/

Click the first “Browse” button and select the image “say.png” in “interaction” directory. This image will be displayed on the newly created box.

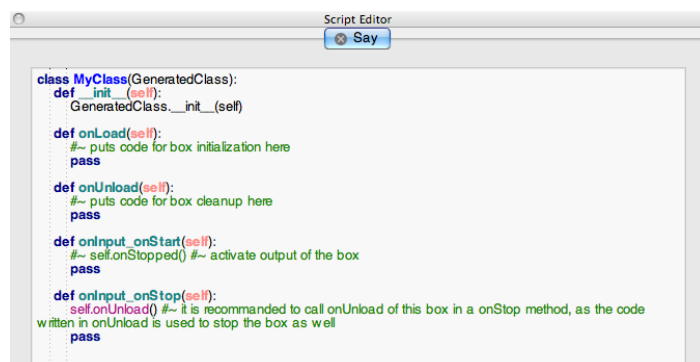
06/

Click “OK” on both of the dialog windows. Your new box will appear in the workspace with the icon you selected.



07/

Double click on the box to open the Script Editor. The Script Editor shows the Python code that is executed by your box.





08/

Look for the line that says “`def onInput_onStart(self) :`”. This line is the definition for a function (or method) named `onInput_onStart`. A function is a procedure which can be called elsewhere in code. In this particular case, the `onInput_onStart` method is called when the box begins executing, and the indented code below this line is called. We will modify the code below this line to make the robot speak.

09/

Replace `pass` in the `onInput_onStart` method with the following two lines of code, as in the picture below:

```
ttsProxy = ALProxy("ALTextToSpeech")
ttsProxy.say("Hello world!")
```

```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        #~ puts code for box initialization here
        pass

    def onUnload(self):
        #~ puts code for box cleanup here
        pass

    def onInput_onStart(self):
        ttsProxy = ALProxy("ALTextToSpeech")
        ttsProxy.say("Hello world!")

    def onInput_onStop(self):
        self.onUnload() #~ it is recommended to call onUnload of this box in a onStop method, as the code
        #~ written in onUnload is used to stop the box as well
        pass
```

Note that in Python, the amount of *whitespace* (spaces and tabs) at the beginning of a line matters. Be sure to put the exact same combination of tabs and spaces on both of these lines (double the amount of whitespace of the line beginning with `def`).

The first line creates an object that gives us access to the robot’s text-to-speech capabilities. This object is assigned to the *variable* `ttsProxy`. We can access this object later through the name `ttsProxy`.

The second line calls a different function, `say`, belonging to the object we just created. This function takes an argument, “Hello world!”. “Hello world!” is a string, or a sequence of characters, denoted by the double quote marks. The robot will speak the string that was passed to the `say` method aloud.

10/

Close the script editor window, and link your Python box to the start arrow.

11/

Connect to the robot and press the play button. The robot should say “Hello world!” aloud.

ADDITIONAL EXERCISES

- 01/** Have the NAO introduce itself and greet the class.
- 02/** Have the NAO play the parts of multiple characters in a play by using different voices for each character.
- 03/** Have the NAO sing the alphabet by varying the voice shaping and pitch that it pronounces each letter with.

MODULE QUESTIONS

BASIC

- 01/** How can you tell if the robot is charging?
- 02/** How should you place the robot when turning it on?
- 03/** How should the robot be held?
- 04/** How is the order the Choregraphe boxes execute determined?

INTERMEDIATE

- 05/** What does the voice shaping parameter for the Say box control?
- 06/** What does the speed parameter for the Say box control?

ADVANCED

- 07/** Name two programming languages that can be used on the NAO.
- 08/** How can the NAO be programmed in Python using Choregraphe?
- 09/** What is a variable?
- 10/** What is a function or method in programming?
- 11/** What is a function argument?
- 12/** Do the number of spaces and tabs matter in python?

2 WALK IT OUT

LEARNING

In this module, students will learn:

- **How to make the NAO walk using Choregraphe**
- **The (x, y) coordinate plane of the NAO**

- * A-REI.3. Solve linear equations and inequalities in one variable, including equations with coefficients represented by letters.

- * 6.EE.6. Use variables to represent numbers and write expressions when solving a real-world or mathematical problem; understand that a variable can represent an unknown number, or, depending on the purpose at hand, any number in a specified set.

- * G-GPE.5. Prove the slope criteria for parallel and perpendicular lines and use them to solve geometric problems (e.g., find the equation of a line parallel or perpendicular to a given line that passes through a given point).

- **How to convert an (x, y) coordinate into an angle to turn, and a distance to walk (polar coordinates)**

- * G-C.2. Identify and describe relationships among inscribed angles, radii, and chords. Include the relationship between central, inscribed, and circumscribed angles; inscribed angles on a diameter are right angles; the radius of a circle is perpendicular to the tangent where the radius intersects the circle.

- **How to make the NAO turn and walk to an (x, y) point using Choregraphe**

- **How to use Python to program the NAO to walk.**

- * G-GPE.7. Use coordinates to compute perimeters of polygons and areas of triangles and rectangles, e.g., using the distance formula.

- **How to use Python to program the NAO to calculate the angle to turn and distance to walk, and then actuate the robot to walk to that point.**

- * RST.11-12.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.

- * RST.11-12.8. Evaluate the hypotheses, data, analysis, and conclusions in a science or technical text, verifying the data when possible and corroborating or challenging conclusions with other sources of information.

- * RST.11-12.9. Synthesize information from a range of sources (e.g., texts, experiments, simulations) into a coherent understanding of a process, phenomenon, or concept, resolving conflicting information when possible.

- * RST.11-12.10. By the end of grade 12, read and comprehend science/technical texts in the grades 11–12 text complexity band *independently and proficiently*.

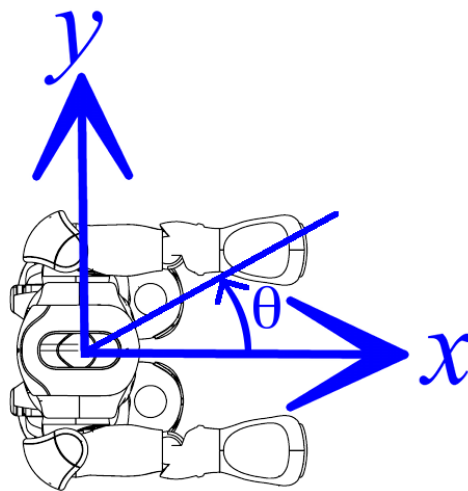
* Reference COMMON CORE Stem standards

CONTENTS

- 01/** The (x, y) Coordinate Plane of the NAO
- 02/** Basic Task: March Forward
- 03/** Intermediate Task: Walk to a Point
- 04/** Advanced Task: Walk to a Point with Python
- 05/** Advanced Task: Turn and Walk to a Point with Python
- 06/** Additional Exercises
- 07/** Additional Exercises
- 08/** Module Questions

THE (X, Y) COORDINATE PLANE OF THE NAO

You may have learned about the (x, y) coordinate plane, and plotting points in the coordinate plane. The NAO also uses the (x, y) plane to refer to places! The figures below show the NAO's coordinate frame. The figure on the left shows the (x, y) coordinate plane, when viewing the NAO from above.



The figure on the right shows the same coordinate frame in 3 dimensions.

The x -axis of the NAO points forward, and the y -axis points to the left of the robot. The z -axis, which is perpendicular to both the x - and y -axes, points up.

The units of the NAO's (x, y) coordinate plane are meters. So, for example, $(1, 2)$ refers to a point 1 meter in front of the NAO, and 2 meters to its left.

In addition to points in the (x, y) coordinate plane, angles are also defined. An angle is measured counterclockwise from the x -axis, as shown by θ in the figure above on the left.

BASIC TASK

MARCH FORWARD

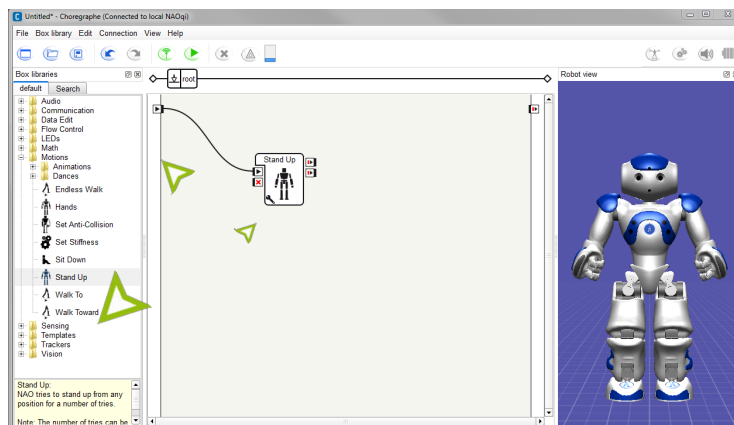
WE'LL START BY MAKING THE NAO STAND UP, WALK FORWARD, AND SIT DOWN.

01/

Make sure the NAO is in a stable position. Turn the NAO on and connect to it with Choregraphe.

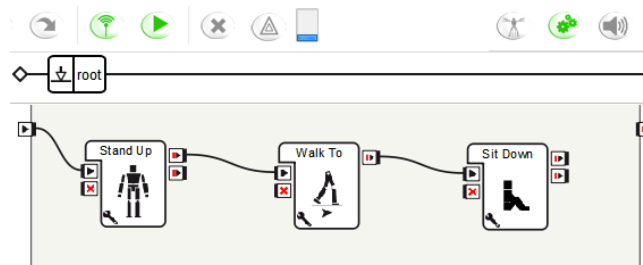
02/

Go to the Motions folder on the box list, and drag the Stand Up box to the workspace. Connect a line from the starting box to the motion box.



03/

Drag a Walk box and then a Sit Down box (both boxes are in the Motion category), and connect them in that order to the Stand Up box.

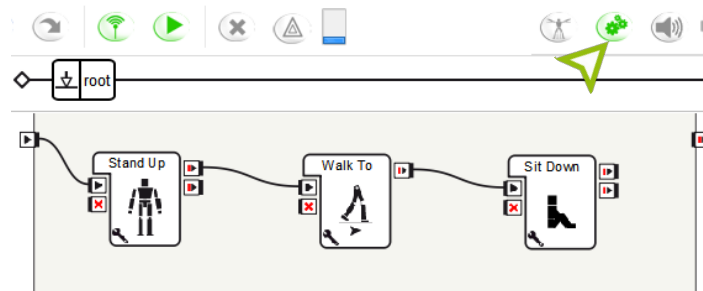




04/

Click the “Stiffness” button (with two gears) marked in the picture below. The button will turn red and the robot will become stiff, such that you cannot move the robot’s joints by hand.

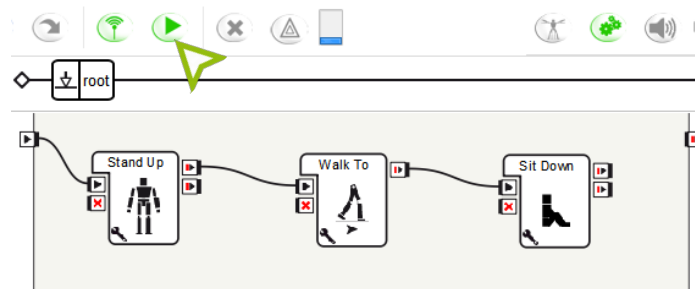
Warning: Do not try to force the joints, and be careful to avoid getting your fingers squished by the robot.



While the joints are stiff, battery power will be consumed at a faster rate, and the motors will heat up.

05/

Finally, click the play button. The robot will stand up, walk forward, and sit down.



After the robot has completed its motion, click the stiffness button again to remove stiffness from the robot, and the button will turn green. **Be careful!** The robot's joints will lose stiffness immediately, and if the robot is in an unstable position it will fall. Hold onto the robot when turning stiffness off.

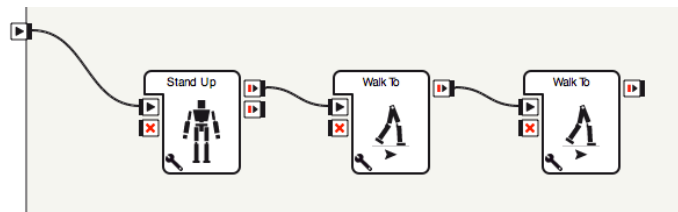
INTERMEDIATE TASK

WALK TO A POINT

IN ADDITION TO WALKING FORWARDS, THE NAO CAN TURN IN PLACE. IN FACT, THE NAO HAS AN OMNIDIRECTIONAL WALK THAT CAN MOVE IN ANY DIRECTION – FORWARD, SIDWAYS, BACKWARDS, OR DIAGONALLY. BUT FIRST WE WILL LEARN HOW TO TURN.

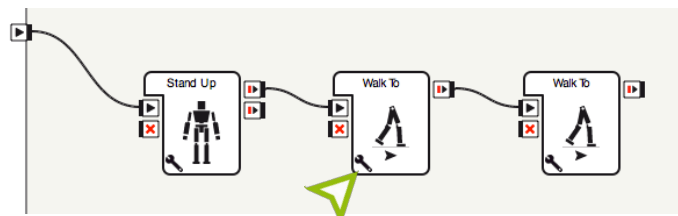
01/

In Choregraphe, string together a “Stand Up” box and two “Walk To” boxes (both found under Motions).

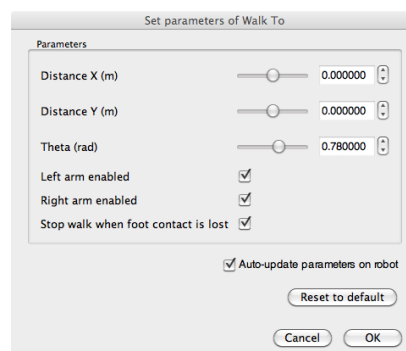


02/

Click the wrench on the first Walk To box to configure its parameters.



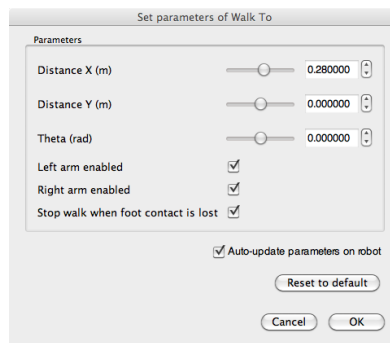
The dialog box allows you to specify how far in the x direction (forward), y direction (sideways), and theta (rotation) the robot should move. Distances are in meters (negative indicates the opposite direction) and rotations are in radians. Set the x and y to 0, and set some amount for the robot to turn by setting theta to be non-zero.





03/

Open the same dialog for the second box. Set the x distance to some positive amount, and the other two values to zero. This will cause the robot to walk forward. With both boxes, the robot will turn and then walk forward. While the joints are stiff, battery power will be consumed at a faster rate, and the motors will heat up.



04/

(Optional) Add a sit down box to the chain of commands.

05/

Turn stiffness on and click the play button. Observe that the robot moves where you told it to.

06/

Consider the floor as a Cartesian plane. The origin is between the NAO's feet, the x -axis is in the forward direction, and the positive y -axis is to the robot's left. Pick some coordinate on the floor, such as 1 meter forward and 0.5 meters to the left.

07/

Using trigonometry, compute what angle the robot should turn and how far it should walk to reach this position. Set these values in the two Walk To boxes.

08/

Run the program again on the NAO. See that it goes where you asked.



09/

In robotics, the estimated change in position of a robot measured from sensors is called its *odometry*. Using a ruler or tape measure, measure the exact position where the robot ended up. How far does it differ from the position you entered? This difference is called odometry error.

Why do you think this error occurs? Would you predict the error would be smaller or greater on a wheeled robot? Much of the difficulty in programming robots as opposed to computers comes from the need to handle errors and noise such as this.

ADVANCED TASK

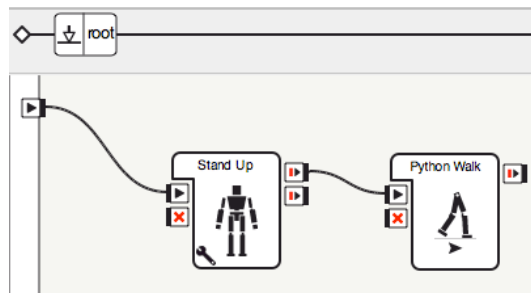
WALK TO A POINT WITH PYTHON

—

NEXT, WE WILL LEARN TO WALK TO A POINT USING PYTHON. PYTHON IS MORE EXPRESSIVE THAN CHAINS OF CHOREGRAPHE BOXES, AND ALLOWS US TO DO THINGS LIKE COMPUTE THE TRIGONOMETRIC CALCULATIONS THAT WE DID MANUALLY ON THE ROBOT.

01/

First, create your own box (right click on workspace, "Add a New Box"). Set a name, description and image, and click OK. Chain together a Stand Up Box with your new box as shown below.



02/

Double click on your custom box to open the script editor.

03/

Enter the following lines of code in the `onInput_onStart` method:

```
motionProxy = ALProxy("ALMotion")
motionProxy.walkTo(0.2, 0.0, 0.0)
self.onStopped()
```



```
Script Editor
Python Walk

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        #~ puts code for box initialization here
        pass

    def onUnload(self):
        #~ puts code for box cleanup here
        pass

    def onInput_onStart(self):
        motionProxy = ALProxy("ALMotion")
        motionProxy.walkTo(0.2, 0, 0)
        # The walk is finished so output
        self.onStopped() #~ activate output of the box

    def onInput_onStop(self):
        self.onUnload() #~ It is recommended to call onUnload of this box in a onStop method, as the code written in onUnload is
        # used to stop the box as well
        pass
```

The first line creates a proxy to `ALMotion`, which allows us to call motion functions. The second line calls the `walkTo` method, which moves the robot a specified distance.

The first argument (number in parentheses) is the distance in meters to walk in the x direction. The second argument is the distance in the y direction, and the third is the amount to turn in radians. These parameters have the same meanings as the parameters to the Choregraphe box. You can change these numbers if you wish.

The final line calls a method to terminate the Choregraphe box and transfers execution to the next box.

Once again, recall that the three lines must have the same indentation level. Note that everything in a line following a `#` sign will be ignored: these are comments. Comments are intended to help the programmer understand the code, but are not executed by the computer.

04/

Turn stiffness on and click play. The robot should walk forward.

05/

(Optional) Try other combinations of parameters to the `walkTo` method, and observe what happens.

ADVANCED TASK

TURN AND WALK TO A POINT WITH PYTHON

NOW, WE WILL IMPLEMENT OUR BEHAVIOR TO TURN AND THEN WALK FORWARD TO A POINT USING PYTHON.

01/

Create a new Choregraphe box by right clicking on the workspace and choosing “Add a New Box.”

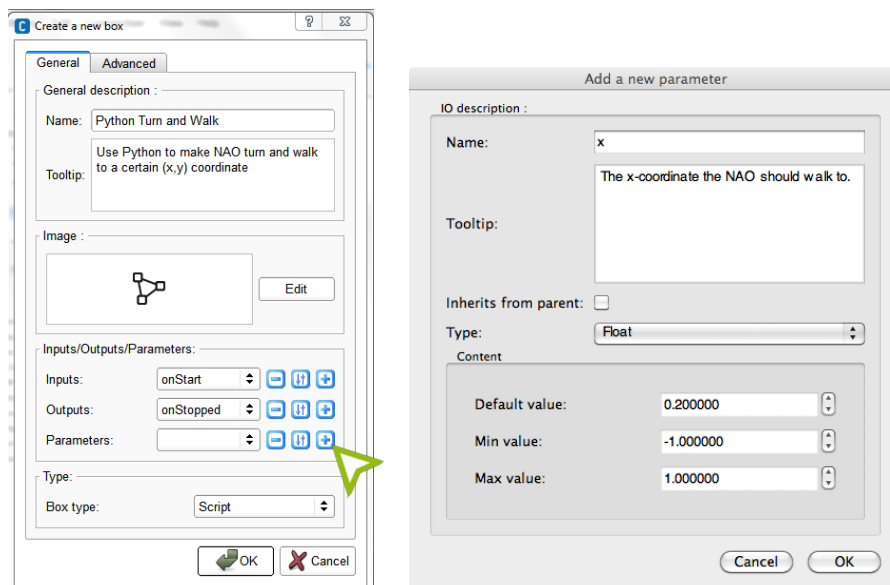
02/

In the Edit Box window, enter an appropriate name and description, and choose an image.

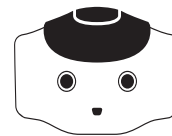
03/

This time, our box will have two parameters: the x and y coordinates on the Cartesian plane that the robot should walk to. Click the plus button on the line that says “Parameters” to add a new parameter.

04/



A dialog box will appear. Set the parameter’s name to “x”, enter a description, set it to type float, and enter a default value of 0.2. Set the minimum and maximum values to -2.0 and 2.0, respectively. Then click OK.



05/

Add a second parameter, “y”. Enter the same values as you did for “x”, aside from the name and description.

IO description :

Name: y

The y-coordinate the NAO should walk to.

Tooltip:

Inherits from parent:

Type: Float

Content

Default value: 0.200000

Min value: -1.000000

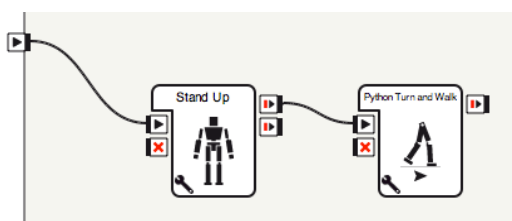
Max value: 1.000000

Cancel OK

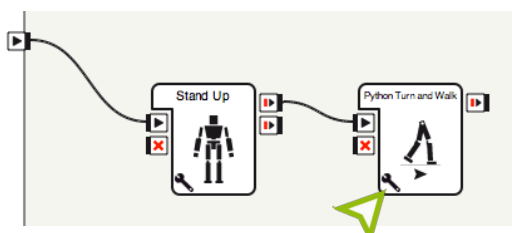
06/

Click OK to construct the box, and connect it to a Stand Up box connected to the starting arrow.

07/



Click the wrench on the box you created. You can set the x and y parameters similarly to how you would for any other box. Enter values of your choice.





08/

Double click on the box you created to edit the python source code. Enter the contents of the `onInput_onStart` method shown below, and add `import math` to the top of the script, which allows usage of some math functions (such as `atan2` and `sqrt`).

This code takes the x and y parameters, computes the angle to turn using the `atan2` (arctangent) function, and computes the distance to walk forward using the Pythagorean theorem, `sqrt` (square root) function, multiplication and addition. It then calls the same `walkTo` function we used before twice in succession: first to turn and then to walk forwards.

```
import math
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
    def onLoad(self):
        #- puts code for box initialization here
        pass
    def onUnload(self):
        #- puts code for box cleanup here
        pass
    def onInput_onStart(self):
        x = self.getParameter("x")
        y = self.getParameter("y")
        amountToTurn = math.atan2(y, x)
        amountToWalk = math.sqrt(x * x + y * y)

        motionProxy = ALProxy("ALMotion")
        motionProxy.walkTo(0, 0, amountToTurn)
        motionProxy.walkTo(amountToWalk, 0, 0)

        # The walk is finished so output
        self.onStopped() #- activate output of the box
    def onInput_onStop(self):
        self.onUnload() #- it is recommended to call onUnload of this box in a onStop method, as the code written in onUnload
        is used to stop the box as well
        pass
```

09/

(Optional) Add a Sit Down box as the final command.

10/

Turn stiffness on and hit play. The robot should walk to the location you specified.

ADDITIONAL EXERCISES

- 01/** Have the NAO walk in a square, that is, walk forward and turn, walk forward and turn, walk forward and turn, and walk forward again.
- 02/** Have the NAO walk in a triangle.
- 03/** Program a Python script that causes the NAO to walk in a square or triangle.
- 04/** Program a Python script that causes the NAO to walk in any regular polygon with n sides ($n > 2$).

MODULE QUESTIONS

THE COORDINATE PLANE

- 01/** Sketch the coordinate plane of the NAO, and plot and label the following points:
- a. The point (0.5, 2).
 - b. The point 3 meters directly in front of the robot.
 - c. The point (5, 3).
 - d. The point 1 meter to the left of the robot.
 - e. The point at an angle of -60° from the robot and 3 m away.
- 02/** Compute the angle from the robot to each of the above five points.

BASIC

- 03/** What are the three parameters in the Walk To box?
- 04/** What unit of measure is used for x and y coordinates?
- 05/** What is theta?
- 06/** What unit of measure is used for theta?
- 07/** For the NAO, what does “stiffness” mean?
- 08/** What problems do you face if stiffness is always set to 100%?

INTERMEDIATE

- 09/** What is odometry?
- 10/** Why do odometry errors occur?
- 11/** Define the Cartesian plane, and explain how it relates to the NAO walking.

ADVANCED

- 12/** What is a parameter?
- 13/** Name the parameters used to make NAO walk.
- 14/** Name three math functions used in programming walking for robot.
- 15/** What do $\text{atan2}(1.0, 0.0)$, $\text{atan2}(0.0, -1.0)$, and evaluate to?
- 16/** Why is atan2 used instead of a single-argument arctan function?

3 HEARING THINGS

LEARNING

In this module, students will learn:

- **What speech recognition is**
- **How to perform speech recognition on the NAO**
- **How to vary the threshold of speech recognition**
- **Boolean (true/false) operations**
- **Branching conditionals (switch statements) for complex logic operations**
- **How to create speech-based interaction behaviors with the NAO**
- **How to process strings in Python**
- **How to use if statements in Python**
 - * RST.11-12.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.
 - * RST.11-12.8. Evaluate the hypotheses, data, analysis, and conclusions in a science or technical text, verifying the data when possible and corroborating or challenging conclusions with other sources of information.
 - * RST.11-12.9. Synthesize information from a range of sources (e.g., texts, experiments, simulations) into a coherent understanding of a process, phenomenon, or concept, resolving conflicting information when possible.
 - * RST.11-12.10. By the end of grade 12, read and comprehend science/technical texts in the grades 11–12 text complexity band independently and proficiently.

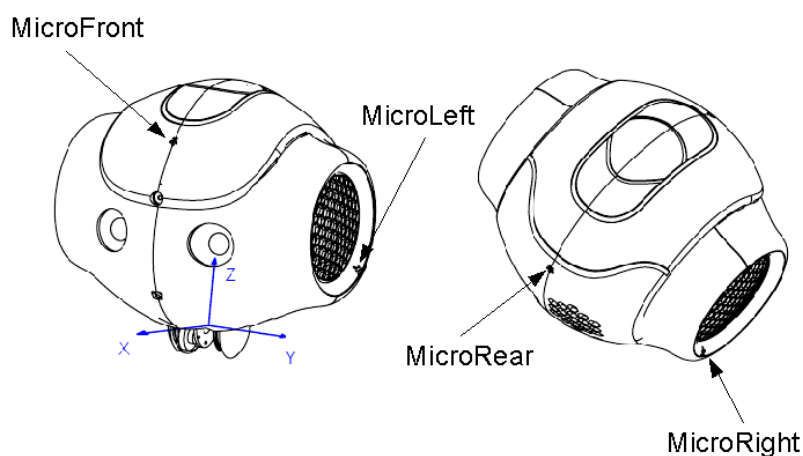
* Reference COMMON CORE Stem standards

CONTENTS

- 01/** Speech Recognition on the NAO
- 02/** Basic Task: Speech Recognition
- 03/** Intermediate Task: Distinguishing Multiple Names
- 04/** Advanced Task: Self-Introductions
- 05/** Advanced Task: Specialized Introductions with if Statements
- 06/** Additional Exercises
- 07/** Module Questions

SPEECH RECOGNITION ON THE NAO

Humans frequently communicate through speech. For example, a common greeting when we meet someone is “hi” or “how are you?” We process speech automatically, and understand the meaning of the words we hear nearly instantaneously. On a robot, this process is more involved. The NAO humanoid robot has microphones on its head, which it uses to listen to sounds around it.



However, unlike our ears that listen for sounds all the time, the NAO has to be programmed to listen for sounds at specific times. After it hears human speech, the NAO performs speech recognition with an algorithm to convert what it hears into words that it knows.

To do so, the NAO requires a library of words that it expects to hear. For example, the library can contain two words, “yes” and “no”. When the NAO processes the sounds it hears, it will classify it as either “yes”, “no”, or neither of the two. You may have had experience with a similar system when using automated phone services or voice control on your cell phone, where you are given a list of options that you can speak to select.

Once a word is recognized, the NAO can then be programmed to react in different ways. After hearing “yes”, the NAO could reply with “I am happy” and after hearing “no”, the NAO could say “I am sad”. If the NAO doesn’t understand the words (it did not sound like “yes” or “no”) then the NAO could reply “I don’t know.” This is called a *conditional* in computer or robot programming, and we will go into more detail in the tasks below.

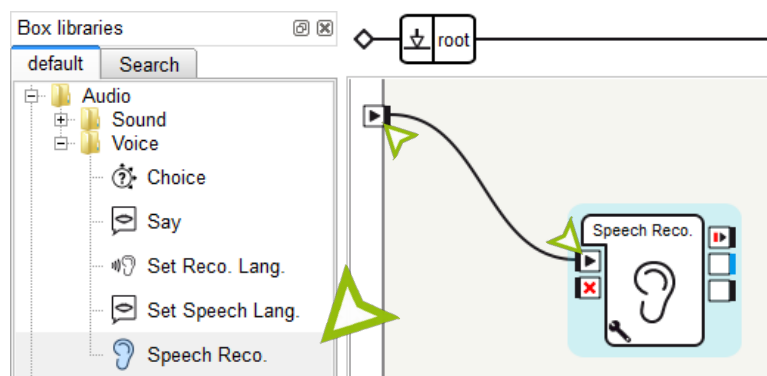
BASIC TASK

SPEECH RECOGNITION

IN THIS LESSON, WE'LL LEARN HOW TO USE SPEECH RECOGNITION ON THE NAO. WE WILL PROGRAM THE NAO TO RECOGNIZE ITS OWN NAME AND TO GIVE A GREETING IN RESPONSE.

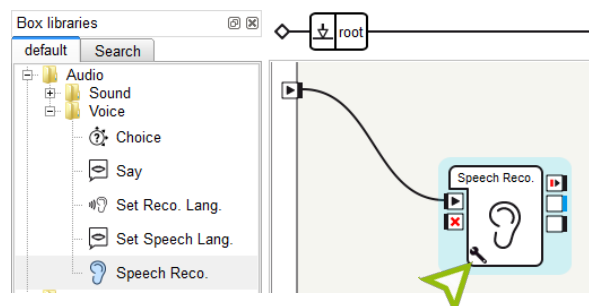
01/

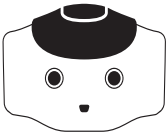
First, drag a Speech Recognition box (found in Audio → Voice) to the workspace and link it to the start arrow.



02/

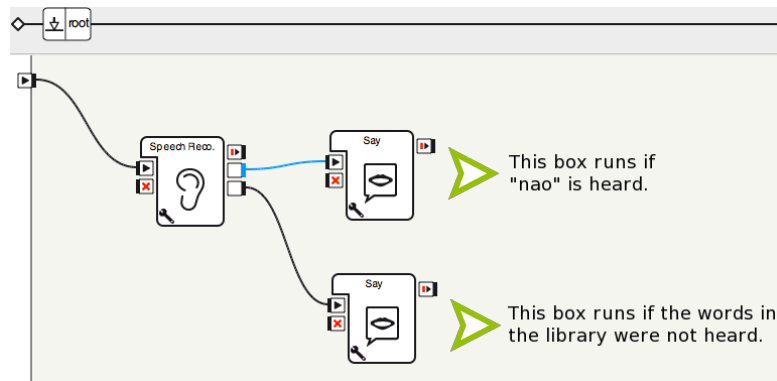
Click on the wrench to examine the parameters for speech recognition. Set the word list to "nao". This defines the library of known words the NAO will listen for. Putting "nao" in the library will make the NAO listen for its own name. Set the threshold slider bar to 10%. The threshold controls how similar the sound has to be for the NAO to recognize it; we will discuss this in more detail later.





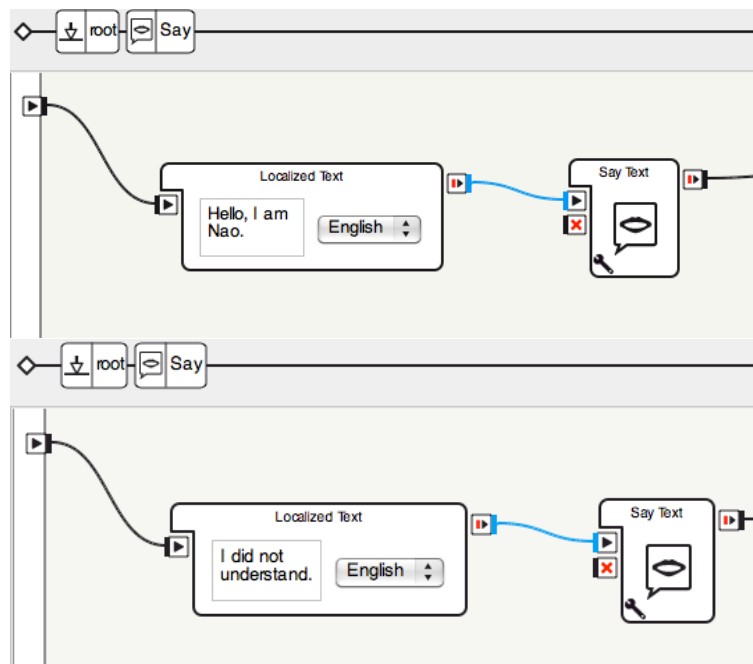
03/

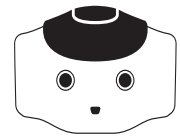
Next, create two Say boxes and connect one to each of the two bottom right connectors on the speech recognition box. The top connector is triggered when a word you entered in the word list is heard, and the bottom box is triggered when it is not. This construct is called a *conditional*.



04/

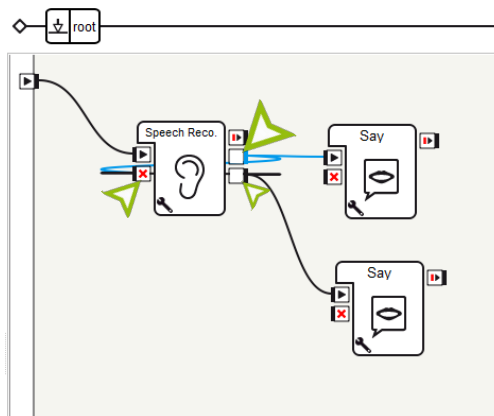
Double click on each of the Say boxes to configure what the NAO will say. The top connector triggers when the robot hears the word "nao". Change this Say box to say a greeting such as "Hello, I am Your Robot." Configure the second box to make the NAO say that it didn't understand.





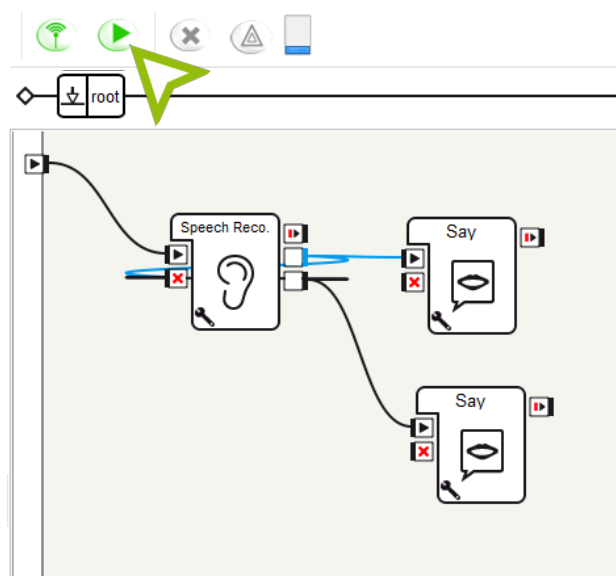
05/

Finally, connect both the boxes on the lower right of the Speech Recognition box to the X mark on the left side of the same box. This will make the robot stop listening for new words after hearing the first one.



06/

Now, hit the play button on Choregraphe. You will hear a sound on the NAO that indicates that it is listening. Its eyes will turn blue in color as well. Once the NAO hears a human speak, its eyes will turn yellow. If it understood the words it heard, the eyes will flash green, and if it did not understand, the eyes will flash red. When the NAO stops listening on its microphones, it makes another sound through its speakers.



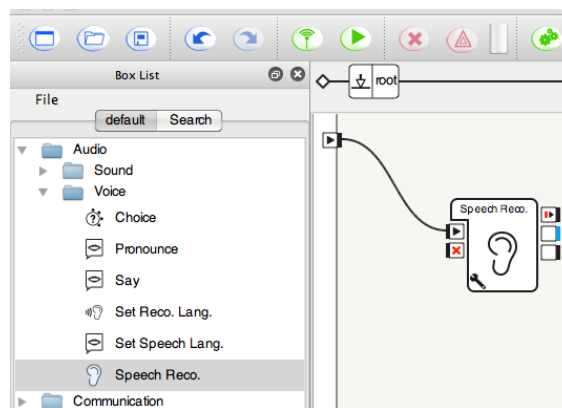
INTERMEDIATE TASK

DISTINGUISHING MULTIPLE NAMES

IN THIS NEXT EXERCISE, WE WILL PROGRAM THE ROBOT TO LISTEN FOR DIFFERENT NAMES AND RESPOND DIFFERENTLY AFTER HEARING EACH ONE.

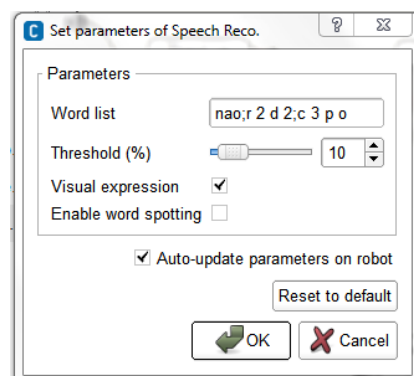
01/

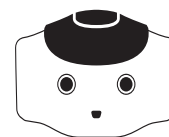
Drag a Speech Recognition box to the workspace.



02/

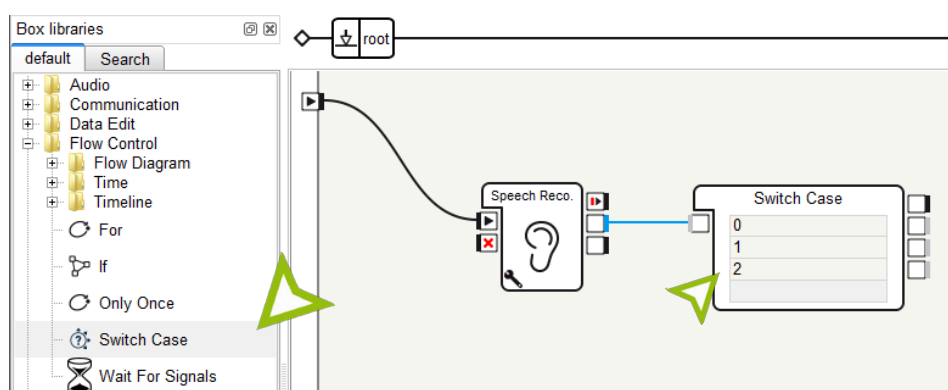
Click the wrench to configure the word list. Set the word list to "nao;r 2 d 2;c 3 p o". The semicolons separate different words in the library. Do not forget to add the spaces. Why are these spaces necessary? Once again, set the threshold to 10%.





03/

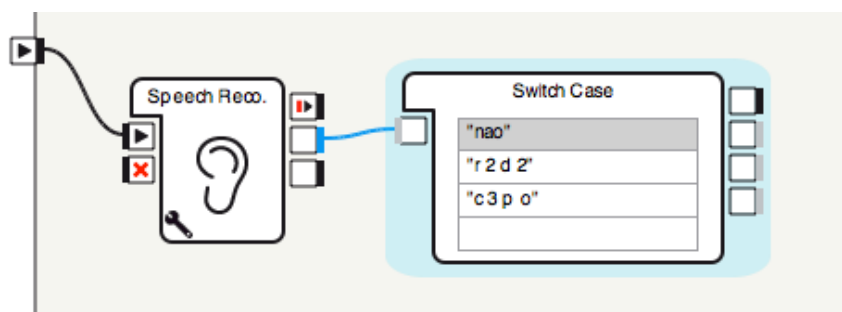
Next, add a Switch Case box. This is found in the list of flow control tools. The Switch Case box compares its input to a list of predefined values, and triggers one of the boxes to the right based on its input.

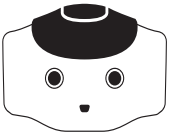


04/

Change the predefined inputs of the Switch Case box to match the words in the speech recognition library. You need to add quotation marks around each word to indicate that it is a string, e.g., "nao". Note also that the capitalization should be the same as in the speech recognition library—the strings must match exactly. When a word in the speech recognition library is heard, the Speech Recognition box sends that word to the Switch Case box.

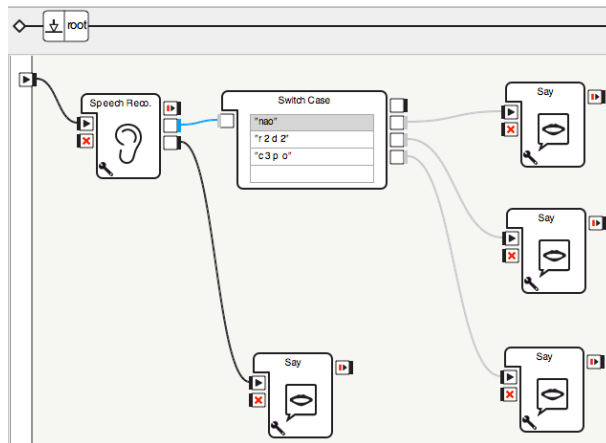
The Switch Case box then compares its input with each of the cases ("nao", "r 2 d 2", and "c 3 p o") and triggers the appropriate box to the right of the matching word.





05/

Now add four Say boxes, and connect the boxes as shown below.

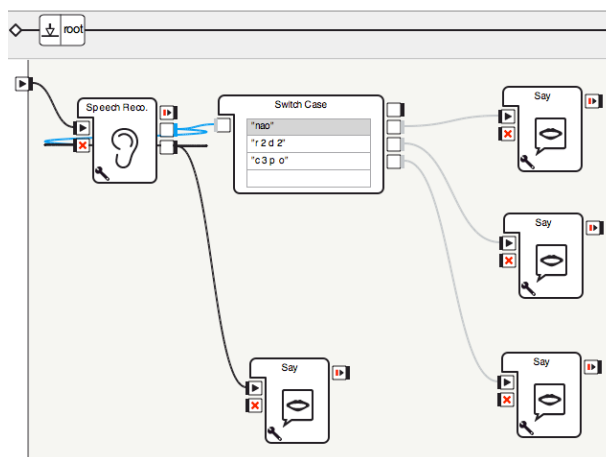


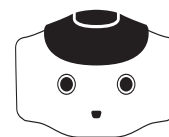
06/

Add messages to the Say boxes. The Say box connected to the Speech Recognition box will be what the robot says if it doesn't understand the human. The three boxes connected to the Switch Case box will be the robot's responses when it hears the corresponding messages. For example, the Say box on the top right will be triggered when "nao" is heard.

07/

Finally, connect the two bottom right boxes on the Speech Recognition box to the X mark on the same box, as we did in the previous exercise.





08/

Press play and try saying the different names.

09/

You may notice that the speech recognition is not perfect. For example, the robot may hear “c 3 p o” when you said “r 2 d 2”. Or, the robot may not understand what you said, even though you said “nao”. The level of recognition can be adjusted via the threshold in the Speech Recognition box.

Recall that we set the threshold to 10% in the Speech Recognition box. This means that the robot has to only be 10% sure of what it hears to recognize the word. Why not set the threshold to a much higher number like 90% then? If the threshold is too high, then the robot may not understand the words you say because it is unsure.

Try changing the value of the threshold, to get a better idea of what it does and how it affects the speech recognition. What value works best?

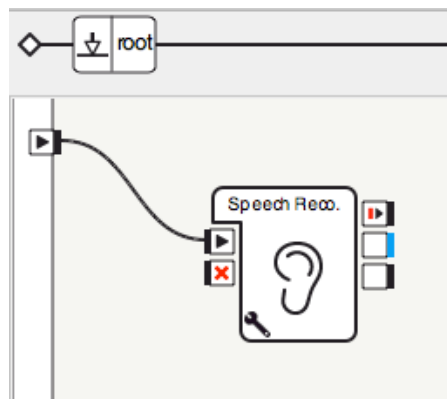
ADVANCED TASK

SELF-INTRODUCTIONS

IN THIS EXERCISE, WE WILL CREATE A BOX USING PYTHON THAT INTRODUCES THE NAO BASED ON WHICHEVER NAME A HUMAN SAYS.

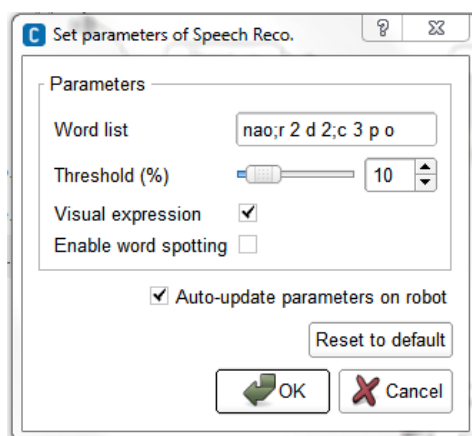
01/

Create a Speech Recognition box and connect it to the start arrow.



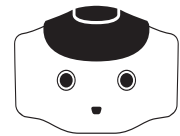
02/

Set the word list and threshold as in the previous exercises.



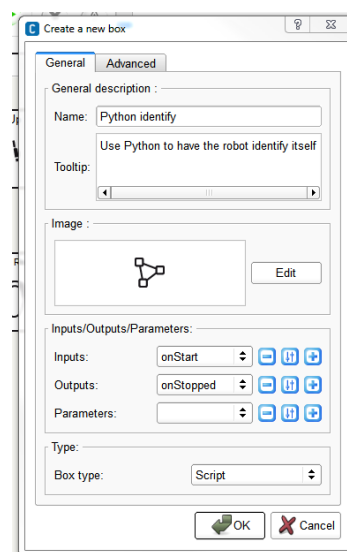
03/

Create a new box



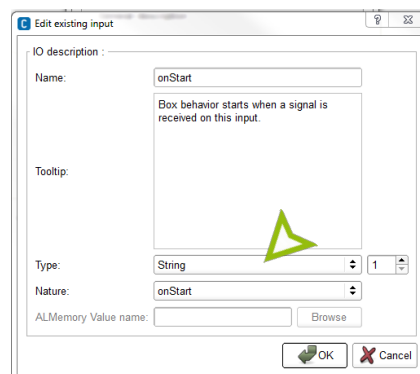
04/

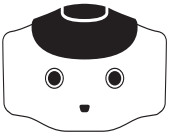
(right click, choose “Add a New Box”). Choose an appropriate (for example Speech box) name, tooltip and image. In the Inputs / Outputs / Parameters section, click the center button to the right of “Inputs: onStart”, the first line. This allows us to set the properties of the onStart input.



05/

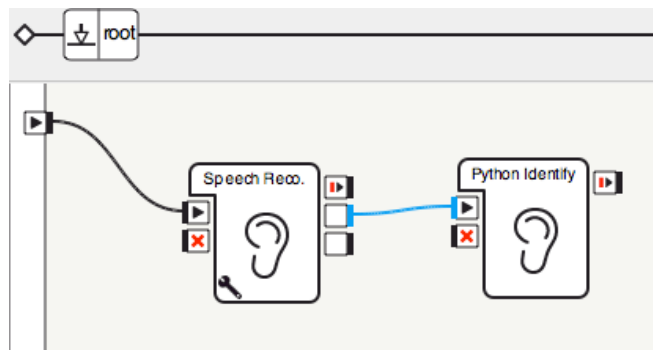
A dialog box appears. Change the type to string, and click OK. This makes the input to the box take a string (a sequence of characters).





06/

The Speech Recognition box outputs a string, the word that has been recognized. Now we can link the two boxes together since the `onStart` input accepts a string.



07/

Double-click on the new box to edit the Python source code. Add the following two lines to the `onInput_onStart` method.

```
ttsProxy = ALProxy("ALTextToSpeech")
ttsProxy.say("Hello, I am " + p)
```

In the earlier exercises, we created an `ALTextToSpeech` proxy and called the `say` method. What's new is the addition of `"Hello, I am " + p`. This is called string concatenation, appending two strings together. The `p` is a parameter to the `onInput_onStart` function. This parameter is set to the value arriving at the `onStart` input, in this case sent by the Speech Recognition box.

For example, if the Speech Recognition box heard "nao", then the expression `"Hello, I am " + p` would evaluate to "Hello, I am nao".

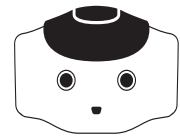
```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        #~ puts code for box initialization here
        pass

    def onUnload(self):
        #~ puts code for box cleanup here
        pass

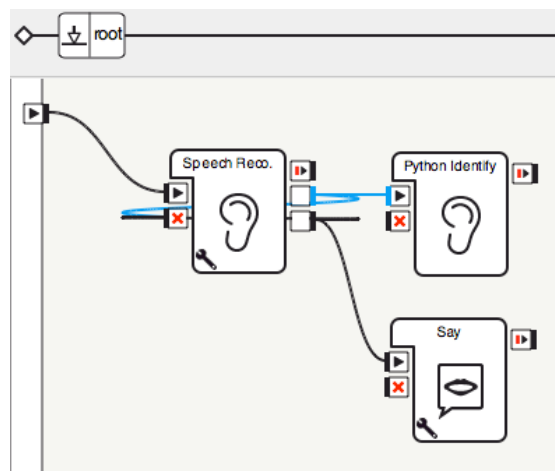
    def onInput_onStart(self, p):
        ttsProxy = ALProxy("ALTextToSpeech")
        ttsProxy.say("Hello, I am " + p)

    def onInput_onStop(self):
        self.onUnload() #~ it is recommended to call
        #~ is used to stop the box as well
        pass
```



08/

Next, add a Say box which says “I did not understand,” and connect it to the bottom output of the Speech Recognition box. Link the two bottom right outputs of the Speech Recognition box to the X mark on the same box, as we have done before.



09/

Hit play and try speaking the different names.

ADVANCED TASK

SPECIALIZED INTRODUCTIONS WITH *IF* STATEMENTS

—

UNTIL NOW, WE HAVE ALWAYS EXECUTED THE SAME CODE IN EACH PROGRAM. BUT WE CAN ALSO USE *CONDITIONALS* TO EXECUTE CODE ONLY IF SOME CONDITION IS SATISFIED. WE WILL MODIFY THE PREVIOUS EXERCISE TO GIVE SPECIALIZED GREETINGS FOR EACH ROBOT NAME.

01/

Begin with the result of the previous exercise.

02/

Double click on the custom box to edit the Python source code.

03/

Enter the code shown below in the `onInput_onStart` method.

```
def onInput_onStart(self, p):
    ttsProxy = ALProxy("ALTextToSpeech")

    if p == "nao":
        ttsProxy.say("Hello, I am a Nao humanoid robot.")
    elif p == "r 2 d 2":
        ttsProxy.say("Hello, I am R 2 D 2.")
    elif p == "c 3 p o":
        ttsProxy.say("Hello, I am C 3 P O, human cyborg relations.")
```

What does this code do? You should already be familiar with the `ALTextToSpeech` proxy and its `say` method. The `if` statement checks if the following condition is satisfied, and if so, executes the code after the colon. The `p == "nao"` condition is satisfied if and only if the string `p` is "nao". The two `elif`s (short for "else if") are the same as the `if` statement, except they are executed only if the previous `if` statement was not satisfied (this is the "else" part). So, this code speaks a specific message based on the output of the speech recognition box.

04/

Run the program and try speaking the different names.

ADDITIONAL EXERCISES

- 01/** Have the NAO ask “How are you?” Depending on what you say, the NAO should reply “That’s good to hear” or “I hope your day gets better!”
- 02/** Create a chain of interactions with the NAO, where it first asks if you like cake or pie. If you like cake, it asks if you like chocolate cake or cheesecake. If you like pie, the NAO asks if you like apple pie or pumpkin pie. Finally, depending on the type of cake or pie you like, the NAO will say something appropriate relevant, such as “I like chocolate cake too, especially with whipped cream on top.”
- 03/** Create a voice-controlled robot, where you can tell the NAO to walk forward, turn left or turn right, and the NAO executes the action you said.
- 04/** Write a Python script to execute the actions in the exercise above.

MODULE QUESTIONS

BASIC

- 01/** What hardware devices does the robot use for speech recognition?
- 02/** What is a conditional statement?
- 03/** What is the purpose of the threshold parameter for speech recognition with NAO? What value did you find to work well?
- 04/** Where is the Speech Recognition box found?
- 05/** Explain the three parameters of the Speech Recognition box.
- 06/** Explain the two outputs of the Speech Recognition box.

INTERMEDIATE

- 07/** Which conditional box in Choregraphe is used to control program flow from multiple answers?
- 08/** What punctuation is used to distinguish strings, in both Choregraphe and Python?

ADVANCED

- 09/** What module is utilized in TextToSpeech (the parameter to ALProxy)?
- 10/** Define string concatenation.
- 11/** What are the differences between inputs and parameters of Choregraphe boxes?
- 12/** What is the syntax in Python for a chain of conditional statements?

4 LET'S DANCE

LEARNING

In this module, students will learn:

- **What a keyframe motion is**
- **Applying the concept of center of gravity on the NAO**
- **How to keep the NAO balanced while adjusting its pose**
- **How to record and execute keyframes on the NAO with Choregraphe**
- **How to make the NAO perform a dance routine**
- **How to use loops in Choregraphe**
- **How to execute joint motions in Python**
- **How to use for loops in Python**

- * RST.11-12.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.
- * RST.11-12.8. Evaluate the hypotheses, data, analysis, and conclusions in a science or technical text, verifying the data when possible and corroborating or challenging conclusions with other sources of information.
- * RST.11-12.9. Synthesize information from a range of sources (e.g., texts, experiments, simulations) into a coherent understanding of a process, phenomenon, or concept, resolving conflicting information when possible.
- * RST.11-12.10. By the end of grade 12, read and comprehend science/technical texts in the grades 11–12 text complexity band independently and proficiently.

* Reference COMMON CORE Stem standards

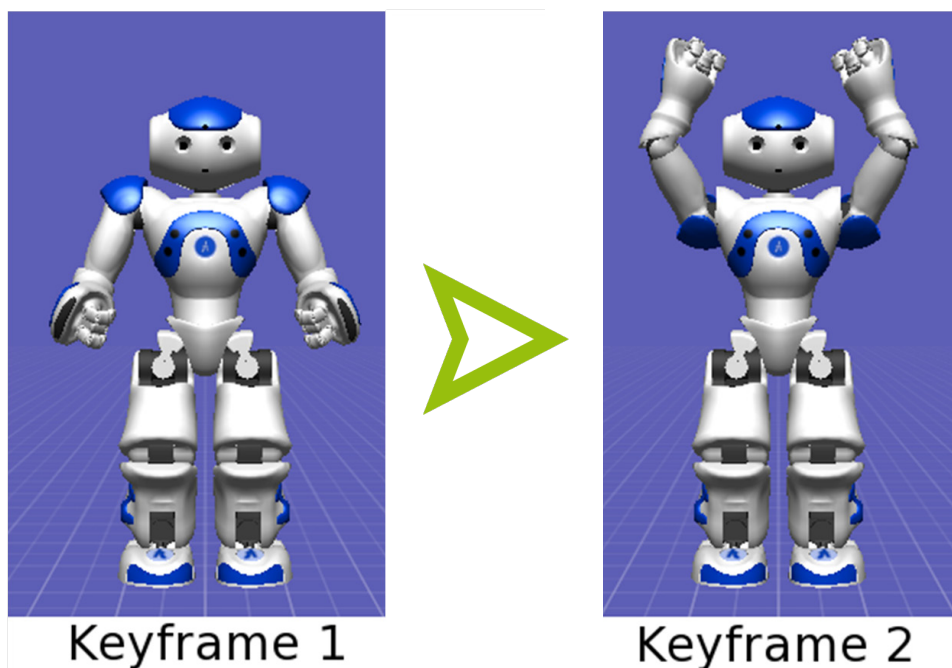
CONTENTS

- 01/** Keyframe Motion
- 02/** Balancing the NAO
- 03/** Basic Task: Macarena Hand Motions
- 04/** Intermediate Task: Do the Macarena
- 05/** Advanced Task: Nodding Off
- 06/** Additional Exercises
- 07/** Module Questions

KEYFRAME MOTION

We move our own bodies fluidly, but robots are stereotyped to move in a clunky “robotic” fashion. Why is this so? Some robots move in a “bang-bang” manner, where they would move a joint (such as an arm) at maximum speed and then suddenly stop. The first “bang” refers to suddenly moving at maximum speed. The second “bang” refers to suddenly stopping.

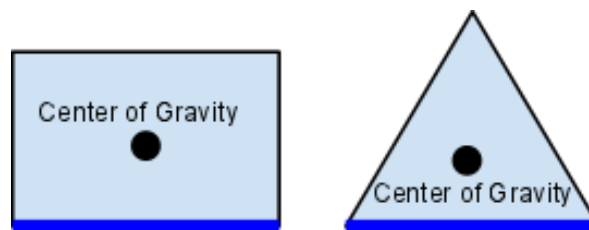
However, robots are capable of more fluid motion. One method to define motions for a robot is through *keyframes*. A keyframe is a set of joint positions of the robot. One keyframe could be the robot in a standing position with its arms by its sides (Keyframe 1 in the figure below). Another keyframe could be the robot in the standing position, with its arms above its head (Keyframe 2 below). In keyframe motion, the robot’s joints transition fluidly from one keyframe to the next. So, for the two keyframes below, the robot would remain standing, and smoothly move its arms from its sides to in the air above its head.



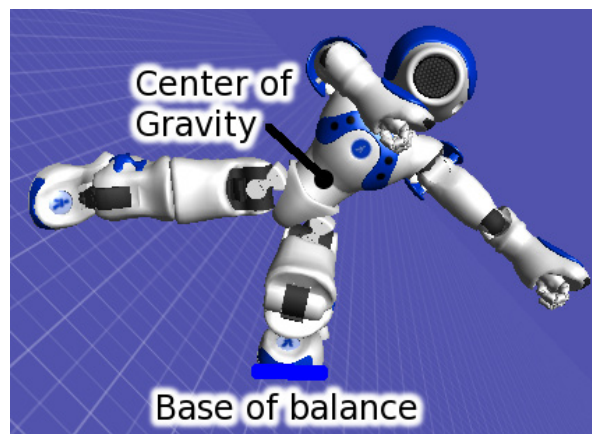
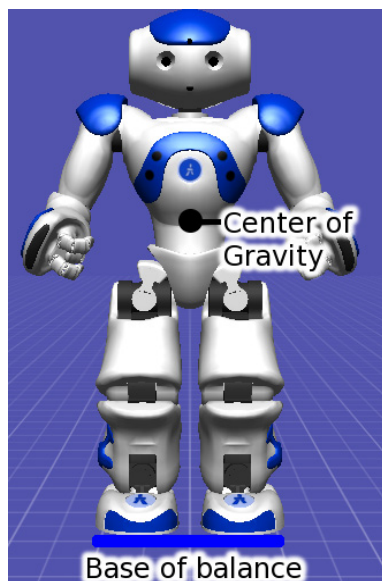
By defining a sequence of keyframes, you can describe complex motions of the robot. Keyframe motion is a simple technique to design motions such as waving the robot’s hands, moving its feet, and even dance routines. Keyframes have also been used in robot soccer. Robots are able to kick the soccer ball in many directions - forward, to the sides, and even backwards!

BALANCING THE NAO

As you may have already learned from physics class, objects have a center of gravity. An object's center of gravity affects whether its pose is stable. For rigid bodies, the object is stable if the center of gravity lies within the base of the object (see figure below).



For the NAO, the same concept applies. When the NAO is standing on both feet, the base lies between both feet. When the NAO is standing on one foot, the base lies only across that foot. The NAO's center of gravity is approximately located in the torso. Because the NAO can change its "shape" when different parts of its body (arms, legs, head) move, the center of gravity will also shift. You can create complex postures with the NAO while keeping it balanced!



When adjusting the pose of the NAO, check that the weight of the NAO is resting on both feet. Sometimes, even when both feet appear to be on the ground, all the weight of the NAO is actually resting on a single foot. Such a position is less stable, and can cause the NAO's motors to heat up more quickly. Also, it becomes more difficult to keep the NAO in balance, since the center of gravity must remain within the bounds of that single foot.

One way to test the weight distribution is to try to move the foot while stiffness is on. If there is weight resting on that foot, then it will be considerably more difficult to move the foot. The foot will feel "heavier" than if no weight was resting on it.

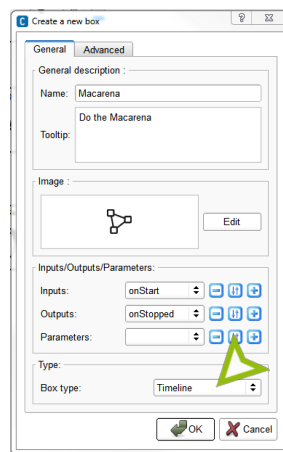
BASIC TASK

MACARENA HAND MOTION

CHOREGRAPHE ALLOWS US TO RECORD BODY POSITIONS (KEYFRAMES) AND COMBINE THEM ON A TIMELINE TO MAKE CONTINUOUS MOTIONS. WE WILL LEARN HOW TO CREATE KEYFRAMES BY MAKING THE NAO DO THE MACARENA.

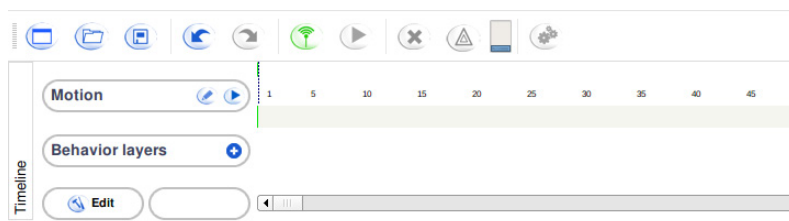
01/

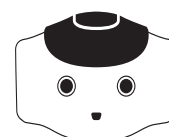
First, create a new box. For “Box type”, select “Timeline”. This will allow us to edit keyframes on a timeline.



02/

Double click on the selected box. The timeline editor will appear, as shown below.





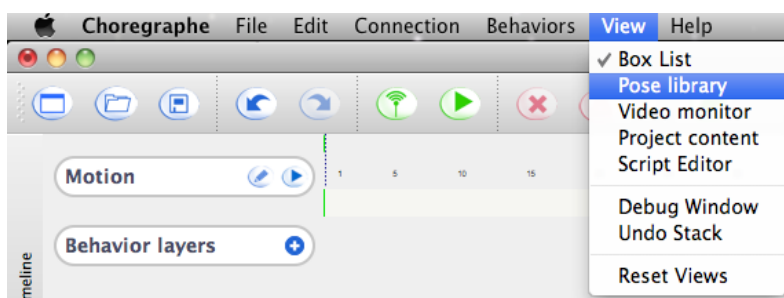
The timeline is used to view and modify individual keyframes, which appear as small rectangular black boxes on the timeline. The numbers on the timeline refer to the sequence of frames - each number represents a duration of 10 milliseconds, so the NAO will take 100 ms to reach frame 10, 150 ms to reach frame 15, and so on. This *frame rate* can be adjusted.

You can click on a keyframe to modify it. The NAO *interpolates* its joints between keyframes. Interpolation means that the NAO will move its body smoothly from one keyframe to another automatically.

If you connect to the NAO, you can view and save keyframes with the actual robot. If you connect to a "local Naoqi", you can view and save keyframes using the virtual 3D NAO on the right side of Choregraphe.

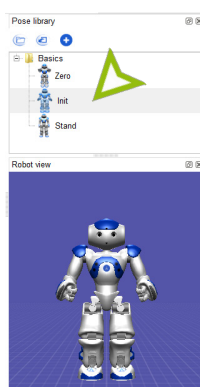
03/

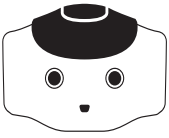
We want to start the dance in a standing position. To do so, open the Pose Library (click View → Pose Library in the top menu). The Pose library will appear above the 3D NAO.



04/

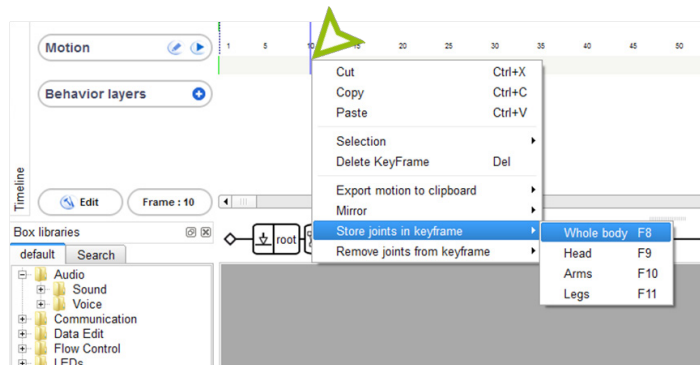
Connect to the NAO, and turn stiffness on. While holding the NAO to prevent it from falling, click on the Init pose in the Pose Library. The NAO will move to a standing position.





05/

We want to save this pose as a keyframe. Right click on frame 10 (100 ms) in the timeline, and select Store joints in keyframe → Whole Body. Now at this keyframe, the NAO will move to the Init pose. The reason we stored this position at frame 10 is so the NAO moves to the keyframe slowly, from whatever position it was initially in. If we had stored the keyframe at frame 1 (10 ms), the NAO would jerk violently to this position and fall over.

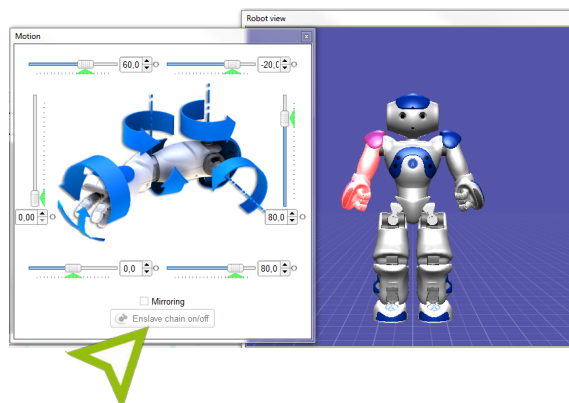


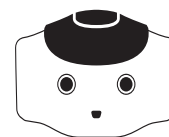
06/

Click on a frame about 10 frames after your first keyframe. Here we will make the robot lift its right arm with its palm down, the first move of the Macarena.

07/

Click the upper part of the NAO's right arm on the 3D view. The dialog below will appear. Click "Enslave Chain On/Off" to disable stiffness in the arm. The icon should turn green.



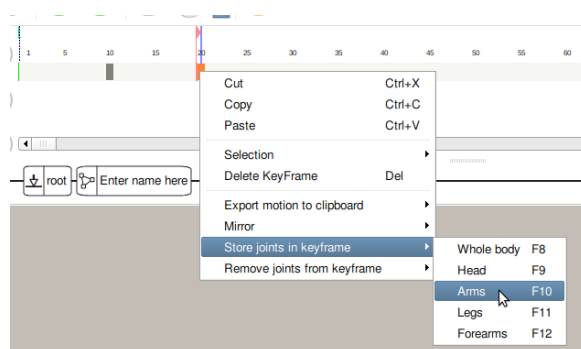


08/

You can now move the NAO's arm freely. Raise the right arm so it is pointing straight ahead with the palm down. Then, click the "Enslave chain on/off" button again to turn the stiffness back on (the icon will turn back to red). The arm will remain raised.

09/

Now right click on the frame you are editing on the timeline (around frame 20). Select Store joints in keyframe → Arms. The arm positions will be stored in that keyframe.



10/

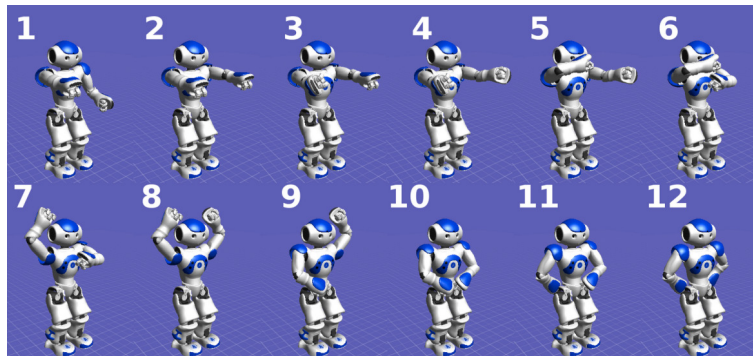
Next, let's check that the keyframes are working correctly. While holding onto the robot, click on the first keyframe. The robot should lower its arm. Then click on the second keyframe. The robot should raise its arm again. Now try clicking the small arrow to the left of the timeline. Both keyframes should play in succession. This is how the keyframes will play with the proper timing when you are finished.



11/

Now that you've learned how to make keyframes, complete the rest of the keyframes for the Macarena hand motions (do not do the turning or the hip shake yet). The steps are:

- 01/ Raise right arm, palm down
- 02/ Raise left arm, palm down
- 03/ Put right palm up
- 04/ Put left palm up
- 05/ Cross right arm
- 06/ Cross left arm
- 07/ Put right arm on head
- 08/ Put left arm on head
- 09/ Put right arm on hip
- 10/ Put left arm on hip
- 11/ Put right arm on rear
- 12/ Put left arm on rear



A few tips:

- a. Save your work often.
- b. Be sure to hold onto the NAO when turning stiffness on or off.
- c. If the NAO says that its motors are hot, turn stiffness off for a few minutes to let it rest.
- d. Be sure that the NAO does not hit itself with its hands. You may need to add intermediate keyframes to prevent this, especially for moving the hands from the hips to the rear in keyframes 11 and 12.

12.

[Optional] Once the NAO can do the Macarena, try to make it dance faster.

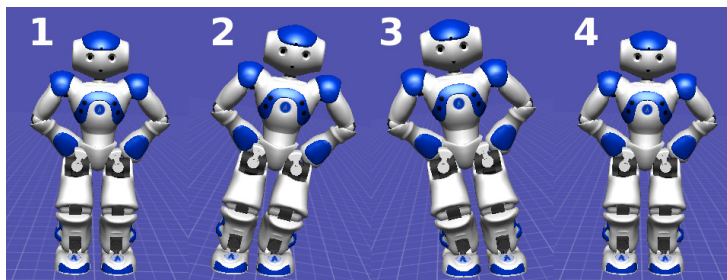
INTERMEDIATE TASK

DO THE MACARENA

WE'VE COMPLETED THE HAND MOTIONS FOR THE MACARENA. NOW WE WILL ADD THE HIP SHAKE AND TURNS. THE TURNS WILL INVOLVE LEARNING HOW TO USE LOOPS IN CHOREGRAPHE.

01/

Add the hip shake in the same way you added the arm motions: with additional keyframes. This time you will need to move the NAO's legs. The figure below shows the NAO performing the Macarena hip shake.



Be sure to hold the NAO tightly around its waist when disabling the leg stiffness. Do not hold the NAO by its limbs or head. Make the NAO shift its hips left, right, and then back to the center.

When adjusting the position of the legs, be careful that the NAO remains balanced. As a humanoid robot with two legs, it is easy for the NAO to fall over to its side if the legs are not positioned correctly. Try out the position on yourself before adjusting the angles on the NAO. It is recommended to remove stiffness on one leg at a time, so that you can rest some weight on one leg while adjusting the other. Check out the tutorial on balancing the NAO above.

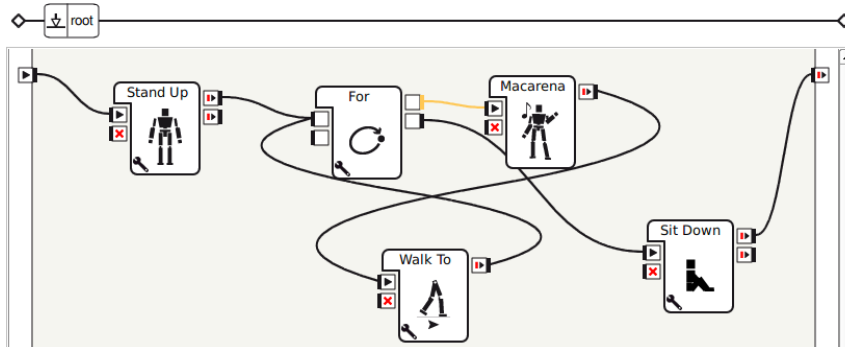
02/

Play the entire sequence to make sure it is **stable** and correct. If certain actions cause the robot to fall, e.g., the hip motion, slowing it down may help; otherwise, the keyframe itself may have to be modified.

You have completed the keyframe motion segment of the Macarena. Next, we will add the quarter rotations.

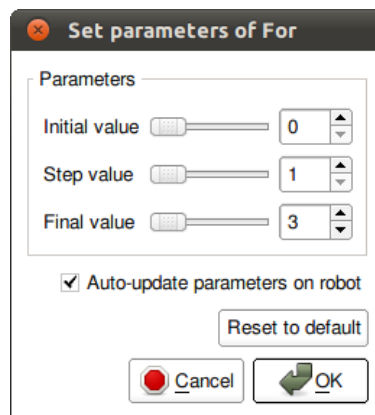
03/

Return to the workspace. In addition to your Macarena box, add a Stand Up, Sit Down, Walk To, and For box (found under Flow Control). Connect the boxes as shown below.



04/

The For box repeats a sequence of boxes a set number of times. Click the wrench on the For box to set the number of times to repeat. Change the “Final Value” to 3. The loop maintains a counter, which begins at the initial value (0) and is incremented by the Step Value (1) after each time the boxes in the loop are executed. The loop stops when the counter goes above the final value (3). How many times will the loop execute?



05/

Configure the Walk To box to do a quarter turn.

06/

Run the program by clicking the Play button. The NAO should do the Macarena.

ADVANCED TASK

NODDING OFF

NEXT, WE'LL LEARN HOW TO CONTROL JOINT ANGLES USING PYTHON. WE'LL MAKE TWO BOXES, ONE TO MAKE THE NAO NOD YES AND ONE TO MAKE IT NOD NO. EACH JOINT OF THE ROBOT CAN BE CONTROLLED. FOR THIS EXERCISE, WE WILL CONTROL THE NECK JOINTS - THE JOINTS ARE CALLED HEADYAW (TURNS THE HEAD LEFT AND RIGHT) AND HEADPITCH (TURNS THE HEAD UP AND DOWN).

01/

Create a new box to nod yes, and open the Python source file by double-clicking the box.

02/

Copy the following source code into the `onInput_onStart` method.

```
def onInput_onStart(self):
    motionProxy = ALProxy("ALMotion")
    names = ['HeadYaw', 'HeadPitch']
    times = [[0.5], [0.5]]
    motionProxy.angleInterpolation(names, [0.0, 0.0], times, True)

    for i in range(3):
        motionProxy.angleInterpolation(names, [0.0, 1.0], times, True)
        motionProxy.angleInterpolation(names, [0.0, -1.0], times, True)

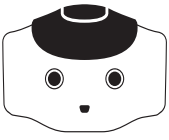
    motionProxy.angleInterpolation(names, [0.0, 0.0], times, True)

    self.onStopped()
```

The `angleInterpolation` method interpolates the NAO's joint angles. This function takes four parameters: the names of the joints we are controlling ("HeadYaw" and "HeadPitch"), the final angles (in radians) to set these two joints to, the time the joints should arrive in the final position (0.5 seconds later for both), and whether the joint positions are absolute or relative (absolute is True).

The parameter that varies is the list of angles we set the joints to. First the head faces straight ahead, then upwards, then downwards, and finally straight ahead again. The HeadPitch joint controls the up and down motion of the robot.

The `for` loop in Python has the same function as the Choregraphe box. It repeats the code in the loop, incrementing the counter variable, `i`, from 0 to 1 to 2 for a total of three iterations through the loop. This makes the NAO nod three times.



03/

Make a new box, and modify the code to have the NAO shake its head left and right. You will want to change the HeadYaw joint instead of the HeadPitch. So, when setting the angles in the for loop, they should be [1.0, 0.0] and [-1.0, 0.0].

04/

```
def onInput_onStart(self):
    motionProxy = ALProxy("ALMotion")
    names = ['HeadYaw', 'HeadPitch']
    times = [[0.5], [0.5]]
    motionProxy.angleInterpolation(names, [0.0, 0.0], times, True)

    for i in range(3):
        motionProxy.angleInterpolation(names, [1.0, 0.0], times, True)
        motionProxy.angleInterpolation(names, [-1.0, 0.0], times, True)

    motionProxy.angleInterpolation(names, [0.0, 0.0], times, True)
    self.onStopped()
```

(Optional) Change the speed of the nodding by adjusting `times`.

ADDITIONAL EXERCISES

- 01/** Make the NAO wave its hand and say hello when someone says hello.
Hint: Use a Speech Recognition box to detect “hello”, and attach the output to a keyframe motion and a Say box.
- 02/** Implement your own dance, and compete in a class dance-off.
- 03/** Use Python to control both head joints at once, and make the NAO’s head move diagonally, e.g., from the bottom-left to top-right.
- 04/** Implement the NAO Crystal Ball - Ask NAO a question, and have it randomly respond by saying yes or no and shaking the head in the appropriate way.
Hint: Use a Speech Recognition box, attach both outputs to a Random Int box (found in the Math section) with range 0 to 1, and attach that to a Switch box.

MODULE QUESTIONS

BASIC

- 01/** What is a keyframe?
- 02/** What does the NAO do in between keyframes?
- 03/** Where is the NAO's center of gravity?
- 04/** Where must the center of gravity be for the NAO to remain balanced?
- 05/** When creating a box in Choregraphe, how do you make it a box where keyframes can be edited?
- 06/** Why is it not suggested to set a keyframe in the first frame of the timeline?

INTERMEDIATE

- 07/** What types of motions will cause the NAO to fall?
- 08/** Explain the behavior of a For loop box in Choregraphe.
- 09/** A For loop box has an initial value of 2, a final value of 10, and a step value of 2. How many times does the loop execute?

ADVANCED

- 10/** Which joint is the HeadYaw joint and which is the HeadPitch?
- 11/** What units are angles passed to the `angleInterpolation` function in?
- 12/** How many radians is 45 degrees?
- 13/** How many degrees is $\pi/3$ radians?

5 SENSE AND ACT

LEARNING

In this module, students will learn:

- **What sensors there are on the NAO**
- **How sensors on the NAO relate to human senses**
- **What actuators are**
- **How the NAO knows its body pose**
- **What an LED is**
- **Where LEDs are located on the NAO**
- **How to control the LEDs of the NAO**
- **How to read sensor values and bumper presses of the NAO**
- **What finite state machines are**
- **How to design a robot behavior using a finite state machine**
- **How to implement a finite state machine**
- **To understand binary numbers**
- **To understand boolean values and bitwise-or**
- **To store and retrieve state**

* RST.11-12.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.

* RST.11-12.8. Evaluate the hypotheses, data, analysis, and conclusions in a science or technical text, verifying the data when possible and corroborating or challenging conclusions with other sources of information.

* RST.11-12.9. Synthesize information from a range of sources (e.g., texts, experiments, simulations) into a coherent understanding of a process, phenomenon, or concept, resolving conflicting information when possible.

* RST.11-12.10. By the end of grade 12, read and comprehend science/technical texts in the grades 11–12 text complexity band independently and proficiently.

* Reference COMMON CORE Stem standards

CONTENTS

- 01/** Senses of the NAO
- 02/** Basic Task: Light Up
- 03/** Finite State Machines
- 04/** Intermediate Task: Switching States
- 05/** Sensors and Actuators
- 06/** Advanced Task: A Bright Idea
- 07/** Advanced Task: Mirror, Mirror on the Wall
- 08/** Additional Exercises
- 09/** Module Questions

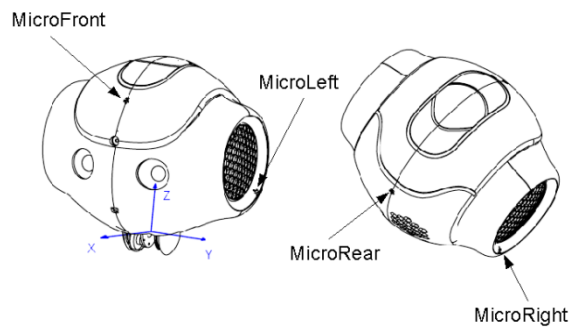
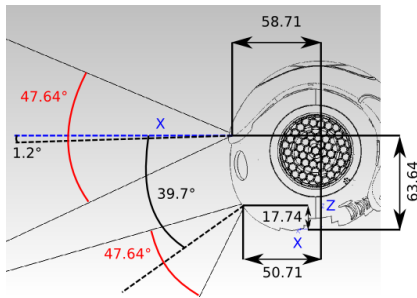
SENSES OF THE NAO

Humans have five senses - hearing, sight, touch, smell and taste. In addition, there are other lesser known human senses. They include the sense of balance, the sense of temperature, and kinesthetic sense. Kinesthetic sense is the ability to know where different parts of your body are without relying on other senses. It is this kinesthetic sense that enables you to close your eyes and touch parts of your body.

There are robots that are capable of performing similar functions to all the senses listed above. The physical devices in robots that sense the environment are called sensors.

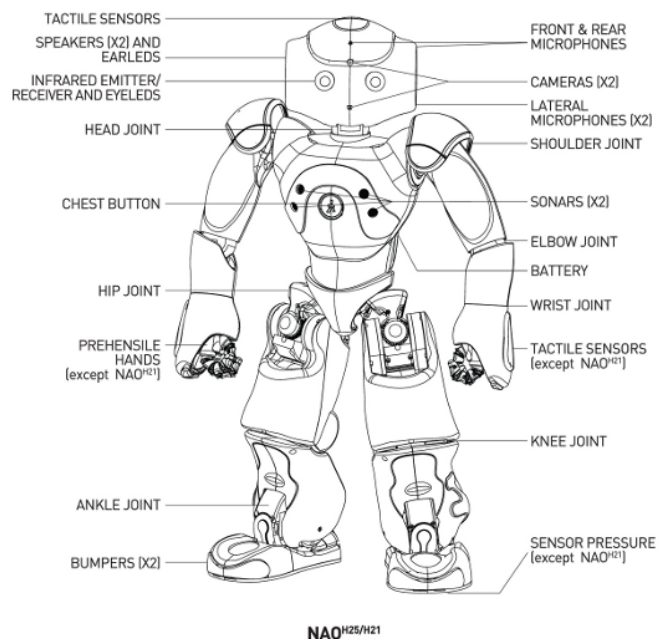
The NAO has two cameras in its head. The cameras are vision sensors, and provide camera images for the NAO's computer to process. This is different from the human "sense of sight" as sight would include processing the image to understand what the robot sees.

The NAO also has 4 microphones built into its head that it uses to listen to sounds around it. In



the earlier exercises, we used the microphones to allow the NAO to perform speech recognition.

The NAO does not have sensors that allow it to smell or taste things. However, it does have touch sensors on its head and body. It has a tactile sensor on its head that is split into three parts. Each part of the head tactile sensor triggers when touched. It also has a chest button and two foot bumpers, which require an actual press to trigger.

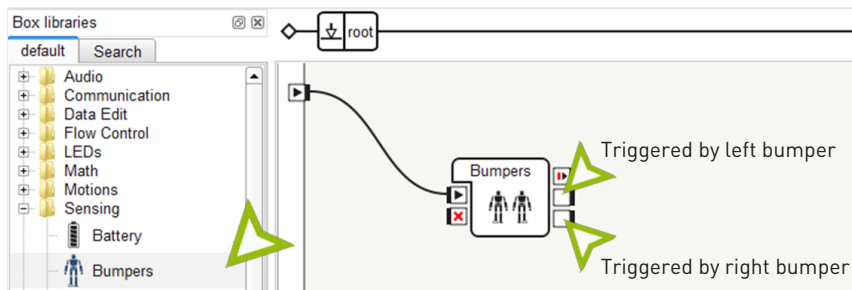


BASIC TASK

LIGHT UP

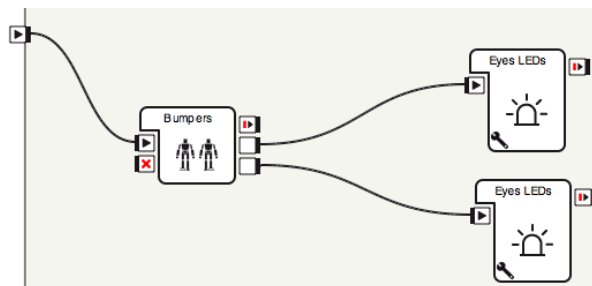
THE NAO HAS FOOT BUMPERS WHICH CAN BE PRESSED, AND VARIOUS LEDS THAT LIGHT UP. IN THIS LESSON, WE WILL MAKE THE NAO'S EYE LEDS CHANGE COLOR WHEN A FOOT BUMPER IS PRESSED.

01/

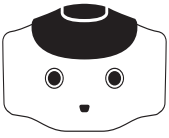


Drag out a Bumpers box from the sensors category. The Bumpers box has two outputs that trigger when the foot bumpers are pressed. The top output triggers on the left foot bumper, and the bottom output triggers on the right foot bumper.

02/

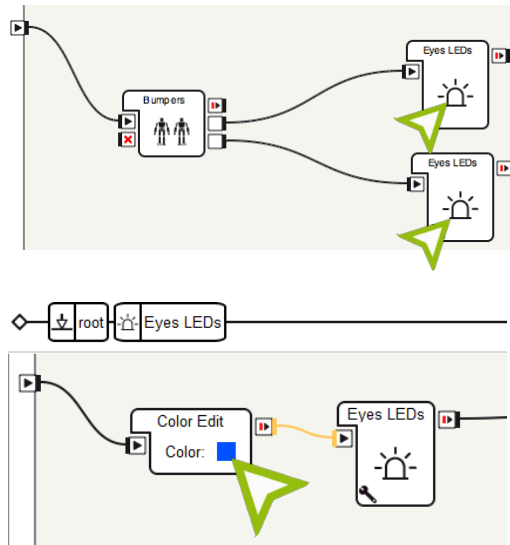


Drag two Eyes LEDs boxes (found in the LEDs category). Connect them as shown below. When the left bumper is triggered, the upper box will be executed. When the right bumper is triggered, the lower box will be executed.



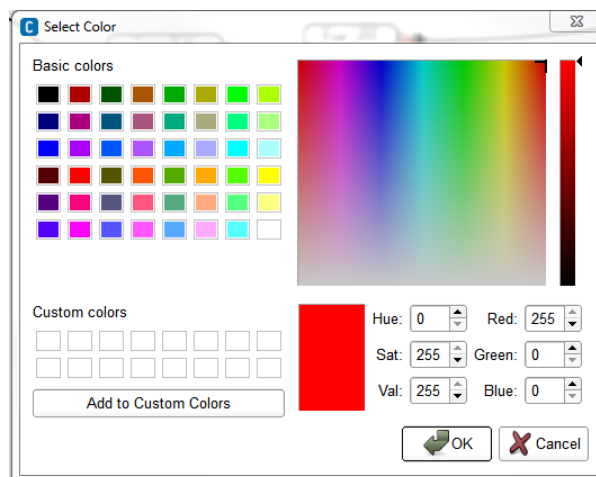
03/

Now we will set the eye colors. Double click on one of the Eyes LEDs boxes, and then click on the color.



04/

Choose any color from the color chooser that appears, but use a different color for each box. You can click anywhere on the color area and the LED will be set to that color.



05/

Play the behavior. Pressing the foot bumpers should switch between the eye colors you selected.

FINITE STATE MACHINES

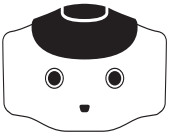
A finite state machine, or FSM, is an abstraction commonly used in computer science and robotics. A finite state machine includes *states* (a finite number of them) and *transitions*. We will use a running example to explain these concepts.

Suppose that an exam is coming up, and you are at home preparing for it over the weekend. Your behaviors (what you do during the weekend) can be described with a finite state machine. Your state can be described using two *features*: how prepared you are for the exam (prepared or unprepared), and how much energy you have (rested or tired). The goal is to ultimately be both prepared and rested at the end of the weekend.

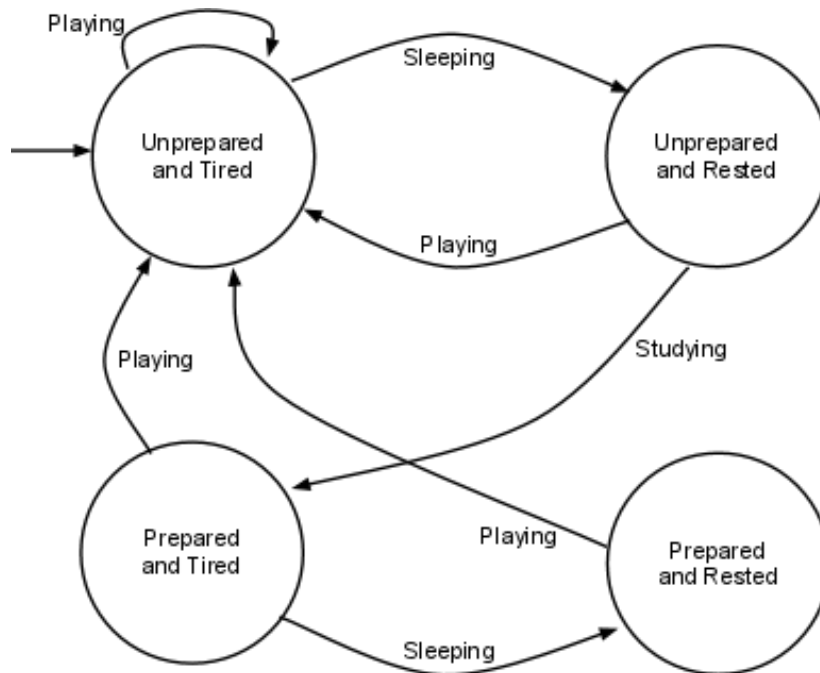
An FSM state must completely describe the situation without any external information. So, one state would be (*prepared and rested*). Notice that we combined both features into a single state. There will be four states in total - (*prepared and rested*), (*prepared and tired*), (*unprepared and rested*) and (*unprepared and tired*). Each of the four states contains all the information about the situation. Within a FSM, only one state is active at a time.

So in our example, say that we start off in the (*unprepared and tired*) state, since a week of classes just ended (so we're tired), and we haven't had time to study for the exam yet. We want to end up in (*prepared and rested*). To go from state to state, we have to define the transitions. Transitions move us from one state to another, and can be triggered through actions or events. Intuitively, actions are things that are performed by choice, and events are things that occur. We will use actions in the example below, and discuss events at the end of this section.

Some actions we can take in our example are *studying*, *playing*, and *sleeping*. If we're tired, then sleeping should make us rested. Thus, the transitions from (*prepared and tired*) to (*prepared and rested*), and from (*unprepared and tired*) to (*unprepared and rested*), are triggered by *sleeping*. Similarly, the action *studying* will make us prepared for the exam but will tire us out. However, we can only perform this action if we are rested. As such, there is a transition from (*unprepared and rested*) to (*prepared and tired*) that is triggered by *studying*. The last action is *playing*, which can be performed at any state. However, playing causes us to both become tired and forget what we've learned. So, there are transitions from all the other states to (*unprepared and tired*) that are triggered by *playing*.



A FSM can be illustrated with a diagram, as shown below. Circles indicate states, and arrows indicate transitions. The straight arrow that points to (unprepared and tired) indicates that it is the initial state, or the state that we start out in.



Besides triggering transitions with actions, events can also be used. Events are things that happen, that are typically caused by something external to the FSM. For example, an event could be the teacher reducing the scope of the exam. This event might transition us from (*unprepared and tired*) to (*prepared and tired*), since we already know the areas covered in the revised exam even without studying.

INTERMEDIATE TASK

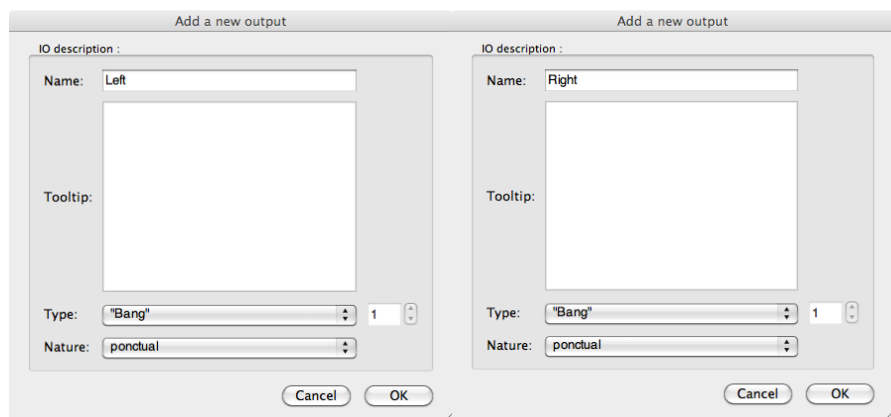
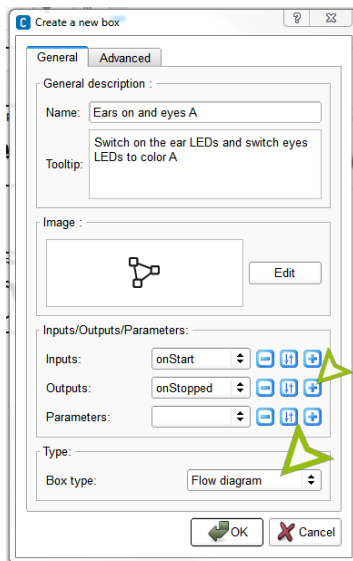
SWITCHING STATES

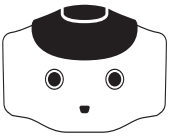
IN THIS TASK, WE WILL MAKE THE ROBOT TURN ITS EAR LEDs ON AND OFF BY PRESSING ONEFOOT BUMPER, AND TOGGLE ITS EYE COLORS BY PRESSING THE OTHER FOOT BUMPER. WE WILL IMPLEMENT THIS USING A FINITE STATE MACHINE.

Similar to our finite state machine example in the section above, we have two features in the states. The features are: whether the ear LEDs are on (on and off), and what color the eye LEDs are set to (A or B). Thus, we have four states: (*ears off and eyes as color A*), (*ears off and eyes as color B*), (*ears on and eyes as color A*), and (*ears on and eyes as color B*). Each press of a foot bumper is an event that will trigger a transition to a different state.

01/

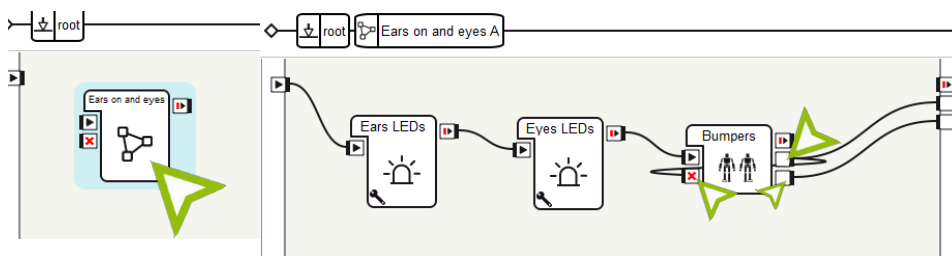
First, we'll create one of the states. Add a custom flow diagram box named BumperState, and add two outputs, "Left", and "Right" to the box.





02/

Double-click on the custom box, and a new flow diagram will show up. Add a Bumpers box, an Eyes LED box, and a Ears LED box, and connect them as shown below. Ensure that the outputs of the Bumpers box also link back to its X.

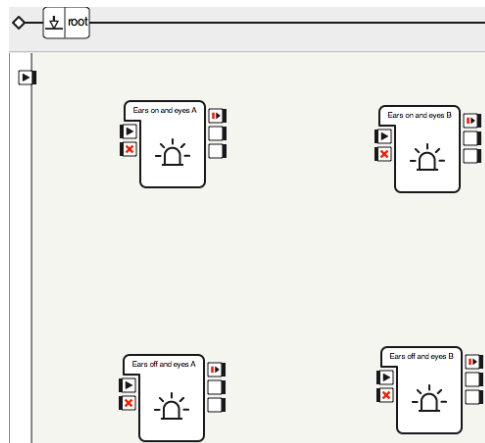


03/

Set the ears intensity to 100%, and set the eye color to a color of your choosing, which we will refer to as color A. The state (ears on and eyes as color A) is now defined with these three boxes. The two outputs, left and right, correspond to the events of the left foot bumper and right foot bumper being pressed.

04/

Click on root to go back to the main flow diagram, then copy and paste these three boxes to create four states, as shown below. Rename each box to correspond to the state it represents.





05/

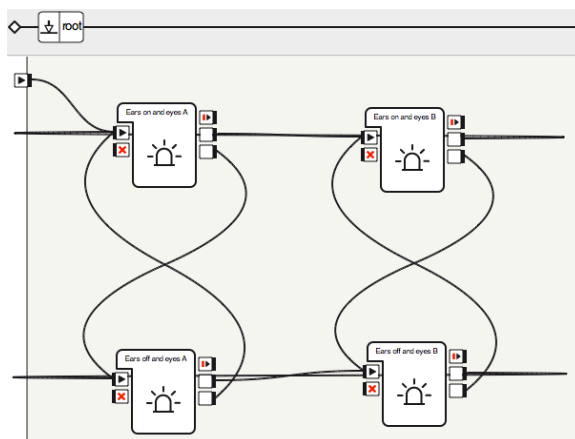
Double-click each of the three new states, and edit the Ears LEDs and Eyes LEDs boxes so that they match the state they are in. In the box “Ears off and eyes B”, the ear LEDs should be switched to 0% and the eye LEDs should be set to color B, and so on for all of the states.

06/

You have created the four states. The states the boxes represent, clockwise from the top-left are: (*ears on and eyes as color A*), (*ears on and eyes as color B*), (*ears off and eyes as color B*) and (*ears off and eyes as color A*).

07/

Now add the transitions between them, as shown below. The left foot bumper event should switch states such that the eyes change color, and the right foot bumper event should switch states so that the ear LEDs switch on and off. Also, connect the start event to the Ears on and eyes A box - this defines the initial state of our FSM.



Notice how the figure above corresponds to the diagram of a finite state machine in the earlier section.

08/

Run the behavior. Press the bumpers to ensure that all the states work as intended. Congratulations! You have implemented a finite state machine.

INTERMEDIATE TASK

SENSING WITH MONITOR

IN THIS LAB, WE WILL LEARN HOW TO OBSERVE THE RAW SENSOR VALUES OF THE ROBOT AND GRAPH THEM WITH MONITOR, A TOOL THAT COMES WITH CHOREGRAPHE.

01/

First, run Monitor. It should be in the same directory that you run Choregraphe from.

02/

Click on Memory. We wish to observe some of the sensors, and this information is stored in the NAO's memory.

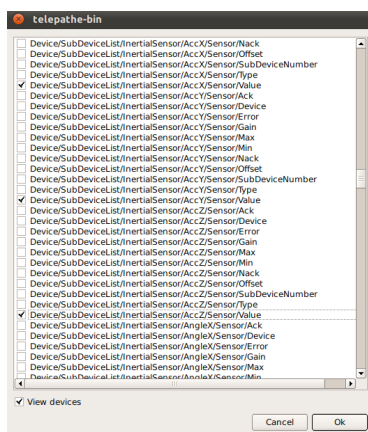
03/

Select your robot from the list, the same way you do in Choregraphe, and connect to it.

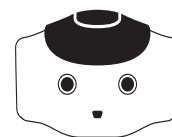
04/

Now Telepathe asks you which memory values you want to read. Select "New Configuration."

05/

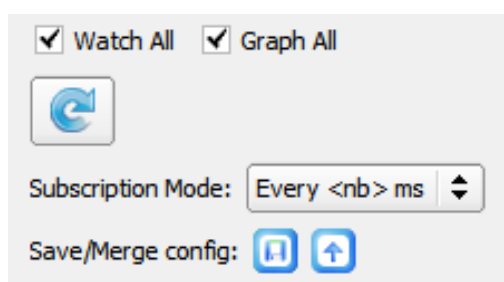


The dialog window below pops up. Check "View Devices" at the bottom, and scroll down until you find "Device/SubDeviceList/InertialSensor/AccX/Sensor/Value", and check this item. Also select ".../AccY/Sensor/Value" and ".../AccZ/Sensor/Value". These are the accelerometer (A sensor that detects acceleration and tilt) (see sensor section) readings along the x, y and z axes. Click OK.



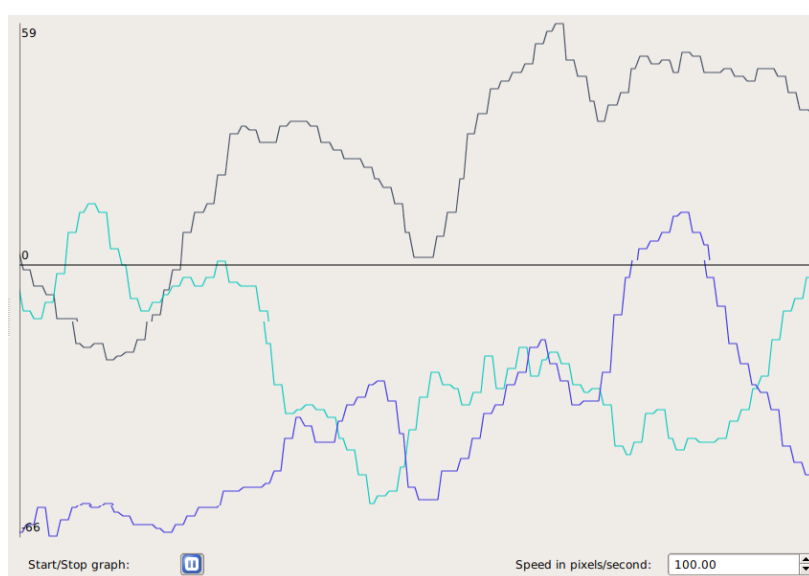
06/

In the bottom left corner of the new window, check the boxes next to “Watch All” and “Graph All”. This will make Telepathe observe and graph the variables we selected. Finally, change the Subscription Mode to “Every \leftarrow nb \rightarrow ms”. The default value in the dialog that pops up is fine. This is how often we refresh the sensor values in the graph.



07/

Now the x, y, and z axis accelerometer readings will appear on the graph. Try turning the NAO in all directions. Determine what direction each of the three axes measure acceleration along.



08/

(Optional) Examine some of the other sensor readings. Some good ones to try are any of the Device/SubDeviceList/JOINT/Position/Sensor/Value, which returns the measured joint angle, or Device/SubDeviceList/InertialSensor/(AngleX, AngleY, AngleZ, GyrX, or GyrY)/Sensor/Value. Try to figure out what these measure on your own.

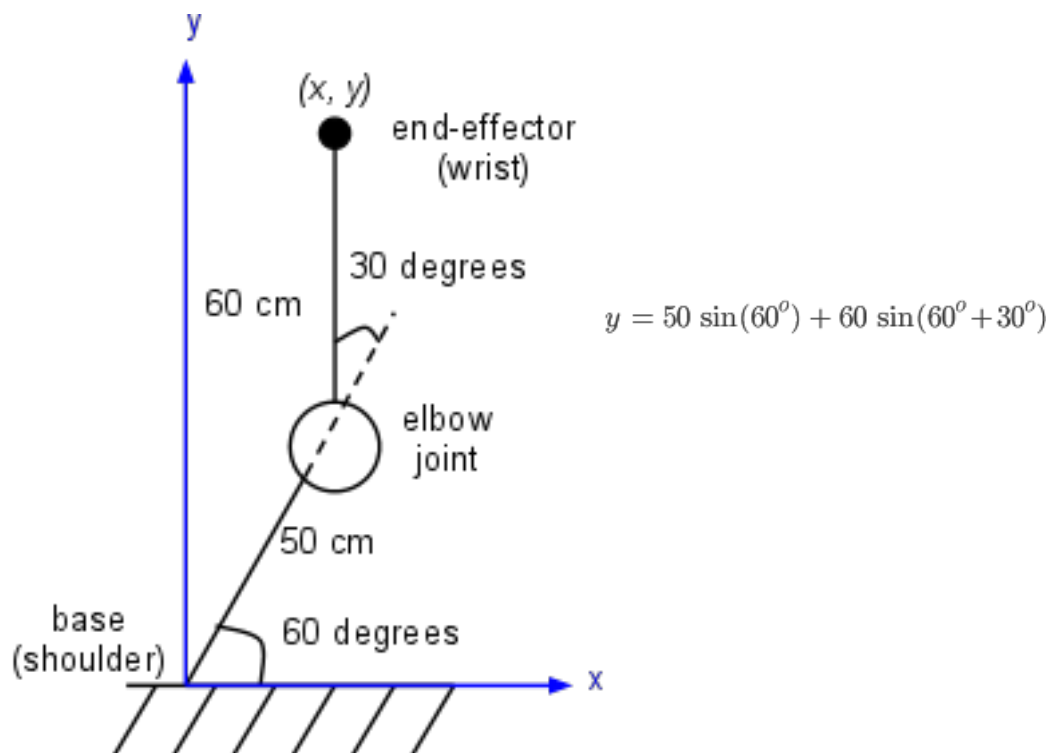
SENSORS AND ACTUATORS

Besides the sensors described earlier, the NAO also has an internal gyroscope and accelerometer inside its torso. The internal gyroscope and accelerometer function like the inner ear, which provides a sense of balance for humans. The gyroscope measures angular velocity (how fast the robot is turning). The accelerometer measures acceleration (which way gravity is pointing). Together, the gyroscope and accelerometer can tell the NAO if it is upright, lying on its back, or lying on its front. Additionally, the two sensors let the NAO know if it is falling down, so it can brace itself for a fall with its arms.

Actuators on a robot refer to its joint motors. The NAO has 21 different motors that can be controlled separately. There are two motors on its head/neck, two for each shoulder, two for each elbow, five for the hips, one for each knee, and two for each ankle. In Module 4, we moved many of these motors to make the NAO dance.

Each motor on the NAO is coupled with a sensor, called an *encoder*. This sensor measures how far each motor has turned. This is known as the motor's angle of rotation. For example, the sensor on the elbow joint is able to tell if the arm is straight or bent at an angle.

Using the angles of rotation of its joints, the NAO is aware of the pose of its entire body. For example, the NAO can calculate how far the hand is from the head. It does so using a *kinematic chain*, which essentially uses trigonometry to calculate relative positions of joints. In the figure below, we show a two-link arm with an elbow joint. Using the length of the arms and the angle of the elbow, we can calculate the position of the *end-effector* (the wrist) relative to the base (the shoulder).



ADVANCED TASK

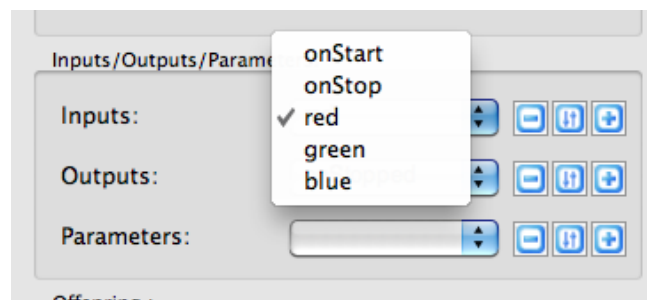
A BRIGHT IDEA

BEFORE, WE USED THE FOOT BUMPERS TO TOGGLE BETWEEN FOUR STATES. NOW WE'LL USE THE THREE BUTTONS ON THE NAO'S HEAD TO TOGGLE BETWEEN EIGHT STATES.

Things are becoming unwieldy using Choregraphe boxes, so we'll switch to using Python. Each of the three head buttons will control one of the three color components: red, green and blue. By combining these colors, we can make others. For example, red and blue together will make purple, and red and green make yellow.

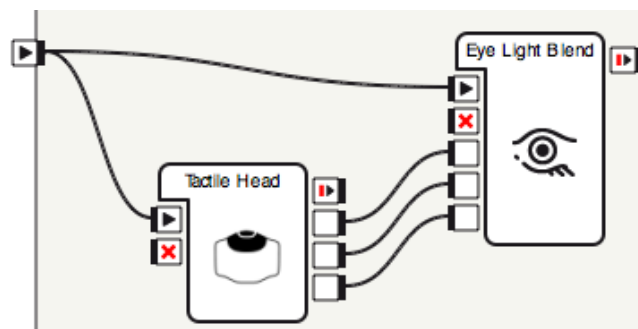
01/

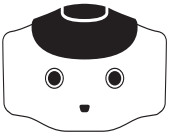
First, add a new box to blend the lights. Add three "bang" inputs, named "red", "green", and "blue".



02/

Add a Tactile Head box (in the Sensors category), and connect the boxes as shown below. The three outputs of the Tactile Head box trigger when the front, middle and rear buttons on the head are touched.





03/

Double-click on your custom box to edit the source code. Add the code shown below.

```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.leds = ALProxy("ALLeds")
        self.blue = True
        self.green = False
        self.red = False

    def onLoad(self):
        self.done = False

    def onUnload(self):
        self.done = True

    def onInput_onStart(self):
        while not self.done:
            color = 0
            if self.blue:
                color = color | 0xFF
            if self.green:
                color = color | 0xFF00
            if self.red:
                color = color | 0xFF0000
            self.leds.fadeRGB("FaceLeds", color, 0.1)

    def onInput_onStop(self):
        self.onUnload() #- it is recommended to call onUnload of this box in a onStop
        method, as the code written in onUnload is used to stop the box as well

    def onInput_red(self):
        self.red = not self.red

    def onInput_green(self):
        self.green = not self.green

    def onInput_blue(self):
        self.blue = not self.blue
```

This script maintains a state consisting of three *boolean* variables: red, green, and blue. Boolean variables can be either `True` or `False`. When a button is pressed, we toggle the corresponding variable using the `not` operator, which performs negation. So, `not True` is `False`, and `not False` is `True`.

We use the state variables to set the LED colors in the `onInput_onStart` method, which continually loops and sets the eye colors. The colors are set using binary 24-bit RGB values. The last eight bits are the blue component, the preceding eight bits are the green component, and the eight bits before that are the red component. So, written in hexadecimal, `0xFF0000` is red, `0x00FF00` is green, and `0x0000FF` is blue.

The vertical bar (`|`) is the bitwise-or operator. This takes two binary numbers, and for each bit, if that bit is set in either number it is set in the output. For example, `100110 | 010100 = 110110`. So the sequence of `if` statements sets the corresponding color component if the `red`, `green` and `blue` variables have been set to `true`.

04/

Now run the behavior. Experiment with all eight different color combinations.

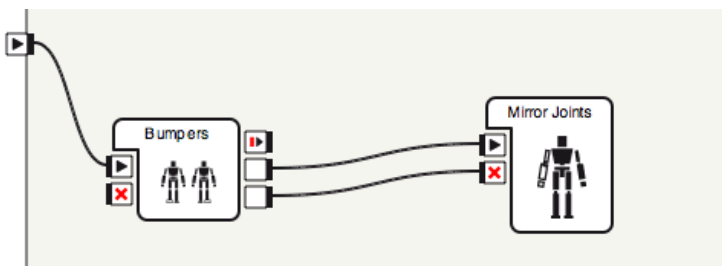
05/

(Optional) Draw the finite state machine to describe the behavior we just created. How many states are there? What are the transitions between the states?

ADVANCED TASK

MIRROR, MIRROR ON THE WALL

ANOTHER SENSOR OF THE NAO, OFTEN OVERLOOKED, IS ITS ENCODERS. THESE SENSORS MEASURE THE ANGLES OF ALL OF THE NAO'S JOINTS. IN THIS TASK, WE WILL DISABLE STIFFNESS ON ONE OF THE NAO'S ARMS SO THAT IT CAN BE MOVED FREELY. THEN, WE WILL USE THE ENCODERS TO READ THE POSITIONS OF THE ARM JOINTS, AND MIRROR THESE SAME POSITIONS ON THE NAO'S OTHER ARM. IN THIS WAY, BY MOVING ONE ARM, THE OTHER ARM WILL FOLLOW.



01/

Create a Bumpers box, and connect it to a custom Mirror Joints box as shown below. Hitting the left bumper will start the behavior and hitting the right bumper will stop it.

```
import time

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.motion = ALProxy("ALMotion")
        self.done = False
        self.stiff = ['RShoulderPitch', 'RShoulderRoll', 'RElbowYaw', 'RElbowRoll']
        self.unstiff = ['LShoulderPitch', 'LShoulderRoll', 'LElbowYaw', 'LElbowRoll']
        self.origStiffness = self.motion.getStiffnesses(self.unstiff)

    def onLoad(self):
        pass

    def onUnload(self):
        self.done = True
        self.motion.setStiffnesses(self.unstiff, self.origStiffness)

    def onInput_onStart(self):
        self.motion.setStiffnesses(self.unstiff, 0.0)
        self.done = False
        while not self.done:
            vals = self.motion.getAngles(self.unstiff, True)
            vals[1] = -vals[1]
            vals[2] = -vals[2]
            vals[3] = -vals[3]
            self.motion.setAngles(self.stiff, vals, 0.5)
            time.sleep(0.1)

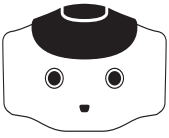
    def onInput_onStop(self):
        self.onUnload() #~ it is recommended to call onUnload of this box in a onStop
        # method, as the code written in onUnload is used to stop the box as well
```

02/

Enter the Python code shown below into your custom box.

The variable `stiff` refers to the joints on the arm which will be doing the mirroring (the right arm). The variable `unstiff` refers to the joints on the arm which will not be stiffened (the left arm). The joints are relaxed when the box begins running, and returned to their initial stiffness when the X is pressed and the box finishes.

The `onInput_onStart` method is where the action happens. We have a loop that continuously measures the



joint angles in the left arm and sets those same angles in the right arm. The `getAngles` method gets the angles in the arm, and the `setAngles` method sets the angles in the other arm.

All of the joint angles except the shoulder pitch are negated because they are mirrored on the opposite side of the body.

03/

Run the behavior. Press the foot bumper, and move the robot's arm around. Check that the other arm mirrors the same position.

ADDITIONAL EXERCISES

01/ Implement the eye blending task (A Bright Idea) using a finite state machine composed of Choregraphe boxes. Compare the difficulty of adding additional states using boxes versus Python code.

02/ Building on the arm mirroring task, make it so tapping the head switches which arm is being mirrored.

Hint: this will require an additional input and calls to `setStiffnesses`. Also, the state of the behavior will now have two features: whether or not the behavior is active, and which arm is being mirrored.

03/ Start with the eye color blending task, and make it so you control the intensity of each color using the foot bumpers (so don't always use `0xFF` as the component). Make one bumper toggle between colors, and show the color component being modified on the feet LEDs. When the other foot bumper is pressed, increment the current color component by `0x10`. Be sure to handle wrap-around properly: no color component can exceed `0xFF`.

04/ Make the robot's head track his own hand. Disable stiffness on the arms, and get the hand and head positions with the `ALMotion` method `getPosition`. Using `setAngles`, make the NAO's head look in the direction of the vector from the head to the hands.
(Optional) Switch which hand the NAO looks at when the head tactile sensor is touched.

MODULE QUESTIONS

BASIC

- 01/** Where are the NAO's two cameras located?
- 02/** How many microphones does the NAO have and where are they?
- 03/** What touch sensors does the NAO possess?

INTERMEDIATE

- 04/** What are the two main components of a finite state machine?
- 05/** Explain when the two outputs of the Bumpers box are triggered.

ADVANCED

- 06/** Explain what the gyroscope and accelerometer measure.
- 07/** Which three body sections can the NAO's motors be grouped into?
- 08/** What do encoders measure?
- 09/** Explain at a high level how the NAO can know the position of its hand relative to its feet.
- 10/** What three color components determine the color emitted from an LED?
- 11/** What is a boolean variable?
- 12/** The finite state machine with buttons toggling two binary variables had four states, and the one with three binary variables had eight states. How many states does an FSM with four binary variables that you can toggle have? Five binary variables? Ten binary variables? N binary variables?
- 13/** True or False: Everything on a digital computer is stored in binary.
- 14/** How many different colors can be represented in the 24-bit RGB format?
- 15/** Write the RGB value for the color purple in hexadecimal.
- 16/** Compute 00110101 | 10011010 (binary), 0x7C | 0x5B (hexadecimal), and 118 | 201 (decimal).

6 DO THE ROBOT

LEARNING

In this module, students will learn:

- **What multi-tasking is on a computer/robot**
- **How an operating system performs multi-tasking with a single core**
- **How to use behavior layers in Choregraphe**
- **How to create motions that execute in parallel on the NAO**
- **How to create complex combinations of actions using behavior layers**
- **What odometry is**
- **How to measure rotational odometry on the NAO**
- **How to use measured odometry to update actions on the NAO**
- **How measured odometry does not match the actual motion perfectly**

- * RST.11-12.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.
- * RST.11-12.8. Evaluate the hypotheses, data, analysis, and conclusions in a science or technical text, verifying the data when possible and corroborating or challenging conclusions with other sources of information.
- * RST.11-12.9. Synthesize information from a range of sources (e.g., texts, experiments, simulations) into a coherent understanding of a process, phenomenon, or concept, resolving conflicting information when possible.
- * RST.11-12.10. By the end of grade 12, read and comprehend science/technical texts in the grades 11–12 text complexity band independently and proficiently.

* Reference COMMON CORE Stem standards

CONTENTS

- 01/** Multi-tasking
- 02/** Behavior Layers in Choregraphe
- 03/** Basic Task: Arms and Head
- 04/** Intermediate Task: Completing the Robot Dance
- Advanced Task: Walking in Circles
- 05/** Additional Exercises
- 06/** Module Questions
- 07/**

MULTI-TASKING

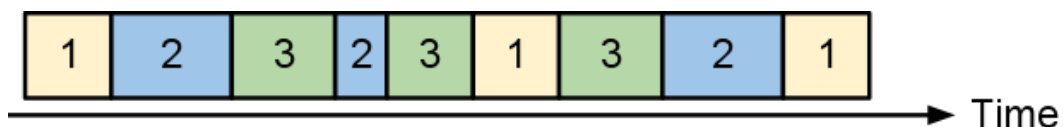
We do many things simultaneously in our day-to-day life. Chatting while walking, and eating while watching television are common examples. In computer science, this process is known as multi-tasking. A task is something that can be performed independently; multi-tasking is the process of performing many tasks in parallel.

Within each computer (and robot), there is a CPU that handles the processing of tasks. CPU stands for Central Processing Unit, and is where all the computational operations are processed in the computer.

How is multi-tasking done on a CPU? A CPU core can only execute one sequence of instructions at a time. One way to do multi-tasking is to use a multi-core CPU, so that multiple sequences of instructions can be executed concurrently.

But the NAO's CPU has only a single core. Although only one sequence of program instructions can be executed at a time, the *operating system* is able to maintain an illusion of true concurrency. The different programs are split into *threads*. Threads are sequences of code to be executed. The operating system switches which thread is currently executing on the CPU thousands of times per second. This switching is faster than a human's perception, and provides the illusion that all of the threads are executing simultaneously.

In the figure below, the operating system rapidly switches between three threads. As such, it appears as if all three threads are running in parallel. Notice that the amount of time each thread runs is not constant, and threads do not always run in the same order. The operating system decides when and how to switch between threads.

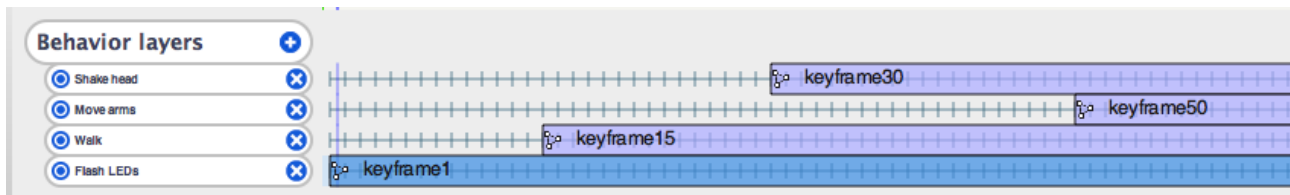


But we don't need to worry about the underlying behavior of threads or the operating system on the NAO, since Choregraphe takes care of this for us.

BEHAVIOR LAYERS IN CHOREGRAPHE

Behavior Layers are a feature in Choregraphe similar to threads. Each layer (or behavior) defines a separate thread that will run on the NAO. Some behaviors could be moving an arm, turning the head, saying something, or turning the LEDs on and off.

You can define when each layer begins playing. For example, you could have the NAO begin shaking its head after 0.3 seconds. Concurrently, another behavior layer could make the NAO move its arms after 0.5 seconds. Recall that in the Choregraphe timeline, each frame is 10 milliseconds, so 0.3 seconds corresponds to 30 frames. In the figure below, there are 4 behavior layers. One layer shakes the head after 0.3s; one moves the arms after 0.5s; one walks after 0.15s; the fourth flashes the LEDs immediately.



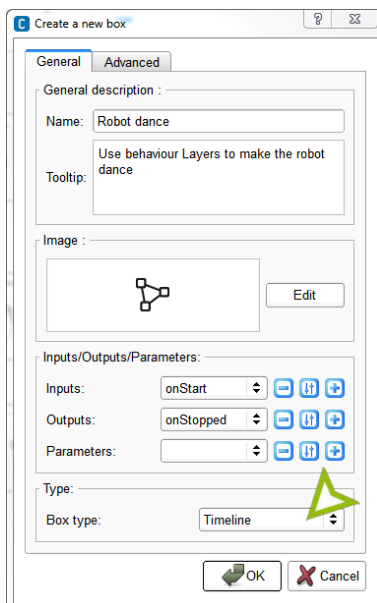
In the previous module, you designed keyframe motions for the Macarena dance. In the Macarena, the arms move first, followed by the legs, before repeating. In some other dances, the arms, legs and head of the robot have to move simultaneously. While it is still possible to create a single keyframe motion for such a dance, considering the repetitiveness of dance routines, behavior layers are a perfect fit for these motions!

In the following exercises, we will show how to create behavior layers in Choregraphe to have the NAO do the robot dance.

BASIC TASK

ARMS AND HEAD

IN THIS LESSON, WE'LL LEARN HOW TO USE BEHAVIOR LAYERS ON THE NAO. BEHAVIOR LAYERS ALLOW THE NAO TO PERFORM MULTIPLE TASKS ON THE NAO AT THE SAME TIME. THIS IS KNOWN AS MULTI-TASKING. LET'S IMPLEMENT THE ROBOT DANCE, AND MAKE THE NAO MOVE ITS ARMS UP AND DOWN AND SHAKE ITS HEAD AT THE SAME TIME.

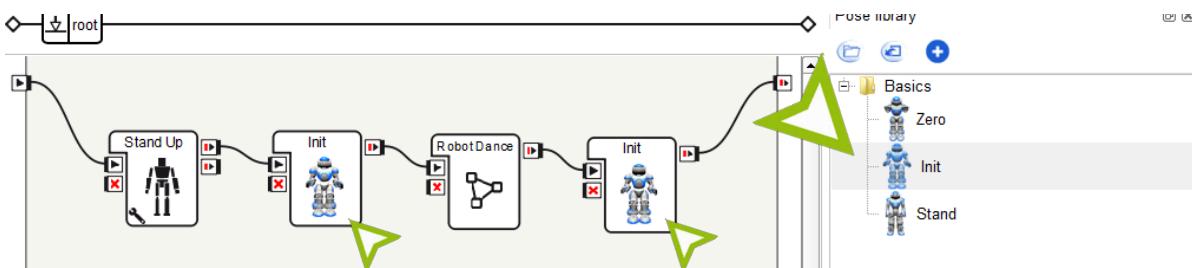


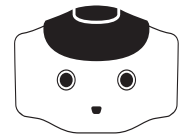
01/

In a new Choregraphe project, create a new “Timeline” box.

02/

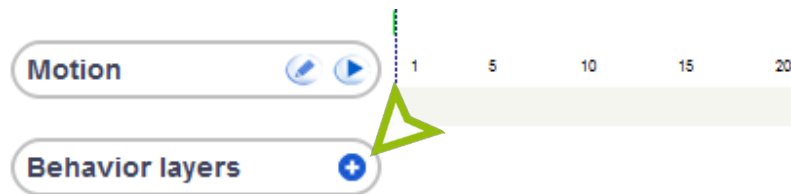
Connect a Stand Up box, an Init Pose box, your new box, and a second Init Pose box, as shown below. The robot will stand up, go to its initial position, dance, and return to the initial position. Recall that you can open the Pose Library by clicking View → Pose library in the top menu bar.





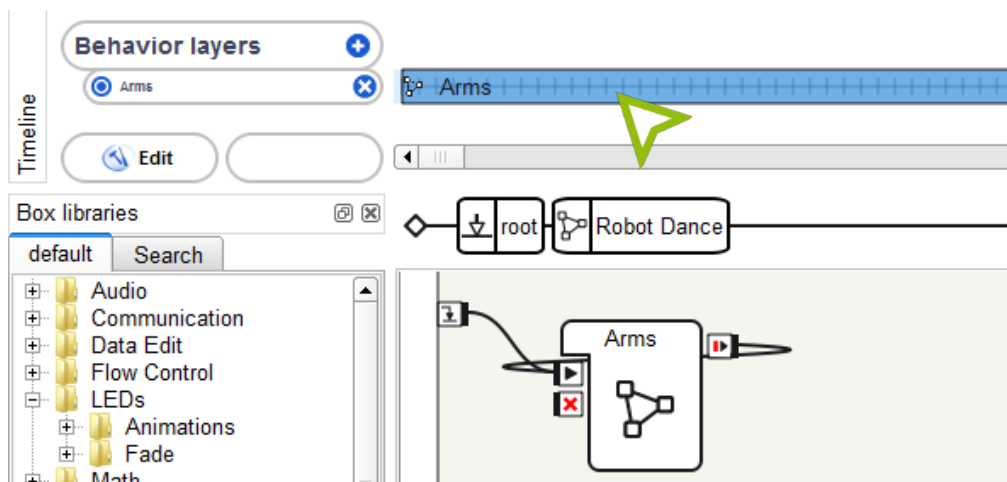
03/

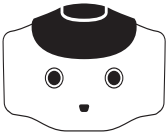
Double click on your new box to edit it. Beneath the timeline and the word “Motion”, you will see the words “Behavior layers” with a plus sign next to it. Click the plus sign to add a new behavior layer. We will make this layer move the arms up and down.



04/

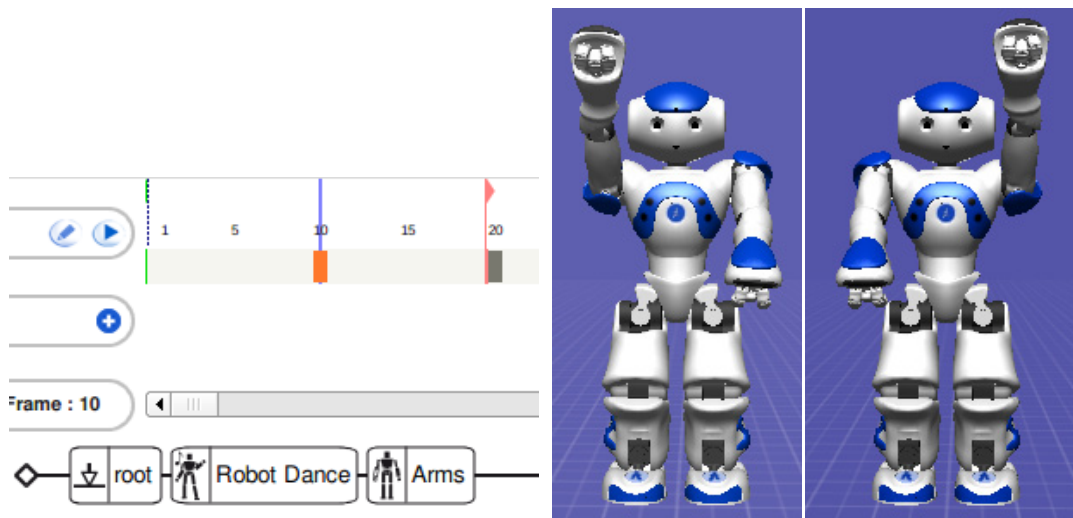
Select the new layer, and add a custom timeline box. Connect it to the initial play arrow, and also connect it to itself. This causes the box to repeat forever.





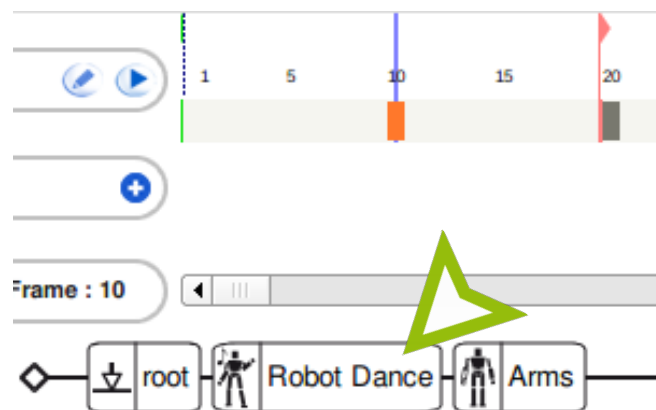
05/

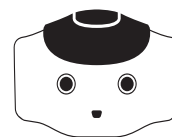
Now double click on the box to edit the timeline. Create two keyframes, as shown below. In the first keyframe, one arm is raised and the other is lowered. In the second keyframe, switch which arm is raised and lowered. Be sure to only save arm angles in these keyframes, since we will set the other body angles in different layers.



06/

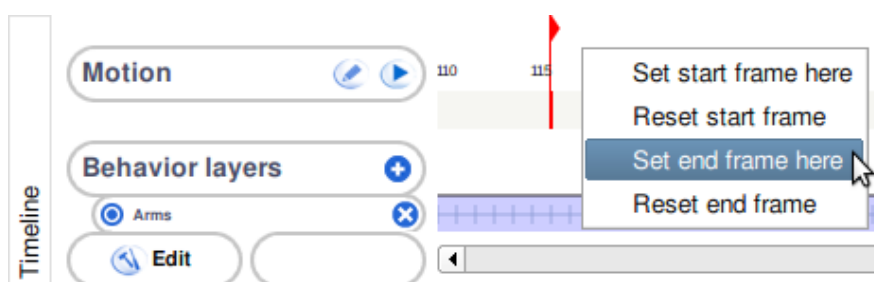
Currently, the arm motion will loop forever. Go back to editing the first custom box you made, the one you added the behavior layer to.





07/

Right click on the timeline at some point at least a hundred frames in the future, and select “Set end frame here.” A red flag will appear, and the entire behavior will stop at this point in time.



08/

Now we will add head shaking. Repeat the same procedure you used to add the arms. Add a new behavior layer, add an infinitely looping custom timeline box, and set two keyframes in that box: one with the head up and the other with the head down. Be sure to only store angles for the head in the keyframes.



09/

Play the behavior. The NAO should move its arms and shake its head, doing the robot dance. If the motions appear unstable, try reducing the frame rate.

10/

[Optional] Try experimenting with the frequencies of the different layers to find a combination that is visually pleasing.

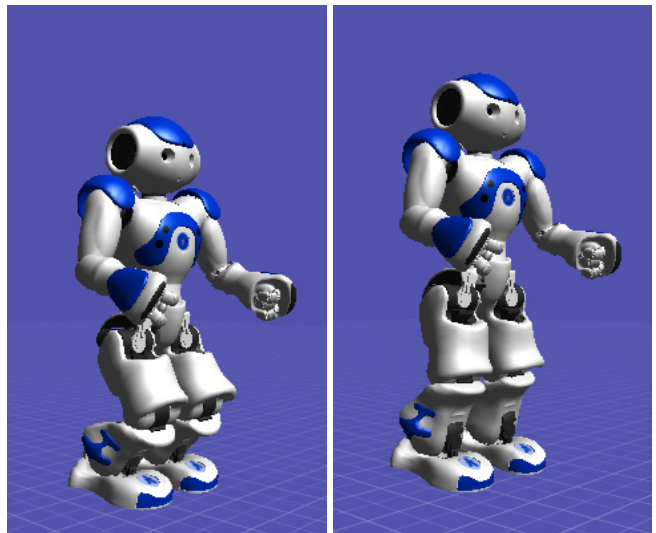
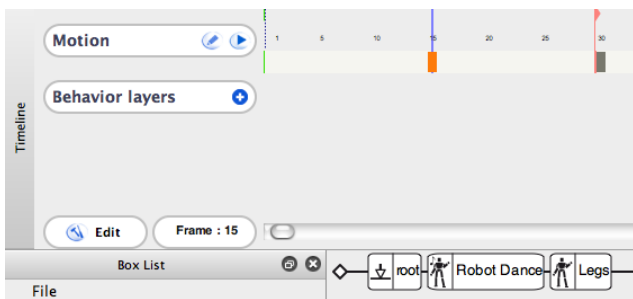
INTERMEDIATE TASK

COMPLETING THE ROBOT DANCE

IN THE PREVIOUS EXERCISE, WE FINISHED THE BASIC ROBOT DANCE. NOW WE WILL ADD IN LEG MOTIONS, FLASHING LIGHTS, AND SPEECH.

01/

First, add leg motions. As in the previous exercise, add a new behavior layer, Legs, with two keyframes. In one of the keyframes, make the robot squat, and in the second, make it stand up straighter. Be sure to only store angles for the legs in these keyframes.

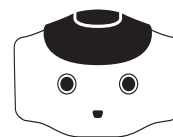


02/

Play the entire dance to make sure that the robot doesn't fall over. If the robot does fall, try slowing down the rate of the squat motion, or make the robot stand up less tall. Changing the frequency of motion of the arm and head may also help. Make a dance that is stable and visually appealing.

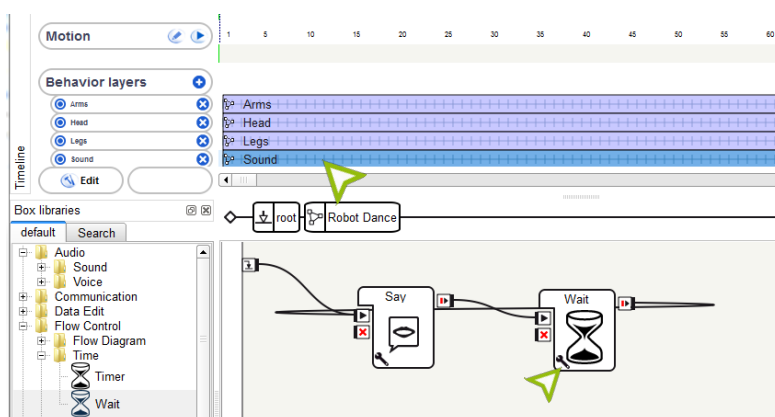
03/

The beauty of using behavior layers is that you can change the speed of one aspect of the dance, while keeping the rest constant. Try changing the frequency of motion of the arm so that it matches the squats, and make the head bob at a different frequency.



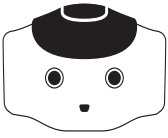
04/

We have finished with the dance motions. Add another layer for the NAO to speak. Place an infinitely repeating Say box in the layer, and have the robot say something of your choosing, such as "Nao." You may also choose to add a Wait box to this layer, if you wish. Click the wrench on the Wait box to set how long the delay should be (in seconds) between saying words.



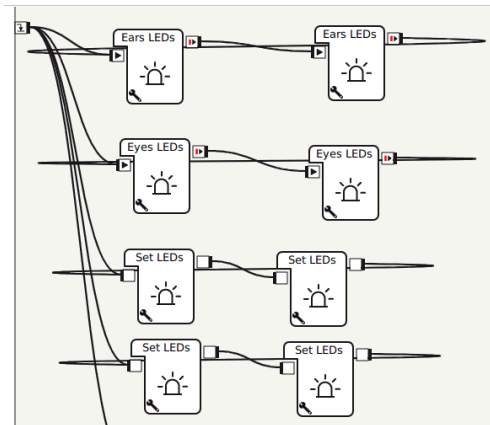
05/

(Optional) With the For box and Wait box, you can have the NAO repeat a behavior a number of times, and then pause for a certain amount of time. Change the head motion of the dance so that the NAO bobs its head up and down 3 times, and then stops for half a second before continuing again.



06/

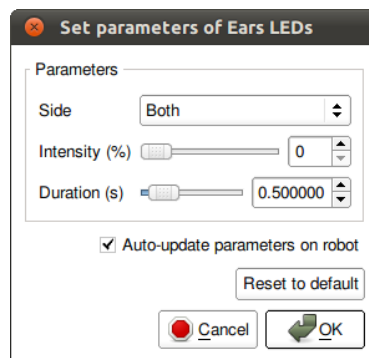
Now add the final behavior layer, which will control the LEDs. Place a bunch of loops with pairs of LED boxes, one to turn the LEDs on and one to turn them off. See the image below. Set the ear and eye LEDs, and use the Set LEDs boxes to set the feet, head and chest LEDs.



07/

Click on the wrenches to set each box's parameters. Choose an appropriate duration (in seconds), and make the intensity 100% in the first box and 0% in the second box. This will make the lights blink continuously. For the eye LEDs, you can double click on the boxes to select colors.

08/



(Optional) Add another layer to play music of your choosing.

09/

Now play the entire dance. It should include arm, head and leg motions, along with speech and flashing lights.

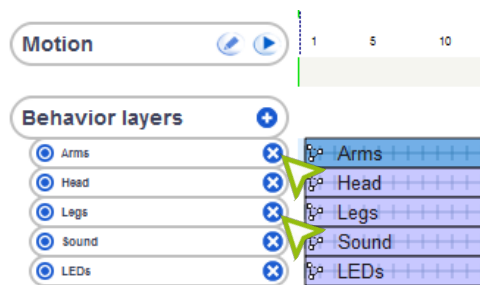
ADVANCED TASK

WALKING IN CIRCLES

WE HAVE FINISHED THE ROBOT DANCE. NOW, LET'S TRY DOING SOMETHING Fancier, AND HAVE THE NAO DO PART OF THE ROBOT DANCE WHILE WALKING IN A CIRCLE. TO DO THIS, WE WILL HAVE THE NAO USE THE OMNI-DIRECTIONAL WALK. WE WILL RECORD THE ODOMETRY INFORMATION AS IT WALKS TO DETERMINE WHEN WE ARE FINISHED.

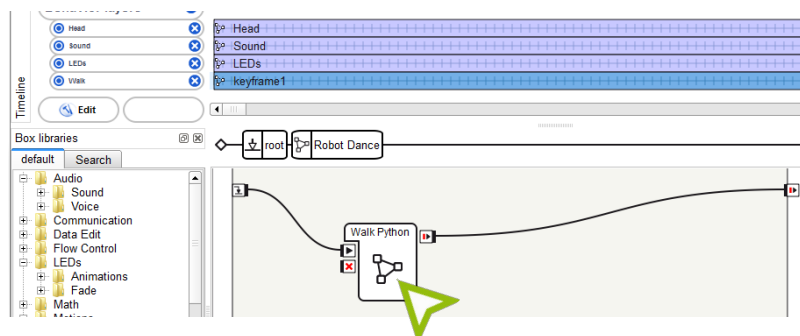
01/

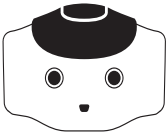
First, remove the behavior layers for controlling the arms and the legs. The NAO cannot do these motions while walking stably.



02/

Next, add a new behavior layer for the walk. Create a new box in this layer, connected to both the start and end arrows. Double click on the box to edit the python code. We will be adding functions into the code, bit by bit.





03/

Before we begin editing and adding functions to the code, we need to import the relevant libraries. Add the following lines before the `class MyClass` line.

```
import math
import time

class MyClass(GeneratedClass):
    def __init__(self):
```

04/

The first function we will edit is `__init__`.

```
import math
import time

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.motionProxy = ALProxy("ALMotion")
```

`__init__` is called only once (init stands for initialization), before the behavior starts executing. You have created a proxy to `ALMotion` in the past within the `onInput_onStart` function. The main difference here is that the variable name is `self.motionProxy` instead of just `motionProxy`. The addition of `self.` means that `motionProxy` is now a member variable of the class - it can be accessed in other functions.

```
def __init__(self):
    GeneratedClass.__init__(self)
    self.motionProxy = ALProxy("ALMotion")

def simplifyAngle(self, theta):
    while theta > math.pi:
        theta -= 2 * math.pi
    while theta < -math.pi:
        theta += 2 * math.pi
    return theta

def updateTheta(self, previousTheta, theta):
    nextTheta = self.motionProxy.getRobotPosition(False)[2]
    theta += self.simplifyAngle(nextTheta - previousTheta)
    return (theta, nextTheta)

def onLoad(self):
    #- puts code for box initialization here
    pass

def onUnload(self):
    #- puts code for box cleanup here
    pass

def onInput_onStart(self):
    self.motionProxy.setWalkArmsEnable(True, True)
    globalTheta = self.motionProxy.getRobotPosition(False)[2]
    theta = 0
    rotationSpeed = 0.25 # range from -1 to 1

    # Walk in a circle
    while abs(theta) < 2 * math.pi:
        # Update the rotation odometry
        (theta, globalTheta) = self.updateTheta(globalTheta, theta)
        # Walk in an arc
        self.motionProxy.setWalkTargetVelocity(0.5, 0.0, rotationSpeed, 1.0)
        # Wait for some time
        time.sleep(0.1)

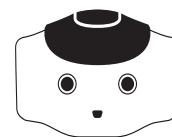
    # Stop walking
    self.motionProxy.setWalkTargetVelocity(0.0, 0.0, 0.0, 1.0)

    self.onStopped() #- activate output of the box
```

05/

Next, we will create a new function, `simplifyAngle`.

The `while` structure is a loop, similar to a `for` loop. However, instead of iterating over some value, it repeats until the condition in the loop evaluates to false. So the first loop will continue subtracting 2π from `theta` (θ) until it is less than π . Similarly, the second loop will continue adding 2π until θ is more than $-\pi$. Thus, `simplifyAngle` normalizes θ (in radians) to between $-\pi$ and π .



06/

We will now create another function, `updateTheta`.

```
def updateTheta(self, previousTheta, theta):
    nextTheta = self.motionProxy.getRobotPosition(False)[2]
    theta += self.simplifyAngle(nextTheta - previousTheta)
    return (theta, nextTheta)
```

The `updateTheta` function computes how much we have rotated so far (also known as odometry). ALMotion's function, `getRobotPosition`, returns the pose of the robot

(`x`, `y`, `theta`) in global coordinates based on where the NAO first started. Since we are only interested in the amount of rotation, we access the third element in the array (array indices start from 0, so we retrieve element [2]).

Also, we only want the change in position since the Circle box started, not since the robot started, and the second line maintains this information.

Finally, we return the amount of rotation since the Circle box started (`theta`), and the current global angle of the robot (`nextTheta`) - that is used for future calculations of `updateTheta`.

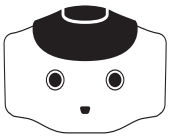
07/

Lastly, we will edit the `onInput_onStart` function - this function is called when the box executes, which we have done in previous exercises. We will use this function to have the NAO walk in a complete circle and then stop walking.

```
def onInput_onStart(self):
    self.motionProxy.setWalkAmsEnable(True, True)
    globalTheta = self.motionProxy.getRobotPosition(False)[2]
    theta = 0
    rotationSpeed = 0.25 # range from -1 to 1

    # Walk in a circle
    while abs(theta) < 2 * math.pi:
        # Update the rotation odometry
        (theta, globalTheta) = self.updateTheta(globalTheta, theta)
        # Walk in an arc
        self.motionProxy.setWalkTargetVelocity(0.5, 0.0, rotationSpeed, 1.0)
        # Wait for some time
        time.sleep(0.1)

    # Stop walking
    self.motionProxy.setWalkTargetVelocity(0.0, 0.0, 0.0, 1.0)
    self.onStopped() #~ activate output of the box
```



The first line calls `setWalkArmsEnable` on `ALMotion`, which enables the swinging of the arms as the NAO walks. The swinging of the arms helps to keep the NAO balanced.

The next three lines initialize variables. `globalTheta` stores the global angle of the robot when the box starts executing; `theta` stores the rotation so far (0 since the robot hasn't started walking yet); `rotationSpeed` is a value from -1 to 1 that controls how quickly the robot will rotate.

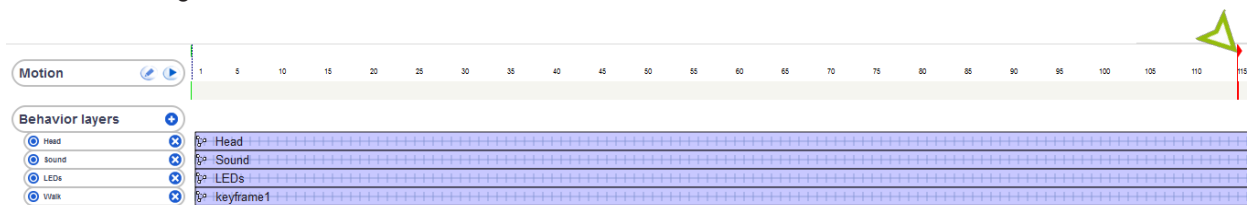
The `while` loop continues executing until the odometry for `theta` exceeds 2π meaning that the circle is complete. Inside the loop, `setWalkTargetVelocity` is called to set the velocity of the NAO's walk. The first 3 parameters are velocity in the x (forward), y (sideways, to the left), and θ (rotation, counter-clockwise) directions. Since we want the robot to walk in an arc, the walk velocity is set to 0.5 in the forward direction and 0.25 in the rotational direction. This makes the robot walk in an arc, where it walks forward and rotates at the same time.

The `time.sleep` function puts the thread to sleep, so the CPU can focus on other tasks. Without this line, the robot would execute this loop as fast as possible, but this is unnecessary and a waste of processing time.

Finally, we set the walk velocity back to zero. This stops the robot's walk once the circle is complete. Otherwise, the function would end but the robot would continue walking.

08/

The behavior will currently stop when it reaches the red flag you placed on the timeline previously. Right click on the red flag, and select "Reset end frame." Now the behavior will end when the robot finishes walking in a circle.



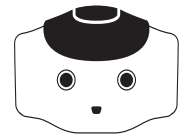
09/

Run the behavior. The robot should walk in a circle and stop.

WARNING: do not hit the red X button to stop the behavior. Otherwise, the robot may continue walking endlessly. If the NAO does walk endlessly, create a new project with a single Walk Toward box, and set the walk velocity in all directions (x , y and θ) to 0.

10/

Measure the odometry error at the end of the behavior. That is, how far does the robot's final position differ from its initial position?



11/

Does the robot walk in a clockwise or counter-clockwise circle? Change the code so that the robot walks in the opposite direction.

12/

Measure the radius of the circle the robot walked in. How can you change the radius of the circle that the NAO walks? Make the NAO walk in circles of radii 0.25 m and 1 m.

13/

Look again at the `simplifyAngle` method. This algorithm works, but it is terribly inefficient. What will happen if a very large number is passed to the function? How many times will the loop repeat? In computer science, this algorithm is called $O(\theta)$ (“big-O of theta”), meaning that the runtime increases linearly with θ . The `simplifyAngle` function can be written to be $O(1)$, meaning the runtime does not depend on θ and the algorithm always runs in constant time. Rewrite the `simplifyAngle` algorithm to be $O(1)$. Your new function should not include any loops.

Hint: Use `int(number)` to truncate (cut off the parts after the decimal) the number to an integer. So, `int(1.2) = 1`, `int(2.0) = 2`, and `int(1.9999) = 1`.

ADDITIONAL EXERCISES

- 01/** Modify the robot dance so that the NAO only bobs its head up and down each time you touch the head sensors.
Hint: add a Tactile Head box that links to the Head keyframe in the behavior layer.
- 02/** Make the NAO walk in a square (building off the earlier exercise) and perform a different combination of speech and/or motion as it traverses each edge of the square.
- 03/** Make the NAO walk in a figure eight while doing the robot dance (minus the arms).
Hint: Modify the advanced exercise to walk in a semicircle for the two ends of the figure eight. For the central crossovers, linearly interpolate the θ in the walk velocity from the turning radial velocity to zero as the angle changes from 0 to 45 degrees. Once 45 degrees is reached, interpolate the radial velocity in the opposite direction until 0 degrees is reached again.

MODULE QUESTIONS

BASIC

- 01/** What does CPU stand for?
- 02/** Explain, at a high level, how a single processor performs multi-tasking.

INTERMEDIATE

- 03/** Explain the use for behavior layers.
- 04/** Why does standing up straighter make the robot more susceptible to falling?

ADVANCED

- 05/** Why did we need to use the `simplifyAngle` method when computing the odometry?
- 06/** Explain the use of the `setWalkTargetVelocity` function.
- 07/** Why did we need to call `time.sleep` in the main loop?
- 08/** Why do we have to set the walk velocity to zero at the end of the loop?
- 09/** Using only division, multiplication, addition, subtraction, and integer truncation, write a function that finds the remainder of x / y .
Note: this function, called modulus, is already implemented in python as `%`. But do not use `%`.
- 10/** Find the big- O runtime cost of the following functions:
- Searching for a name in a telephone book of n pages by:
 - Opening a page you bookmarked previously.
 - Beginning at the first page, and checking each page in order for the name.
 - Open the middle page, see if the name is to the left or the right, then open the middle of the remaining half of the book, and repeat until the name is found.
 - Enumerating every 5-card hand in a deck of n cards.
 - Finding the correct n -bit key to open a lock by trying every possible key (this is called a brute force attack in computer security).
 - Sorting a list of numbers by finding the lowest number of the entire list, placing it first, finding the next lowest number, placing it second, and so on. This is called selection sort.

7 FACE OFF

LEARNING

In this module, students will learn:

→ **How to make the NAO detect faces from Choregraphe**

→ **How to look in the direction of sounds**

→ **How to recognize and distinguish faces**

→ **How to scan with the NAO's head**

→ **What queues are**

→ **How to use queues / lists in python**

→ **How to deal with time in python**

- * RST.11-12.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.
- * RST.11-12.8. Evaluate the hypotheses, data, analysis, and conclusions in a science or technical text, verifying the data when possible and corroborating or challenging conclusions with other sources of information.
- * RST.11-12.9. Synthesize information from a range of sources (e.g., texts, experiments, simulations) into a coherent understanding of a process, phenomenon, or concept, resolving conflicting information when possible.
- * RST.11-12.10. By the end of grade 12, read and comprehend science/technical texts in the grades 11–12 text complexity band independently and proficiently.

* Reference COMMON CORE Stem standards

CONTENTS

01/ Basic Task: Seeing Face to Face

02/ Intermediate Task: Recognizing Faces

03/ Intermediate Task: Seeking Out Faces

04/ Advanced Task: Remembering Faces

05/ Additional Exercises

06/ Module Questions

07/ Module Question Solutions

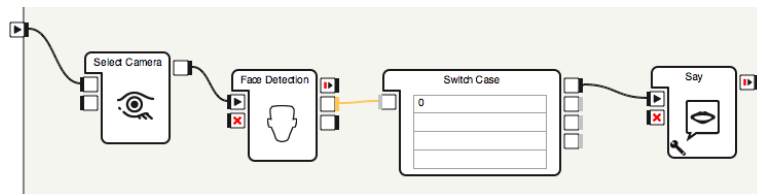
BASIC TASK

SEEING FACE TO FACE

IN THIS MODULE WE'LL EXPERIMENT WITH NAO'S ABILITY TO DETECT HUMAN FACES. FIRST, WE WILL HAVE THE NAO SPEAK WHEN IT SEES A HUMAN FACE.

01/

Link up a select camera box, a face detection box, a switch box, and a say box as shown below. The select camera box will activate the NAO's top camera (in its forehead) instead of the camera in its chin. The face detection box outputs the number of faces the robot sees. If this number is zero, the robot does nothing. Otherwise, it executes the Say box.



02/

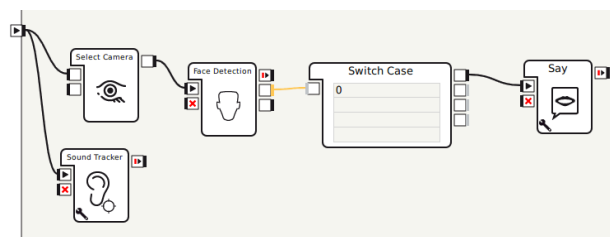
Set a message for the say box, such as "Hello, human."

03/

Execute the behavior. Put your face in the line of sight of the robot's camera (keep in mind that this is quite narrow). The robot should greet you. If the robot doesn't see you, try moving your head and/or the robot's head slightly until it does.

04/

Now add a Sound Tracker box (in the Trackers section) and link it to the start arrow in parallel to the Select Camera box. Run the behavior again. Snap your fingers or make a sound, and the NAO should look in the direction of the sound.



INTERMEDIATE TASK

SEEKING OUT FACES

—

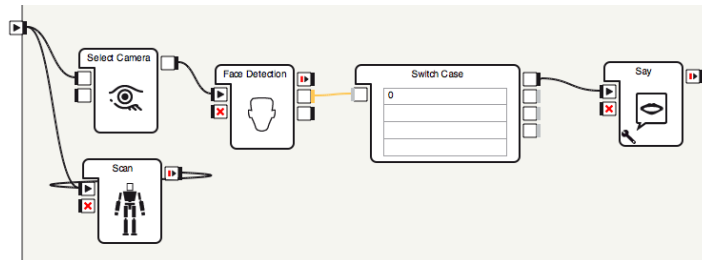
THE NAO CAN SEE A FACE THAT HAPPENS TO PLACE ITSELF IN FRONT OF ITS CAMERA. NOW WE WILL MAKE IT SCAN ITS HEAD TO LOOK FOR FACES.

01/

Begin with the results of the first exercise, which detects faces.

02/

Add a new timeline box to do a head scan, as shown below.



03/

Add keyframes to the custom box to make the head move from side to side.

04/

Run the behavior and see if the robot can see faces. If not, you may need to slow down the head motion.

ADVANCED TASK

REMEMBERING FACES

—

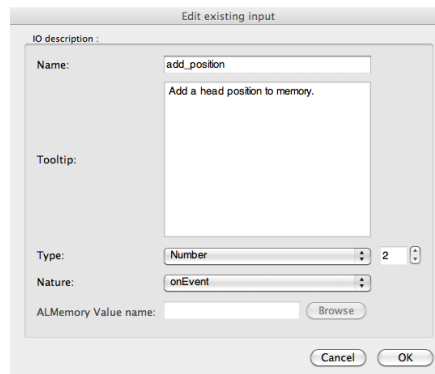
BEGIN FROM THE BASIC TASK, WHERE THE NAO LOOKS IN THE DIRECTION OF A NOISE. WE WILL CHANGE THIS BEHAVIOR TO MAKE THE ROBOT REMEMBER THE LAST TWO POSITIONS IT HAS HEARD A NOISE IN, AND TO CYCLE THROUGH THESE POSITIONS.

01/

Begin with the basic task. Double click the Sound Tracker Box on the workspace. You will see a Sound Loc. box. Copy this box, and replace the Sound Tracker Box in the original workspace with a Sound Loc. box.

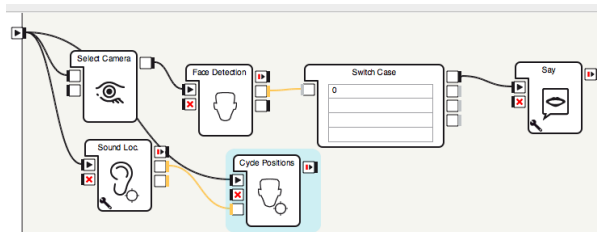
02/

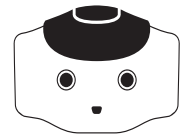
Create a new box, and add an input named `add_position` which takes two Number parameters as shown below.



03/

Now connect the output of the sound Loc. box to your new boxes input. The Sound Loc. box doesn't move the robot's head, but only outputs where a sound was heard.





04/

Add the following code to the custom box.

```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)
        self.motion = ALProxy("ALMotion")
        self.positions = [[0.0, 0.0]]
        self.pos = 0
        self.last_time = 0

    def onLoad(self):
        #~ puts code for box initialization here
        pass

    def onUnload(self):
        self.running = False

    def onInput_onStart(self):
        self.running = True
        while self.running:
            while time.time() < self.last_time + 2.0:
                time.sleep(self.last_time + 2.0 - time.time())
                self.last_time = time.time()
                self.pos = self.pos + 1
            if self.pos >= len(self.positions):
                self.pos = 0
            self.motion.setAngles(["HeadYaw", "HeadPitch"], self.positions[self.pos], 0.5)

    def onInput_onStop(self):
        self.onUnload() #~ it is recommended to call onUnload of this box in a onStop
        # method, as the code written in onUnload is used to stop the box as well
        pass

    def onInput_add_position(self, p):
        self.positions.append([p[0], p[1]])
        if len(self.positions) > 2:
            self.positions = self.positions[1:]
        self.motion.setAngles(["HeadYaw", "HeadPitch"], [p[0], p[1]], 0.5)
        self.last_time = time.time()
```

In this code, we maintain `self.positions` as a queue of head positions where we have heard a sound recently. A queue is a list which is like a line: the first thing to come in is the first thing to go out. Only the two most recent head positions are stored. The `append` method adds a new value to the end of the list. In the `if` statement, `len(self.positions)` returns the length of the list. If this is greater than two, we shorten the list. Then `self.positions[1:]` gets everything in the list except the first (zeroth) element, from element one to the end of the list. The variable `self.pos` indicates which index in the list we are currently looking at, and is incremented every two seconds.

The only other thing that should be new in this example is the function `time.time()`. This returns the number of seconds that have passed since midnight on Jan. 1, 1970 (called "time since the epoch"). We use this value to make sure we stay in each state for at least two seconds, including states we jump to after hearing a sound.

05/

Now run the behavior. See that the robot looks at sounds and oscillates between looking at the two latest places it heard noises from.

06/

(Optional) As it stands now, we may jump to look at a new position, wait two seconds, and then look at that new position again immediately. Modify the code so that we do not look at the same position twice in a row.

ADDITIONAL EXERCISES

- 01/** When the robot sees a face, in addition to giving a greeting, make it wave and flash its lights.
- 02/** Make the NAO recognize two different faces and greet the people differently.
- 03/** While the robot is scanning for humans, make it stop scanning if it sees a face and look at that person. To make it look in the correct direction, you may need to reduce the scanning speed further.

MODULE QUESTIONS

BASIC

01/ What does the face box output?

02/ Speculate as to why the robot does not always detect your face.

INTERMEDIATE

03/ What is the difference between face detection and recognition?

04/ What does the face recognition box output?

ADVANCED

05/ What areas will the search not see? How could you expand the robot's search, using both the head and by walking?

8 OBJECT RECOGNITION

LEARNING

In this module, students will learn:

- **How images are stored on a computer/robot**
- **What object recognition is**
- **How object recognition is performed**
- **What a logical-AND operation is**
- **How to loop a behavior in Choregraphe until a condition is met**
- **How to parse strings to search for prefixes**

- * RST.11-12.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.
- * RST.11-12.8. Evaluate the hypotheses, data, analysis, and conclusions in a science or technical text, verifying the data when possible and corroborating or challenging conclusions with other sources of information.
- * RST.11-12.9. Synthesize information from a range of sources (e.g., texts, experiments, simulations) into a coherent understanding of a process, phenomenon, or concept, resolving conflicting information when possible.
- * RST.11-12.10. By the end of grade 12, read and comprehend science/technical texts in the grades 11–12 text complexity band independently and proficiently.

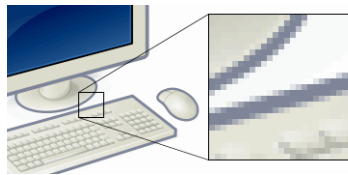
* Reference COMMON CORE Stem standards

CONTENTS

- 01/** Digital Images and Pixels
- 02/** Computer Vision
- 03/** Basic Task: NAO-Mark-Controlled Robot
- 04/** Intermediate Task: Object Recognition
- 05/** Advanced Task: Walk to an Object
- 06/** Additional Exercises
- 07/** Module Questions

DIGITAL IMAGES AND PIXELS

You may have used a digital camera and heard of phrases such as “10 megapixels”. What is a megapixel? A megapixel is 1,000,000 pixels. What is a pixel then? ‘Pixel’ stands for picture element. A pixel is the smallest part of an image or screen. In a digital image or photo, a pixel is the smallest area that is a single color. The figure below is taken from the Wikimedia Commons. It shows an image of a computer, with a zoomed-in section of the individual pixels.



Similarly, a digital photo is composed of pixels. Thus, photos taken by a “10 megapixel” camera have 10 million pixels. Every digital camera has a sensor that converts the light coming in through the lens into digital signals. These digital signals are then used to create the pixels of the image.

The NAO has video cameras that do the same thing. However, the NAO’s camera doesn’t take a single image when a button is pressed. Instead, it takes a video - a continuous stream of images. The NAO’s cameras are capable of a *resolution* of 1288 pixels by 968 pixels, at a *frame rate* of 30 frames per second (fps).

The resolution of an image refers to how many pixels there are along the length and breadth of the image. So, for the NAO’s camera, the resolution is 1288 by 968. This means that the images are 640 pixels wide, and 968 pixels high. The frame rate refers to how many images are taken per second. So, the frame rate of 30 fps means that the camera takes 30 images every second.

COMPUTER VISION

From a single digital image, we humans are able to identify the objects within it. For example, when presented with a picture of a beach, we can readily say that it shows a beach. We can also point out the sand, sea, clouds, and trees in the image.

For a computer, this task, known as computer vision, is not easy at all. Firstly, the image is entirely made up of pixels. This means that all the computer has are a series of numbers that represent the colors of each pixel. Given an image of a cup on a table, it is hard for the computer to tell that one pixel is part of the cup while the next pixel is the table. Also, the computer must have some concept of “cup” and “table” in its memory. As humans, we have a large memory, or database, of objects that we know and have interacted with. This database helps us to identify objects just from a quick, small view of it.

In computer vision, the algorithm first finds *features* in the image. Features are distinguishing parts of an image that aid the computer vision algorithm in deciding what object is present. Typical features include edges and texture. Edges occur because objects tend to look different from the background. Some objects tend to have uniform textures. Another feature that usually comes to mind is color. However, it is difficult for an algorithm to use color as a feature. This is because the same color looks very different under various lighting conditions. In most situations, our eyes are capable of adjusting for the color of the light source so we can tell if an object is red, but computers are unable to do so robustly.

Another aspect of features used in computer vision is that a feature should be *scale-invariant*. Invariant means “never changing”. So, a scale-invariant feature means that the same feature is detected regardless of how big or how small the object is in the image. A common feature set used in computer vision is the scale-invariant feature transform (SIFT) feature set. We will not describe the details of what SIFT features are. The main idea is that these features are extracted from an image. They are then used to detect the objects in the image. One method is to have a database of images that are labeled. Given a new image, the features of the new image are compared to the features of images in the database. The object in the database with the closest features is then chosen. For example, suppose there is an image of a cup with SIFT features A, B and C. If the new image also has SIFT features A, B and C, then it is likely that the new image is an image of a cup.

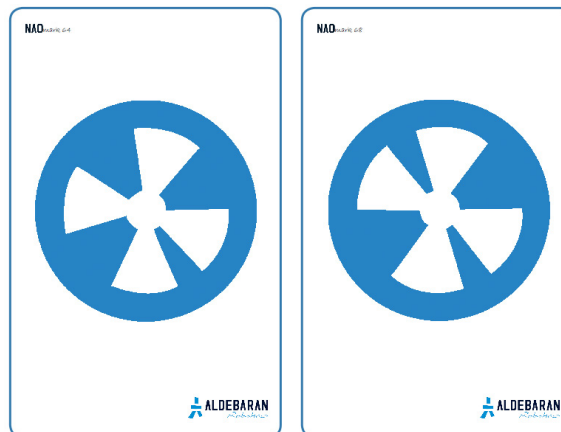
BASIC TASK

NAOMARK- CONTROLLED ROBOT

—

IN THIS EXERCISE, WE WILL BE USING THE NAO TO PERFORM COMPUTER VISION. IN YOUR DVD, YOU WILL FIND A FILE CONTAINING NAOMARKS. NAOMARKS ARE CIRCULAR DESIGNS THAT LOOK LIKE THE FIGURE BELOW:

Choregraphe comes with an automated algorithm that detects these NAOMarks. The program returns their identification number, which is located on the top left of each card. The algorithm detects the unique shape of the NAOMarks, and divides them up based on the sizes of the white and blue regions within the mark.

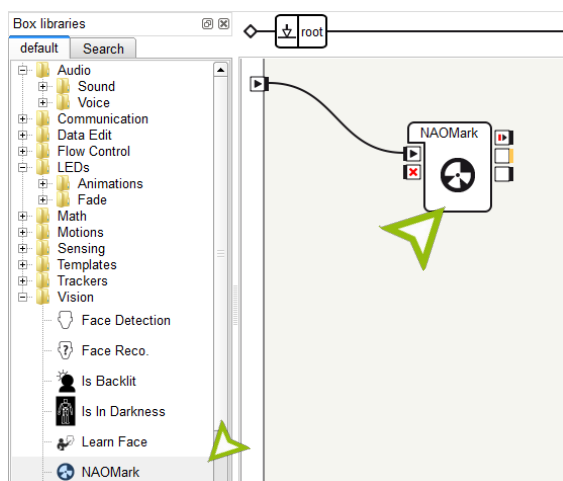


We will be using these NAOMarks as input to control the NAO's actions. In the earlier modules, we created a voice-controlled robot. In this exercise, we will build a robot that walks based on visual cues.



01/

Create a NAOMark box from the Vision category, and connect it as shown below.



The NAOMark box has two outputs. The first (middle box on the right) returns the identification number of the NAOMark detected (if any). The second (bottom-right box) triggers if no NAOMarks are detected.

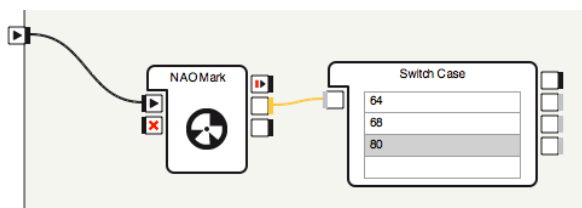
02/

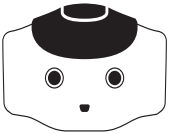
Try running the behavior as is. Print out 3 NAOMarks, and show it to the robot. You should see the NAOMark number appear in the middle-right box of the NAOMark box in Choregraphe.

Hint: you may notice that other numbers sometimes get detected as well. For example, NAOMark 64 may be detected as 64, and sometimes 79 or 127. Make sure the three NAOMarks you choose look different from one another. So, if you picked NAOMark 1, 2 and 3, then ensure that NAOMark 1 does not ever get detected as 2 or 3.

03/

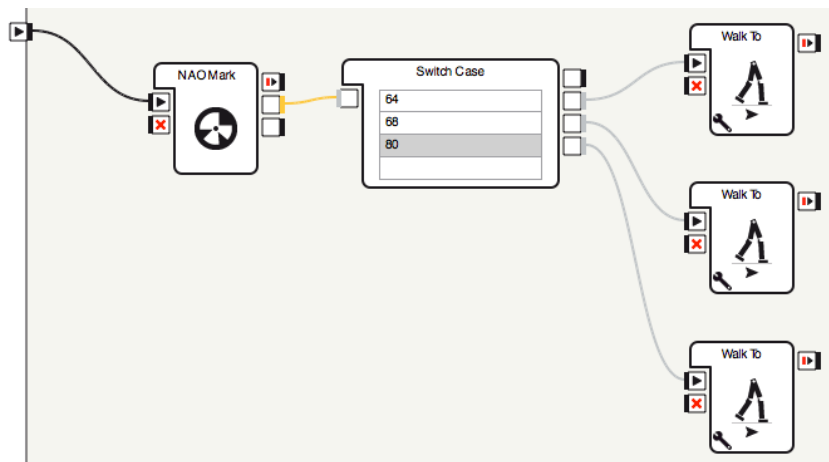
Connect a Switch Case box to the output of the NAOMark box. Within the Switch Case box, enter the identification numbers of the three NAOMarks that you chose. In the figure below, we used NAOMarks 64, 68 and 80.





04/

When a particular NAOMark is detected, we want the robot to perform a unique action. Drag three Walk To boxes and connect them to the Switch Case box, as shown below.



05/

For each of the three Walk To boxes, enter different values for x, y and theta, so that the actions taken for each NAOMark is unique.

06/

Run the behavior. Show the different NAOMarks to the robot, and ensure that it performs the correct action for each mark.

INTERMEDIATE TASK

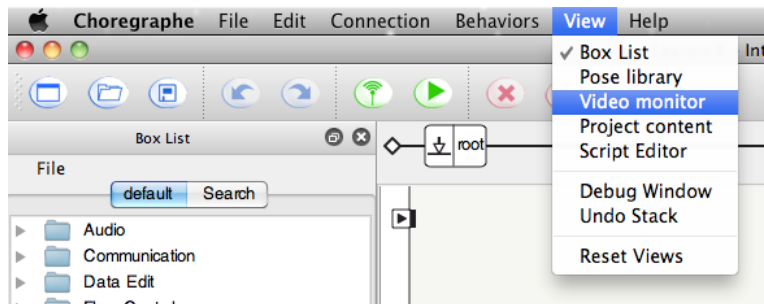
OBJECT RECOGNITION

—

IN THE PREVIOUS EXERCISE, WE DETECTED NAOMARKS WITH AN ALGORITHM IN-BUILT INTO CHOREGRAPHE. IN THIS EXERCISE, WE WILL EXPLORE DEFINING AND DETECTING GENERAL IMAGES.

01/

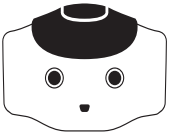
First, we need to create a library of known objects for the NAO. To do so, connect to the robot in Choregraphe, and then click View → Video Monitor in the top menu bar. The window below should pop up. Click the New Vision Recognition Database button if it is not grayed out.



02/

Click the play button in the Video Monitor (not the “play behavior” button in the Choregraphe toolbar, although it looks the same), and you should see a video stream from the NAO. Move your hand in front of the NAO’s face and ensure that you are streaming the video correctly. The NAO has two cameras, so check both of them to see which camera the NAO is using. You can switch the cameras using the Select Camera box under the Vision category.





03/

Now, place the object you want the NAO to learn in front of the camera, and click the learn button. A countdown will appear on the Video Monitor, after which a single image will be paused and shown in the window.

Hint: For this exercise, we recommend using a flat object with pictures, such as the NAO DVD sleeve that is included in the NAO package. Other alternatives could be the cover of a book, or a page from the newspaper.

Video monitor



04/

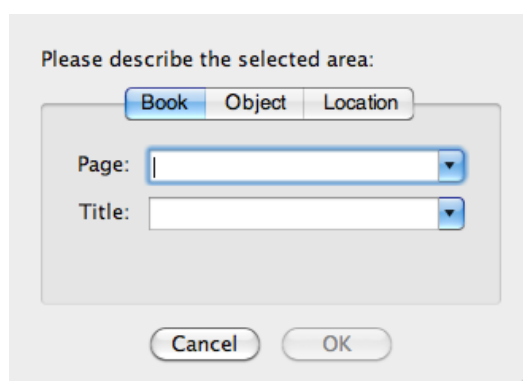
You can now click on the border of the object to trace its outline. To end the trace, click on the first point of the border again. The features of the object will then be extracted within this outline and saved.





05/

Enter a name for the object, and the features will be associated with this object in the local Choregraphe database.



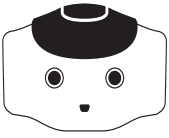
06/

If you want to learn more objects, you can repeat steps 2 to 5.

07/

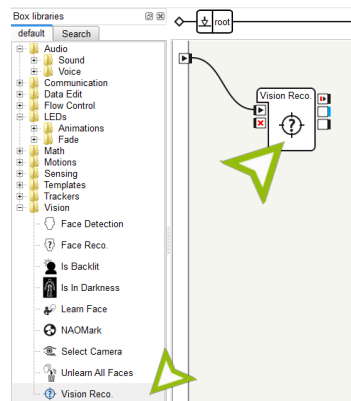
Before we can do object recognition on the NAO, we need to send it the database of objects that we have created. Click the right-most button to send current vision recognition database to NAO.





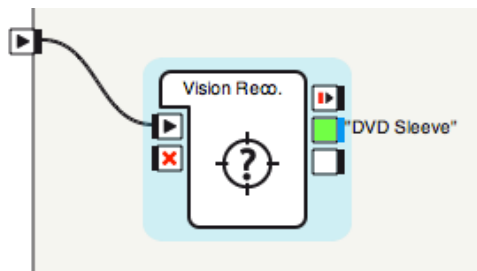
08/

Now that the NAO has a database of objects and their features, we can perform object recognition in Choregraphe. To do so, drag a Vision Reco box under the Vision category.



09/

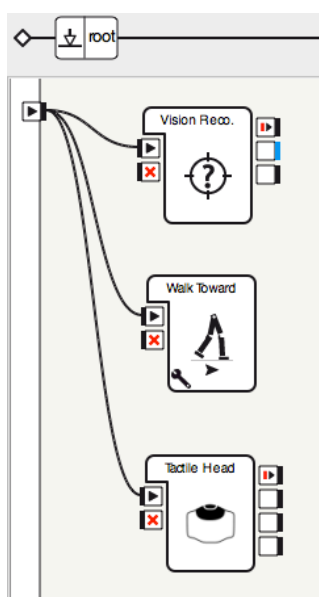
Start the behavior, and place the object in the NAO's camera view. Ensure that the object gets detected, and the Vision Reco box returns the object's name in its output box. If the object does not get detected, repeat the steps above to create a new database and send it to the NAO.





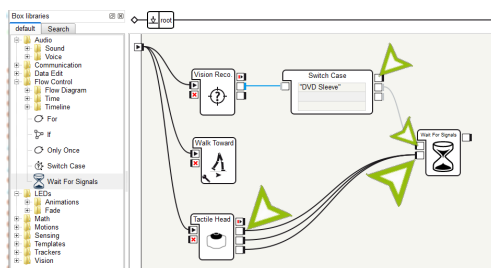
10/

We will now use the Vision Reco box to have the NAO walk until it sees an object that it recognizes, and is touched on the head. We will use a Walk Toward box, from the Motions category, and a Tactile Head box from the Sensors category.

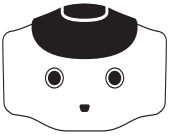


11/

We want the NAO to stop walking when it sees an object it recognizes *and* a head sensor is touched. To do so, add a Switch Case box to check the output of Vision Reco box, and the Wait For Signals box from the Flow Control category. Connect the output of the Switch Case box to one of the inputs of the Wait For Signals box, and connect the outputs of the Tactile Head box to the other input of the Wait For Signals box.

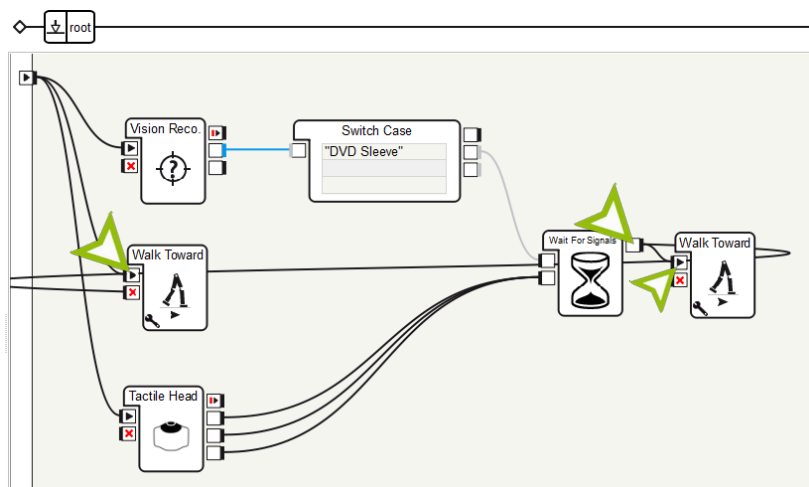


The Wait for Signals box waits for both of its inputs before triggering its output - it performs a logical-AND operation. A logical-AND checks that all its inputs are true, e.g., true AND true is true, but true AND false is false, false AND false is false.



12/

When the Wait for Signals box triggers, it means that the desired object was detected, and the head sensor was touched. In this case, we want the robot to stop walking. To do so, drag another Walk Toward box, and connect it to the output of the Wait for Signals box. Set the parameters of the Walk Toward box to have x, y and theta set to 0. Also, connect the output of the Wait for Signals box to stop the previous Walk Toward box.



13/

Run the behavior. Check that the NAO starts walking, and stops only when the object is detected, and the head sensor is touched.

INTERMEDIATE TASK

OBJECT RECOGNITION

WE MADE THE NAO PERFORM OBJECT RECOGNITION ON A FLAT OBJECT LIKE A DVD SLEEVE IN THE PREVIOUS EXERCISE. HOWEVER, MANY OBJECTS IN OUR ENVIRONMENT HAVE COMPLEX SHAPES, AND LOOK DIFFERENT FROM DIFFERENT PERSPECTIVES. FOR EXAMPLE, A MUG LOOKS DIFFERENT WHEN YOU LOOK AT IT FROM THE SIDE COMPARED TO LOOKING AT IT FROM ABOVE. A BOOK LOOKS DIFFERENT DEPENDING ON WHICH PAGE IT IS OPEN TO.

01/

For this exercise, we will use a complex object such as a book or a brochure, that either can change its shape (like opening a book) or has a complex shape (like a mug).
Hint: use an object with multiple colors or a complex texture.

02/

Create a new vision recognition database in Choregraphe (refer to step 1 of the previous exercise).

03/

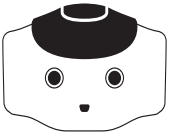
Repeat steps 2-5 of the previous exercise to save different images of the object. However, besides giving the object the same name, also fill in the field on the page of the book, or side of the object.

The image shows two side-by-side screenshots of a dialog box titled "Please describe the selected area:". Each dialog has three tabs: "Book", "Object", and "Location".

The left dialog has the "Book" tab selected. It contains two dropdown menus: "Page:" with the value "1" and "Title:" with the value "Title of book".

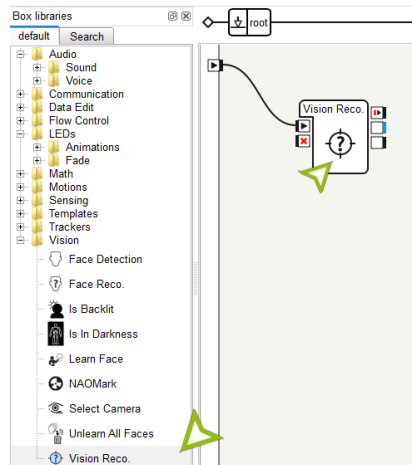
The right dialog has the "Object" tab selected. It contains two dropdown menus: "Side:" with the value "Front" and "Name:" with the value "Mug".

Both dialogs have "Cancel" and "OK" buttons at the bottom.



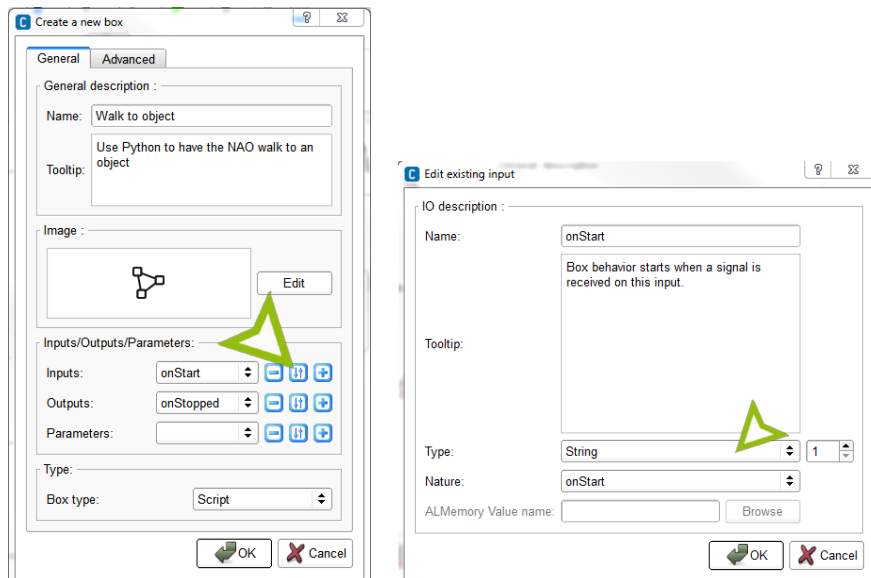
04/

Drag out a Vision Reco box, and run the behavior. Ensure that the object can be detected from different distances and angles.



05/

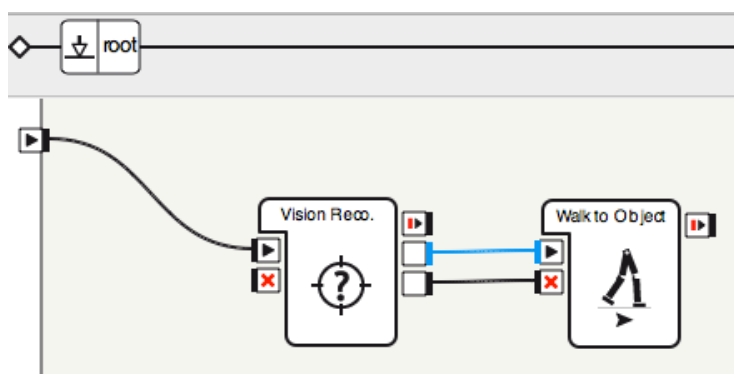
Create a custom Python box and connect it to the output of the Vision Reco box. Change the input type of the Python box to be String, instead of "Bang".





06/

Connect the outputs of the Vision Reco box to the Python box, such that onStart is triggered when an object is detected, and the x is triggered when no object is detected.



07/

When the object is detected, the Vision Reco box returns the name of the object as well as the side/page. For example, page 1 of the book would be returned as a string “book 1”, and page 2 would be “book 2”. For the purposes of this exercise, we don’t distinguish between the pages and sides and only care that the book was detected. However, we do care that the book was detected and not the mug, if there are multiple objects in the database.

To do so, we can check that the input starts with the name of the object, e.g., “book 1” and “book 2” both start with “book”.

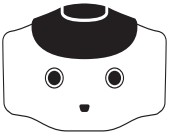
08/

Enter the following Python code into the custom Python box. The startswith function, as its name suggests, return true if the string does start with the prefix you provide. In the example below, we check that the string p starts with the prefix “Brochure”. So, inputs such as “Brochure 1”,

```
def onInput_onStart(self, p):
    if p.startswith('Brochure'):
        motionProxy = ALProxy("ALMotion")
        motionProxy.walkTo(0.10, 0, 0)
```

“Brochure 2”, “Brochure front” would be accepted, but “Mug top” and “Mug side” would not.

Thus, when the object is detected, we use the motionProxy to walk the robot forward 10cm. Otherwise, the behavior doesn’t do anything.



09/

Run the behavior. Place the object within the camera view of the NAO, and it should walk towards it. Once the NAO is close to the object, it will be outside the camera's field-of-view and not be seen by the NAO. As such, it will not be detected and the NAO will no longer walk forward.

Warning: be careful that the NAO does not walk over the object. It may cause damage to the object, the NAO, or both.

ADDITIONAL EXERCISES

- 01/** Use one NAOMark to start the NAO turning in one direction, and another NAOMark to stop its motion. This will allow you to use visual landmarks to direct the NAO's motions.
- 02/** Use Python to do the same behavior as above. You will still have to use the NAOMark box to detect the NAOMarks.
- 03/** Use the Switch Case box and enumerate all the sides/pages of the 3D object of the Advanced Task, in order to have the NAO walk to an object, without using Python.
- 04/** Using Python, have the NAO search for the object with its head when it doesn't detect an object. Once it sees the object, use the HeadYaw angles of the head to determine which direction to turn and walk. For example, if head is pointed to the left, then the NAO should turn to the left and then walk forward.

MODULE QUESTIONS

BASIC

- 01/** What is a pixel?
- 02/** What does a NAOMark box in Choregraphe do?

INTERMEDIATE

- 03/** How does object recognition work?
- 04/** Is color a good feature to be used in computer vision? Why or why not?
- 05/** When learning a complex object, why do we need to provide many different images of the object?

ADVANCED

- 06/** How do we use Python to determine if a string starts with a prefix, e.g., how do we check that a variable called `myst_r` starts with "Abc"?
- 07/** In the Advanced Task, why does the robot stop walking once it reaches the object?

9 GAMES AND STORIES

LEARNING

In this module, students will learn:

- **What human-robot interaction is**
- **Why human-robot interaction is important**
- **How to make robots interact with humans**
- **How to do cooperative motions (hand shakes and high fives) with humans**
- **To act out a play on the NAOs.**
- **To share the results with children**

CONTENTS

- 01/** Human-Robot Interaction
- 02/** Basic Task: Greetings
- 03/** Intermediate Task: Peek-a-boo
- 04/** Advanced Task: Storytelling
- 05/** Additional Exercises
- 06/** Module Questions

HUMAN-ROBOT INTERACTION

Robots can do many things on their own. As humans, we want to build robots that interact with us. You may already interact with robots every day. Some cars can parallel park themselves. Other robots assemble the manufactured goods you use.

Human-robot interaction studies the interactions between humans and robots. This field combines robotics and psychology. It seeks to answer a number of questions. How can humans better control robots? How can humans and robots work together? And, in general, how can we improve the human experience and make robots more effective tools?

In this module, we will program the NAO to interact with humans. The NAO appeals to humans because of its humanoid shape. It is also designed to look “cute”. By the end of this lesson, you will have completed three tasks in which the robot interacts with humans. You will make the Nao communicate with hand motions, including shaking hands, giving a high five, and waving goodbye. You will program the NAO to play peek-a-boo with children. Finally, you will re-enact a scene from a play. After finishing this module, show off the results to children, adults or fellow students outside your class!

BASIC TASK

GREETINGS

IN THIS MODULE, WE'LL FOCUS ON HOW ROBOTS CAN INTERACT WITH HUMANS. WE'LL TEACH THE NAO TO SHAKE HANDS, GIVE A HIGH FIVE, AND WAVE GOODBYE.

01/

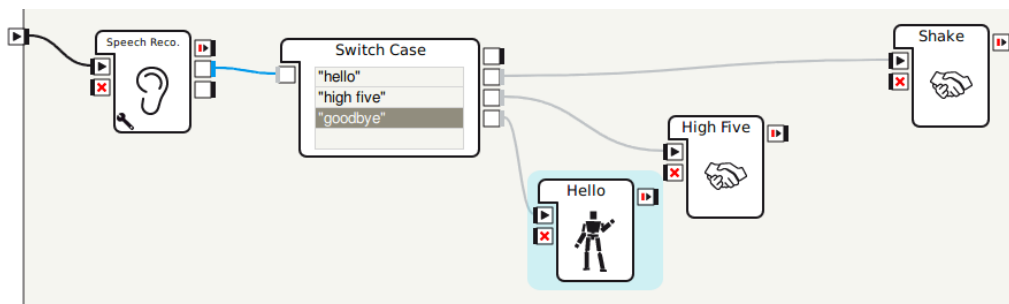
Create a new Choregraphe project, with three new Timeline boxes for keyframe motions.

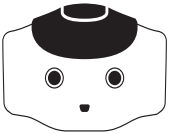
02/

You should be well-versed by now at creating keyframe motions. For one keyframe motion, make the NAO do a hand shaking motion. For the second, make the NAO give a high five. For the third, make the NAO wave goodbye (you can use the Hello motion box).

03/

Now add a Speech Recognition box and a Switch Case box. Set the word list for the speech recognition box to be "hello;high five;goodbye", and set these three words in the Switch Case box. Connect these two boxes to the motion boxes as shown below.

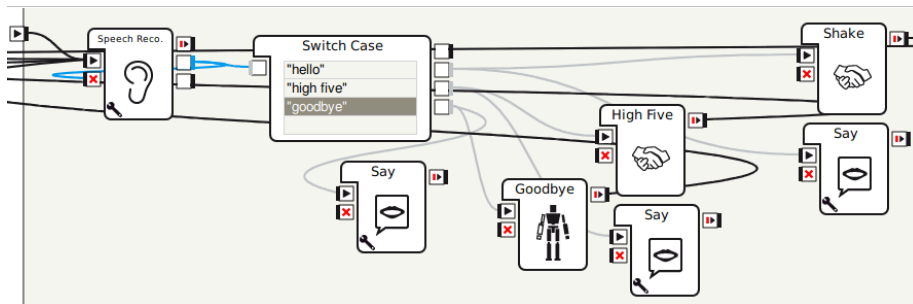




04/

Now add Say boxes, one for each action, connected to the same output of the Switch Case box. For each Say box, make the robot say a message related to the associated action.

With this setup, the Speech Recognition box continues running while the robot speaks. If our greeting when we shake hands includes the word “hello”, the robot will recognize itself as having said “hello” and continue shaking hands forever. To fix this, connect the output of the speech recognition box to the red X on the same box. Then connect each of the motion boxes to the speech recognition box to make the behavior run continuously. The final result should be similar to that shown below.



05/

Now run the behavior, and try saying each phrase on the word list.

INTERMEDIATE TASK

PEEK-A-BOO

—

NEXT, WE'LL MAKE THE NAO PLAY THE CHILDREN'S GAME "PEEK-A-BOO".

01/

Create a new Choregraphe project, with two new Timeline boxes for keyframe motions.

02/

For one keyframe motion, make the NAO hide his face with his hands. For the other, make him put his hands at his side. Be sure to make these gradual transitions.

03/

Add a speech recognition box, which recognizes the phrases "peek a boo" and "I see you". Also add a face recognition box connected to a switch case box, which does nothing on receiving 0 and continues on in the default case. The robot should play the game in a cycle.

04/

The order of actions should be as follows. The final result is shown below.

- a. Hide the NAO's face with its hands.
- b. Say "Where am I?"
- c. Recognize the speech. This box should link to its own X on completion.
- d. Remove the hands from the robot's face.
- e. Say "You found me! Now you hide."
- f. Recognize a face. Again, link this box to its own X on completion.
- g. Say "Peek a boo! I see you!" after detecting a face.
- h. Hide the NAO's face with its hands again and repeat.

05/

Try playing peek-a-boo with the NAO.

ADVANCED TASK

STORYTELLING

—

NEXT, WE'LL MAKE THE NAO EITHER TELL A STORY OR ACT OUT A SCENE FROM A PLAY.

Choose either a short scene or a skit that interests you. Use a combination of voices and motions on the robot. You have already learned all the skills needed to do this exercise, so no step-by-step instructions will be provided. A few hints and suggestions:

01/

If the scene has multiple characters, try changing the voice parameters and head lights to differentiate between them.

02/

Use expressive hand gestures and move the NAO's head.

03/

Use a variety of lights.

04/

Try including sound effects.

05/

Use the "Wait for Signals" box to wait for multiple boxes to finish (i.e., speech and motion) before beginning the next action.

06/

Use "Wait" boxes to insert dramatic pauses into the script.

07/

Be creative!

ADDITIONAL EXERCISES

- 01/** Make the NAO play rock paper scissors. Think carefully about how to modify the hand gestures for NAO's hand and arm.
- 02/** Divide the class into several groups, and make each group responsible for delivering the lines of a character in a specific scene of a play. Combine the groups' results to enact the play. If available, use multiple robots.
- 03/** Bring the NAOs to an elementary school, and show the class projects to the children at the school.

MODULE QUESTIONS

- 01/** Describe human-robot interaction.
- 02/** Give an example of a robot that interacts with people.
- 03/** Give a reason why humans find the NAO an appealing robot.
- 04/** Discuss how multiple robots would need to cooperate to enact a play together.

10 FINDING YOUR WAY

LEARNING

In this module, students will learn:

→ **How to solve mazes**

→ **What the dead-end filling algorithm for solving a maze is**

→ **How to solve a maze without a map using the wall-following algorithm**

→ **How to find the shortest path from the start to the goal in a maze using the breadth-first search algorithm**

→ **What a visual cue is**

→ **How to use visual cues to instruct a robot to solve a maze**

→ **How to use cues of multiple types to solve a maze**

→ **How to implement a maze-solving algorithm with Choregraphe**

→ **How to implement a maze-solving algorithm in Python.**

CONTENTS

01/ Solving Mazes with a Map

02/ Basic Task: Maze Solving with Visual Cues

03/ Intermediate Task: Maze Solving with Multiple Cues

04/ Completing Mazes without a Map

05/ Advanced Task: The Right Way to Maze Solving

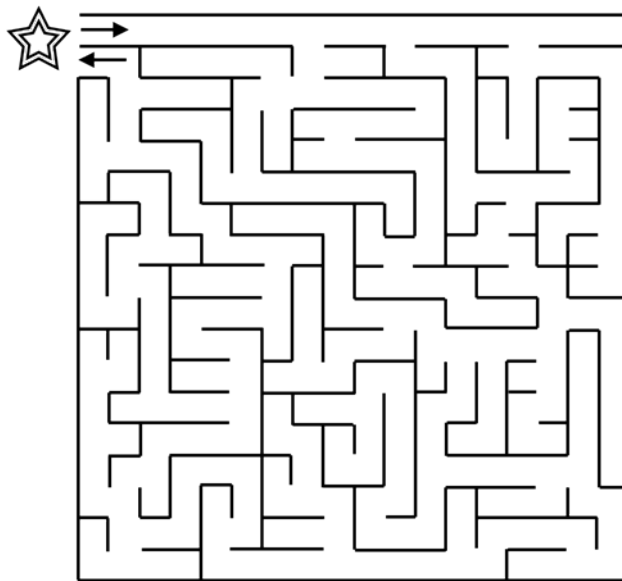
06/ Finding the Shortest Path in a Maze

07/ Additional Exercises

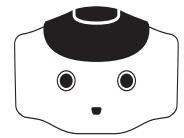
08/ Module Questions

SOLVING MAZES

Many of us have had experience solving mazes on paper. The figure below, taken from the Wikimedia Commons, is one such maze. There is a defined start and goal, and we want to find a path from the start to the end. However, solving a maze on paper is somewhat different from solving a maze while you are in it. If you have walked in a hedge maze, you should know what it is like. For one thing, there is no map of the entire maze, so you don't know beforehand if taking a left or right turn is the correct thing to do. Furthermore, it is difficult to keep track of where you have been in the maze, which can cause you to go around in circles.

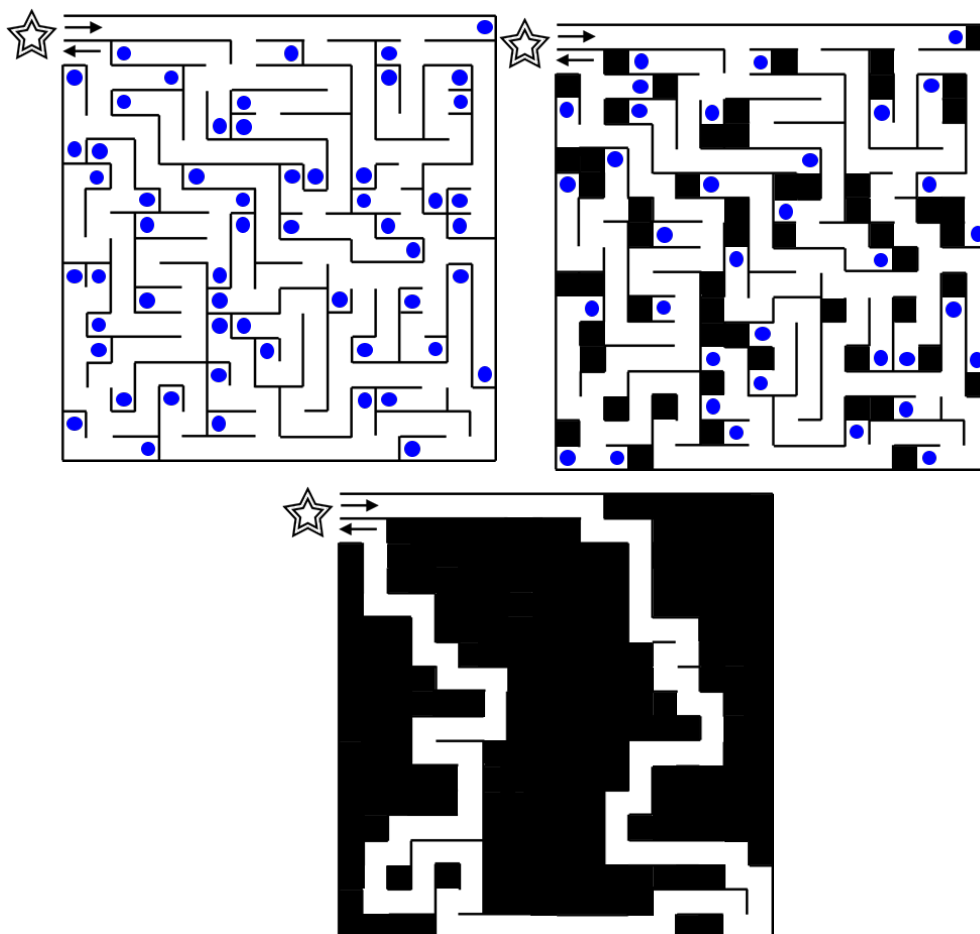


There are many algorithms to help solve mazes. There are algorithms for cases where you know the map of the maze (such as when you do it on paper), and when you don't (like when you are walking in a hedge maze). We will explore a number of these algorithms in this module, as we go through the exercises.



Dead-End Filling

One algorithm to solve a maze when you have the map is called dead-end filling. Essentially, the idea is to find dead ends in the maze, and fill it up. This process repeats until no more dead ends exist. The remaining squares then form the solution from the start to the goal. In the figure to the left, the dead ends of the maze are shown with dots, and on the right, those dead ends have been filled, creating new dead ends which are shown with new dots. When the process of filling all the dead ends are completed, the resulting maze is shown at the bottom, which corresponds to the solution path.



BASIC TASK

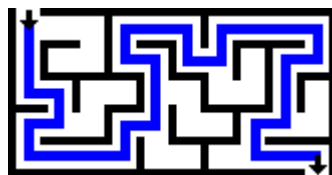
MAZE SOLVING WITH VISUAL CUES

IN THIS EXERCISE, THE NAO WILL BE PLACED IN A MAZE, AND THE GOAL IS FOR THE NAO TO WALK FROM THE START TO THE END.

To do so, we can make use of the human brain to solve the maze. We will then use visual cues to help the NAO navigate the maze. Cues are objects or events that provide information and instruction for the NAO. Visual cues are cues that are detected by vision, such as a NAOMark.

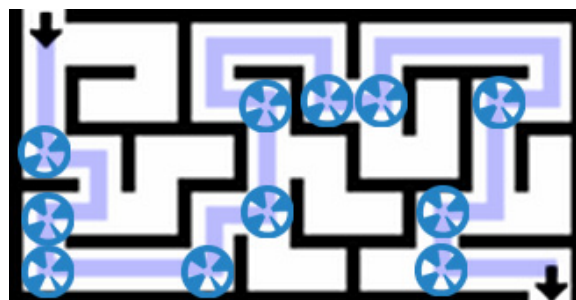
01/

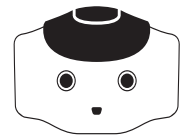
With the map of the maze, find a path from the start position to the goal. The figure below shows a maze, and the path. Do this on the maze map that you have been provided with. You can use the dead-end filling algorithm described above.



02/

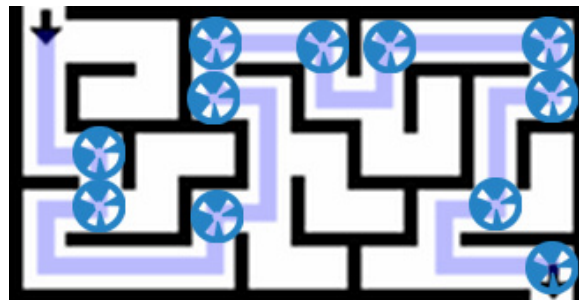
In addition to the map of the maze, you should have been given a number of NAOMarks. You can place these visual cues in the maze to instruct the NAO what to do. For example, In the figure below, NAOMarks are placed at every junction that the NAO should make a left turn.





03/

Similarly, you can use a different set of NAOMarks to instruct the NAO to make right turns, as



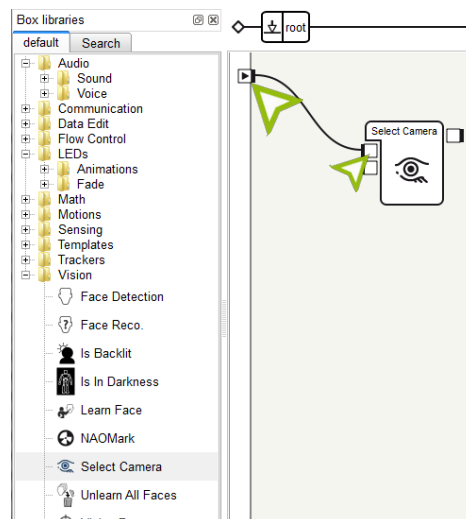
shown below.

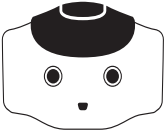
04/

With both sets of NAOMarks in place, we can now program the NAO to complete the maze. Essentially, the NAO should walk forward if it doesn't see a NAOMark. When a NAOMark is detected, it should turn 90 degrees to the left or to the right, depending on the identification number of the NAOMark.

05/

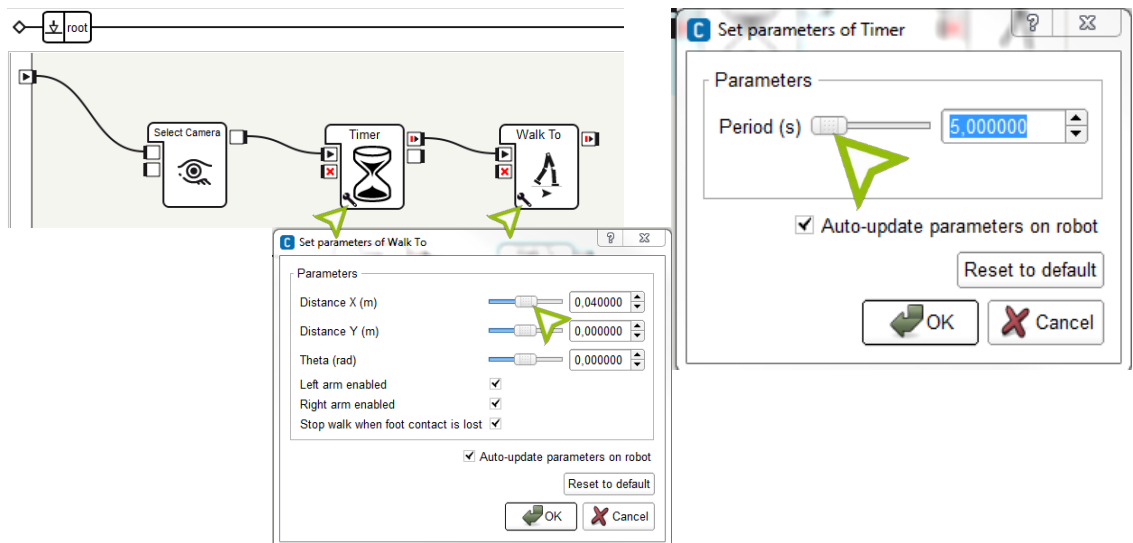
However, to see the NAOMark and make a turn, the NAO should make use of its bottom camera, instead of the top camera. To do so, use a Select Camera box from the Vision category, and connect the bottom-left input to select the bottom camera.





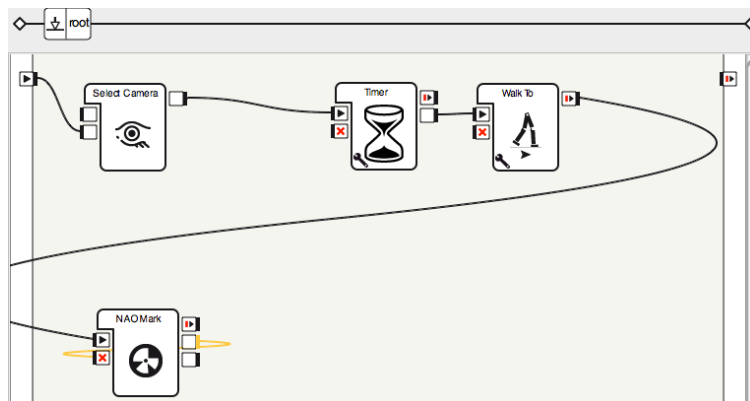
06/

When the NAO has no NAOMarks in sight, it should continue walking forwards. Thus, we can use a Timer box and Walk To box to make the NAO walk forward every few seconds. The Timer box (in the Flow Control → Time category) triggers periodically, and for this exercise, we will set the period to 5 seconds. The Walk To box should be set to a small distance, such as 0.04m forward.



07/

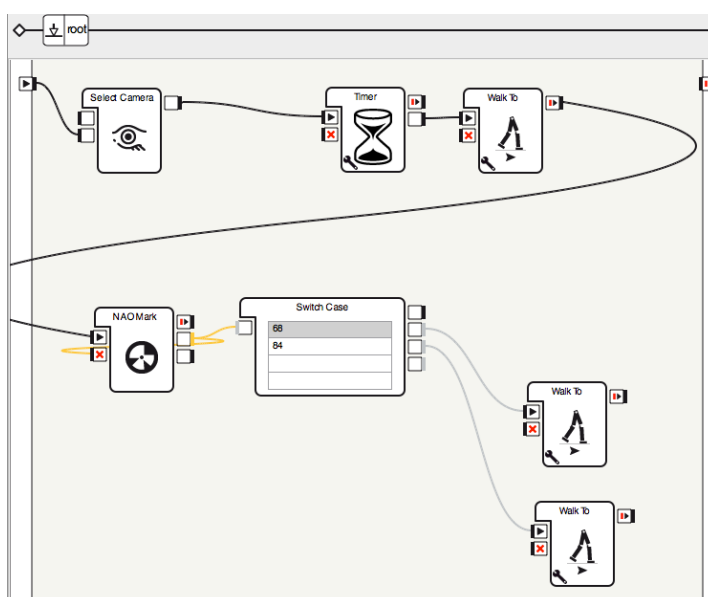
When the Walk To box completes, we want the NAO to perform object recognition, and check if a NAOMark is visible. To do so, we can use a NAOMark box. Notice how the output of the NAOMark box links back to its itself. This ensures that a NAOMark is detected only once if it exists.





08/

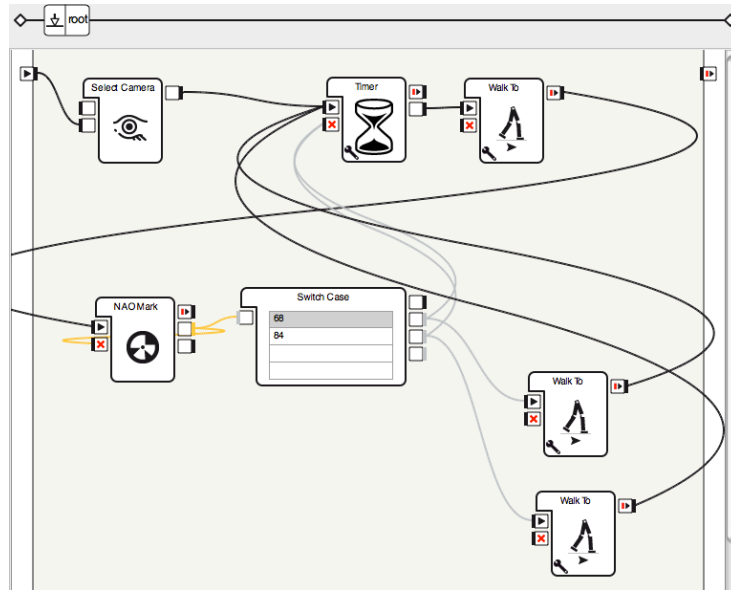
Now, when a NAOmark is detected, we need to check that which NAOmark was detected. If the identification number matches the mark we chose to do a left turn, a Walk To box performs the turn. Similarly, another Walk To box performs the right turn when the expected mark is detected. In the figure below, NAOmark 68 indicates a left turn, and NAOmark 84 indicates a right turn.





09/

There is still the possibility of the Timer box triggering while the robot is performing the turn. To prevent this from happening, we can link the output of the Switch Case box to the x of the Timer, to stop the Timer from triggering. In addition, we connect the output of the Walk To boxes that turn



the NAO to the start of the Timer box, so that the robot resumes walking forward.

10/

Run the behavior. The robot should walk forward every 5 seconds. When it sees NAO Mark 68, it makes a left turn and then continues walking forward, and it makes a right turn and then walks forward when it sees NAO Mark 84.

11/

Now, by placing NAO Marks in the right places, you can give visual cues to the NAO in order to complete the maze!

INTERMEDIATE TASK

MAZE SOLVING WITH MULTIPLE CUES

Besides NAOMarks, object recognition can also be used as visual cues. Together, they provide actions for the NAO. We have previously covered learning object features into the object recognition library on the NAO and using the Object Reco box to control the NAO.

Besides visual cues, you can also make use of the other sensors on the NAO, such as the touch sensors on its head, and the foot bumpers on its feet. For example, touching the head could make the NAO walk forward a short distance, and hitting a foot bumper will make the NAO turn 90 degrees in that direction.

Audio cues can also be used. You can tell the NAO to “turn left”, “turn right” and “walk forward”.

In this exercise, like the exercise above, you will be given the map of the maze. You can solve the maze on paper before working on the NAO. The NAO does not need to solve the maze automatically. Instead, it uses cues to decide what actions to take.

You can provide any sort of cue to the NAO, such as visual, sensory or audio cues. The score of a team will be calculated as follows:

- +100 if the NAO reaches the goal position by navigating through the maze
- +30 for each new type of cue, e.g.,
- NAOMarks
- visual objects
- button presses
- speech recognition
- face recognition
- +1 for every second under 5 minutes

For example, a team that only uses NAOMarks to complete the task in 4 minutes will have a score of $100 + 30 + 60 = 190$. Another team that uses all 5 types of cues, and completes the maze in 4 minutes and 35 seconds will achieve $100 + 5 * 30 + 25 = 280$.

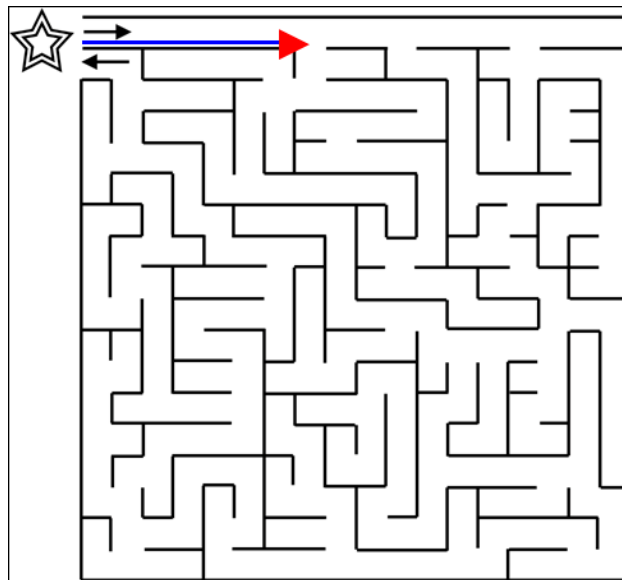
As such, there is an incentive to develop an algorithm that is fast to execute and is less susceptible to errors.

SOLVING A MAZE WITHOUT A MAP

We will now explore an algorithm that can be used when the map of the maze is unknown. For example, a person physically walking in a hedge maze can use it. Similarly, the algorithm can be used by a robot in a maze. The algorithm has a straightforward concept - as you walk in the maze, keep one of your hands, say the right hand, in contact with a wall of the maze at all times. Thus, when there is a turn to the right, in order to maintain contact with the right turn, the algorithm will take the right turn. In this fashion, the algorithm is guaranteed to find the exit provided a condition is met. The condition is that all the walls of the maze are connected together or to the boundary of the maze. This is also known as a *simply connected* maze.

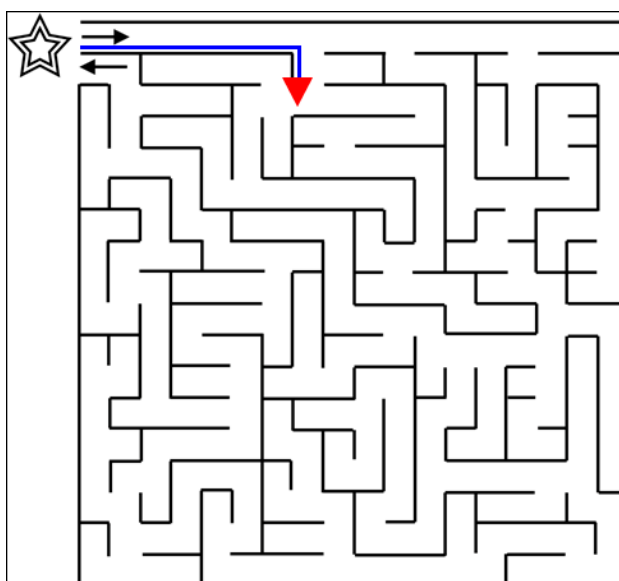
In the figures below, the algorithm will be performing the *right-hand-rule*, i.e., keeping the right side in contact with a wall. The parts of the walls that have been in contact are highlighted in blue.

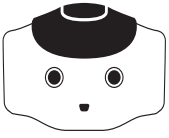
In this first figure, the algorithm follows the right wall, up until the first junction.



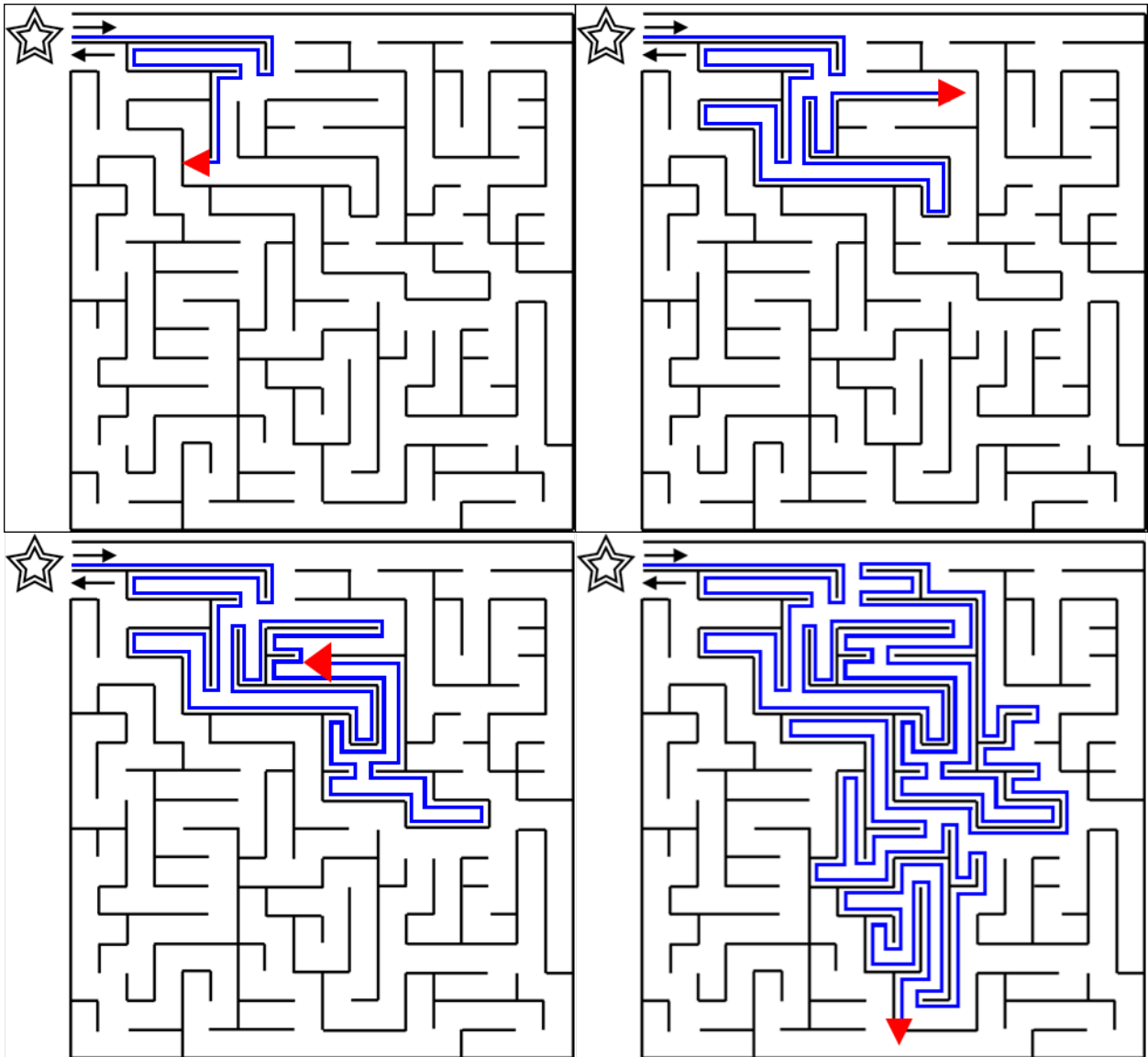


Since the right wall makes a turn, the algorithm will continue following the right wall and so it turns right. After making the first right turn, the algorithm follows the wall again to reach a new junction. Here, the algorithm continues to follow the wall, and go around the junction.



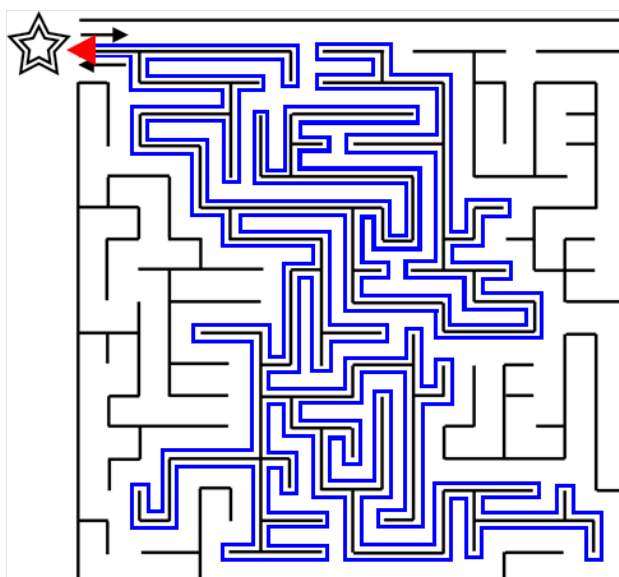


The algorithm continues on and follows the right wall whenever it makes a turn. The following four figures show the state of the algorithm as it keeps going.





Eventually, the algorithm reaches the goal position and completes.



As you can see from the figure above, the algorithm takes a very long path from the start to the goal. However, while there are other algorithms that work when the map of the maze is not available, the wall-following algorithm is one of the easiest to implement.

ADVANCED TASK

THE RIGHT WAY TO MAZE SOLVING

In the previous two tasks, you were given the map of the maze, and gave cues for the NAO to walk from the start position to the goal. In this exercise, you will not be given the map of the maze. Instead, the NAO has to explore the maze and find its way to the goal automatically.

We will be using the Wall Following algorithm in order to do this, and we will use the right-hand-rule. The algorithm can be performed as follows:

- 01.** If there is no wall to the right, turn 90 degrees to the right and then walk forward.
- 02.** If there is a wall to the right, but no wall in front, then walk forward.
- 03.** If there is a wall to the right, and a wall in front, then turn 90 degrees to the left.

In order for the algorithm to function, the NAO needs to know the *state* of the environment around it, i.e., whether there are walls to its right and/or in front of it. The state can be detected through a variety of methods, such as visual cues, audio, and sensors. We will leave it to you to decide what sort of cues you want to use for this module.

In the instructions below, we will be using the head touch sensor to inform the NAO about the state of the environment. Tapping the front head button indicates that there is a wall in front of it, and tapping the rear head button indicates that there is a wall to the right. Tapping the middle button means that the input is done, and the NAO should execute the next action.

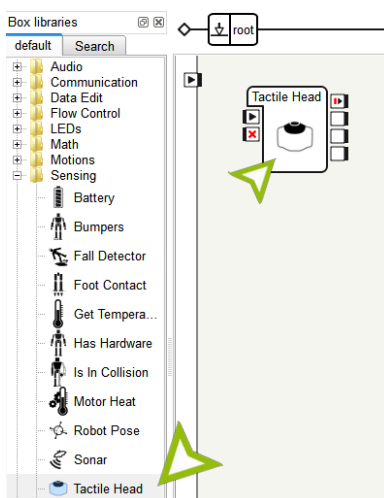
Thus, if we only tapped the middle button, then there are no walls on the right and front of the NAO. If we tapped the top then middle button, then there is a wall to the front, but none on the right. If we tapped the rear then middle button, there is a wall to the right, but not the front. And lastly, if we tapped the front, rear, and then the middle button, there are walls to the front and the right.

Once the state of the environment is known, the NAO can then execute the correct action.



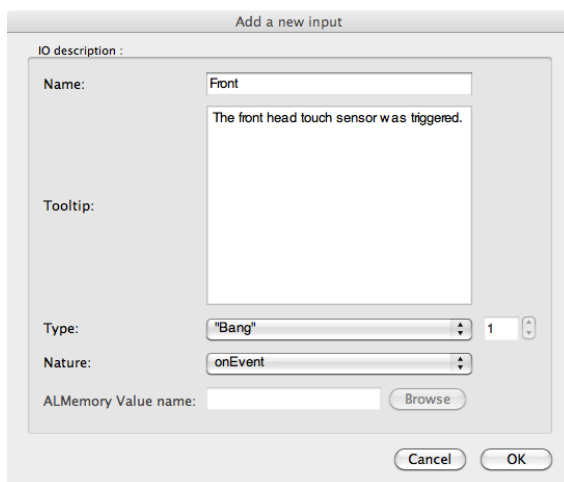
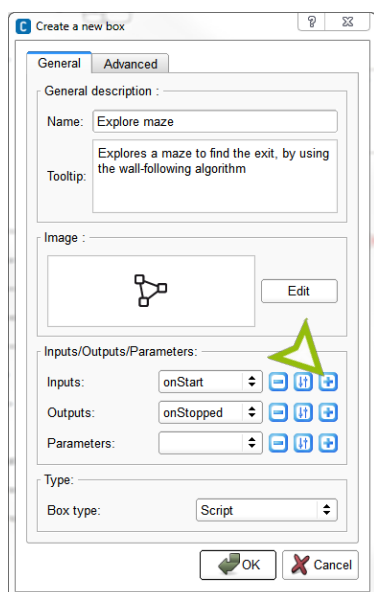
01/

First, we will use the Tactile Head box from the sensing category, to detect touches on the head. Recall that the 3 outputs to the right of the box trigger when the front, middle, and rear touch sensors on the head are touched respectively.



02/

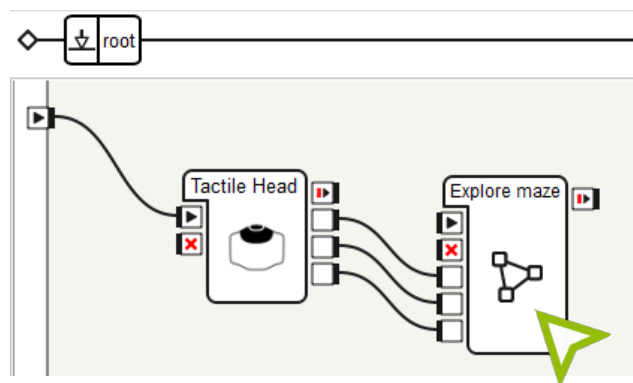
Now, create a custom Python box, and add 3 inputs of type “bang”.





03/

Connect each of the outputs of the Tactile Head box to the inputs of the custom Python box. Then, double-click the Python box to edit the code.



04/

First, in the `onLoad` function, we initialize a few variables. We create a proxy to `ALMotion`, and 2 boolean variables, `front` and `rear`, to store whether the front and rear head touch sensors were triggered. Also, `forwardDist` stores the distance in meters that the NAO should walk to move from one node in the maze to another.

```
class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        self.motionProxy = ALProxy("ALMotion")
        self.front = False
        self.rear = False
        self.forwardDist = 0.1 # How far is each cell to walk across in m?
```



05/ Next, edit the 3 functions that are triggered by the Python box's inputs, `onInput_Front`, `onInput_Middle` and `onInput_Rear`. The code for `onInput_Front` and `onInput_Rear` set the relevant boolean variables to `True`, to indicate that a wall is present in front and to the right of the NAO respectively. When the middle button is triggered, the algorithm computes the next action for the NAO and executes it. Then, the variables storing the states of the walls are reset to **False**.

```
def onInput_Front(self):
    self.front = True

def onInput_Middle(self):
    self.computeNextAction()
    self.front = False
    self.rear = False

def onInput_Rear(self):
    self.rear = True
```

06/

Finally, the function `computeNextAction` needs to be filled in. In this function, the different cases of the walls are checked, and the relevant action is taken.

```
def computeNextAction(self):
    if not self.rear: # no wall to the right
        self.motionProxy.walkTo(0, 0, -1.57) # Turn to the right
        self.motionProxy.walkTo(self.forwardDist, 0, 0) # Walk forward
    elif self.rear and not self.front: # Wall to the right, no wall in front
        self.motionProxy.walkTo(self.forwardDist, 0, 0) # Walk forward
    else: # Wall to the right, wall to the front
        self.motionProxy.walkTo(0, 0, 1.57) # Turn to the left
```

07/

Run the behavior without putting the NAO in the maze. Try pressing the head buttons and check that the NAO performs the correct action.

08/

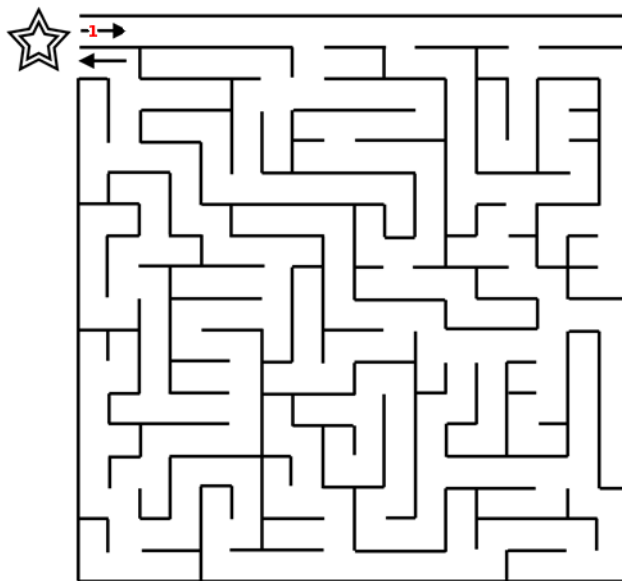
Now, place the NAO in the maze. Touch the head sensors to indicate whether there is a wall to the front and/or right of the NAO, and ensure that the NAO successfully executes the wall-following algorithm and reaches the goal position.

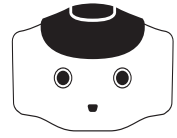
FINDING THE SHORTEST PATH IN A MAZE

Another algorithm to solve a known maze is called breadth-first search. The search algorithm visits cells in a queue data structure (discussed in an earlier module), and adds neighboring cells into the queue as it goes along. By exploring the maze in this fashion, the algorithm is guaranteed to find the shortest path from the start to the goal.

The algorithm can be described with the following steps:

- 01.** Add the starting position into the queue, and give it a value 1.
- 02.** Remove the front node from the queue, and save its value as i . $i = \text{iteration index}$
- 03.** If the node is the goal position, go to step 7.
- 04.** Otherwise, examine all the neighbors of that position. If the neighbor hasn't been visited before, add it into the queue with value $i + 1$.
- 05.** If the queue is empty, there isn't a path from the start to the goal.
- 06.** Otherwise, go to step 2.
- 07.** The goal position has been found, and has value i .
- 08.** Look at the neighbors of the node and pick one that has value $i - 1$.
- 09.** Repeat step 8, reducing i by 1 in each step, until the starting position is reached.
- 10.** The order of nodes corresponds to the reverse of the path from the start to the goal.



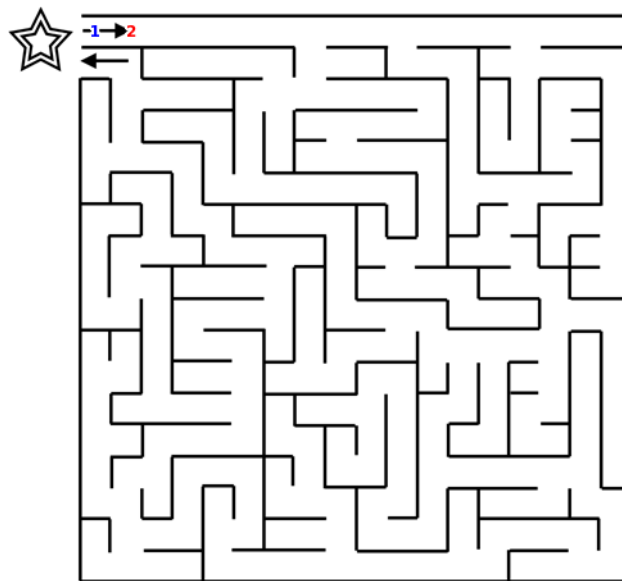


The figures below illustrate the steps in the algorithm, and the numbers over each node correspond to the value given to that node in the queue. We will describe how the algorithm proceeds using the figures. Numbers in red indicate nodes that are in the queue, and numbers in blue indicate nodes that were visited and are no longer in the queue.

Initially, the only node in the queue is the start position, and it has value 1. (Step 1)

The front node of the queue is removed (node 1), and its neighbors are examined. It has only 1 neighbor, and it is given a value 2, and placed in the queue. (Steps 2 to 6)

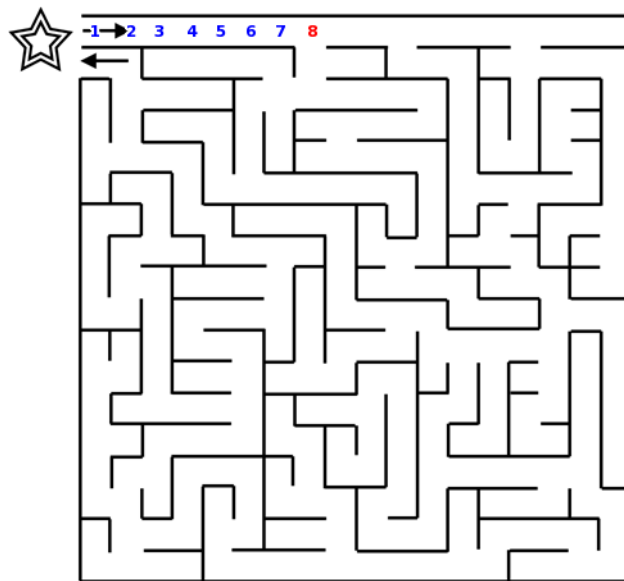
Note: Node 1 is no longer in the queue, and it is shown in blue in the figure below.

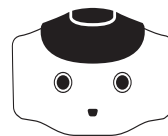




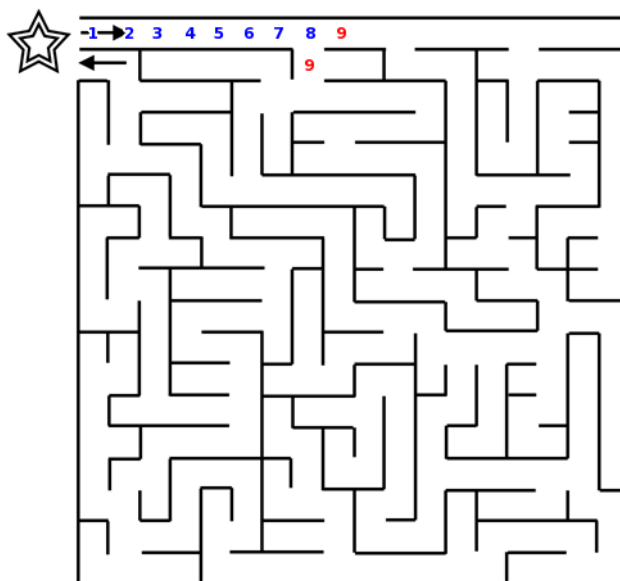
The process is repeated and more nodes are added into the queue. In the figure below, a node with value 8 has just been added to the queue. (Steps 2 to 6)

Note: Nodes with values 1-7 are no longer in the queue, but they are shown in the figure below.



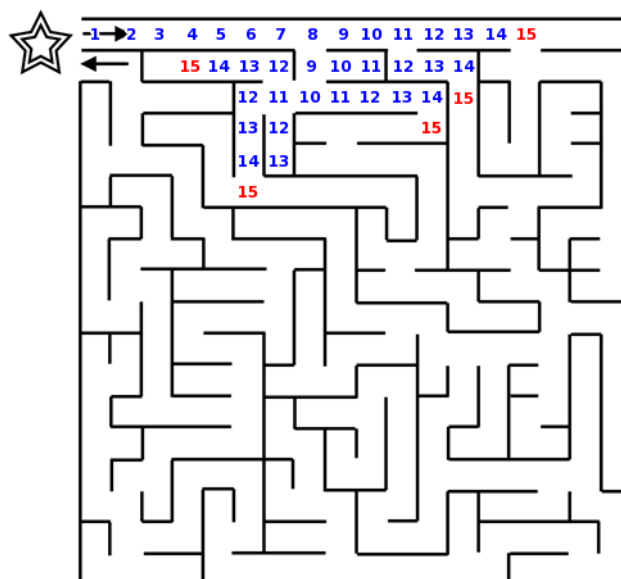


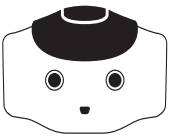
Node 8 has two neighbors that have not been visited, so both of them get value 9 and are placed in the queue. (Steps 2 to 6)



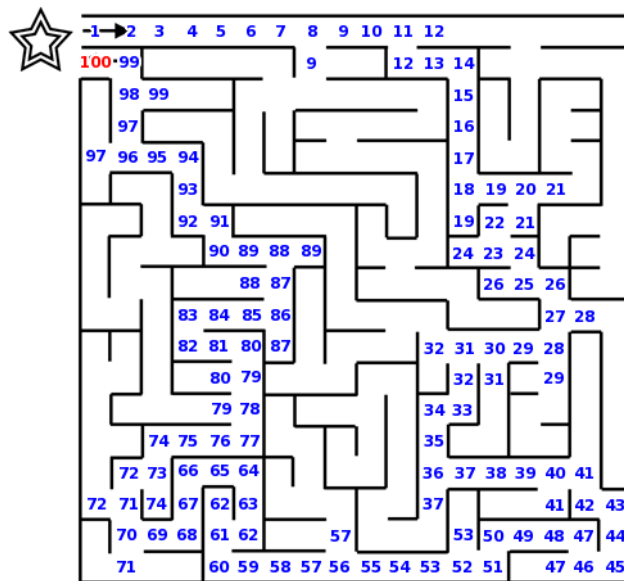


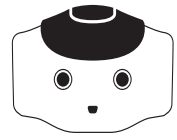
This process continues on as nodes are removed from the queue, and their neighbors are given higher values. The figure below shows the values of visited nodes up to 14, and the nodes in the queue with value 15. (Steps 2 to 6)





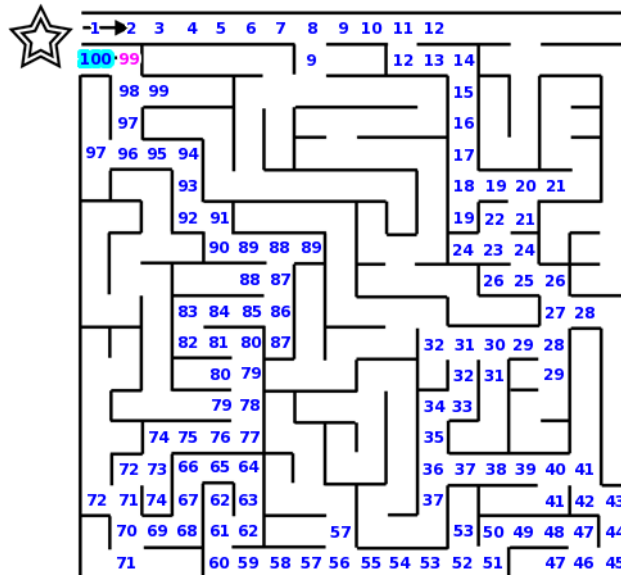
Eventually, the goal position receives a value of 100, and the algorithm jumps from step 3 to step 7. The figure below shows the values of the nodes on the path to the goal and their immediate neighbors. (Steps 2 to 7)





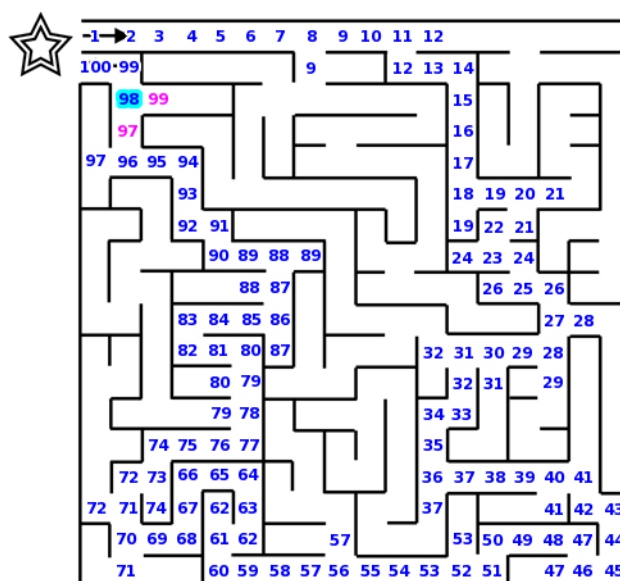
Now that the goal position has been reached, the path from the start position to the goal has to be found. To do so, we start from the goal position and store its value as i . In this case $i = 100$. From that state, we look at its immediate neighbors to find one with value $i - 1$, i.e., 99. There is only one neighbor with that value, so that node gets added to the final path. (Step 8)

In the figure below, the currently active node is highlighted in light blue, and its neighbors are shown in purple.





The algorithm is now at the node with value 98, and so $i = 98$. The node has two neighbors that have not been considered, one with value 97 and one with value 99. Thus, the algorithm picks the node with value 97, i.e., $i - 1$. If there were more nodes with value 97, then it would not matter which node of value 97 the algorithm picked. (Step 8)



The algorithm continues in this fashion until the starting node is reached (value 1). The nodes that were selected by the algorithm along the way then form the path from the starting position to the goal. (Steps 9 to 10)

ADDITIONAL EXERCISES

- 01/** Complete the advanced task, without using Python.
Hint: you can use a finite state machine to keep track of which buttons have been pressed.
- 02/** Change the algorithm of the advanced task to do left-hand-following instead of right-hand-following.
- 03/** Implement the breadth-first search algorithm in Python.
Hint: use the starter code in “Lesson 10 - Exercise - BFS starter code.crg”.

MODULE QUESTIONS

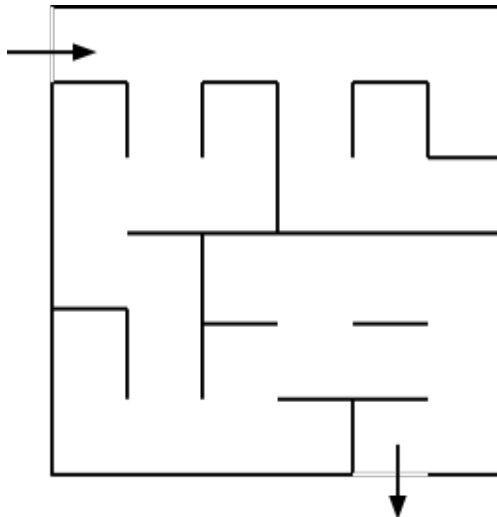
BASIC

01/ Describe the dead-end filling algorithm in your own words.

02/ Why does the dead-end algorithm work?

INTERMEDIATE

03/ Perform dead-end filling on the maze below.



04/ After all the dead-ends have been filled in the maze above, there isn't a single direct path to the goal. What do the non-filled squares represent?

ADVANCED

05/ Describe the wall-following algorithm in your own words.

In what cases does the wall-following algorithm fail? Why?

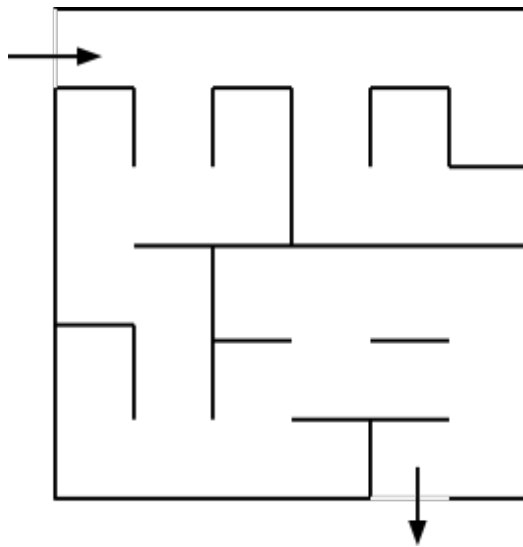
06/

Create a maze where the right-hand-rule will find a much shorter path than the left-hand-rule.

Does the wall-following algorithm find the shortest path from the start position to the goal?

09/

Perform the breadth-first search algorithm by hand on the graph below, filling in the values for each node, and then finding a path from the starting position to the goal.



10/

Does the breadth-first search algorithm always find the shortest path?

MODULE

QUESTION

SOLUTIONS

1

HELLO WORLD

BASIC:

01/ How can you tell if the robot is charging?

The light on the charger should be red.

02/ How should you place the robot when turning it on?

Sitting upright with its feet flat on the floor.

03/ How should the robot be held?

With both hands around its chest.

04/ How is the order the Choregraphe boxes execute determined?

They execute in the order they are connected with arrows.

INTERMEDIATE:

05/ What does the voice shaping parameter for the Say box control?

The tone and pitch of the voice.

06/ What does the speed parameter for the Say box control?

How slow or fast the robot speaks.

ADVANCED:

07/ Name two programming languages that can be used on the NAO.

C++ and Python.

08/ How can the NAO be programmed in Python using Choregraphe?

Right-click to create a new box, set the name, description and picture, and click ok. Double-click on the box to edit the python code.

09/ What is a variable?

A symbolic name assigned to a value which can be modified.



10/ What is a function or method in programming?

A section of code that performs an operation, and can be called by name elsewhere in the code.

11/ What is a function argument?

An input to a function, which can be used within.

12/ Do the number of spaces and tabs matter in python?

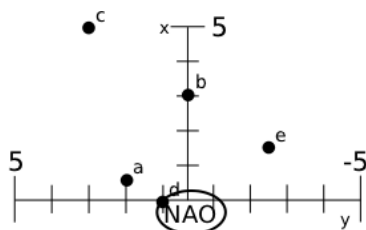
Yes, the spacing at the beginning of a line denotes scope.

2 WALK IT OUT

THE COORDINATE PLANE

01/ Sketch the coordinate plane of the NAO, and plot and label the following points:

- The point (0.5, 2).
- The point 3 meters directly in front of the robot?
- The point (5, 3)?
- The point 1 meter to the left of the robot?
- The point at an angle of -60° from the robot and 3 m away.



02/ Compute the angle from the robot to each of the above five points.

- a) 75.96 degrees, b) 0 degrees, c) 30.96 degrees, d) 90 degrees, e) -60 degrees

BASIC:

03/ What are the three parameters in the Walk To box?

The amounts in the x , y and theta directions to walk. x is forward, y is side to side, and theta is rotation.

04/ What unit of measure is used for x and y coordinates?

Meters.

05/ What is theta?

The amount the robot should rotate in degrees. Positive angles are counterclockwise.

06/ What unit of measure is used for theta?

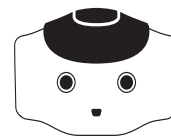
Radians.

07/ For the NAO, what does “stiffness” mean?

How much resistance should be provided by the motors.

08/ What problems do you face if stiffness is always set to 100%?

High stiffness consumes battery power more quickly and heats up the motors.



INTERMEDIATE:

09/ What is odometry?

The robot's model of how it has moved.

Why do odometry errors occur?

10/ Effects of the real world: imperfectly modeled motors, imperfect assembly of the robot, unexpected floor conditions (the floor surface matters a great deal), and outside forces acting on the robot.

Define the Cartesian plane, and explain how it relates to the NAO walking.

Points on a plane are specified using coordinates along two perpendicular axes. The coordinates for NAO to walk to are specified on a Cartesian plane.

11/

ADVANCED:

12/ What is a parameter?

An input to a function.

13/ Name the parameters used to make NAO walk.

The x, y and theta coordinates of its destination.

14/ Name three math functions used in programming walking for robot.

Three of multiplication, division, `sqrt`, `atan2`.

15/ What do `atan2(1.0, 0.0)`, `atan2(0.0, -1.0)`, and `atan2(sqrt(3)/2, 1/2)` evaluate to?
 $\pi/2$, π , and $\pi/3$.

16/ Why is `atan2` used instead of a single-argument arctan function?

The arctan function is undefined at $x = 0$. Also arctan returns values in $[-\pi/2, \pi/2]$, while `atan2` covers the entire circle.

3

HEARING THINGS

BASIC:

01/ What hardware devices does the robot use for speech recognition?

Microphones in its head.

02/ What is a conditional statement?

Depending on the value of a true / false condition, a segment of code is or is not executed.

03/ What is the threshold for speech recognition with NAO? What value did you find to work well?

The threshold is how confident the NAO must be to claim it recognized speech. Lower values are more tolerant.

04/ Where is the Speech Recognition box found?

In the Audio → Voice section of the box list.

05/ Explain the three parameters of the Speech Recognition box.

The word list is the set of words the NAO attempts to listen for. The threshold is how confident the NAO must be to decide it recognized speech. The Visual Expression checkbox enables setting the lights to indicate it is listening or heard a word.

06/ Explain the two outputs of the Speech Recognition box.

The top output is triggered when a word is recognized with that word. The bottom output is triggered if speech is heard but no word is recognized from the word list.

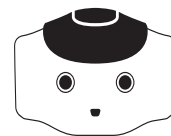
INTERMEDIATE

07/ Which conditional box is used to control program flow from multiple answers?

The Switch Case box.

08/ What punctuation is used to distinguish strings, in both Choregraphe and Python?

Quotes, double “ or single ‘.



ADVANCED

09/ What module is utilized in TextToSpeech (the parameter to ALProxy)?

ALTextToSpeech.

10/ Define string concatenation.

Appending two strings together. For example, "Hello " + "NAO." gives "Hello NAO."

11/ What are the differences between inputs and parameters of Choregraphe boxes?

Parameters can be set by clicking on the wrench, inputs must be sent from another box. Additionally, parameters are values which can be set in advance

12/ What is the syntax in Python for a chain of conditional statements?

```
if condition_1:  
...  
elif condition_2:  
...  
else:  
...
```

4 LET'S DANCE

BASIC:

01/ What is a keyframe?

A set of fixed positions the robot moves its joints to.

02/ What does the NAO do in between keyframes?

It interpolates between the joint angles of adjacent keyframes.

03/ Where is the NAO's center of gravity?

In its torso.

04/ Where must the center of gravity be for the NAO to remain balanced?

Directly above the robot's feet (or foot) which are touching the floor.

05/ When creating a box in Choregraphe, how do you make it a box where keyframes can be edited?

Change the "Box offspring" to "Timeline".

06/ Why is it not suggested to set a keyframe in the first frame of the timeline?

The NAO will jump immediately to that position, and is likely to fall due to the rapid motion.

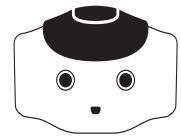
INTERMEDIATE:

07/ What types of motions will cause the NAO to fall?

Rapid motions which give it momentum, or motions which shift the center of gravity away from above the feet.

08/ Explain the behavior of a For box in Choregraphe.

It executes the loop connected to the box multiple times.



09/ A For loop box has an initial value of 2, a final value of 10, and a step value of 2. How many times does the loop execute?

5 times. The counter value will be 2, 4, 6, 8 and 10.

ADVANCED:

10/ Which joint is the HeadYaw joint and which is the HeadPitch?

The HeadYaw is for side to side motion, the HeadPitch for up and down motion.

11/ What units are angles passed to the `angleInterpolation` function in?

Radians.

12/ How many radians is 45 degrees?

$\pi/4$

13/ How many degrees is $\pi/3$ radians?

60 degrees

5 SENSE AND ACT

BASIC:

01/ Where are the NAO's two cameras located?

On its chin and forehead.

02/ How many microphones does the NAO have and where are they?

Four. Two in the ears, one on the back of the head and one on the forehead.

03/ What touch sensors does the NAO possess?

Two foot bumpers, three head touch sensors, a chest button, and foot pressure sensors.

INTERMEDIATE:

04/ What are the two main components of a finite state machine?

States and transitions.

05/ Explain when the two outputs of the Bumpers box are triggered.

One is triggered when the left bumper is pressed, the other is triggered when the right bumper is pressed.

ADVANCED:

06/ Explain what the gyroscope and accelerometer measure.

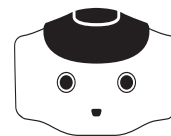
The gyrometer measures angular velocity and the accelerometer measures acceleration (e.g., gravity).

07/ Which three body sections can the NAO's motors be grouped into?

Legs, arms and head/neck.

08/ What do encoders measure?

The position/angle of a joint.



09/ Explain at a high level how the NAO can know the position of its hand relative to its feet.

It knows the position of each joint, and can compute the transformation from this information.

10/ What three primary color components determine the color emitted from an LED?

Red, green and blue.

11/ What is a boolean variable?

A variable that is either true or false.

12/ The finite state machine with buttons toggling two binary variables had four states, and the one with three binary variables had eight states. How many states does an FSM with four binary variables that you can toggle have? Five binary variables? Ten binary variables? N binary variables?

16, 32, 1024, 2^n

13/ True or False: Everything on a digital computer is stored in binary.

True.

14/ How many different colors can be represented in the 24-bit RGB format?

2^{24}

15/ Write the RGB value for the color purple in hexadecimal.

0xFF00FF

16/ Compute 00110101 | 10011010 (binary), 0x7C | 0x5B (hexadecimal), and 118 | 201 (decimal).

10111111, 0x7F, 0x76 | 0xC9 = 0xFF = 255

6 DO THE ROBOT

BASIC:

01/ What does CPU stand for?

Central Processing Unit.

02/ Explain, at a high level, how a single processor performs multi-tasking.

The operating system executes one thread for a tiny slice of time, and then quickly switches to the next one to provide the illusion of true concurrency.

INTERMEDIATE:

03/ Explain the use for behavior layers.

Behavior layers allow multiple behaviors using different body parts to be executed simultaneously in Choregraphe.

04/ Why does standing up straighter make the robot more susceptible to falling?

Because the robot's center of gravity is higher, a smaller tilt will make it leave the stable base region of the feet.

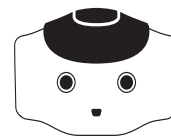
ADVANCED:

05/ Why did we need to use the `simplifyAngle` method when computing the odometry?

To prevent wraparound issues. Let's say that in one frame, the angle returned for the robot's position is $\pi-0,1$, and the next frame the robot rotates 0.2 radians. Then the new angle is $-\pi+0,1$. Without the `simplifyAngle` method, we would calculate that in this frame, the robot turned nearly 2π radians as opposed to the actual rotation of 0.2 radians. This makes quite a difference!

06/ Explain the use of the `setWalkTargetVelocity` function.

This sets a velocity - a direction and speed - for the robot to walk in. The velocity has x, y and rotational components. The robot continues to walk at this velocity until a new command is given.



07/ Why did we need to call `time.sleep` in the main loop?

Otherwise the main loop will needlessly consume all the computation available, which is wasteful and prevents other important tasks from executing (such as the software making the robot walk).

08/ Why do we have to set the walk velocity to zero at the end of the loop?

Otherwise the robot won't stop walking.

09/ Using only division, multiplication, addition, subtraction, and integer truncation, compute the remainder of x / y in python. (This function, called modulus, is already implemented in python as the `%` operator. But do not use the `%` operator.)

`x - int(x / y) * y`

10/ Find the big-0 runtime cost of the following functions:

a. Searching for a name in a telephone book of n pages by:

i. Opening a page you bookmarked previously.

$O(1)$

ii. Beginning at the first page, and checking each page in order for the name.

$O(n)$

iii. Open the middle page, see if the name is to the left or the right, then open the middle of the remaining half of the book, and repeat until the name is found.

$O(\log n)$. This algorithm is called *binary search*.

b. Enumerating every 5-card hand in a deck of n cards.

$O(n^5)$

c. Finding the correct n -bit key to open a lock by trying every possible key (this is called a brute force attack in computer security).

$O(2^n)$

d. Sorting a list of numbers by finding the lowest number of the entire list, placing it first, finding the next lowest number, placing it second, and so on. This is called selection sort.

$O(n^2)$. There are faster sorting algorithms which are $O(n \log n)$.

7 FACE OFF

BASIC:

01/ What does the face detection box output?

The number of faces that were detected.

02/ Speculate as to why the robot does not always detect your face.

Possible responses include: poor algorithms, changing lighting conditions, variations in angle and position.

INTERMEDIATE:

03/ What is the difference between face detection and recognition?

Detection is realizing that we see a face: recognition is knowing *whose* face we see.

04/ What does the face recognition box output?

The name of the face that was detected.

ADVANCED:

05/ What areas will the search not see? How could you expand the robot's search, using both the head and by walking?

It won't see anything behind the robot or above or below the search plane. The search could be expanded by also changing the HeadPitch angle, and/or by turning around.

8 OBJECT RECOGNITION

BASIC:

01/ What is a pixel?

A pixel is a picture element, and stores a small single-colored section of an image or computer screen.

02/ What does a NAOMark box in Choregraphe do?

The NAOMark box uses computer vision to detect specially-designed NAOMarks. When a NAOMark is in the camera view, the NAOMark box triggers and returns the identification number of that NAOMark.

INTERMEDIATE:

03/ How does object recognition work?

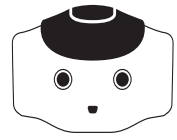
Object recognition works by comparing the features detected by computer vision, against a library of known objects and their features.

04/ Is color a good feature to be used in computer vision? Why or why not?

In general, color is not a good feature for computer vision, because the color detected by the camera depends on the color of the object, and the color of the light. For example, a red piece of paper looks red under white light, and a white piece of paper looks red under red light.

05/ When learning a complex object, why do we need to provide many different images of the object?

A complex object looks different from different perspectives, and so it is important to provide the features of these perspectives to the vision algorithm when creating the object database, so that it can recognize the object from any perspective.



ADVANCED:

06/ How do we use Python to determine if a string starts with a prefix, e.g., how do we check that a variable called `myst` starts with “Abc”?

```
myst.startswith("Abc")
```

07/ In the Advanced Task, why does the robot stop walking once it reaches the object?

When the object is very close to the robot, the object is no longer in the NAO’s camera’s field-of-view, i.e., the camera does not see the object. Thus, the vision algorithm does not detect any object and so the robot stops walking.

9 GAMES AND STORIES

01/ Describe human-robot interaction.

The field studying how humans and robots interact, and how robots can be made both easier and more pleasing to use for humans.

02/ Give an example of a robot that interacts with humans.

A car that parallel parks itself (or other examples).

03/ Give a reason why humans find the NAO an appealing robot.

It is shaped like a human and looks cute.

04/ Discuss how multiple robots would need to cooperate to enact a play together.

They would communicate wirelessly to remain in sync, executing the same parts of the script in unison.

10 FINDING YOUR WAY

BASIC:

01/ Describe the dead-end filling algorithm in your own words.

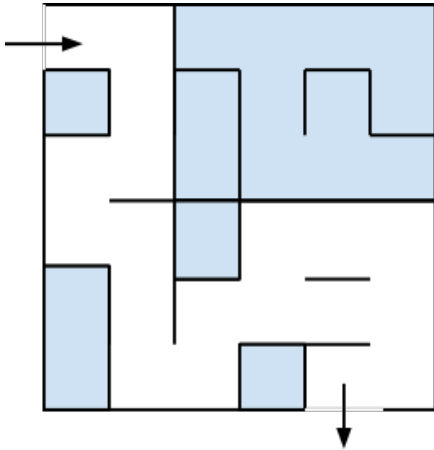
The dead-end algorithm repeatedly fills in dead ends in the maze. When all dead ends have been filled, the path to the goal is shown.

02/ Why does the dead-end algorithm work?

The path to the goal cannot have any dead ends, so filling in dead ends will never cover the path.

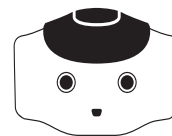
INTERMEDIATE:

03/ Perform dead-end filling on the maze below.



04/ After all the dead-ends have been filled in the maze above, there isn't a single direct path to the goal. What do the non-filled squares represent?

The remaining squares represent all possible paths to the goal. Multiple paths to the goal can be achieved by branching on the non-filled squares.



ADVANCED:

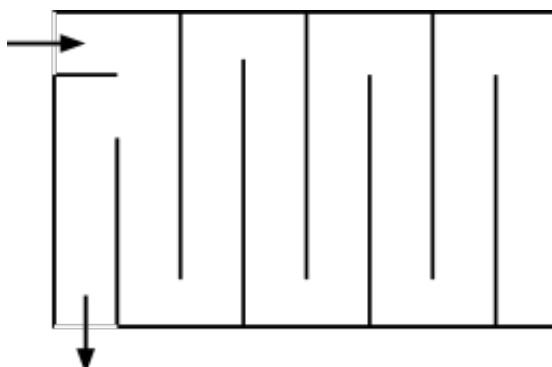
05/ Describe the wall-following algorithm in your own words.

The wall-following algorithm follows one wall (either on the left or right) and moves around corners until the goal is found.

06/ In what cases does the wall-following algorithm fail? Why?

The wall-following algorithm fails if walls are not connected together. Since the algorithm follows walls, it cannot reach a goal that is not connected by walls.

07/ Create a maze where the right-hand-rule will find a much shorter path than the left-hand-

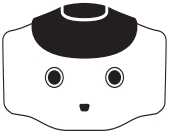


rule.

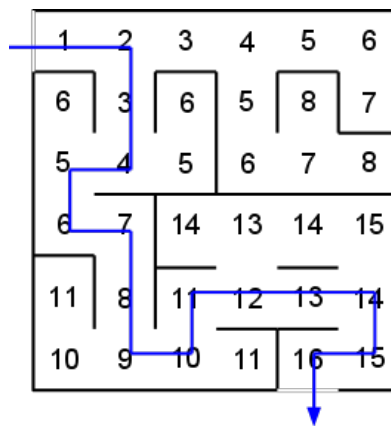
08/

Does the wall-following algorithm find the shortest path from the start position to the goal?

In general, no.



09/ Perform the breadth-first search algorithm by hand on the graph below, filling in the values for each node, and then finding a path from the starting position to the goal.



10/ Does the breadth-first search algorithm always find the shortest path?

Yes.

ANNEX

ACADEMIC

LESSON PLANS

1 HELLO WORLD

LESSON PLAN

hello world

OBJECTIVE

In this module, students will learn:

→ How to switch on the NAO humanoid robot

→ How to connect to the NAO with Choregraphe on a computer

→ How to make the NAO speak with Choregraphe

** RST.9-10.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks, attending to special cases or exceptions defined in the text.*

→ How to vary the pitch and speed of the NAO's voice

** RST.9-10.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks, attending to special cases or exceptions defined in the text.*

→ How to program the NAO to speak with Python

** RST.11-12.3. Follow precisely a complex multistep procedure when carrying out experiments, taking measurements, or performing technical tasks; analyze the specific results based on explanations in the text.*

** RST.11-12.8. Evaluate the hypotheses, data, analysis, and conclusions in a science or technical text, verifying the data when possible and corroborating or challenging conclusions with other sources of information.*

** RST.11-12.9. Synthesize information from a range of sources (e.g., texts, experiments, simulations) into a coherent understanding of a process, phenomenon, or concept, resolving conflicting information when possible.*

** RST.11-12.10. By the end of grade 12, read and comprehend science/technical texts in the grades 11-12 text complexity band independently and proficiently.*

KEY VOCABULARY

Choregraphe, Say Box, Parameter, Variable, Function Method

MATERIALS NEEDED

Choregraphe, NAO robot, Internet Connection (wired or wireless)

ACTIVITY OUTLINE

01/ Students should read the "Hello World Module"

02/ Complete the module questions and review

03/ Prepare a short written algorithm (set of step by step instructions) of what they plan to do once they have access to Choregraphe and NAO

04/ Teacher may want to demonstrate safe methods for handling NAO and discuss protecting the NAO joints from damage.

2 WALK IT OUT

LESSON PLAN

Cartesian Coordinate Plane

OBJECTIVE:

The students will be introduced to the coordinate system through a human Cartesian coordinate plane.

COMMON CORE STATE STANDARDS:

6.NS.6. Understand a rational number as a point on the number line. Extend number line diagrams and coordinate axes familiar from previous grades to represent points on the line and in the plane with negative number coordinates.

- Understand signs of numbers in ordered pairs as indicating locations in quadrants of the coordinate plane; recognize that when two ordered pairs differ only by signs, the locations of the points are related by reflections across one or both axes.

- Find and position integers and other rational numbers on a horizontal or vertical number line diagram; find and position pairs of integers and other rational numbers on a coordinate plane.

6.NS.8. Solve real-world and mathematical problems by graphing points in all four quadrants of the coordinate plane. Include use of coordinates and absolute value to find distances between points with the same first coordinate or the same second coordinate.

KEY VOCABULARY

Abscissa, axes, coordinate, coordinate plane, ordinate, ordered pair, origin, quadrant, x-axis, y-axis

MATERIALS NEEDED

Index cards with coordinates

Masking tape

Example of a coordinate plane (transparency, on chalkboard, LCD projector)

PROCEDURE

01/

Set up classroom as a coordinate plane. Use masking tape to make an x-axis and y-axis. Put desks in rows over the axes, making sure one of the desks is on the origin.

02/

Make a card for each desk with the correct ordered pair.

03/

Class discussion of the Cartesian coordinate system. Many students have already been exposed to the coordinate plane. Use their prior knowledge to create an example of a coordinate plane, and label all parts. As plane is created, students should copy example in their notes, as teacher creates it on the board, transparency, or computer.

04/

Ask students such questions as

- Who is sitting at the origin?
- Who is on the x-axis?
- Who is on the y-axis?
- Who is in Quadrant I?

Students could also be asked to stand if at origin, on x-axis, etc.

-
- 05/** After reviewing all key vocabulary, have students stand along wall and give each an index card. One by one have students find their new location starting at the origin. Students need to walk along x-axis, then y-axis to find seat. Discuss any problems that arise, such as two students at one seat.
- 06/** Students can move back to original seats after everyone is seated.
- 07/** Students then should be called on to give coordinates of their original seat.
- 08/** Discussion for later class: Was there a shorter walk to the new desk? How would you direct someone to the desk without giving coordinates (assuming you could walk through desks)?

ASSIGNMENT/EXTENSION:

- Assign a worksheet or book problems that has students label missing parts of the coordinate plane, plot coordinates, and name coordinates.
- Use graphing software, such as Geometer's Sketchpad, to plot and name coordinates. This could be done as a group activity, or individually on computers.
- Have students visit graphing websites and practice skills or see more examples. Some examples are:

<http://www.purplemath.com/modules/plane.htm>

<http://www.wisc-online.com/Objects/ViewObject.aspx?ID=ABM201>

<http://mathforum.org/cgraph/cplane/pexample.html>

5 SENSE AND ACT

LESSON PLAN

Trigonometric Functions of an Angle - Extension

OBJECTIVE:

The students will be able to find the x-coordinate, y-coordinate of a point, given (r_1, θ_1) and (r_2, θ_2) .

COMMON CORE STATE STANDARDS:

6.NS.6. Understand a rational number as a point on the number line. Extend number line diagrams and coordinate axes familiar from previous grades to represent points on the line and in the plane with negative number coordinates.

- Understand signs of numbers in ordered pairs as indicating locations in quadrants of the coordinate plane; recognize that when two ordered pairs differ only by signs, the locations of the points are related by reflections across one or both axes.

- Find and position integers and other rational numbers on a horizontal or vertical number line diagram; find and position pairs of integers and other rational numbers on a coordinate plane.

6.NS.8. Solve real-world and mathematical problems by graphing points in all four quadrants of the coordinate plane. Include use of coordinates and absolute value to find distances between points with the same first coordinate or the same second coordinate.

6.G.3. Draw polygons in the coordinate plane given coordinates for the vertices; use coordinates to find the length of a side joining points with the same first coordinate or the same second coordinate. Apply these

techniques in the context of solving real-world and mathematical problems.

8.G.8. Apply the Pythagorean Theorem to find the distance between two points in a coordinate system.

G-SRT.6. Understand that by similarity, side ratios in right triangles are properties of the angles in the triangle, leading to definitions of trigonometric ratios for acute angles.

G-SRT.8. Use trigonometric ratios and the Pythagorean Theorem to solve right triangles in applied problems.*

MATERIALS NEEDED

Cartesian coordinate system on transparency, chalkboard, or LCD projector

Scientific calculator

Graph paper

Centimeter rulers

PROCEDURE:

09/

Review the Sine, Cosine, and Tangent functions with students.

10/

Give students a point A (10 cm, 30°). Have students draw picture to scale, labeling r and θ , and x and y . Have students measure and record x and y . Students should check measurements by calculating x and y using trigonometric functions. Discuss any differences. Repeat this with B (5 cm, 60°) and C (12 cm, 45°), each point being plotted on new graphs.



11/

Now have students plot point A, and then from A go the distance and angle of point B. (This is essentially adding vectors tail to tail, but we are looking for coordinates of B, not the resultant vector. This can later be discussed when teaching vectors). Have students come up with a real world example to illustrate what is happening. An example would be: I started hiking from my campsite at a 30° for 10 miles. I then turned 60° and continued walking 5 more miles to my friend's campsite. How many miles would I have walked to my friend's camp, had I gone due east and then north from my campsite?

12/

Have students share their examples or come up with a class example.

13/

Students need to calculate missing distances by creating two right triangles. The first right triangle is made by dropping a perpendicular from point A to the x -axis. Students can find x_A value by using the Cosine function, and y_A value by using the Sine function. The second right triangle is made by dropping a perpendicular to the horizontal line containing point B (students may have to find a different angle than the given one to calculate distances). Students can find x_B value by using the Cosine function, and y_B value by using the Sine function. To find the location of point B from the origin, add x and y values together ($x_A + x_B, y_A + y_B$). Students should realize the x -coordinate is how many miles I hiked due east and the y -coordinate is how many miles I walked north. Students can check answers with their scale drawing and compare results.

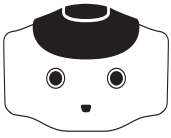
14/

Class discussion:

- What was the total distance hiked in the initial problem?
- What would the total distance hiked have been, going due east and then north?
- Which was the longer walk?
- How easy is it to walk a straight line distance in the woods? How are streets laid out in a town, at angles, or perpendicular and parallel?

ASSIGNMENT/EXTENSION:

Have students repeat process with points B and C. Students should also create a real world example to go with it.



LESSON PLAN

Trigonometric Functions of an Angle

OBJECTIVE:

The students will be able to locate a point in a coordinate plane by distance (r) and angle (θ) given its coordinates (x, y) .

COMMON CORE STATE STANDARDS:

6.NS.6. Understand a rational number as a point on the number line. Extend number line diagrams and coordinate axes familiar from previous grades to represent points on the line and in the plane with negative number coordinates.

- Understand signs of numbers in ordered pairs as indicating locations in quadrants of the coordinate plane; recognize that when two ordered pairs differ only by signs, the locations of the points are related by reflections across one or both axes.

- Find and position integers and other rational numbers on a horizontal or vertical number line diagram; find and position pairs of integers and other rational numbers on a coordinate plane.

6.NS.8. Solve real-world and mathematical problems by graphing points in all four quadrants of the coordinate plane. Include use of coordinates and absolute value to find distances between points with the same first coordinate or the same second coordinate.

6.G.3. Draw polygons in the coordinate plane given coordinates for the vertices; use coordinates to find the length of a side joining points with the same first coordinate or the same second coordinate. Apply these techniques in the context of solving real-world and mathematical problems.

8.G.8. Apply the Pythagorean Theorem to find the distance between two points in a coordinate system.

G-SRT.6. Understand that by similarity, side ratios in right triangles are properties of the angles in the triangle, leading to definitions of trigonometric ratios for acute angles.

G-SRT.8. Use trigonometric ratios and the Pythagorean Theorem to solve right triangles in applied problems.*

MATERIALS NEEDED

Cartesian coordinate system on transparency, chalkboard, or LCD projector
Scientific calculator

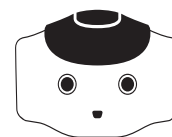
VOCABULARY:

adjacent, angle, Cosine, degrees, distance formula, initial side, opposite, Pythagorean Theorem, radians, Sine, Tangent, terminal side

PROCEDURE:

15/

Plot the point A (3, 4) on coordinate plane. Ask students to find the distance from origin to point A. Class discussion: Based on their previous experience, students may use Pythagorean Theorem or distance formula. Go over both methods, drawing the straight line distance from origin to A and labeling it r . Have students come up with the formula of r to be $r = \sqrt{x^2 + y^2}$. Students should add both methods and formula to notes. Give students a few points from first quadrant to calculate r for.



16/

Refer to previous class discussion after the human Cartesian plane activity, about what else is needed to get to point A, given only r . Students should come up with the need of an angle. Have students label the x-axis as the initial side and y-axis as the terminal side. Students should define both terms in their notebook. Students should label the missing angle θ .

17/

Define Sine, Cosine, and Tangent using x and y . Define again using side opposite, side adjacent, and hypotenuse. Students are to record in notebook.

18/

Review radians with students. Go over procedure for putting calculators in radian mode. Using the previous points that students calculated r for, have students find θ in radians.

19/

Discussion for later class: How would we locate points in a Quadrant II, III, and IV? Would r change or θ ? How would they change? If given (r, θ) in a human Cartesian plane, would you move left or right then walk or vice versa? Define Cosecant, Cotangent, and Secant.

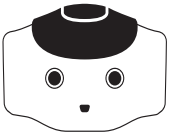
ASSIGNMENT/EXTENSION:

- Assign a worksheet or book problems that has students plotting points and finding the distance and angle.
- Give students a point in Quadrant II, III, or IV and have them find the distance and angle.
- Have students visit these two websites

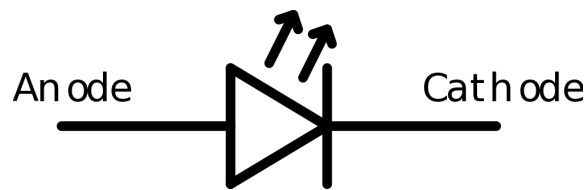
<http://zonalandeducation.com/mmts/trigonometryRealms/TrigFuncPointDef/TrigFuncPointDefinitions.html> (can check worksheet answers approximately)

<http://www.slideshare.net/guest793408/trigonometric-function-of-any-angle>

(explains points in other quadrants, use as an overview, future lesson still needs to be taught)

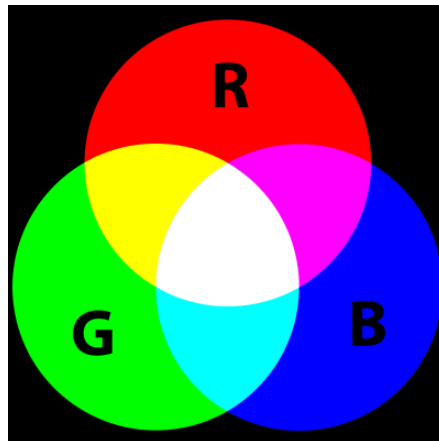


Besides its motors, the NAO has other forms of output. It has speakers on both sides of its head that it can use to play sounds and speak - we have used the text-to-speech capabilities in earlier modules. The NAO also has a variety of light-emitting diodes (LEDs) that can change brightness and color. As you may have learned from physics, diodes allow current to flow in only one direction. LEDs are a class of diodes that emit light when current flows through. The figure below shows the electronic symbol of an LED.



Around each of the NAO's speakers are 10 LEDs. These LEDs are blue in color and form a ring around the speaker. You may have noticed these LEDs switch on as the NAO is booting up. Besides turning these LEDs on and off, you can also vary their brightness. There are also 3 similar LEDs on the top of the NAO's head that are blue in color.

In each of the NAO's "eyes" there are 8 color LEDs. Color LEDs can show any color, by combining different intensities of three primary color LEDs - red, green and blue. For example, when yellow is displayed, the red and green are on and blue is off. You may have seen the figure below, which shows how the primary colors combine to form any color.



The NAO has a color LED in its chest, and one on each foot. Similar to the eye LEDs, we can choose what color the LED should display by changing the intensity of red, green and blue.

ABOUT

THE AUTHORS

This Textbook was created under the direction of Mike Beiter, Computer Science teacher at Central Career and Technical School, , and 2 PHD Students from Carnegie Mellon University, who created the individual lessons: Somchaya Liemhetcharat, and Brian Coltin.



MIKE BEITER

Mike is a High School Computer Science teacher at Central Career and Technical School in Erie, Pennsylvania. He is also an adjunct professor of computer science at Penn State University, and Gannon University. Mike is on the STEM development team leading the school in integrated project based learning in areas of computer science, Programmable Logic Controllers, and robotics.



BRIAN COLTIN

Brian Coltin is a PhD student in the Robotics Institute at Carnegie Mellon University. His research interests include multi-robot coordination, scheduling and path planning, sensor networks, and robot localization. He has participated in the RoboCup Standard Platform League with the NAO robots for four years as a member of the Carnegie Mellon team.



SOMCHAYA LIEMHETCHARAT

Somchaya Liemhetcharat is a PhD candidate at the Robotics Institute of Carnegie Mellon University. His research interests are in artificial intelligence and robotics, and in multi-robot coordination in particular, i.e., how multiple robots of different types can work together to solve complex problems.



www.aldebaran-robotics.com

AMERICAS - americas@aldebaran-robotics.com

EUROPE MIDDLE EAST AFRICA - emea@aldebaran-robotics.com

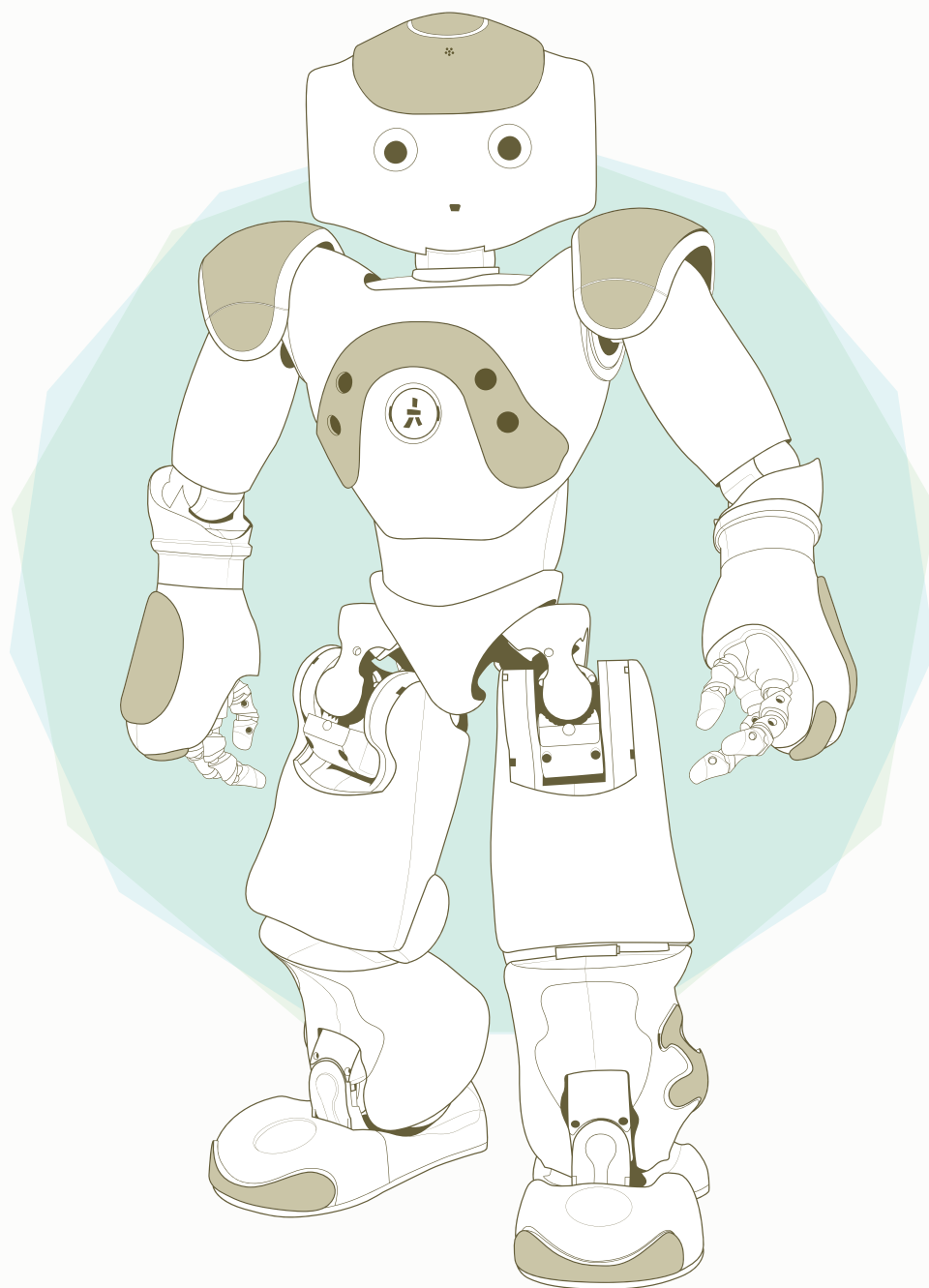
ASIA PACIFIC - asia-pacific@aldebaran-robotics.com

©2012 ALDEBARAN Robotics

www.aldebaran-robotics.com

Published by Aldebaran Robotics.
Designed by Romain Belotti for Aldebaran Robotics.
Printed in France by Icones.

ALDEBARAN Robotics, the ALDEBARAN Robotics logo, and NAO are trademarks of ALDEBARAN Robotics. Other trademarks, trade names and logos used in this document refer either to the entities claiming the marks and names, or to their products. ALDEBARAN Robotics disclaims proprietary interest in the marks and names of others. Choregraphe® & NAO® are registered trademarks of ALDEBARAN Robotics. The design of NAO® is the property of ALDEBARAN Robotics. All the photos featured in this document are noncontractual and are the property of ALDEBARAN Robotics.



www.aldebaran-robotics.com

AMERICAS - americas@aldebaran-robotics.com

EUROPE MIDDLE EAST AFRICA - emea@aldebaran-robotics.com

ASIA PACIFIC - asia-pacific@aldebaran-robotics.com