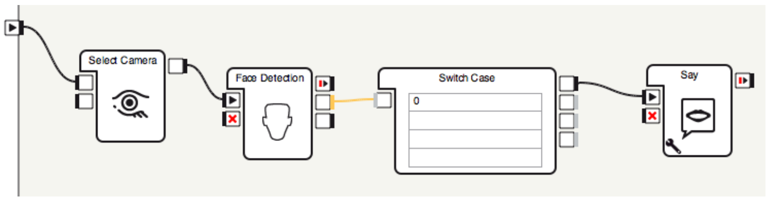
BASIC TASK
SEEING FACE TO FACE

–

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/

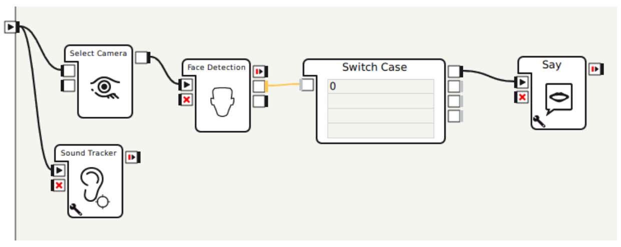
Set a message for the say box, such as "Hello, human."

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**

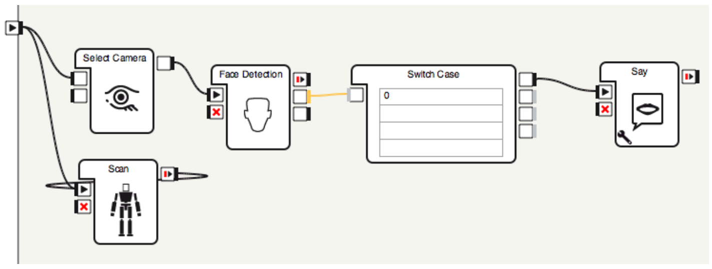
THE NAO CAN SEE A FACE
THAT HAPPENS TO PLACE ITSELF IN FRONT
OF ITS CAMERA. NOW WE WILL MAKE IT
SCAN ITS HEAD TO LOOK FOR FACES.

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

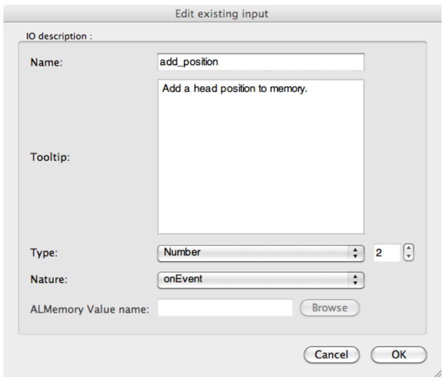
BEGIN FROM THE BASIC TASK, WHERE THE NAO LOOKS IN THE DIRECTION OF A NOISE. WE WILL CHANGE THIS BEHAVIOR TO MAKE THE ROBOT REMEMBER THE LAST TWO POSITIONS IT HAS HEARD A NOISE IN, AND TO CYCLE THROUGH THESE POSITIONS.

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.



Edit existing input

IO description :

Name: add_position

Tooltip: Add a head position to memory.

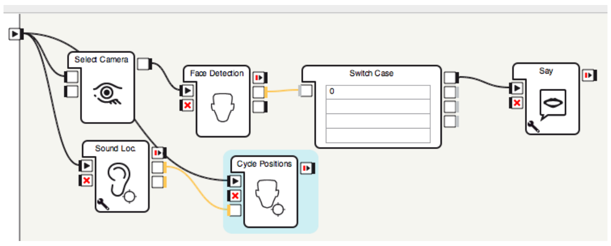
Type: Number 2

Nature: onEvent

ALMemory Value name:

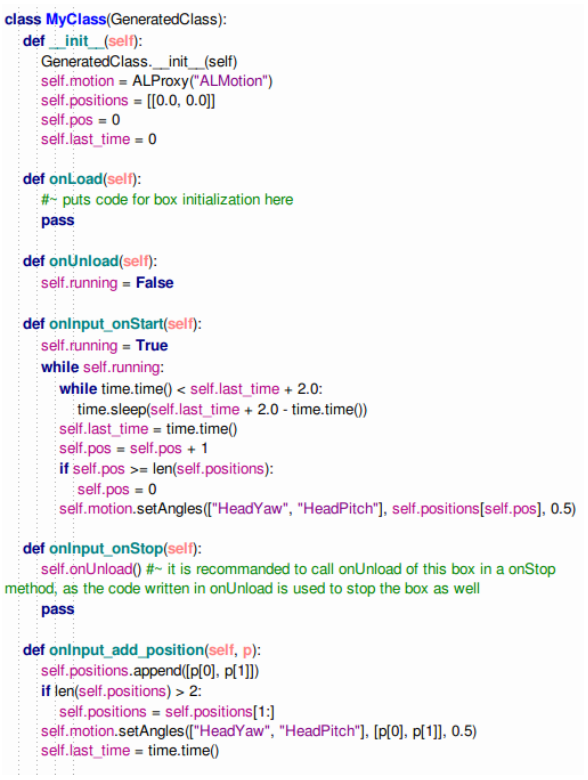
03/

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 recommanded 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.