overview

March 30, 2024

1 FullControl overview

FullControl is used to design changes to the ***state*** of ***things***

- ***state*** is any property of interest that can change (position, speed, power, temperature, etc.)
- ***things*** are anything with ***state*** - this initial release of FullControl is focused on ***things*** being extrusion 3D printers that are instructed by gcode

gcode is a list of instructions that change the ***state*** of a ***thing*** (3D printer, laser cutter, etc.)

a FullControl ***design*** dictates how the ***states*** of ***things*** change during a procedure (e.g. a manufacturing procedure)

for this release, the FullControl ***design*** is a 1D list of sequential 'steps' to change ***state***. The designer creates simple python code to generate the list. Each 'step' in the list is created using pre-defined templates for objects built into FullControl (described in later tutorial notebooks)

FullControl inspects the ***design*** and converts it into a ***result***

a ***result*** is gcode or a 3D plot in this initial release, but future releases will allow different types of ***designs*** and ***results***. - e.g. to FEA simulations - e.g. to documentation to support certification

at present, gcode can be formatted for a selection of printers and the 3D plot is implemented in plotly, but the range of printers is intended to be extended along with plotting software options

FullControl contains a set of tools to guide and support the generation of the ***design*** and the ***result***. e.g. geometry functions to support the generation of the ***design***. e.g. different variants of the gcode ***result*** to suit different printers

<this document is a jupyter notebook - if they're new to you, check out how they work: link, link, link>

run all cells in this notebook in order (keep pressing shift+enter)

2 scope of this notebook

this notebook gives a brief overview of FullControl capabilities with minimal technical explanations

other tutorial notebooks give full details of the FullControl features demonstrated here

2.1 I - import the FullControl python package

this gives you access to FullControl's functions and objects, etc.

make sure FullControl is installed first (very simple) - see the github readme for instructions

[ ]:
```
import fullcontrol as fc
```

2.2 II - create a FullControl *design*

as described above, the ***design*** is a list of steps using pre-defined FullControl objects as templates for ***state***-changes

minimal python knowledge is required

this notebook introduces basic python features/functions including 1D arrays (*'lists'*), *'append'* and *'extend'* functions, for-loops and the *'math'* module

complex FullControl designs can be created with only these functions

the *design* is a list of steps in this example, we create a three-step 'design' each step uses a FullControl 'Point' object, which tells the printer where to move to

[ ]:
```
point_1 = fc.Point(x=10, y=10, z=0)
point_2 = fc.Point(x=20, y=10, z=0)
point_3 = fc.Point(x=10, y=20, z=0)
steps = [point_1, point_2, point_3]
```

transform the *design* into a 'gcode' *result* use the fc.transform() function to transform the list of steps into gcode, then use the print() function to print the gcode to screen

saving gcode to a .gcode file directly is demonstrated later in this notebook

[ ]:
```
steps = [point_1, point_2, point_3]
gcode = fc.transform(steps, 'gcode')
print(gcode)
```

use the python 'append' and 'extend' functions to add steps to a *design*

[ ]:
```
# first, create an empty list
steps = []

# then add single items to it with 'append'
steps.append(fc.Point(x=10, y=10, z=0))
steps.append(fc.Point(x=20))
steps.append(fc.Point(x=10, y=20))

# to add multiple items to the list use 'extend'
extra_steps = [fc.Point(x=50, y=50),fc.Point(x=60, y=60),fc.Point(x=70, y=70)]
steps.extend(extra_steps)
```

```
# transform the design to gcode and print to screen
print(fc.transform(steps, 'gcode'))
```

use a python loop to concisely add steps to a *design*

```
[ ]: steps = []
     for i in range(11):
         steps.append(fc.Point(x=10+i,y=10+i,z=0))
     print(fc.transform(steps, 'gcode'))
```

transform a *design* into a 'plot' *result* plot are created from the ***design*** data. it does **not** inspect and plot a gcode file. this means it can utilize design data that may not be included in gcode (e.g. color)

info about changing the style of the plot (colors, axes, etc.) can be found in a plot formatting notebook

the following design loops 25 times to achieve 25 layers with 4 points in each

```
[ ]: layer_height = 0.2
     steps = []
     for i in range(25):
         steps.append(fc.Point(x=50,y=55,z=i*layer_height))
         steps.append(fc.Point(x=55+i*layer_height/2,y=50,z=i*layer_height/2))
         steps.append(fc.Point(x=50,y=45,z=i*layer_height))
         steps.append(fc.Point(x=45-i*layer_height/2,y=50,z=i*layer_height/2))
     fc.transform(steps, 'plot')
```

use mathematical design to make complex print paths this design creates a helix print path that fluctuates in height and radius

the tau, sin and cos functions need to be imported from python's built-in math module

```
[ ]: from math import sin, cos, tau
     steps = []
     for i in range(10000):
         angle = tau*i/200
         offset = (1.5*(i/10000)**2)*cos(angle*6)
         steps.append(fc.Point(x=(6+offset)*sin(angle), y=(6+offset)*cos(angle),␣
      ↪z=((i/200)*0.1)-offset/2))
     fc.transform(steps, 'plot')
```

use python *'list comprehension'* to create the list of steps efficiently

```
[ ]: from random import random
     steps = [fc.Point(x=50*random(),y=50*random(),z=i*0.01) for i in range(1000)]
     fc.transform(steps, 'plot')
```

2.3 III - common types of ***state***

you can change the ***state*** of more than the nozzle position

a few examples are shown here - more details about the various types of ***state*** are given in the state objects notebook

some changes to ***state*** result in a new line of gcode (e.g. changing fan speed)

other changes do not, but influence future lines of gcode (e.g. changing print speed only manifests in gcode when the next G1 movement command occurs)

e.g. print speed, fan speed and hotend temperature

```
[ ]: steps = []
steps.append(fc.Point(x=0,y=0,z=0))
steps.append(fc.Point(x=20))
steps.append(fc.Point(x=40))
steps.append(fc.Printer(print_speed=750))
steps.append(fc.Point(x=60))
steps.append(fc.Point(x=80))
steps.append(fc.Fan(speed_percent=50))
steps.append(fc.Hotend(temp=205))
steps.append(fc.Point(x=100))
print(fc.transform(steps, 'gcode'))
```

turn the extruder off and on

```
[ ]: steps = []
steps.append(fc.Point(x=0,y=0,z=0.2))
steps.append(fc.Point(x=5, y=20))
steps.append(fc.Point(x=10, y=0))
steps.append(fc.Extruder(on=False))
steps.append(fc.Point(x=0,y=0,z=0.4))
steps.append(fc.Extruder(on=True))
steps.append(fc.Point(x=5, y=20))
steps.append(fc.Point(x=10, y=0))
steps.extend(fc.travel_to(fc.Point(x=0,y=0,z=0.6)))
steps.append(fc.Point(x=5, y=20))
steps.append(fc.Point(x=10, y=0))
fc.transform(steps, 'plot')
```

2.4 IV - annotations and custom commands

add comments for the gcode *result*

```
[ ]: steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.GcodeComment(text='the next line of gcode will print to x=20'))
steps.append(fc.Point(x=20))
steps.append(fc.Point(x=40))
steps.append(fc.GcodeComment(end_of_previous_line_text='this line of gcode␣
  ↪prints to x=40'))
print(fc.transform(steps, 'gcode'))
```

add custom gcode commands gcode commands can be manually written

alternatively, the printer has a list of commands that can be called by their id, which allows automatic conversion of commands for different printers' gcode styles

```
[ ]: steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Point(x=20))
steps.append(fc.ManualGcode(text="G4 P2000 ; pause for 2 seconds"))
steps.append(fc.PrinterCommand(id='retract'))
print(fc.transform(steps, 'gcode'))
```

add annotations to the 'plot' *result* more details about plot annotations can be found in the plot formatting notebook

```
[ ]: steps = []
for i in range(3):
    steps.append(fc.Fan(speed_percent=50*i))
    for j in range (3):
        steps.append(fc.Point(x=20, y=20+5*j+30*i, z=0+0.1*j))
        steps.append(fc.PlotAnnotation(label="Height: " + str(0+0.1*j) + " mm"))
        steps.append(fc.Point(x=50, y=20+5*j+30*i, z=0+0.1*j))
    steps.append(fc.PlotAnnotation(label="Fan speed: " + str(50*i) + "%"))
fc.transform(steps, 'plot')
```

2.5 V - adjust the way the *design* is converted into the *result*

2.5.1 1. *GcodeControls* adjust gcode creation

some examples are given below

for more info about gcode controls, see the gcode formatting notebook

save gcode to file run the next cell to save gcode as a file in the same folder as this jupyter notebook

```
[ ]: steps = [fc.Point(x=10, y=10, z=0), fc.Point(x=20), fc.Point(y=20)]
fc.transform(steps, 'gcode', fc.GcodeControls(save_as="my_design"))
```

change initial print settings in addition to changing the ***state*** during the printing process, as shown in the above examples, you can set the ***state*** of initial print settings

```
[ ]: steps = [fc.Point(x=10, y=10, z=0), fc.Point(x=20), fc.Point(y=20)]
print('########\n######### Default initial conditions:\n########')
print(fc.transform(steps, 'gcode'))
```

```
gcode_controls = fc.GcodeControls(initialization_data={"print_speed": 600,␣
 ↪"travel_speed": 5750})
print('\n########\n######## Modified initial conditions (see F8000 changed to␣
 ↪F5750 and F1000 changed to F600):\n########')
print(fc.transform(steps, 'gcode', gcode_controls))
```

change format of gcode *result* for different printers running the code in the next cell will generate gcode for two different printers and print the first 8 lines to screen

```
[ ]: steps = [fc.Point(x=10, y=10, z=0), fc.Point(x=20), fc.Point(y=20)]
     prusa_gcode = fc.transform(steps, 'gcode', fc.
      ↪GcodeControls(printer_name='prusa_i3', initialization_data={'relative_e':␣
      ↪False}))
     ulti2plus_gcode = fc.transform(steps, 'gcode', fc.
      ↪GcodeControls(printer_name='ultimaker2plus',␣
      ↪initialization_data={'relative_e': True}))

     print('########\n######## prusa gcode - first 8 lines:\n######## ')
     gcode_list = (prusa_gcode.split('\n'))
     print('\n'.join(gcode_list[0:8]))

     print('\n\n########\n######## ultimaker gcode - first 8 lines:\n######## ')
     gcode_list = (ulti2plus_gcode.split('\n'))
     print('\n'.join(gcode_list[0:8]))
```

2.5.2 2. *PlotControls* adjust how plot data is created and displayed

output the raw plot data for use in alternative plottings modules/software it's also possible to change color, line-width, etc. - for more info about plot controls, see the plot formatting notebook

the next code cell prints the raw plot data to screen and also creates the associated 3D plot for comparison

```
[ ]: plot_controls = fc.PlotControls(raw_data=True)
     steps = [fc.Point(x=10, y=10, z=0), fc.Point(x=30, z=0.5), fc.Point(x=10, z=1),␣
      ↪fc.PlotAnnotation(label="End")]
     plot_data = fc.transform(steps, 'plot', plot_controls)
     print(plot_data)
     fc.transform(steps, 'plot')
```

2.6 VI - use FullControl geometry functions to create the *design*

a few demo functions are shown here - for more details about geometry functions, see the geometry functions notebook

e.g. rectangle

[ ]:
```
steps = fc.rectangleXY(fc.Point(x=0, y=0, z=0.2), 20, 4)
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence'))
```

e.g. copy geometry to make a linear array

[ ]:
```
steps = fc.rectangleXY(fc.Point(x=0, y=0, z=0.2), 20, 4)
steps = fc.move(steps,fc.Vector(z=0.2),copy=True, copy_quantity=25)
fc.transform(steps, 'plot')
```

e.g. helix

[ ]:
```
centre_point = fc.Point(x=50, y=50, z=0)
steps = fc.helixZ(centre_point, 8, 6, 0, 30, 0.15, 20*64)
fc.transform(steps, 'plot')
```

combine geometry functions to quickly achieve interesting print paths

- create a squarewave
- copy it with 180-degree rotation
- repeat it for 25 layers

[ ]:
```
steps = fc.squarewaveXY(fc.Point(x=20, y=50, z=0), fc.Vector(x=1, y=0), 10, 5,␣
 ↪10)
steps = fc.move_polar(steps,fc.Point(x=67.5, y=45, z=0), 0, tau/2, copy=True)
steps = fc.move(steps, fc.Vector(z=0.2), copy=True, copy_quantity=60)
fc.transform(steps, 'plot')
```

design in polar coordinates the FullControl 'polar_to_point' function converts polar coordinates into Cartesian points

combining it with python's built-in 'list comprehension' capabilities allows a complex list of steps to be created with one line of code

[ ]:
```
steps=[fc.polar_to_point(centre=fc.Point(x=0, y=0, z=i*0.001),␣
 ↪radius=10+5*random(), angle=i*tau/13.8) for i in range(4000)]
fc.transform(steps, 'plot', fc.PlotControls(neat_for_publishing=True, zoom=0.7))
```

create custom geometry functions the next example is similar to the earlier squarewave example, except it uses a **custom** triangle wave function instead of a function built into FullControl

if you create useful geometry functions, add them to FullControl so everyone can benefit (see contribution guidelines on github)

[ ]:
```
def tri_wave(start_point: fc.Point, amplitude: float, period_length: float,␣
 ↪periods: int) -> list:
    tri_wave_steps = []
    for i in range(periods*2+1):
        tri_wave_steps.append(fc.Point(x=start_point.x+i*period_length/2,␣
 ↪y=start_point.y+amplitude*(i % 2), z=start_point.z))
    return tri_wave_steps
```

```
steps = tri_wave(fc.Point(x=20, y=50, z=0), 10, 10, 10)
steps.extend(tri_wave(fc.Point(x=120, y=40, z=0), -10, -10, 10))
steps = fc.move(steps, fc.Vector(z=0.2), copy=True, copy_quantity=60)
fc.transform(steps, 'plot')
```

2.7 VII - next steps

enhanced functionality has been developed for in-house research and is intended for public release as time allows: - multi-axis - full walk-through documentation for a 5-axis tool changer, including hardware, configuration, calibration, and more (release imminent) - toolpath design directly in FullControl - preliminary version already included in the 'FullControl lab' - see the example below, and the 5-axis demo notebook - multi-tool - multi-hardware - geometry import and interrogation (STL and similar) - in-process inspection and correction - upload of models to www.fullcontrol.xyz - if you're able and interested in turning www.fullcontrol.xyz into the ***best website ever*** for additive manufacturing, please get in touch: `info@fullcontrol.xyz`

additional functionality beyond that listed above is planned for ongoing research by Andy Gleadall, which will also be made open-source whenever possible

please improve FullControl and add capabilities to it - e.g. to support journal papers that present new methods for additive manufacturing by making those methods available to everyone via FullControl

five_axis example this example shows a wavey helical print path, where the model is continuously rotating while the nozzle gradually moves away from the print platform

the part is tilted to orient the nozzle perpendicular(ish) to the wavey walls at all points

color data is added to visualize the b axis

[ ]:
```
import lab.fullcontrol.fiveaxis as fc5
from math import sin, cos, tau
steps = []
for i in range(10001):
    angle = tau*i/200
    offset = (1.5*(i/10000)**2)*cos(angle*6)
    steps.append(fc5.Point(x=(6+offset)*sin(angle), y=(6+offset)*cos(angle),␣
  ↪z=((i/200)*0.1)-offset/2, b=(offset/1.5)*30, c=angle*360/tau))
for step in steps:
    if type(step).__name__ == 'Point':
        # color is a gradient from B=0 (blue) to B=45 (red)
        step.color = [((step.b+30)/60), 0, 1-((step.b+30)/60)]
steps.append(fc5.PlotAnnotation(point=fc5.Point(x=0, y=0, z=8.75), label='color␣
  ↪indicates B axis (tilt)'))
steps.append(fc5.PlotAnnotation(point=fc5.Point(x=0, y=0, z=7.5), label='-30␣
  ↪deg (blue) to +30 deg (red)'))
gcode = fc5.transform(steps,'gcode')
print('final ten gcode lines:\n' + '\n'.join(gcode.split('\n')[-10:]))
```

```
fc5.transform(steps, 'plot', fc5.PlotControls(color_type='manual',␣
 ↪hide_axes=False, zoom=0.75))
```