ADD PECK TO RIGID TAPPING

Prepared by: Tanay Gahlot, BTech IV Year, Computer Science, NIT Goa, India
Email: tanaygahlot@gmail.com
Mobile: +91-8552871793
IRC: TanayG
Github: http://github.com/TanayGahlot
Brief Background Info:

- Final year BTech student in Computer Science at National Institute of Technology, Goa, India
- Google Student Ambassador
- Author of "Automating Toolpath Generation For 3-Axis CNC"(selected in ICIT2016)

Brief Summary

This project is about adding peck tapping to rigid tapping(G33.1) in LinuxCNC so that G-Code command for adding pecking cycle looks like:

G33 Z-5.0 Q1.0 Kxx.xx

(tap 5mm deep, with 1mm peck depth at XX feed, repeat until we reach -5mm)

An additional parameter "Q" specifies the depth of peck. Peck tapping comes in handy while machining deep holes or hard to tap metal.

Essential Vocabulary

G-Code: Its a language in which people tell computerized cutting tools 'how to make something'

CNC: Computerized Numeric Control, is the automation of machine tools that are operated by precisely programmed commands.

Why This Project?

Peck tapping is required for drilling deep holes and tap hard metals. Currently LinuxCNC users have to do it manually by writing a series of G33.1 command with increasing depth. This leads to time wastage since spindle has to be synchronized and repositioned to index. This is why we need peck tapping functionality.

Overview of Approach

Peck tapping can be added to G33.1 in inter_convert.cc by wrapping it in a six line utility in interp_cycles.cc. Also the spindle can be synchronized, but to implement this, changes have to be made from interpreter to motion planner, which becomes hard to implement reliably. Since machining is an area that demands very high reliability, i will stick to adding G33.1 in pecking utility. Ultimate decision on this remains in the hand of mentor.

Detailed Approach

Since i am not familiar with the entire codebase, i will invest first few days in understanding LinuxCNC. My primary focus would be to understand:

- How G-Code are converted to electrical signals?
- How various modules like motion planning, G-Code interpreter, kinematics, trajectory planner work together to achieve that?
- Coding style used throughout the code.
- Testing methods and technique used to test other G-Code commands

Once i have a thorough grasp of code i will go on to implement peck tapping by wrapping it in a six line utility. In doing so i will ensure it adheres to coding style and doesn't break the code by performing constant build. Once the code is reviewed, i would go on to test it in the simulation software and then on the machine to create few working examples. Then i will move on to writing test cases in line with standards in place for testing other G-Code commands. All this while i will keep iterating on the code. Finally i will update users documentation to inform them about usage of peck tapping and developers documentation to help future developers understand my work.

Timeline

Week	Task
Week 1(23-29)	Familiarize myself with the codebase and particularly understand how G-Code are converted to machine signals
Week 2(30-5)	Familiarize myself with the codebase and particularly understand how test cases are written for them
Week 3(6-12)	Wrapping G33.1 in six line utility
Week 4(13-19)	Wrapping G33.1 in six line utility
Week 5(20-26)	Testing the generated output on a machine to show working example and iterating
Week 6(27-3)	Testing the generated output on a machine to show working example and iterating
Week 7(4-10)	Testing the generated output on a machine to show working example and iterating
Week 8(11-17)	Write test case to ensure changes made doesn't break existing code and iterating
Week 9(18-24)	Write test case to ensure changes made doesn't break existing code and iterating
Week 10(25-31)	Write test case to ensure changes made doesn't break existing code and iterating
Week 11(1-7)	Adding usage to users documentation
Week 12(8-14)	Adding methods to developers documentation
Week 13(15-21)	Final review

Time Availability

I will be available for 40 hours/week. I am flexible in terms of work hours, if situation demands i don't mind working extra hours.

Why LinuxCNC?

My thesis is in the area of CNC machining, ever since i started working on CNC's i have heard about LinuxCNC capability and reliability extensively. Therefore i want to contribute in my capacity to this noble initiative.

Why me?

Since I am trained in computer science i have thorough grasp of C++ and Python which is used in LinuxCNC. I am also a great fan and tinkerer of linux. I have been working in machining for almost an year, particularly desktop fabrication, so i have decent understanding of CNC machining. I have contributed to open source organization like MNE-PYTHON and RHOK. Therefore i feel i am a good fit for this project.