# lab_five_axis_demo

March 30, 2024

## 1 lab five-axis demo

this documentation gives a brief overview of 5-axis capabilities - it will be expanded in the future

most of this tutorial relates to a system with B-C rotation stages, where C-axis rotations do not affect the B axis, but B-axis rotations do alter the C axis orientation (i.e. a rotating stage [C] mounted onto a tilting platform [B])

the final section of this notebook demosntrates how to generate gcode for a system with a rotating nozzle (B) and rotating bed (C)

the generated gcode would work on other 5-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)*

**five axis import**

```
[ ]: import lab.fullcontrol.fiveaxis as fc5
     import fullcontrol as fc
     import lab.fullcontrol as fclab
```

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

the point x=0, y=0, z=0 in the model's coordinate system represents the intercept point of B and C axes

FullControl translates them to the 3D printer XYZ coordinates, factoring in the effect of rotations to B and C axes

a full explanation of this concept is out of scope for this brief tutorial notebook - google 5-axis kinematics for more info

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

```
[ ]: steps=[]
     steps.append(fc5.Point(x=0, y=0, z=0, b=0, c=0))
     steps.append(fc5.GcodeComment(end_of_previous_line_text='start point'))
     steps.append(fc5.Point(x=1))
```

```
steps.append(fc5.GcodeComment(end_of_previous_line_text='set x=1 - since b=0␣
 ↪and c=0, the model x-axis is oriented the same as the system x-axis'))
steps.append(fc5.Point(b=45))
steps.append(fc5.GcodeComment(end_of_previous_line_text='set b=45 - this causes␣
 ↪a change to x and z in system coordinates'))
steps.append(fc5.Point(b=90))
steps.append(fc5.GcodeComment(end_of_previous_line_text='set b=90 - although x␣
 ↪and z change, E=0 because the nozzle stays in the same point on the model'))
steps.append(fc5.Point(b=0))
steps.append(fc5.GcodeComment(end_of_previous_line_text='set b=0'))
steps.append(fc5.Point(c=90))
steps.append(fc5.GcodeComment(end_of_previous_line_text='set c=90 - this causes␣
 ↪a change to x and y in system coordinates'))
steps.append(fc5.Point(y=1))
steps.append(fc5.GcodeComment(end_of_previous_line_text='set y=1 - this causes␣
 ↪a change to x in system coordinates since the model is rotated 90 degrees'))
print(fc5.transform(steps, 'gcode'))
```

**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(fc5.Point(x=0, y=0, z=0, b=0, c=0))
steps.append(fc5.PlotAnnotation(label='B0'))
steps.append(fc5.Point(x=10, y=5, z=0, b=0, c=0))
steps.append(fc5.PlotAnnotation(label='B0'))
steps.append(fc5.Point(y=10, z=0, b=-180, c=0))
steps.append(fc5.PlotAnnotation(label='B-180'))
steps.append(fc5.Point(x=0, y=15, b=-180, c=0))
steps.append(fc5.PlotAnnotation(label='B-180'))
steps.append(fc5.Point(y=20, b=180, c=0))
steps.append(fc5.PlotAnnotation(label='B+180'))
steps.append(fc5.Point(x=10, y=25, b=180, c=0))
steps.append(fc5.PlotAnnotation(label='B+180'))
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+180)/360), 0, 1-((step.b+180)/360)]
fc5.transform(steps, 'plot', fc5.PlotControls(color_type='manual'))
```

**a more complex color 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

```
[ ]: 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))
```

**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 fc5.xyz_geom()

but they must be translated to have 5-axis methods for gcode generation - achieved via fc5.xyz_add_bc()

this conversion does not set any values of B or C attributes for those points - the BC 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 90

therefore, the 3D printer actually prints a circle in the YZ plane since the model coordinate system has been rotated by 90 degrees about the B axis

hence, when the ***design*** is transformed to a 'gcode' ***result***, Y and Z values vary in gcode while X is constant (of course this would not print well - it's just for simple demonstration)

in contrast, when the ***design*** is transformed to a 'plot' ***result***, the plot shows model coordinates because 5-axis plots in the 3D-printer's coordinates system often make no sense visually

```
[ ]: steps=[]
steps.append(fc5.Point(x=10, y=0, z=0, b=90, c=0))
xyz_geometry_steps = fc5.xyz_geom.circleXY(fc5.Point(x=0, y=0, z=0), 10, 0, 16)
xyz_geometry_steps_with_bc_capabilities = fc5.xyz_add_bc(xyz_geometry_steps)
steps.extend(xyz_geometry_steps_with_bc_capabilities)
steps.append(fc5.PlotAnnotation(point=fc5.Point(x=0, y=0, z=5), label='normal␣
 ↪FullControl geometry functions can be used via fc5.xyz_geom'))
```

```
steps.append(fc5.PlotAnnotation(point=fc5.Point(x=0, y=0, z=3.5), label='but␣
 ↪points must be converted to 5-axis variants via fc5.xyz_add_bc'))
print(fc5.transform(steps, 'gcode'))
fc5.transform(steps, 'plot')
```

**bc_intercept** if the machine's coordinate system is **not** set up so that the b and c axes intercept at the point x=0, y=0, z=0, the bc_intercept data can be provided in a GcodeControls object to ensure correct gcode generation

the GcodeControls object has slightly less functionality for 5-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

note that although the system does not need the b and c axes to intercept at the point x=0, y=0, z=0, the model coordinate system must still be implemented such that the point x=0, y=0, z=0 represents the intercept point of b and c axes

[ ]:
```
gcode_controls = fc5.GcodeControls(bc_intercept = fc5.Point(x=10, y=0, z=0),␣
 ↪initialization_data={'nozzle_temp': 250})
steps=[]
steps.append(fc5.Point(x=0, y=0, z=0, b=0, c=0))
steps.append(fc5.GcodeComment(end_of_previous_line_text='start point (x=0 in␣
 ↪the model but x=10 in gcode due to the bc_intercept being at x=10)'))
steps.append(fc5.Point(x=1))
steps.append(fc5.GcodeComment(end_of_previous_line_text='set x=1 - since b=0␣
 ↪and c=0, the model x-axis is oriented the same as the system x-axis'))
steps.append(fc5.Point(b=45))
steps.append(fc5.GcodeComment(end_of_previous_line_text='set b=45 - this causes␣
 ↪a change to x and z in system coordinates'))
print(fc5.transform(steps, 'gcode', gcode_controls))
```

**rotating-nozzle 5-axis system** if the nozzle rotates about the Y axis, as opposed to the rotating bed tilting about the Y axis (as was the case for the code above), but the print bed still rotates about the Z axis, you can import `lab.fullcontrol.fiveaxisC0B1` as fc5 instead of `lab.fullcontrol.fiveaxis`

this is shown in the code cell below. note that the new import statement means the previous import of fc5 is nullified, so don't try to run the above code cells after running the next code cell or they won't work

the simple instructions below show how rotation of the bed and of the nozzle independently result in necessary changes to X and Y in gcode

in the future, the way of defining which type of multiaxis printer you change will be made more intuitive and procedural, but this works for now.

[ ]:
```
import lab.fullcontrol.fiveaxisC0B1 as fc5
b_offset_z = 40
steps=[]
```

```
steps.append(fc5.Point(x=0, y=0, z=0, b=0, c=0))
steps.append(fc5.GcodeComment(end_of_previous_line_text='start point'))
steps.extend([fc5.Point(x=1), fc5.
 ↪GcodeComment(end_of_previous_line_text='x=1')])
steps.extend([fc5.Point(c=90), fc5.
 ↪GcodeComment(end_of_previous_line_text='c=90')])
steps.extend([fc5.Point(c=180), fc5.
 ↪GcodeComment(end_of_previous_line_text='c=180')])
steps.extend([fc5.Point(c=270), fc5.
 ↪GcodeComment(end_of_previous_line_text='c=270')])
steps.extend([fc5.Point(x=0, y=1, c=0), fc5.
 ↪GcodeComment(end_of_previous_line_text='x=0, y=1, c=0')])
steps.extend([fc5.Point(c=90), fc5.
 ↪GcodeComment(end_of_previous_line_text='c=90')])
steps.extend([fc5.Point(c=180), fc5.
 ↪GcodeComment(end_of_previous_line_text='c=180')])
steps.extend([fc5.Point(c=270), fc5.
 ↪GcodeComment(end_of_previous_line_text='c=270')])
steps.extend([fc5.Point(b=90), fc5.
 ↪GcodeComment(end_of_previous_line_text='b=90')])
steps.extend([fc5.Point(b=-90), fc5.
 ↪GcodeComment(end_of_previous_line_text='b=-90')])
print(fc5.transform(steps, 'gcode', fc5.GcodeControls(b_offset_z=b_offset_z)))
```

to keep the nozzle directly to the hand side of the bed (Y=0) for every point, which is useful for nozzle tilting about Y when printing a funnel for example, you need to design C to rotate for each point

[ ]:
```
from math import degrees

circle_segments = 16
points_per_circle = circle_segments+1
steps = []
xyz_geometry_steps = fc5.xyz_geom.circleXY(fc5.Point(x=0, y=0, z=0), 10, 0,␣
 ↪circle_segments)
xyz_geometry_steps_with_bc_capabilities = fc5.xyz_add_bc(xyz_geometry_steps)
steps.extend(xyz_geometry_steps_with_bc_capabilities)
steps[0].b, steps[0].c = 0.0, 0.0
gcode_without_c_rotation = fc5.transform(steps, 'gcode', fc5.
 ↪GcodeControls(b_offset_z=b_offset_z))
fc5.transform(steps, 'plot')

for i in range(len(steps)): steps[i].c = -360/circle_segments*i
# instead of the above for loop, you can use the following function to␣
 ↪constantly vary c automatically. This is good for more complex geometry,␣
 ↪where c cannot be 'designed' easily.
```

```
# steps = fclab.constant_polar_angle_with_c(points=steps, centre=fc5.Point(x=0,␣
 ↪y=0, z=0), initial_c=-90)
gcode_with_c_rotation = fc5.transform(steps, 'gcode', fc5.
 ↪GcodeControls(b_offset_z=b_offset_z))

print(gcode_without_c_rotation +
      '\n\n\ngcode with C rotation to keep nozzle directly in +X direction from␣
 ↪bed centre:\n\n' + gcode_with_c_rotation)
```

**next steps** this tutorial notebook gives a brief introduction to five-axis use of FullControl for interest, but it is not an expansive implementation. it is included as an initial step towards translating in-house research for 5-axis gcode generation into a more general format compatible with the overall FullControl concept