lab_four_axis_demo

March 30, 2024

1 lab four-axis demo

this documentation gives a brief overview of 4-axis capabilities - it will be expanded in the future

it currently works for a system with a nozzle rotating about the y axis, for which open-source documentation will be released soon

the generated gcode would work on other 4-axis systems but this would likely require minor tweaks and a good understanding of the gcode requirements

<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)

four axis import

[ ]:
```
import lab.fullcontrol.fouraxis as fc4
```

basic demo points in fullcontrol are designed in the model's XYZ coordinate system

rotation of the b axis will cause the nozzle to move in the x and z directions, and the amount that it moves depends on how far the tip of the nozzle is away from the axis of rotation. therefore it is important to set this distance with `GcodeControls(b_offset_z=...)` to allow fullcontrol to determine the correct x z values to send to the printer

if the nozzle is below the axis of rotation b_offset_z should be positive

there is also the potential for the nozzle to be offset from the axis of rotation in the x direction when it is vertical (b=0). this is not currently programmed in fullcontrol but will be in the future and will be set by the user with `b_offset_x`

the GcodeControls object has slightly less functionality for 4-axis FullControl compared to 3-axis FullControl - there are no printer options to choose from at present (the 'generic' printer is always used) and no built-in primer can be used

[ ]:
```
b_offset_z = 46.0 # mm
```

the following code cell briefly demonstrates how changes to the model coordinates and orientation affect the machine coordinates in interesting ways

[ ]:
```
steps=[]
steps.append(fc4.Point(x=0, y=0, z=0, b=0))
```

```
steps.append(fc4.GcodeComment(end_of_previous_line_text='start point'))
steps.append(fc4.Point(x=1))
steps.append(fc4.GcodeComment(end_of_previous_line_text='set x=1 - gcode for␣
 ↪this is simple... just move in x'))
steps.append(fc4.Point(b=60))
steps.append(fc4.GcodeComment(end_of_previous_line_text='set b=45 - this causes␣
 ↪a change to x and z in system coordinates'))
steps.append(fc4.Point(b=90))
steps.append(fc4.GcodeComment(end_of_previous_line_text="set b=90 - although x␣
 ↪and z change, the nozzle tip doesn't move (hence E=0)"))
steps.append(fc4.Point(z=1))
steps.append(fc4.GcodeComment(end_of_previous_line_text="set z=1 - just like␣
 ↪the x-movement above, this z-movement is simple. it's only changes to nozzle␣
 ↪angle that affect other axes"))
steps.append(fc4.Point(b=-90))
steps.append(fc4.GcodeComment(end_of_previous_line_text='set b=-90 - the print␣
 ↪head moves to the opposite side when the nozzle rotates 180 degrees to␣
 ↪ensure the nozzle stays at x=1'))
print(fc4.transform(steps, 'gcode', fc4.GcodeControls(b_offset_z=b_offset_z)))
```

add custom color to preview axes this code cell demonstrates a convenient way to add color for previews - it is not supposed to be a useful print path, it's just for demonstration

after creating all the steps in the design, color is added to each Point object based on the Point's orientation in B

[ ]:
```
steps = []
steps.append(fc4.Point(x=0, y=0, z=0, b=0))
steps.append(fc4.PlotAnnotation(label='B0'))
steps.append(fc4.Point(x=10, y=5, z=0, b=0))
steps.append(fc4.PlotAnnotation(label='B0'))
steps.append(fc4.Point(y=10, z=0, b=-180))
steps.append(fc4.PlotAnnotation(label='B-45'))
steps.append(fc4.Point(x=0, y=15, b=-180))
steps.append(fc4.PlotAnnotation(label='B-45'))
steps.append(fc4.Point(y=20, b=180))
steps.append(fc4.PlotAnnotation(label='B+45'))
steps.append(fc4.Point(x=10, y=25, b=180))
steps.append(fc4.PlotAnnotation(label='B+45'))
for step in steps:
    if type(step).__name__ == 'Point':
        # color is a gradient from B=-180 (blue) to B=+180 (red)
        step.color = [((step.b+45)/90), 0, 1-((step.b+45)/90)]
fc4.transform(steps, 'plot', fc4.PlotControls(color_type='manual'))
```

a more complex color example this example shows a wavey helical print path, where the tilts to easy side (oscialtes once per layer)

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

[ ]:
```
from math import sin, cos, tau
EH = 0.4
EW = 1.2

rad = 12  # nominal radius of structure before offsets
max_offset = rad

start_x, start_y = 75, 75
initial_z = 0.5*EH

steps = []
segs, segs_per_layer = 10000, 200
max_z = (segs/segs_per_layer)*EH

for i in range(segs+1):
    angle = tau*i/segs_per_layer
    offset = (max_offset*(i/segs)**2)*(0.5+0.5*cos(angle*2))
    steps.append(fc4.Point(x=start_x+(rad+offset)*cos(angle),␣
 ↪y=start_y+(rad+offset)*sin(angle),
                z=initial_z+((i/segs_per_layer)*EH)-offset/2,␣
 ↪b=cos(angle)*(offset/max_offset)*45))
for step in steps:
    if type(step).__name__ == 'Point':
        # color is a gradient from B=-45 (blue) to B=45 (red)
        step.color = [((step.b+45)/90), 0, 1-((step.b+45)/90)]
steps.append(fc4.PlotAnnotation(point=fc4.Point(
    x=start_x, y=start_y, z=max_z*1.2), label='color indicates B axis (tilt)'))
steps.append(fc4.PlotAnnotation(point=fc4.Point(
    x=start_x, y=start_y, z=max_z), label='-45 deg (blue) to +45 deg (red)'))
gcode = fc4.transform(steps, 'gcode', fc4.GcodeControls(b_offset_z=b_offset_z,␣
 ↪initialization_data={
                    'print_speed': 500, 'extrusion_width': EW,␣
 ↪'extrusion_height': EH}))
print('final ten gcode lines:\n' + '\n'.join(gcode.split('\n')[-10:]))
fc4.transform(steps, 'plot', fc4.PlotControls(
    color_type='manual', hide_axes=False, zoom=0.75))

design_name = 'fouraxis'
open(f'{design_name}.gcode', 'w').write(gcode)

# activate the next line to download the gcode if using google colab
# files.download(f'{design_name}.gcode')
```

use 3-axis geometry functions from FullControl (with caution!) this functionality should be considered experimental at best!

geometry functions that generate 3-axis points can be used - accessed via fc4.xyz_geom()

but they must be translated to have 4-axis methods for gcode generation - achieved via fc4.xyz_add_b()

this conversion does not set any values of B attributes for those points - the B values will remain at whatever values they were in the ***design*** before the list of converted points

in the example below, a circle is created in the XY plane in the model's coordinate system, but the b-axis is set to 45

hence, when the ***design*** is transformed to a 'gcode' ***result***, X and Z values vary from the design X Z values in gcode to accomodate the true required position of the printhead (to get the desired nozzle location)

in contrast, when the ***design*** is transformed to a 'plot' ***result***, the plot shows model coordinates (e.g. Z=0) because 4-axis plots in the 3D-printer's coordinates system often make no sense visually

[ ]:
```
steps=[]
steps.append(fc4.Point(x=10, y=0, z=0, b=45))
xyz_geometry_steps = fc4.xyz_geom.circleXY(fc4.Point(x=0, y=0, z=0), 10, 0, 16)
xyz_geometry_steps_with_bc_capabilities = fc4.xyz_add_b(xyz_geometry_steps)
steps.extend(xyz_geometry_steps_with_bc_capabilities)
steps.append(fc4.PlotAnnotation(point=fc4.Point(x=0, y=0, z=5), label='normal␣
 ↪FullControl geometry functions can be used via fc4.xyz_geom'))
steps.append(fc4.PlotAnnotation(point=fc4.Point(x=0, y=0, z=3.5), label='but␣
 ↪points must be converted to 4-axis variants via fc4.xyz_add_b'))
print(fc4.transform(steps, 'gcode', fc4.GcodeControls(b_offset_z=30)))
fc4.transform(steps, 'plot')
```