

Steps to Programming the Gestalt Nodes;
(Mac OS X El Capitan; should work similarly for other Os)

List of useful links:

Essential:

<https://github.com/nadya/pygestalt>

<http://monograph.io/james/m-mtm>

<http://fabacademy.org/archives/2015/doc/MachineMakingNotes.html>

Links to needed software:

<https://learn.adafruit.com/arduino-lesson-17-email-sending-movement-detector/installing-python-and-pyserial>

<http://www.ftdichip.com/Drivers/VCP.htm>

Documentation to implement the Python code:

<http://www.fabacademy.org/archives/2015/doc/projects/gestalt-doc/api/modules.html>

Other projects needed for inspiration and debugging:

http://www.fabacademy.org/archives/2015/eu/labs/vienna_hl/machine_week.html

<http://fabacademy.org/archives/2015/eu/labs/frosinone/mechanical.html>

<http://fabacademy.org/archives/2015/as/labs/vigyam/machine/project.html>

<http://fabacademy.org/archives/2015/eu/labs/amsterdam/mtm/machinebuilding2015.html>

The Process:

The first step of the programming phase was setting up the hardware.

Having a total of 4 Gestalt nodes and 1 Netfab¹ I was able to get into the programming directly after understanding how to connect up the board correctly:

fortunately for me Emma sat with me and showed me how to connect up the board.

After the necessary basic introduction I was left on my own to figure out the programming basing myself on the documentation from the previous years:

¹ <http://fabacademy.org/archives/2015/eu/labs/amsterdam/mtm/machinebuilding2015.html>

Connecting the boards:

Connecting the Gestalt Nodes to each other are amply documented on the official fab academy site dedicated to them (<http://monograph.io/james/m-mtm>) the only difference for us was that they had been already flashed, so there was no need for us to do it, and the FabNet had already been milled and stuffed following this documentation; this was done by the students from the previous year (<http://fabacademy.org/archives/2015/eu/labs/amsterdam/mtm/machinebuilding2015.html> + <http://fabacademy.org/archives/2015/doc/MachineMakingNotes.html>)

Programming:

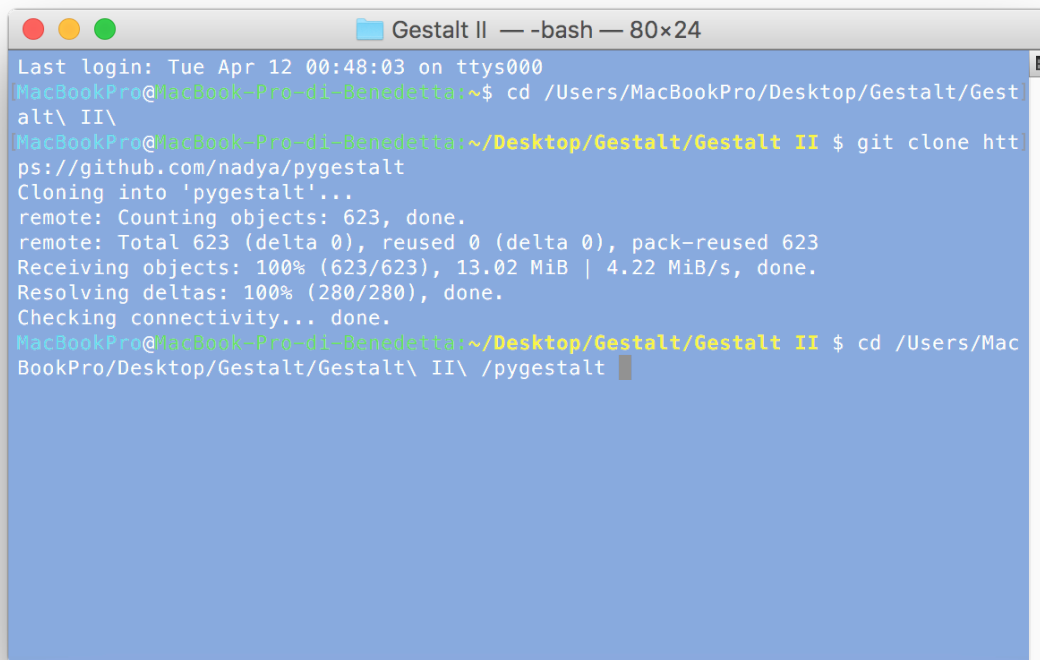
There are some software which I have installed before hand which some of the guides to use the node suggest to use, while some others don't, because I can't verify whether having installed them makes a difference, but I suggest you do:

=><https://learn.adafruit.com/arduino-lesson-17-email-sending-movement-detector/installing-python-and-pyserial> {Follow the instructions on this page to install pyserial}

=><https://www.python.org/downloads/> {Download and install the version with the closest release date; for me 3.4.4}

The first step with programming was cloning the *"pygestalt git repository"* (<https://github.com/nadya/pygestalt>). This is done through the terminal by choosing the folder you want to save the repository in with *"cd /[desired directory]"* (ex. *cd /Users/MacBookPro/Desktop*) and using the command *"git clone https://github.com/nadya/pygestalt"*.

Once this is completed we open with *"cd"* the directory where we saved
[All steps shown in the picture below on each line]

A terminal window titled "Gestalt II -- -bash -- 80x24" with a blue background. The terminal shows the following text:

```
Last login: Tue Apr 12 00:48:03 on ttys000
MacBookPro@MacBook-Pro-di-Benedetta:~$ cd /Users/MacBookPro/Desktop/Gestalt/Gestalt\ II\
MacBookPro@MacBook-Pro-di-Benedetta:~/Desktop/Gestalt/Gestalt II $ git clone https://github.com/nadya/pygestalt
Cloning into 'pygestalt'...
remote: Counting objects: 623, done.
remote: Total 623 (delta 0), reused 0 (delta 0), pack-reused 623
Receiving objects: 100% (623/623), 13.02 MiB | 4.22 MiB/s, done.
Resolving deltas: 100% (280/280), done.
Checking connectivity... done.
MacBookPro@MacBook-Pro-di-Benedetta:~/Desktop/Gestalt/Gestalt II $ cd /Users/MacBookPro/Desktop/Gestalt/Gestalt\ II\ /pygestalt
```

The next step is installing the “`setup.py`” this is done by writing in the command “`sudo python setup.py install`” in the terminal. If you have a password on your computer it will ask for it, just type it in and hit enter, it should work.

[This is what it should look like]

Once you hit enter and no errors pop-up the next step, if you are on OS X is to check the ID of your HDMI cable, this is done by inserting the command “`ls /dev/tty.usb*`” in the terminal when the FTDI cable is connected. It should give an output similar to => `/dev/tty.usbserial-FTXW4F9H` (which is the ID of my own HDMI cable) all of this name is essential!

We can then open to edit the “`xy_plotter.py`” file, this can be found in the following directory “`pygestalt/examples/machines/htmaa`” (what you use to edit this file is up to you, I used Xcode): You should modify the code to add `/dev/tty.usbserial-FTXW4F9H` to the definition of portName [Therefore it should look like the following]

```

def initInterfaces(self):
    if self.providedInterface: self.fabnet = self.providedInterface    #providedInterface is defined in the virtualMachine class.
    else: self.fabnet = interfaces.gestaltInterface('FABNET', interfaces.serialInterface(baudRate = 115200, interfaceType = 'ftdi',
        portName = '/dev/tty.usbserial-FTXW4F9H'))

```

```

Last login: Tue Apr 12 00:48:03 on ttys000
MacBookPro@MacBook-Pro-di-Benedetta:~$ cd /Users/MacBookPro/Desktop/Gestalt/Gestalt\ II\
MacBookPro@MacBook-Pro-di-Benedetta:~/Desktop/Gestalt/Gestalt II $ git clone https://github.com/nadya/pygestalt
Cloning into 'pygestalt'...
remote: Counting objects: 623, done.
remote: Total 623 (delta 0), reused 0 (delta 0), pack-reused 623
Receiving objects: 100% (623/623), 13.02 MiB | 4.22 MiB/s, done.
Resolving deltas: 100% (280/280), done.
Checking connectivity... done.
MacBookPro@MacBook-Pro-di-Benedetta:~/Desktop/Gestalt/Gestalt II $ cd /Users/MacBookPro/Desktop/Gestalt/Gestalt\ II\ /pygestalt
MacBookPro@MacBook-Pro-di-Benedetta:~/Desktop/Gestalt/Gestalt II /pygestalt$ sudo python setup.py install
Password:

```

[As mentioned on ‘getting started with Gestalt nodes’ you will also need to install the FTDI drivers to match with your OS (Found here: <http://www.ftdichip.com/Drivers/VCP.htm>)]

After this was done, what is left is personalizing the code to our need.

Since we are going to use three ‘axis’, and the xy_plotter example only controls two, I needed to add the capability of controlling another one.

The other examples in the folder helped in understanding the underlying logic, also the official documentation was helpful

(<http://www.fabacademy.org/archives/2015/doc/projects/gestalt-doc/api/modules.html>).

Essentially I modified the original program to contain references to a third ‘U axis’ by mimicking the existing axis. [This consisted of the following changes]

```

def initControllers(self):
1 self.xAxisNode = nodes.networkedGestaltNode('X Axis', self.fabnet, filename = '086-005a.py', persistence
  = self.persistence)
  self.yAxisNode = nodes.networkedGestaltNode('Y Axis', self.fabnet, filename = '086-005a.py', persistence
  = self.persistence)
  self.uAxisNode = nodes.networkedGestaltNode('U Axis', self.fabnet, filename = '086-005a.py', persistence
  = self.persistence)
3 self.xyuNode = nodes.compoundNode(self.xAxisNode, self.yAxisNode, self.uAxisNode) 2

```

1. Added a self.uAxisNode statement, where I define the third node
2. Added the self.uAxis node to the definition of the xyuNode variable which controls the nodes

```

def initCoordinates(self):
  self.position = state.coordinate(['mm', 'mm', 'mm'])

```

Added a third 'mm' parameter to the possible 'move' function that controls the displacement of the motors (before it was ['mm','mm']

```

self.stageKinematics = kinematics.direct(3) #direct drive on all axes

```

Set the kinematics to 3 instead of 2

```

def setPosition(self, position = [None, None, None]):
  self.position.future.set(position)

```

Added a third 'None' to the definition of setPostion

```

def initFunctions(self):
  self.move = functions.move(virtualMachine = self, virtualNode = self.xyuNode, axes = [self.xAxis, self.
  yAxis, self.uAxis], kinematics = self.stageKinematics, machinePosition = self.position,planner =
  'null')
  self.jog = functions.jog(self.move) #an incremental wrapper for the move function
  pass

```

There are two changes in the above, firstly I updated the name of the self.xyNode to self.xyuNode to match our previous change. Then I added in the definition the self.uAxis parameter.

```

# Some random moves to test with
ans_one = [0,100,0]
ans_two = [100,0,0]
ans_three = [0,0,100]

ans_list = [ans_one, ans_two, ans_three]
rnd_ans = random.choice(ans_list)
rest_position = [0,0,0]
extrusion = [rnd_ans[0] + 10, rnd_ans[1] + 10, rnd_ans[2] + 10]

moves = [rest_position, rnd_ans, extrusion, rnd_ans, rest_position]

```

This perhaps is my only true original contribution to the code;

With this we can set an indefinite amount of coordinates which define answer positions (here we only have three answers represented as 'ans_one', 'ans_two' & 'ans_three') this then puts them in a list and a random selector picks an answer and executes the moves in a standard manner to ensure that the extrusion and returning to the zero point are standardize amongst all answers.

I have not yet tested this, so I don't know if it works. I'm also working on a GUI.

(I would also like to note that now there are three parameters instead of two, to control all three motors)