Lesson Seven:
Holding Gestures

Myo™

Lesson Seven:
Holding Gestures

Overview

In the previous lesson, we made our functions more useful by allowing them to output through the keyboard. By assigning different functions to different poses, we built the foundation for any Myo Script.

This lesson will complete the basics and then explore vibration and locking, two details that come built into the Myo armband. How the armband locks and unlocks is important depending on what application you want to use.

Objective

After this lesson, you will have everything you need to take things into your own hands. Whether you want to use Myo for a game, a presentation, or on an app that you made yourself, your imagination is the only limit!

Goals

- Use vibration feedback from Myo
- Explore different locking policies
- Stop Myo from locking when a pose is held
- Write a complete script

Lesson Length (Time)

60 Minutes

PART 1: VIBRATION

Watching the debug console is not always easy when you are building a script. A useful technique is to give vibration feedback whenever a gesture is detected. The best way to do that is with `myo.notifyUserAction()`. You can make the Myo armband vibrate with the `myo.vibrate(vibrationType)` function, where `vibrationType` can be one of `short`, `medium`, or `long`.

Try putting in different combinations of vibration for each different pose function. If your code has `myo.vibrate(short)` twice, the Myo armband will vibrate twice consecutively.

Once you have made this change, this is what it may look like:

```
function onWaveOut()
myo.debug("Next")
myo.vibrate("short")
   myo.keyboard("tab", "press")
end

function onWaveIn()
   myo.debug("Previous")
   myo.vibrate("short")
   myo.vibrate("short")
   myo.keyboard("tab","press","shift")
end

function onFist()
   myo.debug("Enter")
   myo.notifyUserAction()
   myo.keyboard("return","press")
end

function onFingersSpread()
   myo.debug("Escape")
   myo.vibrate("long")
   myo.keyboard("escape", "press")
end
```

Think about whether you really need the Myo armband to vibrate before adding it to your own script.

PART 2: LOCKING POLICY

Until the user does the unlock gesture with an application running that can accept Myo input, none of the other poses are detected. This is the standard unlock policy by default. If that flow doesn't work for your script (for example, if you are controlling a game where the Myo should be active at all times), you can disable this behaviour by setting the lock policy.

```
myo.setLockingPolicy("none")
```

You can turn it back on with

```
myo.setLockingPolicy("standard")
```

The standard policy is designed to easily let the user input a command before the armband locks again. However, sometimes you may want to let the user hold a single gesture (like to adjust the volume) or enter a series of commands.

PART 3: HOLDING A POSE

We can hold a single gesture with the `myo.unlock(unlockType)` function. This will let you programmatically unlock (or keep unlocked) a Myo armband. You can choose either `timed` or `hold` as the `unlockType`. `timed` will reset the clock and keep the armband unlocked for a few more seconds, and `hold` will keep it unlocked until you manually lock it (or set it back to `timed` and wait).

Now we can change our script so that the user can hold any key press as long as they hold the pose that triggers it. To do that, we have to modify `onPoseEdge` and our pose handlers (`onWaveOut`, etc) so that we use the edge value of `onPoseEdge` to control the edge value of the key press.

The first problem is that `onPoseEdge` gives you either `on` or `off`, while `myo.keyboard()` needs either `down` or `up`. What we need to do is write some code that converts the one into the other and saves it in a local variable.

To do this, we'll use a local variable:

```
local keyEdge
if (edge == “on”) then
   keyEdge = “down”
else
   keyEdge = “up”
end
```

“Local” means that the variable only exists in the current “scope” (ie, in the current execution of the function or code block we're in. After that, it is gone forever and you can reuse the name). All the arguments to a function like `pose` and `edge` are local automatically. Every other variable is available anywhere in your script, which can be confusing. It's best to keep things local when you have the option.

Another way to do this relies on how Lua handles logical operators and order of operations. This one line means the same thing as the above solution:

```
local keyEdge = edge == “off” and “up” or “down”
```

the formatting of this line follows an if/else statement of the form:

```
myVariable = <condition> and <value if true> or <value if false>
```

PART 4: INTEGRATION

If we update `onPoseEdge` and the pose functions, our code should look something like below. We have also added the vibrations for different poses in this example, which can easily be taken out or commented out using two dashes ("--").

```
function onPoseEdge(pose, edge)
   myo.debug("onPoseEdge: " .. pose .. ": " .. edge)

   pose = conditionallySwapWave(pose)

   local keyEdge = edge == "off" and "up" or "down"

   if (pose == "waveOut") then
         onWaveOut(keyEdge)
   elseif (pose == "waveIn") then
         onWaveIn(keyEdge)
   elseif (pose == "fist") then
         onFist(keyEdge)
   elseif (pose == "fingersSpread") then
         onFingersSpread(keyEdge)
   end
end

function onWaveOut(keyEdge)
   myo.debug("Next")
   myo.vibrate("short")
   myo.keyboard("tab", keyEdge)
end

function onWaveIn(keyEdge)
   myo.debug("Previous")
   myo.vibrate("short")
   myo.vibrate("short")
   myo.keyboard("tab",keyEdge,"shift")
end
```

```
function onFist(keyEdge)
   myo.debug(“Enter”)
   myo.vibrate(“medium”)
   myo.keyboard(“return”,keyEdge)
end

function onFingersSpread(keyEdge)
   myo.debug(“Escape”)
   myo.vibrate(“long”)
   myo.keyboard(“escape”, keyEdge)
end

function conditionallySwapWave(pose)
   if myo.getArm() == “left” then
         if pose == “waveIn” then
               pose = “waveOut”
         elseif pose == “waveOut” then
               pose = “waveIn”
         end
   end
   return pose
end
```

If you try to run this script, you'll notice the armband locking in the middle of holding a pose, and you will only be able to do one or two of gestures. To keep the Myo armband unlocked while we are using it, we need to call `myo.unlock(unlockType)` every time a valid pose is detected. Unlock type will be `hold` if the user is holding a pose, or `timed` if they've released it.

Challenge Activities

For the questions below, you can copy this code[1] into your text editor and use it to complete each question. You should save it as a .lua file.

1. Using `myo.setLockingPolicy("none")` from Part 2, modify the script so that the Myo armband does not lock. Verify your code by waiting 10 seconds and then making a gesture. Observe the debug console to ensure that your gestures are being read.

2. Add different vibrations to each gesture with the `myo.vibrate(vibrationType)` function. You can keep `myo.setLockingPolicy("none")` so that Myo does not lock and is easier to test.

a) Test each gesture and verify the vibration type

b) Add 5 "short" vibrations to a gesture. Hold the gesture until it stops vibrating while counting the total amount of vibrations. Release the gesture and count the amount of vibrations again.

3. Write an if statement that calls `myo.unlock(unlockType)` every time a valid pose is detected. Remember that `pose` can have two invalid values, `rest` and `unknown`. This is to keep the Myo armband unlocked while we are using it, and should exist inside of your `onPoseEdge` function. You can refer to the end of Part 4 for help. Remember to remove `myo.setLockingPolicy("none")` from your script for this question.

Footnotes

[1] http://download.thalmic.com/education/Lesson7Sample.lua

Solutions

1. https://download.thalmic.com/education/Lesson7Q1.lua

2. https://download.thalmic.com/education/Lesson7Q2.lua

3. https://download.thalmic.com/education/Lesson7Q3.lua