Project Report On CNC Machine

Project Team

Pubudu Tharuka – CODSE172F-041

Tishan Ravisanka– CODSE172F-019

Rafeek Natha– CODSE172F-044

Sajith Hemantha– CODSE172F-017

Project Title: CNC Machine

Authors **:** CODSE172F-041 Pubudu Tharuka

CODSE172F-019 Tishan Ravisanka

CODSE172F-044 Rafeek Natha

CODSE172F-017 Sajith Hemantha

Program **:** Diploma in Software Engineering

Supervisor **:** Mr. O K G C Weerasekara

Institute **:** Management Information System Division

National Institute of Business Management Colombo 07

Date **:** 21.03.2019

"The project is submitted in partial fulfillment of the requirement of the Diploma in Software Engineering of National Institute of Business Management."

"We certify that this project does not incorporate without acknowledgement, any material previously submitted for a Diploma in any institution and to the best of our knowledge and belief, it does not contain any material previously published or written by another person or ourselves except where due reference is made in the text. We also hereby give consent for our project report, if accepted, to be made available for photocopying and for interlibrary loans and for the title and summary to be made available to outside organizations."

Abstract

The object of this project is to develop a writing and drawing machine which can write by a pen or pencil and print images as a drawing artist,

Acknowledgement

We would like to express our gratitude to Mr K.V. Narangoda Lecturer of NIBM and Mr. O.K.G.C. Weerasekara, Director of Diploma in Software Engineering of NIBM-Colombo, who enabled us to complete this project successfully in all aspects. We would like to thank to all the staff members, both teaching and non-teaching who helped us in all possible way to make this project successfully.

Contents

1.Introduction

- CNC Machine means computer numerical control machine.
- Numerical control is a method of automatically operating a machine based on a code of letters, numbers, and special characters.
- The numerical data required to produce a part is provided to a machine in the form of a program called, *CNC program*.
- The program is translated into the appropriate electrical signals for input to motors that run the machine.

1.1. History of CNC

Before CNC there was NC, machines being numerically controlled long before the introduction of the modern computer. The first numerically controlled machines were invented in the 1940's and 1950's, these machines now being obsolete and having been replaced by computers. CNC first came about in the late 1950's, the first CNC engineered parts being manufactured for the aerospace industry after the Second World War. These first computers were very rudimentary when compared with the computers we find today, a slow but steady progression over the decades seeing more and more computers being used in everyday industries all over the globe.

1.2. What is CNC Engineering? Modern CNC

Modern CNC engineering is a specialty that is found within the field of mechanical and precision engineering, and nowadays it accounts for the production of a hugely wide and varied range of parts for hundreds of different industries. Modern CNC can mass produce parts to a very high level of precision, CNC lathes, cutters, milling machine, and plasma cutters being used.

When you consider how a factory floor looks today in comparison with what it looked like half a century ago, you will notice a stark contrast from one that was brimming with human activity to one that has relatively no human presence at all. Manpower is no longer used to mill or turn parts and components; CNC machines being programmed to do the job to a much greater level of accuracy and to a much-increased speed.

CNC machining can include the production of anything as simple as a nut and bolt, to anything as complex as a metal part for a modern aircraft. The machines used to mill and turn these parts are advancing in technology on almost a daily basis, and the real masterminds behind these machines are the CAD software specialists.

Although the factory floor may nowadays have far less human workers on it, behind each and every machine that is working is a CNC expert who is experienced in CAD technology. CAD stands for computer aided design, and the design for all the parts produced on the factory floor will have been designed by a CAD expert. CAD design allows for not just 3 dimensional but 4 dimensional images to be displayed on the computer screen, software then being used to convey messages to the CNC machines which in turn produce exact replicas of the image displayed on the screen.

By altering the image on the computer, one single CNC machine can produce an unlimited number of different parts, made from metal, wood, glass, and other materials. These machines make no errors, are controlled by highly trained CNC and CAD experts, and allow for the mass production of the very highest quality of precision engineered parts.

2.Why Use CNC Machines?

- Increase production throughput
- Improve the quality and accuracy of manufactured parts
- Stabilize manufacturing costs
- Manufacture complex or otherwise impossible jobs 2D and 3D model

3.Our machine

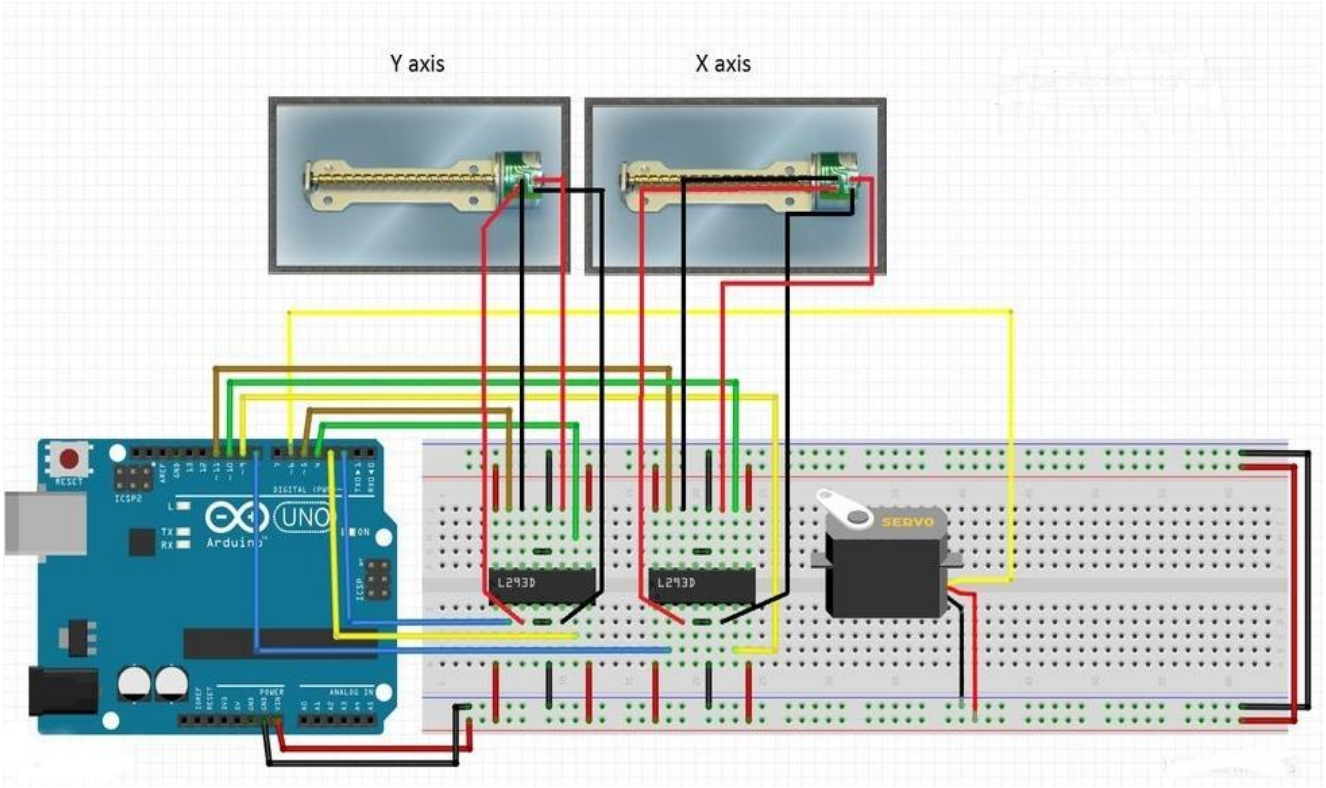
- Our machine understands G-code language which is commonly used in CNC's World.
  - We can convert any picture, character and shape into G-code.

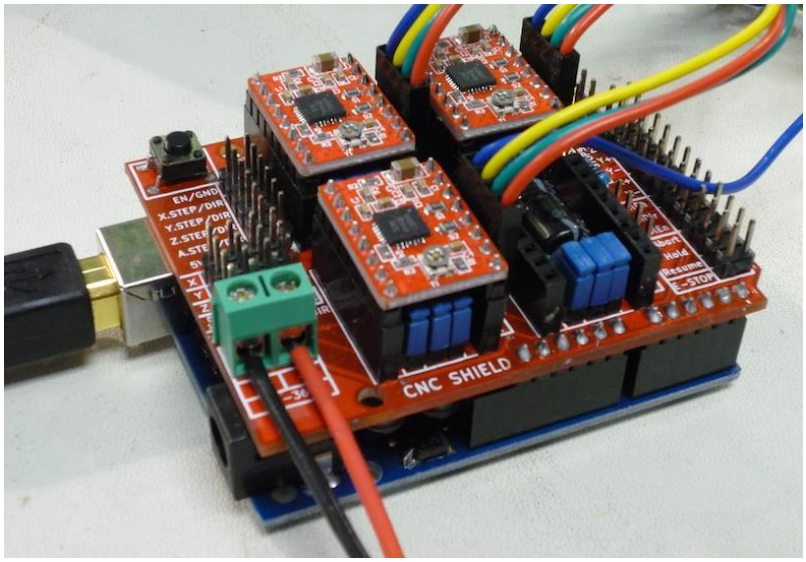
- We can print any character and logos
- We can cut woods with a drill motor
- Can use as a rigifoam cutter
- Take picture, edit it and draw
- Type words, add front styles and draw
- Draw simple shapes

4.Hardware Part

- three movable Axis (x, y, and z)
- 12W power supply
- Metal body
- three Stepper motors
- Driver circuit for steppers
- Arduino Uno board
- CNC shield
- Cutting motor
- Drill chuck and a bit
- Element of a 12v bulb with a bit
- Pen or a pencil

4.1. Basic Circuit Diagram





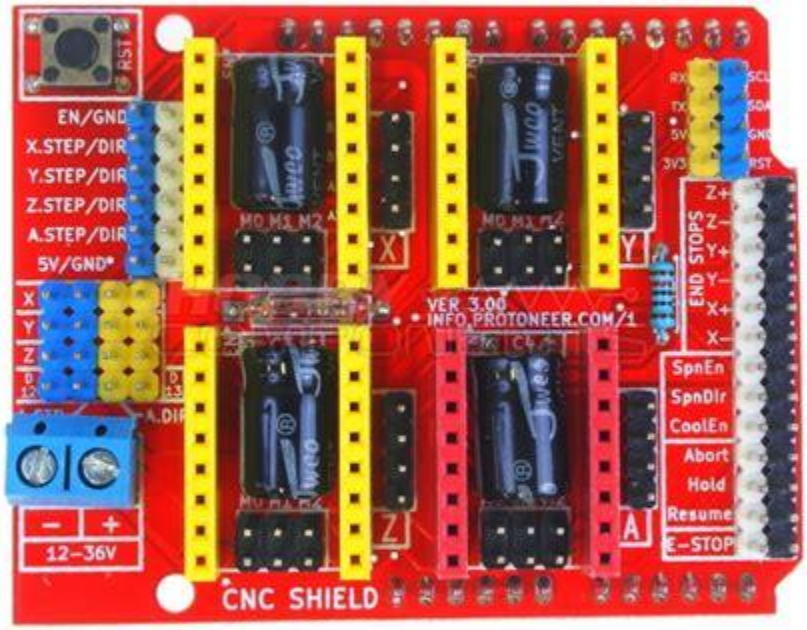
Connecting motors to shield

4.2. Arduino Uno

The Arduino UNO is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with sets of digital and analog input/output pins that may be interfaced to various expansion boards and other circuits.

- CNC shield can only be plugged into Arduino Uno board

4.3. CNC Shield v3

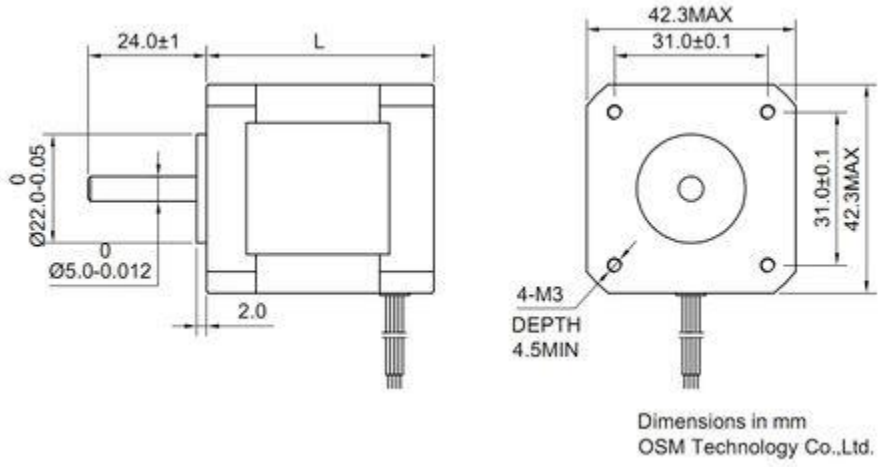


The Arduino CNC Shield V3, is an Arduino compatible board that turns Arduino into a CNC controller. Using an open source firmware, it can control up to 4 Stepper motor using DRV8825 or A4988 stepper motor driver making it easy to get your CNC projects up and running in a few hours.

Features:

- GRBL 0.8c compatible. (Open source firmware that runs on an Arduino UNO that turns G-code commands into stepper signals)
- 4-Axis support (X, Y, Z, A-Can duplicate X,Y,Z or do a full 4th axis with custom firmware using pins D12 and D13)
- 2 x End stops for each axis (6 in total)
- Spindle enable and direction
- Coolant enables
- Uses removable Pololu A4988 compatible stepper drivers. (A4988, DRV8825 and others)
- Jumpers to set the Micro-Stepping for the stepper drivers. (Some drivers like the DRV8825 can do up to 1/32 micro-stepping)
- Compact design.
- Stepper Motors can be connected with 4 pin Molex connectors.
- Runs on 12-36V DC. (At the moment only the Pololu DRV8825 drivers can handle up to 36V so please consider the operation voltage when powering the board.)

4.4. Nema 17 stepper motor



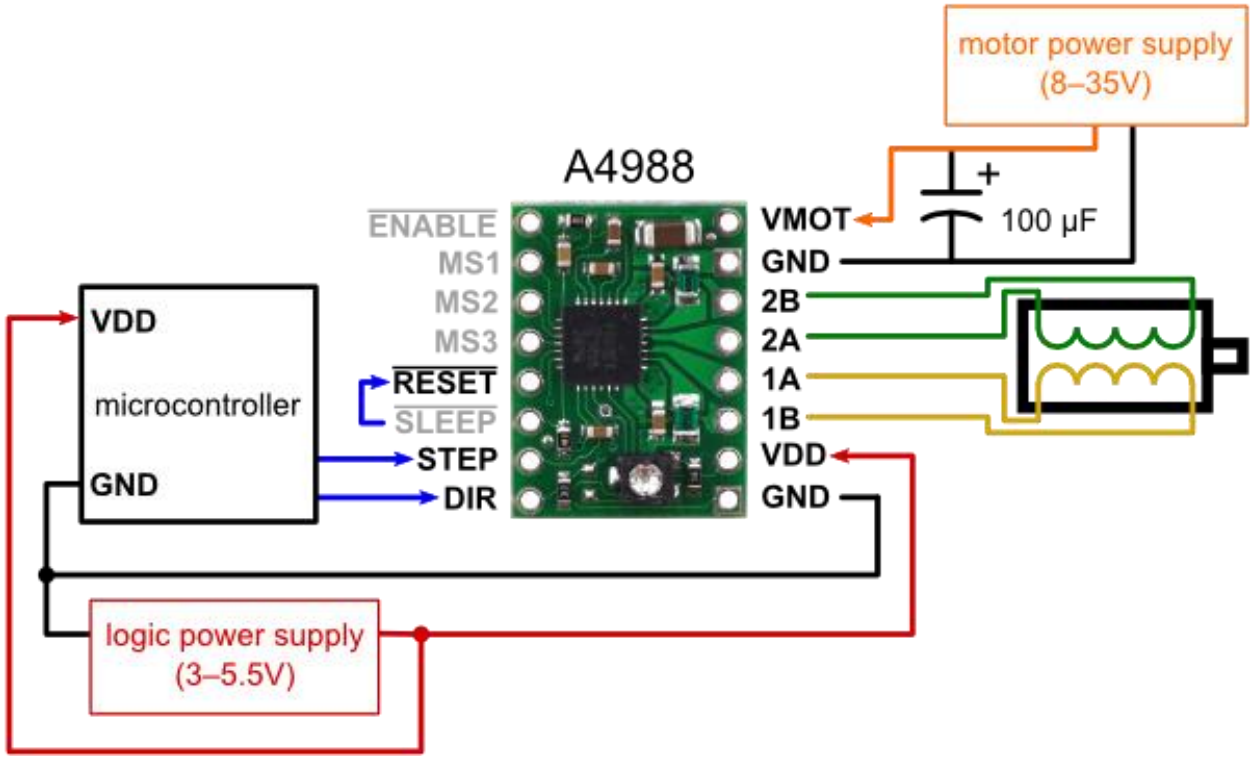
- A NEMA 17 stepper motor is a stepper motor with a 1.7 x 1.7 inch (43.2 x 43.2 mm) faceplate. The NEMA 17 is larger and generally heavier than for example a NEMA 14, but this also means it has more room to put a higher torque. However, its size is not an indication of its power.

Common Stepper Motor Models

- The most commonly used stepper motors in Reprap–based 3D printers are the Kysan 1124090/42BYGH4803, Rattm 17HS8401, and Wantai 42BYGHW609.

- However, motors close to NEMA 17 size, with approximately the following specifications, can also work:
- 1.5A to 1.8A current per phase
- 1-4 volts
- 3 to 8 mH inductance per phase
- 44 N·cm (62oz·in, 4.5kg·cm) or more holding torque
- 1.8 or 0.9 degrees per step (200/400 steps/rev respectively)

4.5. Motor Driver A988



- Simple step and direction control interface
- Five different step resolutions: full-step, half-step, quarter-step, eighth-step, and sixteenth-step
- Adjustable current control lets you set the maximum current output with a potentiometer, which lets you use voltages above your stepper motor's rated voltage to achieve higher step rates
- Intelligent chopping control that automatically selects the correct current decay mode (fast decay or slow decay)
- Over-temperature thermal shutdown, under-voltage lockout, and crossover-current protection
- Short-to-ground and shorted-load protection
-

Step (and microstep) size

- Stepper motors typically have a step size specification (e.g. 1.8° or 200 steps per revolution), which applies to full steps. A microstepping driver such as the A4988 allows higher resolutions by allowing intermediate step locations, which are achieved by energizing the coils with intermediate current levels. For instance, driving a motor in quarter-step mode will give the 200-step-per-revolution motor 800 microsteps per revolution by using four different current levels.

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

- The resolution (step size) selector inputs (MS1, MS2, and MS3) enable selection from the five step resolutions according to the table below. MS1 and MS3 have internal 100kΩ pull-down resistors and MS2 has an internal 50kΩ pull-down resistor, so leaving these three microstep selection pins disconnected results in full-step mode.
- For the microstep modes to function correctly, the current limit must be set low enough (see below) so that current limiting gets engaged. Otherwise, the intermediate current levels will not be correctly maintained, and the motor will skip microsteps.

Control inputs

- Each pulse to the STEP input corresponds to one microstep of the stepper motor in the direction selected by the DIR pin. Note that the STEP and DIR pins are not pulled to any particular voltage internally, so you should not leave either of these pins floating in your application. If you just want rotation in a single direction, you can tie DIR directly to VCC or GND. The chip has three different inputs for controlling its many power states: RST, SLP, and EN. For details about these power states, see the datasheet. Please note that the RST pin is floating; if you are not using the pin, you can connect it to the adjacent SLP pin on the PCB to bring it high and enable the board

4.6. Power supply

Power supply will convert 230 A/C current into 12V 2A output by connecting the com line and ps-on line together.

4.7. Other Components

GT2 Synchronous timing belt

- width: 6mm
- length: 4m

GT2 Timing Pulley

Bore Size	5 mm
Belt Size	6 mm
No of Teeth	20

Iron Box Bar

iron round Bar

5.Software- C#

- Main process in C# application is to Get the user inputs for drawing, convert it to G-Code read the text file which contain the G-Code and send the values to Arduino, then the microcontroller gets the values and move the motors according to the values received.
- Image tuning
    - Change basic tunings of an image
    - Convert to black and white
    - Add dither effect
    - Change size and rotation
- Text formatting
    - Change font
    - Set character distance, height
- Create shapes
    - Set width, height or radius
- Capture picture(developing)
    - Select video source
    - Shape recognition (for PCB drawing)
    - Zooming

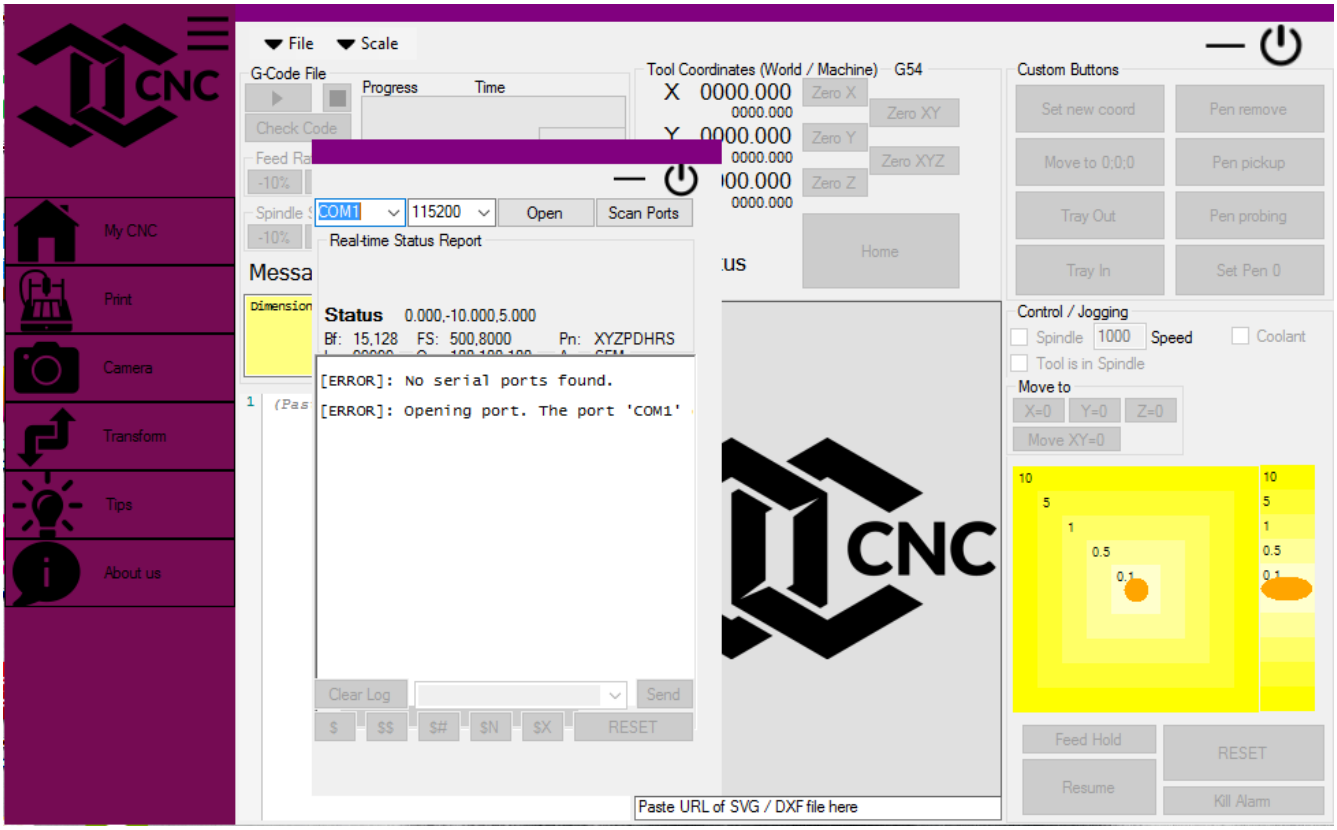


Figure: Main form window

5.1. G-code

G-codes, also called preparatory codes, are any word in a CNC program that begins with the letter G. Generally, it is a code telling the machine tool what type of action to perform, such as:

- Rapid movement (transport the tool as quickly as possible in between cuts)
- Controlled feed in a straight line or arc
- Series of controlled feed movements that would result in a hole being bored, a workpiece cut (routed) to a specific dimension, or a profile (contour) shape added to the edge of a workpiece
- Set tool information such as offset
- Switch coordinate systems

There are other codes; the type codes can be thought of like registers in a computer.

People have pointed out over the years that the term "G-code" is imprecise. It comes from the literal sense of the term, referring to one letter address and to the specific codes that can be formed with it (for example, G00, G01, G28). But every letter of the English alphabet is used somewhere in the language. Nevertheless, "G-code" is established as the common name of the language.

General G-codes

Some letter addresses are used only in milling or only in turning; most are used in both. Below are the letters seen most frequently throughout a program.

Variable	**Description**	**Corollary info**
A	Absolute or incremental position of A axis (rotational axis around X axis)	Positive rotation is defined as a counterclockwise rotation looking from X positive towards X negative.
B	Absolute or incremental position of B axis (rotational axis around Y axis)	
C	Absolute or incremental position of C axis (rotational axis around Z axis)	
D	Defines diameter or radial offset used for cutter compensation. D is used for depth of cut on lathes. It is used for aperture selection and commands on photoplotters.	G41: left cutter compensation, G42: right cutter compensation

E	Precision feedrate for threading on lathes	
F	Defines feed rate	Common units are distance per time for mills (inches per minute, IPM, or millimeters per minute, mm/min) and distance per revolution for lathes (inches per revolution, IPR, or millimeters per revolution, mm/rev)
G	Address for preparatory commands	G commands often tell the control what kind of motion is wanted (e.g., rapid positioning, linear feed, circular feed, fixed cycle) or what offset value to use.
H	Defines tool length offset; Incremental axis corresponding to C axis (e.g., on a turn-mill)	G43: Negative tool length compensation, G44: Positive tool length compensation
I	Defines arc center in X axis for G02 or G03 arc commands. Also used as a parameter within some fixed cycles.	The arc center is the relative distance from the current position to the arc center, not the absolute distance from the work coordinate system (WCS).
J	Defines arc center in Y axis for G02 or G03 arc commands. Also used as a parameter within some fixed cycles.	Same corollary info as I above.
K	Defines arc center in Z axis for G02 or G03 arc commands. Also used as a parameter within some fixed cycles, equal to L address.	Same corollary info as I above.
L	Fixed cycle loop count; Specification of what register to edit using G10	*Fixed cycle loop count:* Defines number of repetitions ("loops") of a fixed cycle at *each* position. Assumed to be 1 unless programmed with another integer. Sometimes the K address is used instead of L. With incremental positioning (G91), a series of equally spaced holes can be programmed as a loop rather than as individual positions. *G10 use:* Specification of what register to edit (work offsets, tool radius offsets, tool length offsets, etc.).
M	Miscellaneous function	Action code, auxiliary command; descriptions vary. Many M-codes call for machine functions, which is why people often say that the "M" stands for "machine", although it was not intended to.

N	Line (block) number in program; System parameter number to change using G10	*Line (block) numbers:* Optional, so often omitted. Necessary for certain tasks, such as M99 P address (to tell the control which block of the program to return to if not the default) or GoTo statements (if the control supports those). N numbering need not increment by 1 (for example, it can increment by 10, 20, or 1000) and can be used on every block or only in certain spots throughout a program. *System parameter number:* G10 allows changing of system parameters under program control.
O	Program name	For example, O4501. For many years it was common for CNC control displays to use slashed zero glyphs to ensure effortless distinction of letter "O" from digit "0". Today's GUI controls often have a choice of fonts, like a PC does.
P	Serves as parameter address for various G and M codes	• With G04, defines dwell time value. • Also serves as a parameter in some canned cycles, representing dwell times or other variables. • Also used in the calling and termination of subprograms. (With M98, it specifies which subprogram to call; with M99, it specifies which block number of the main program to return to.)
Q	Peck increment in canned cycles	For example, G73, G83 (peck drilling cycles)
R	Defines size of arc radius, or defines retract height in milling canned cycles	For radii, not all controls support the R address for G02 and G03, in which case IJK vectors are used. For retract height, the "R level", as it's called, is returned to if G99 is programmed.
S	Defines speed, either spindle speed or surface speed depending on mode	Data type = integer. In G97 mode (which is usually the default), an integer after S is interpreted as a number of rev/min (rpm). In G96 mode (CSS), an integer after S is interpreted as surface speed—sfm (G20) or m/min (G21). See also Speeds and feeds. On multifunction (turn-mill or mill-turn) machines, which spindle gets the input (main spindle or subspindles) is determined by other M codes.
T	Tool selection	To understand how the T address works and how it interacts (or not) with M06, one must study the various methods, such as lathe turret programming, ATC fixed tool selection, ATC random memory tool selection, the concept of "next tool waiting", and empty tools. Programming on any particular machine tool requires knowing which method that machine uses.

U	Incremental axis corresponding to X axis (typically only lathe group A controls) Also defines dwell time on some machines (instead of "P" or "X").	In these controls, X and U obviate G90 and G91, respectively. On these lathes, G90 is instead a fixed cycle address for roughing.
V	Incremental axis corresponding to Y axis	Until the 2000s, the V address was very rarely used, because most lathes that used U and W didn't have a Y-axis, so they didn't use V. (Green *et al.* 1996 did not even list V in their table of addresses.) That is still often the case, although the proliferation of live lathe tooling and turn-mill machining has made V address usage less rare than it used to be (Smid 2008 shows an example). See also G18.
W	Incremental axis corresponding to Z axis (typically only lathe group A controls)	In these controls, Z and W obviate G90 and G91, respectively. On these lathes, G90 is instead a fixed cycle address for roughing.
X	Absolute or incremental position of X axis. Also defines dwell time on some machines (instead of "P" or "U").	
Y	Absolute or incremental position of Y axis	
Z	Absolute or incremental position of Z axis	The main spindle's axis of rotation often determines which axis of a machine tool is labeled as Z.

List of G-codes commonly found on FANUC and similarly designed controls for milling and turning

Note: ***Modal*** means a code stays in effect until replaced, or cancelled, by another permitted code. ***Non-Modal*** means it executes only once. See, for example, codes G09, G61 & G64 below.

Code	Description	Milling (M)	Turning (T)	Corollary info
G00	Rapid positioning	M	T	On 2- or 3-axis moves, G00 (unlike G01) traditionally does not necessarily move in a single straight line between start point and end point. It moves each axis at its max speed until its vector quantity is achieved. Shorter vector usually finishes first (given similar axis speeds). This matters because it may yield a dog-leg or hockey-stick

				motion, which the programmer needs to consider, depending on what obstacles are nearby, to avoid a crash. Some machines offer interpolated rapids as a feature for ease of programming (safe to assume a straight line).
G01	Linear interpolation	M	T	The most common workhorse code for feeding during a cut. The program specs the start and end points, and the control automatically calculates (interpolates) the intermediate points to pass through that yield a straight line (hence "linear"). The control then calculates the angular velocities at which to turn the axis leadscrews via their servomotors or stepper motors. The computer performs thousands of calculations per second, and the motors react quickly to each input. Thus the actual toolpath of the machining takes place with the given feedrate on a path that is accurately linear to within very small limits.
G02	Circular interpolation, clockwise	M	T	Very similar in concept to G01. Again, the control interpolates intermediate points and commands the servo- or stepper motors to rotate the amount needed for the leadscrew to translate the motion to the correct tool tip positioning. This process repeated thousands of times per minute generates the desired toolpath. In the case of G02, the interpolation generates a circle rather than a line. As with G01, the actual toolpath of the machining takes place with the given feedrate on a path that accurately matches the ideal (in G02's case, a circle) to within very small limits. In fact, the interpolation is so precise (when all conditions are correct) that milling an interpolated circle can obviate operations such as drilling, and often even fine boring. **Addresses for radius or arc center:** G02 and G03 take either an R address (for the radius desired on the part) or IJK addresses (for the component vectors that define the vector from the arc start point to the arc center point). **Cutter comp:** On most controls you cannot start G41 or G42 in G02 or G03 modes. You must already have compensated in an earlier G01 block. Often, a short linear lead-in movement is programmed, merely to allow cutter compensation before the main action, the circle-cutting, begins. **Full circles:** When the arc start point and the arc end point are identical, the tool cuts a 360° arc (a full circle). (Some older controls do not support this because arcs cannot cross between quadrants of the cartesian system. Instead, they require four quarter-circle arcs programmed back-to-back.)

G03	Circular interpolation, counterclockwise	M	T	Same corollary info as for G02.
G04	Dwell	M	T	Takes an address for dwell period (may be X, U, or P). The dwell period is specified by a control parameter, typically set to milliseconds. Some machines can accept either X1.0 (s) or P1000 (ms), which are equivalent. **Choosing dwell duration**: Often the dwell needs only to last one or two full spindle rotations. This is typically much less than one second. Be aware when choosing a duration value that a long dwell is a waste of cycle time. In some situations it won't matter, but for high-volume repetitive production (over thousands of cycles), it is worth calculating that perhaps you only need 100 ms, and you can call it 200 to be safe, but 1000 is just a waste (too long).
G05 P10000	High-precision contour control (HPCC)	M		Uses a deep look-ahead buffer and simulation processing to provide better axis movement acceleration and deceleration during contour milling
G05.1 Q1.	AI Advanced Preview Control	M		Uses a deep look-ahead buffer and simulation processing to provide better axis movement acceleration and deceleration during contour milling
G06.1	Non-uniform rational B-spline (NURBS) Machining	M		Activates Non-Uniform Rational B Spline for complex curve and waveform machining (this code is confirmed in Mazatrol 640M ISO Programming)
G07	Imaginary axis designation	M		
G09	Exact stop checks, non-modal	M	T	The modal version is G61.
G10	Programmable data input	M	T	Modifies the value of work coordinate and tool offsets
G11	Data write cancel	M	T	
G17	XY plane selection	M		
G18	ZX plane selection	M	T	
G19	YZ plane selection	M		
G20	Programming in inches	M	T	Somewhat uncommon except in USA and (to lesser extent) Canada and UK. However, in the global marketplace, competence with both G20 and G21 always stands some chance of being necessary at any time. The usual minimum increment in G20 is one ten-thousandth of an inch (0.0001"), which is a larger distance than the usual minimum increment in G21 (one thousandth of a millimeter, .001 mm, that

				is, one micrometer). This physical difference sometimes favors G21 programming.
G21	Programming in millimeters (mm)	M	T	Prevalent worldwide. However, in the global marketplace, competence with both G20 and G21 always stands some chance of being necessary at any time.
G28	Return to home position (machine zero, aka machine reference point)	M	T	Takes X Y Z addresses which define the intermediate point that the tool tip will pass through on its way home to machine zero. They are in terms of part zero (aka program zero), NOT machine zero.
G30	Return to secondary home position (machine zero, aka machine reference point)	M	T	Takes a P address specifying *which* machine zero point to use *if* the machine has several secondary points (P1 to P4). Takes X Y Z addresses that define the intermediate point that the tool tip passes through on its way home to machine zero. These are expressed in terms of part zero (aka program zero), NOT machine zero.
G31	Feed until skip function	M		Used for probes and tool length measurement systems.
G32	Single-point threading, longhand style (if not using a cycle, e.g., G76)		T	Similar to G01 linear interpolation, except with automatic spindle synchronization for single-point threading.
G33	Constant-pitch threading	M		
G33	Single-point threading, longhand style (if not using a cycle, e.g., G76)		T	Some lathe controls assign this mode to G33 rather than G32.
G54	Variable-pitch threading	M		
G40	Tool radius compensation off	M	T	Turn off cutter radius compensation (CRC). Cancels G41 or G42.
G41	Tool radius compensation left	M	T	Turn on cutter radius compensation (CRC), left, for climb milling. **Milling:** Given righthand-helix cutter and M03 spindle direction, G41 corresponds to climb milling (down milling). Takes an address (D or H) that calls an offset register value for radius. **Turning:** Often needs no D or H address on lathes, because whatever tool is active automatically calls its geometry offsets with it. (Each turret station is bound to its geometry offset register.) G41 and G42 for milling has been partially automated and obviated (although not completely) since CAM programming has

				become more common. CAM systems let the user program as if using a zero-diameter cutter. The fundamental concept of cutter radius compensation is still in play (i.e., that the surface produced will be distance R away from the cutter center), but the programming mindset is different. The human does not choreograph the toolpath with conscious, painstaking attention to G41, G42, and G40, because the CAM software takes care of that. The software has various CRC mode selections, such as *computer*, *control*, *wear*, *reverse wear*, *off*, some of which do not use G41/G42 at all (good for roughing, or wide finish tolerances), and others that use it so that the wear offset can still be tweaked at the machine (better for tight finish tolerances).
G42	Tool radius compensation right	M	T	Turn on cutter radius compensation (CRC), right, for conventional milling. Similar corollary info as for G41. Given righthand-helix cutter and M03 spindle direction, G42 corresponds to conventional milling (up milling).
G43	Tool height offset compensation negative	M		Takes an address, usually H, to call the tool length offset register value. The value is *negative* because it will be *added* to the gauge line position. G43 is the commonly used version (vs G44).
G44	Tool height offset compensation positive	M		Takes an address, usually H, to call the tool length offset register value. The value is *positive* because it will be *subtracted* from the gauge line position. G44 is the seldom-used version (vs G43).
G45	Axis offset single increase	M		
G46	Axis offset single decrease	M		
G47	Axis offset double increase	M		
G48	Axis offset double decrease	M		
G49	Tool length offset compensation cancel	M		Cancels G43 or G44.
G50	Define the maximum spindle speed		T	Takes an S address integer, which is interpreted as rpm. Without this feature, G96 mode (CSS) would rev the spindle to "wide open throttle" when closely approaching the axis of rotation.
G50	Scaling function cancel	M		

G50	Position register (programming of vector from part zero to tool tip)		T	Position register is one of the original methods to relate the part (program) coordinate system to the tool position, which indirectly relates it to the machine coordinate system, the only position the control really "knows". Not commonly programmed anymore because G54 to G59 (WCSs) are a better, newer method. Called via G50 for turning, G92 for milling. Those G addresses also have alternate meanings (*which see*). Position register can still be useful for datum shift programming. The "manual absolute" switch, which has very few useful applications in WCS contexts, was more useful in position register contexts, because it allowed the operator to move the tool to a certain distance from the part (for example, by touching off a 2.0000" gage) and then declare to the control what the distance-to-go shall be (2.0000).
G52	Local coordinate system (LCS)	M		Temporarily shifts program zero to a new location. It is simply "an offset from an offset", that is, an additional offset added onto the WCS offset. This simplifies programming in some cases. The typical example is moving from part to part in a multipart setup. With **G54** active, **G52 X140.0 Y170.0** shifts program zero 140 mm over in X and 170 mm over in Y. When the part "over there" is done, **G52 X0 Y0** returns program zero to normal G54 (by reducing G52 offset to nothing). The same result can also be achieved (1) using multiple WCS origins, G54/G55/G56/G57/G58/G59; (2) on newer controls, G54.1 P1/P2/P3/etc. (all the way up to P48); or (3) using G10 for programmable data input, in which the program can write new offset values to the offset registers. The method to use depends on shop-specific application.
G53	Machine coordinate system	M	T	Takes absolute coordinates (X,Y,Z,A,B,C) with reference to machine zero rather than program zero. Can be helpful for tool changes. Nonmodal and absolute only. Subsequent blocks are interpreted as "back to G54" even if it is not explicitly programmed.
G54 to G59	Work coordinate systems (WCSs)	M	T	Have largely replaced position register (G50 and G92). Each tuple of axis offsets relates program zero directly to machine zero. Standard is 6 tuples (G54 to G59), with optional extensibility to 48 more via G54.1 P1 to P48.
G54.1 P1 to P48	Extended work coordinate systems	M	T	Up to 48 more WCSs besides the 6 provided as standard by G54 to G59. Note floating-point extension of G-code data type (formerly all integers). Other examples have also evolved (e.g.,

				G84.2). Modern controls have the hardware to handle it.
G61	Exact stop check, modal	M	T	Can be canceled with G64. The non-modal version is G09.
G62	Automatic corner override	M	T	
G64	Default cutting mode (cancel exact stop check mode)	M	T	Cancels G61.
G68	Rotate coordinate system	M		Rotates coordinate system in the current plane given with G17, G18, or G19. Center of rotation is given with two parameters, which vary with each vendor's implementation. Rotate with angle given with argument R. This can be used, for instance, to align the coordinate system with a misaligned part. It can also be used to repeat movement sequences around a center. Not all vendors support coordinate system rotation.
G69	Turn off coordinate system rotation	M		Cancels G68.
G70	Fixed cycle, multiple repetitive cycle, for finishing (including contours)		T	
G71	Fixed cycle, multiple repetitive cycle, for roughing (Z-axis emphasis)		T	
G72	Fixed cycle, multiple repetitive cycle, for roughing (X-axis emphasis)		T	
G73	Fixed cycle, multiple repetitive cycle, for roughing, with pattern repetition		T	
G73	Peck drilling cycle for milling – high-speed (NO full retraction from pecks)	M		Retracts only as far as a clearance increment (system parameter). For when chipbreaking is the main concern, but chip clogging of flutes is not. Compare G83.

G74	Peck drilling cycle for turning		T	
G74	Tapping cycle for milling, lefthand thread, M04 spindle direction	M		See notes at G84.
G75	Peck grooving cycle for turning		T	
G76	Fine boring cycle for milling	M		Includes OSS and shift (oriented spindle stop and shift tool off centerline for retraction)
G76	Threading cycle for turning, multiple repetitive cycle		T	
G80	Cancel canned cycle	M	T	**Milling:** Cancels all cycles such as G73, G81, G83, etc. Z-axis returns either to Z-initial level or R level, as programmed (G98 or G99, respectively). **Turning:** Usually not needed on lathes, because a new group-1 G address (G00 to G03) cancels whatever cycle was active.
G81	Simple drilling cycle	M		No dwell built in
G82	Drilling cycle with dwell	M		Dwells at hole bottom (Z-depth) for the number of milliseconds specified by the P address. Good for when hole bottom finish matters. Good for spot drilling because the divot is certain to clean up evenly. Consider the "choosing dwell duration" note at G04.
G83	Peck drilling cycle (full retraction from pecks)	M		Returns to R-level after each peck. Good for clearing flutes of chips. Compare G73.
G84	Tapping cycle, righthand thread, M03 spindle direction	M		G74 and G84 are the righthand and lefthand "pair" for old-school tapping with a non-rigid toolholder ("tapping head" style). Compare the rigid tapping "pair", G84.2 and G84.3.
G84.2	Tapping cycle, righthand thread, M03 spindle direction, rigid toolholder	M		See notes at G84. Rigid tapping synchronizes speed and feed according to the desired thread helix. That is, it synchronizes degrees of spindle rotation with microns of axial travel. Therefore, it can use a rigid toolholder to hold the tap. This feature is not available on old machines or newer low-end machines, which must use "tapping head" motion (G74/G84).
G84.3	Tapping cycle, lefthand thread, M04 spindle direction, rigid toolholder	M		See notes at G84 and G84.2.

G85	boring cycle, feed in/feed out	M		• Good cycle for a reamer. • In some cases good for single-point boring tool, although in other cases the lack of depth of cut on the way back out is bad for surface finish, in which case, G76 (OSS/shift) can be used instead. • If need dwell at hole bottom, see G89.
G86	boring cycle, feed in/spindle stop/rapid out	M		Boring tool leaves a slight score mark on the way back out. Appropriate cycle for some applications; for others, G76 (OSS/shift) can be used instead.
G87	boring cycle, backboring	M		For backboring. Returns to initial level only (G98); this cycle cannot use G99 because its R level is on the far side of the part, away from the spindle headstock.
G88	boring cycle, feed in/spindle stop/manual operation	M		
G89	boring cycle, feed in/dwell/feed out	M		G89 is like G85 but with dwell added at bottom of hole.
G90	Absolute programming	M	T (B)	Positioning defined with reference to part zero. **Milling:** Always as above. **Turning:** Sometimes as above (Fanuc group type B and similarly designed), but on most lathes (Fanuc group type A and similarly designed), G90/G91 are not used for absolute/incremental modes. Instead, U and W are the incremental addresses and X and Z are the absolute addresses. On these lathes, G90 is instead a fixed cycle address for roughing.
G90	Fixed cycle, simple cycle, for roughing (Z-axis emphasis)		T (A)	When not serving for absolute programming (above)
G91	Incremental programming	M	T (B)	Positioning defined with reference to previous position. **Milling:** Always as above. **Turning:** Sometimes as above (Fanuc group type B and similarly designed), but on most lathes (Fanuc group type A and similarly designed), G90/G91 are not used for absolute/incremental modes. Instead, U and W are the incremental addresses and X and Z are the absolute addresses. On these lathes, G90 is a fixed cycle address for roughing.
G92	Position register (programming of vector from part zero to tool tip)	M	T (B)	Same corollary info as at G50 position register. **Milling:** Always as above. **Turning:** Sometimes as above (Fanuc group type B and similarly designed), but on most lathes (Fanuc

				group type A and similarly designed), position register is G50.
G92	Threading cycle, simple cycle		T (A)	
G94	Feedrate per minute	M	T (B)	On group type A lathes, feedrate per minute is G98.
G94	Fixed cycle, simple cycle, for roughing (X-axis emphasis)		T (A)	When not serving for feedrate per minute (above)
G95	Feedrate per revolution	M	T (B)	On group type A lathes, feedrate per revolution is G99.
G96	Constant surface speed (CSS)		T	Varies spindle speed automatically to achieve a constant surface speed. See speeds and feeds. Takes an S address integer, which is interpreted as sfm in G20 mode or as m/min in G21 mode.
G97	Constant spindle speed	M	T	Takes an S address integer, which is interpreted as rev/min (rpm). The default speed mode per system parameter if no mode is programmed.
G98	Return to initial Z level in canned cycle	M		
G98	Feedrate per minute (group type A)		T (A)	Feedrate per minute is G94 on group type B.
G99	Return to R level in canned cycle	M		
G99	Feedrate per revolution (group type A)		T (A)	Feedrate per revolution is G95 on group type B.
G100	Tool length measurement	M		

List of M-codes commonly found on FANUC and similarly designed controls for milling and turning

Some older controls require M codes to be in separate blocks (that is, not on the same line).

Code	**Description**	**Milling (M)**	**Turning (T)**	**Corollary info**
M00	Compulsory stop	M	T	Non-optional—machine always stops on reaching M00 in the program execution.
M01	Optional stop	M	T	Machine only stops at M01 if operator pushes the optional stop button.
M02	End of program	M	T	Program ends; execution may or may not return to program top (depending on the control); may or may not reset register values. M02 was the original program-end code, now considered obsolete, but still supported for backward

				compatibility. Many modern controls treat M02 as equivalent to M30. See M30 for additional discussion of control status upon executing M02 or M30.
M03	Spindle on (clockwise rotation)	M	T	The speed of the spindle is determined by the address S, in either revolutions per minute (G97 mode; default) or surface feet per minute or [surface] meters per minute (G96 mode [CSS] under either G20 or G21). The right-hand rule can be used to determine which direction is clockwise and which direction is counter-clockwise. Right-hand-helix screws moving in the tightening direction (and right-hand-helix flutes spinning in the cutting direction) are defined as moving in the M03 direction, and are labeled "clockwise" by convention. The M03 direction is always M03 regardless of local vantage point and local CW/CCW distinction.
M04	Spindle on (counterclockwise rotation)	M	T	See comment above at M03.
M05	Spindle stop	M	T	
M06	Automatic tool change (ATC)	M	T (some-times)	Many lathes do not use M06 because the T address itself indexes the turret. Programming on any particular machine tool requires knowing which method that machine uses. To understand how the T address works and how it interacts (or not) with M06, one must study the various methods, such as lathe turret programming, ATC fixed tool selection, ATC random memory tool selection, the concept of "next tool waiting", and empty tools.
M07	Coolant on (mist)	M	T	
M08	Coolant on (flood)	M	T	
M09	Coolant off	M	T	
M10	Pallet clamp on	M		For machining centers with pallet changers
M11	Pallet clamp off	M		For machining centers with pallet changers
M13	Spindle on (clockwise rotation) and coolant on (flood)	M		This one M-code does the work of both M03 and M08. It is not unusual for specific machine models to have such combined commands, which make for shorter, more quickly written programs.
M19	Spindle orientation	M	T	Spindle orientation is more often called within cycles (automatically) or during setup (manually), but it is also available under program control via **M19**. The abbreviation OSS (oriented spindle stop) may be seen in reference to an oriented stop within cycles.

				The relevance of spindle orientation has increased as technology has advanced. Although 4- and 5-axis contour milling and CNC single-pointing have depended on spindle position encoders for decades, before the advent of widespread live tooling and mill-turn/turn-mill systems, it was not as often relevant in "regular" (non-"special") machining for the operator (as opposed to the machine) to know the angular orientation of a spindle as it is today, except in certain contexts (such as tool change, or G76 fine boring cycles with choreographed tool retraction). Most milling of features indexed around a turned workpiece was accomplished with separate operations on indexing head setups; in a sense, indexing heads were originally invented as separate pieces of equipment, to be used in separate operations, which could provide precise spindle orientation in a world where it otherwise mostly didn't exist (and didn't need to). But as CAD/CAM and multiaxis CNC machining with multiple rotary-cutter axes becomes the norm, even for "regular" (non-"special") applications, machinists now frequently care about stepping just about *any* spindle through its 360° with precision.
M21	Mirror, X-axis	M		
M21	Tailstock forward		T	
M22	Mirror, Y-axis	M		
M22	Tailstock backward		T	
M23	Mirror OFF	M		
M23	Thread gradual pullout ON		T	
M24	Thread gradual pullout OFF		T	
M30	End of program, with return to program top	M	T	Today, M30 is considered the standard program-end code, and returns execution to the top of the program. Most controls also still support the original program-end code, M02, usually by treating it as equivalent to M30. **Additional info:** Compare M02 with M30. First, M02 was created, in the days when the punched tape was expected to be short enough to splice into a continuous loop (which is why on old controls, M02 triggered no tape rewinding). The other program-end code,

				M30, was added later to accommodate longer punched tapes, which were wound on a reel and thus needed rewinding before another cycle could start. On many newer controls, there is no longer a difference in how the codes are executed—both act like M30.
M41	Gear select – gear 1		T	
M42	Gear select – gear 2		T	
M43	Gear select – gear 3		T	
M44	Gear select – gear 4		T	
M48	Feedrate override allowed	M	T	MFO (manual feedrate override)
M49	Feedrate override NOT allowed	M	T	Prevent MFO (manual feedrate override). This rule is also usually called (automatically) within tapping cycles or single-point threading cycles, where feed is precisely correlated to speed. Same with SSO (spindle speed override) and feed hold button. Some controls are capable of providing SSO and MFO during threading.
M52	Unload Last tool from spindle	M	T	Also empty spindle.
M60	Automatic pallet change (APC)	M		For machining centers with pallet changers
M98	Subprogram call	M	T	Takes an address P to specify which subprogram to call, for example, "M98 P8979" calls subprogram O8979.
M99	Subprogram end	M	T	Usually placed at end of subprogram, where it returns execution control to the main program. The default is that control returns to the block following the M98 call in the main program. Return to a different block number can be specified by a P address. M99 can also be used in main program with block skip for endless loop of main program on bar work on lathes (until operator toggles block skip).

Example program

This is a generic program that demonstrates the use of G-Code to turn a part that is 1" diameter by 1" long. Assume that a bar of material is in the machine and that the bar is slightly oversized in length and diameter and that the bar protrudes by more than 1" from the face of the chuck.

Block	Code	Description
%		Signals start of data during file transfer. Originally used to stop tape rewind, not necessarily start of program. For some controls (FANUC) the first LF (EOB) is start of program. ISO uses %, EIA uses ER (0x0B).
	O4968 (OPTIONAL PROGRAM DESCRIPTION OR COMMENT)	Sample face and turn program. Comments are enclosed in parentheses.
N01	M216	Turn on load monitor
N02	G20 G90 G54 D200 G40	Inch units. Absolute mode. Activate work offset. Activate tool offset. Deactivate tool nose radius compensation. *Significance:* This block is often called the **safe block** or safety block. Its commands can vary but are usually similar to the ones shown here. The idea is that a safety block should always be given near the top of any program, as a general default, unless some very specific/concrete reason exists to omit it. The safety block is like a sanity check or a preflight checklist: it explicitly ensures conditions that otherwise would be implicit, left merely to assumption. The safety block reduces risk of crashes, and it can also helpfully refocus the thinking of the humans who write or read the program under hurried conditions.
N03	G50 S2000	Set maximum spindle speed in rev/min — This setting affects Constant Surface Speed mode
N04	T0300	Index turret to tool 3. Clear wear offset (00).
N05	G96 S854 M03	Constant surface speed [automatically varies the spindle speed], 854 sfm, start spindle CW rotation
N06	G41 G00 X1.1 Z1.1 T0303 M08	Enable cutter radius compensation mode, rapid position to 0.55" above axial centerline (1.1" in diameter) and 1.1 inches positive from the work offset in Z, activate flood coolant
N07	G01 Z1.0 F.05	Feed in horizontally at rate of 0.050" per revolution of the spindle until the tool is positioned 1" positive from the work offset
N08	X-0.016	Feed the tool slightly past center—the tool must travel by at least its nose radius past the center of the part to prevent a leftover scallop of material.
N09	G00 Z1.1	Rapid positioning; retract to start position
N10	X1.0	Rapid positioning; next pass
N11	G01 Z0.0 F.05	Feed in horizontally cutting the bar to 1" diameter all the way to the datum, 0.05in/rev
N12	G00 X1.1 M05 M09	Clear the part, stop the spindle, turn off the coolant
N13	G91 G28 X0	Home X axis — return the machine's home position for the X axis
N14	G91 G28 Z0	Home Z axis — return to machine's home position for the Z axis
N15	G90	Return to absolute mode. Turn off load monitor
N16	M30	Program stop, rewind to top of program, wait for cycle start
%		Signal end of data during file transfer. Originally used to mark end of tape, not necessarily end of program. ISO uses %, EIA uses ER (0x0B).

G28 – Perform Homing Routine

This command tells the printer to run its homing sequence, which will move the tool head to the far edges of the machine until it contacts the end stops at these locations. Most of your print files will begin with this command so that the printer starts from a known location. This is also a useful way to quickly move one axis out of the way, which may be useful at the end of a print so that you can remove your part.

Arguments:

If no arguments are provided, the machine will home all 3 axes. You can also specify which exact axes you want to home by adding an X, Y, or Z to the command.

Example usage:

```
G28 ; home all axes (X, Y, and Z)
G28 X Y ; home X and Y axes
G28 Z ; home Z axis only
```

G90 and G91 – Set Positioning Mode

Printer can use either absolute or relative positioning. Absolute positioning means that you will be telling your 3D printer to move an exact XYZ coordinate. Relative positioning is used when you want to tell the printer how far it should move from the current location. Send a G90 command to tell your printer to use absolute positioning, or a G91 for relative positioning. The majority of your gcode file will likely use absolute positioning, since the slicer has already determined the exact XYZ coordinates to move to. However, if you don't know the previous position of the tool head, or you simply know that you want to move the head a certain distance along an axis, you can use relative positioning. While G90 and G91 control the positioning mode for the X, Y, and Z axes, you can also use M82 or M83 to set your extruder (E-axis) to absolute or relative positioning.

Example usage:

```
G90 ; use absolute positioning for the XYZ axes
```

```
G1 X10 F3600 ; move to the X=10mm position on the bed
G1 X20 F3600 ; move to X=20mm
G91 ; use relative positioning for the XYZ axes
G1 X10 F3600 ; move 10mm to the right of the current location
G1 X10 F3600 ; move another 10mm to the right
```

G1 – Linear Movement

This command probably makes up 95% of g-code files. The G1 command tells your printer to move in a straight line to the location that you specify. You can use this to move just a single axis, or multiple axes at once. Extruder is controlled just like any other axis, so you can also use this command to extrude or retract filament from the nozzle.

Arguments:

Use X, Y, or Z values to tell the printer what position to move to. These values will obey the current positioning mode, so you can specify them using either absolute or relative coordinates. Include an E value if you want to move the extruder as well. The E value corresponds to the position of your filament spool, so if you move the E axis by 10mm, that would cause 10mm of your filament to be pushed into the nozzle. Since the nozzle diameter is usually much smaller than your filament diameter, 10mm of filament pushed into the nozzle may create an extrusion that is hundreds of millimeters long! For this reason, the E values that you will see in your file are typically quite small compared to the X, Y, and Z values. Finally, you can use an F value to tell the printer what speed to use for the movement. This speed must always have specified in units of mm/min, so even if you use mm/s in your slicing software, you will still need use mm/min anytime you are sending a command directly to the printer.

Most printers support “sticky” coordinates, which means that you only need to specify the arguments for the axes you actually want to move. So if you only wanted to move the Z axis, you would just include the Z argument as well as an F value to define the speed.

Example usage:

```
G1 X0 Y0 F2400 ; move to the X=0 Y=0 position on the bed at a speed of 2400 mm/min
G1 Z10 F1200 ; move the Z-axis to Z=10mm at a slower speed of 1200 mm/min
G1 X30 E10 F1800 ; push 10mm of filament into the nozzle while moving to the X=30
position at the same time
```

G92 – Set Current Position

Use this command to set the current position of your axes. This can be useful if you want to change or offset the location of one of your axes. One of the most common uses for this command is actually with your E axis (the filament position). You can quickly override the current filament position so that all future commands will now be relative to this new value. It is common to do this at the start of each layer or right before a prime or retraction command.

Arguments:

Specify the absolute coordinate for any axis that you wish to overwrite. You can include the X, Y, Z, and E axes. If you do not include one of these axes in the command, the position will remain unchanged.

Example usage:

```
G92 E0 ; set the current filament position to E=0
G1 E10 F800 ; extrude 10mm of filament
```

M104 and M109 – Extruder Heating Commands

Use these commands to set the temperature of your extruder. The M104 command starts heating the extruder, but then allows you to run other commands immediately afterwards. The M109 command will actually wait until the desired temperature is reached before allowing any other commands to run. For this reason, you will frequently see an M109 at the top of your Simplify3D g-code files, as this allows the extruder to reach the necessary temperature before the print begins.

While most machines use M104 and M109, some firmware's may use slightly different commands. For example, if you are using a machine that reads x3g files, then you may use an M133 command for stabilizing your extruder instead of M109.

Arguments:
The S value specifies the extruder temperature in degrees Celsius. The T value can be used if you have more than one extruder, as it allows you to specify which exact extruder temperature you want to set. If you have a dual extrusion machine, typically T0 is the right extruder, and T1 is the left extruder. If you only have a single extruder machine, you can typically omit the T parameter entirely.

Example usage:

```
M104 S190 T0 ; start heating T0 to 190 degrees Celsius
G28 X0 ; home the X axis while the extruder is still heating
M109 S190 T0 ; wait for T0 to reach 190 degrees before continuing with any other commands
```

M140 and M190 – Bed Heating Commands

Use these commands to set the temperature of your heated build platform. The syntax is very similar to the M104 and M109 commands mentioned above. Sending the M140 command begins heating the bed, but allows you to run other commands immediately afterwards. The M190 command will wait until the bed temperature is reached before allowing any other commands to run. The heated bed on your printer may take several minutes to reach elevated temperatures. So don't be surprised if you see your printer pausing while waiting on an M190 command to finish heating the bed. Because this process can take a long time, it may be a good idea to start heating the bed at the beginning of your routine using an M140 command, which would allow you to do other actions such as homing or nozzle purging while the bed is still pre-heating. Just make sure to include an M190 before the print begins, as the bed temperature can be an important factor for first layer adhesion.

As with the M104 and M109 commands, these bed heating commands can differ depending on what firmware you are using. If your machine reads x3g files, then you can use the M134 command for stabilizing your bed instead of M190. If you are using a variant of the Flash Forge Dreamer or Dremel firmware's, you'll want to use an M7 command to stabilize your bed.

Arguments:

The S value specifies the bed temperature in degrees Celsius. No other arguments are typically needed, as most machines only have a single heated build platform.

Example usage:

```
M140 S50 ; start heating the bed to 50 degrees Celsius
G28 ; home all 3 axes while the bed is still heating
M190 S50 ; wait until the bed reaches 50 degrees before continuing
```

M106 – Set Fan Speed

This command allows you to set the speed of your printer's part cooling fan. This is an external cooling fan that is pointed towards the part that you are printing. Keep in mind that your printer may also have an extruder fan that helps cool the extruder drive mechanism, so make sure you are looking at the correct fan first. While most printers have an external cooling fan, there are a few exceptions, so check your machine first to make sure it has an external cooling fan that you can control.

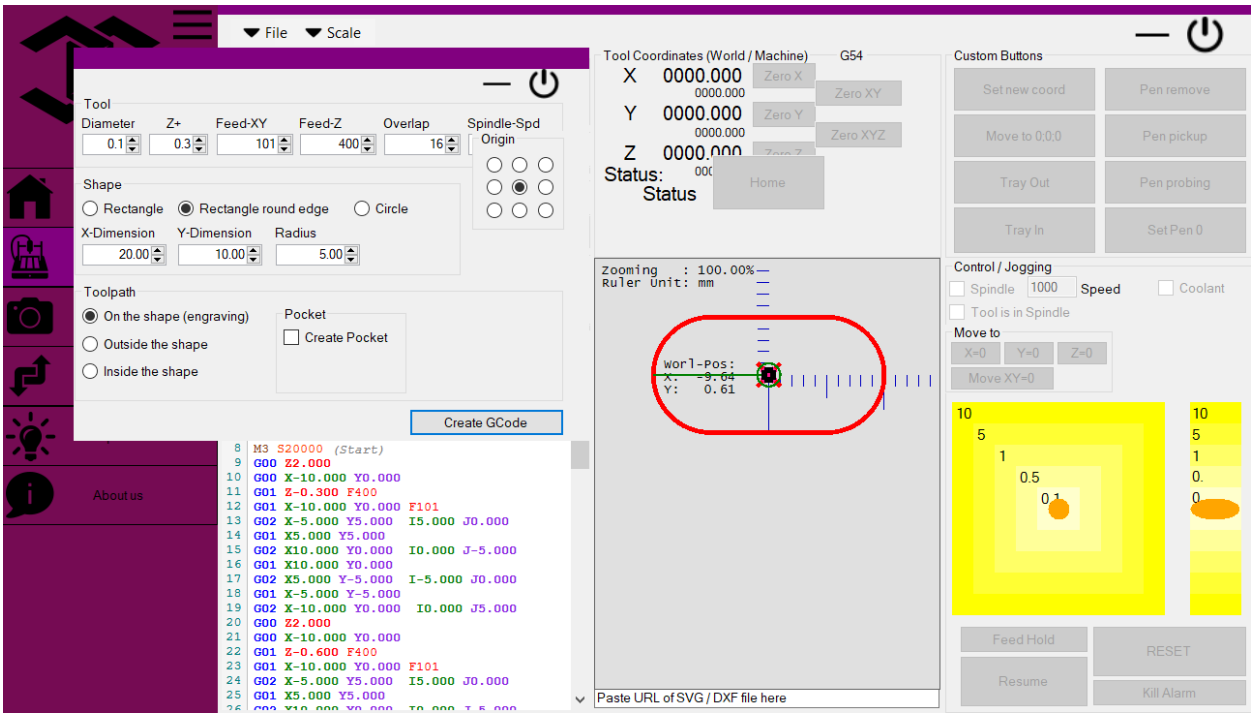
Arguments:

The S value sets the speed of the cooling fan in a range between 0 (off) and 255 (full power).

Example usage:

```
M106 S255 ; set the fan to full speed
M106 S127 ; set the fan to roughly 50% power
M106 S0 ; turn off the fan completely
```

5.2. Create shapes



When creating shapes three types of shapes can be generated by our software. Shapes can be drawn using different types of tools. For an example if we select a pen or a pencil, diameter of the tool is very small and if we select a select a tool like a highlighter which has a large diameter, the tool type can be changed using the software. Also the pen up value and the speed of sending g-code which is known as the xy-feed rate can be changed.

Part of code – Create Shapes

```
private void btnApply_Click(object sender, EventArgs e)
    {
        saveSettings();
        gcode.setup();                  // load defaults from setup-tab
        gcode.gcodeXYFeed = (float)nUDToolFeedXY.Value;    // override devault values
        gcode.gcodeZFeed  = (float)nUDToolFeedZ.Value;    // override devault values
```

```
        gcode.gcodeSpindleSpeed = (float)nUDToolSpindleSpeed.Value;      // override devault values
        gcode.gcodeZDown = (float)nUDImportGCZDown.Value;

        gcodeString.Clear();
        //          gcodeLines = 0;
        //       gcodePause = 0;
        gcode.Tool(gcodeString, 0, "");
        if  (!Properties.Settings.Default.importGCSpindleToggle)  gcode.SpindleOn(gcodeString, "Start");
 //          gcode.PenUp(gcodeString);

        float x, y, rShape,d,dTool,overlap,rTool,zStep;
        float zStart = 0;
        x = (float)nUDShapeX.Value;
        y = (float)nUDShapeY.Value;
        rShape = (float)nUDShapeR.Value;
        d = 2 * rShape;
        dTool = (float)nUDToolDiameter.Value;          // tool diameter;
        overlap = dTool * (float)nUDToolOverlap.Value/100;  // tool overlap
        if (rBToolpath1.Checked) { dTool = 0; }        // engrave
        if (rBToolpath3.Checked) { dTool = -dTool; }      // outside
        rTool = dTool / 2;                             // tool radius

        int counter=0,safety = 100;
        float dx = 0, dy = 0, rDelta=0;
        if (rBShape1.Checked)              // rectangle
        {   getOffset(x,y);
            offsetX -= rTool; offsetY -= rTool;
```

```
            x += dTool;y += dTool;
            zStep = zStart;
            while (zStep > (float)nUDImportGCZDown.Value)
            {
               zStep -= (float)nUDToolZStep.Value;
               if (zStep < (float)nUDImportGCZDown.Value)
                  zStep = (float)nUDImportGCZDown.Value;
               gcode.PenUp(gcodeString);
               if (cBToolpathPocket.Checked)
                  gcode.MoveToRapid(gcodeString, offsetX + overlap, offsetY + overlap, "");
               else
                  gcode.MoveToRapid(gcodeString, offsetX, offsetY, "");
               gcode.gcodeZDown = zStep;              // adapt Z-deepth
               gcode.PenDown(gcodeString);
               if (cBToolpathPocket.Checked)
               {
                  dx = overlap; dy = overlap;
                  while (((dx < x/2) && (dy < y/2)) && (counter++ < safety))
                  {
                     makeRect(offsetX+dx, offsetY+dy, offsetX + x - dx, offsetY + y - dy, 0, false);
// rectangle clockwise
                     dx += overlap; dy += overlap;
                     if ((dx < x / 2) && (dy < y / 2))
                        gcode.MoveTo(gcodeString, offsetX + dx, offsetY + dy, "");
                  }
                  gcode.PenUp(gcodeString);
                  gcode.MoveToRapid(gcodeString, offsetX, offsetY, "");
                  gcode.PenDown(gcodeString);
               }
```

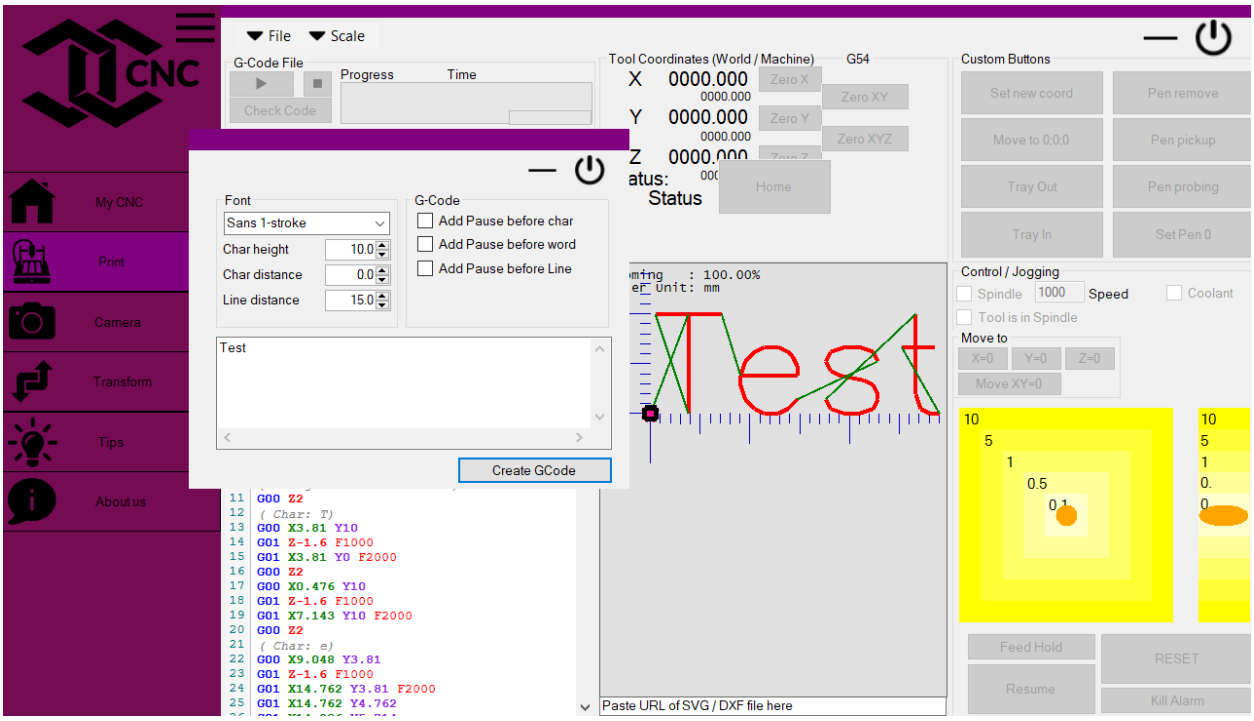
```
            makeRect(offsetX, offsetY, offsetX + x, offsetY + y, 0, true);  // rectangle clockwise
        }
        gcode.PenUp(gcodeString);
    }
    if (rBShape2.Checked)          // rectangle with round edge
    {
        getOffset(x, y);
        offsetX -= rTool; offsetY -= rTool;
        x += dTool; y += dTool;
//              gcode.Move(gcodeString, 0, offsetX, offsetY + r, false, "");
        zStep = zStart;
        while (zStep > (float)nUDImportGCZDown.Value)
        {
            zStep -= (float)nUDToolZStep.Value;
            if (zStep < (float)nUDImportGCZDown.Value)
                zStep = (float)nUDImportGCZDown.Value;
            gcode.PenUp(gcodeString);
            if (cBToolpathPocket.Checked)
                gcode.MoveToRapid(gcodeString, offsetX + overlap, offsetY + rShape , "");
            else
                gcode.MoveToRapid(gcodeString, offsetX, offsetY + rShape, "");
            gcode.gcodeZDown = zStep;              // adapt Z-deepth
            gcode.PenDown(gcodeString);
            if (cBToolpathPocket.Checked)
            {
                dx = overlap; dy = overlap; rDelta = rShape - overlap;
                while (((dx < x / 2) && (dy < y / 2)) && (counter++ < safety))
                {
```

```
                    makeRect(offsetX + dx, offsetY + dy, offsetX + x - dx, offsetY + y - dy, rDelta,
false);  // rectangle clockwise
                    dx += overlap; dy += overlap; rDelta -= overlap;
                    if (dx > x / 2) { dx = x / 2; }
                    if (dy > x / 2) { dy = y / 2; }
                    if (rDelta < 0) { rDelta = 0; }
                    if ((dx < x / 2) && (dy < y / 2))
                        gcode.MoveTo(gcodeString, offsetX + dx, offsetY + dy + rDelta, "");
                }
                gcode.PenUp(gcodeString);
                gcode.MoveToRapid(gcodeString, offsetX, offsetY + rShape, "");
                gcode.PenDown(gcodeString);
            }
            makeRect(offsetX, offsetY, offsetX + x, offsetY + y, rShape, true);   // rectangle
clockwise
        }
        gcode.PenUp(gcodeString);
    }
    if (rBShape3.Checked)          // circle
    {
        getOffset(d, d);
        offsetX -= rTool; offsetY -= rTool;
        rShape += rTool;                  // take care of tool diameter if set
        zStep = zStart;
        while (zStep > (float)nUDImportGCZDown.Value)
        {
            gcode.PenUp(gcodeString);
            if (cBToolpathPocket.Checked)
                gcode.MoveToRapid(gcodeString, offsetX + rShape-overlap, offsetY + rShape, "");
```

```
        else
            gcode.MoveToRapid(gcodeString, offsetX, offsetY + rShape, "");
        zStep -= (float)nUDToolZStep.Value;
        if (zStep < (float)nUDImportGCZDown.Value)
            zStep = (float)nUDImportGCZDown.Value;
        gcode.gcodeZDown = zStep;               // adapt Z-deepth
        gcode.PenDown(gcodeString);
        rDelta = overlap;
        counter = 0;
        if (cBToolpathPocket.Checked)
        {   while ((rDelta < rShape) && (counter++ < safety))
            {
          gcode.Arc(gcodeString, 2, offsetX + rShape - rDelta, offsetY + rShape, rDelta, 0, "");
                rDelta += overlap;
                if (rDelta < rShape)
                    gcode.MoveTo(gcodeString, offsetX + rShape - rDelta, offsetY + rShape, "");
            }
            gcode.MoveTo(gcodeString, offsetX , offsetY + rShape, "");
        }
        gcode.Arc(gcodeString, 2, offsetX, offsetY + rShape, rShape, 0, "");
    }
    gcode.PenUp(gcodeString);
}

string header = gcode.GetHeader("Simple Shape");
string footer = gcode.GetFooter();

gcodeString.Replace(',', '.');
```

```
    shapegcode = header + gcodeString.ToString() + footer;
}

private void makeRect(float x1, float y1, float x2, float y2, float r, bool cw=true )
{   // start bottom left
    if (cw)
    {
        gcode.MoveTo(gcodeString, x1, y2 - r, "");          //BL to TL
        if (r > 0) { gcode.Arc(gcodeString, 2, x1 + r, y2, r, 0,  "");}
        gcode.MoveTo(gcodeString, x2 - r, y2, "");          // TL to TR
        if (r > 0) { gcode.Arc(gcodeString, 2, x2, y2 - r, 0, -r,  ""); }
        gcode.MoveTo(gcodeString, x2, y1 + r, "");          // TR to BR
        if (r > 0) { gcode.Arc(gcodeString, 2, x2 - r, y1, -r, 0,  ""); }
        gcode.MoveTo(gcodeString, x1 + r, y1, "");          // BR to BL
        if (r > 0) { gcode.Arc(gcodeString, 2, x1, y1 + r, 0, r,  ""); }
    }
    else
    {
        if (r > 0) { gcode.Arc(gcodeString, 3, x1 + r, y1, r, 0,  ""); }
        gcode.MoveTo(gcodeString, x2 - r, y1, "");          // to BR
        if (r > 0) { gcode.Arc(gcodeString, 3, x2, y1 + r, 0, r,  ""); }
        gcode.MoveTo(gcodeString, x2, y2 - r, "");          // to TR
        if (r > 0) { gcode.Arc(gcodeString, 3, x2 - r, y2, -r, 0,  ""); }
        gcode.MoveTo(gcodeString, x1 + r, y2, "");          // to TL
        if (r > 0) { gcode.Arc(gcodeString, 3, x1, y2 - r, 0, -r,  ""); }
        gcode.MoveTo(gcodeString, x1, y1 + r, "");          // to BL
    }
}
```

5.3Text formatting



In text formatting we can change the font style, the font size, character distance and line spacing of a text. And also we can add a pause before a char, or a word, or a line for change the color of the pen. Any .iff type font file can be added to our software.

Part of code – Text printing

```
private void TextForm_Load(object sender, EventArgs e)
    {
        cBFont.Items.AddRange(GCodeFromFont.fontNames);
        cBFont.Items.AddRange(GCodeFromFont.fontFileName());

        cBFont.SelectedIndex = Properties.Settings.Default.textFontIndex;
        tBText.Text = Properties.Settings.Default.textFontText;
        nUDFontSize.Value = Properties.Settings.Default.textFontSize;
```

```
        Location = Properties.Settings.Default.locationTextForm;
        Size desktopSize = System.Windows.Forms.SystemInformation.PrimaryMonitorSize;
        if ((Location.X < -20) || (Location.X > (desktopSize.Width - 100)) || (Location.Y < -20) || (Location.Y > (desktopSize.Height - 100))) { Location = new Point(0, 0); }
        getSettings();
    }
    private void getSettings()
    {
        gcodeXYFeed = (float)Properties.Settings.Default.importGCXYFeed;
        gcodeUseSpindle = Properties.Settings.Default.importGCZEnable;
    }
    private void TextForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        Properties.Settings.Default.textFontIndex = cBFont.SelectedIndex;
        Properties.Settings.Default.textFontSize= nUDFontSize.Value;
        Properties.Settings.Default.textFontText = tBText.Text;
        Properties.Settings.Default.locationTextForm = Location;
        Properties.Settings.Default.Save();
    }

    // get text, break it into chars, get path, etc... This event needs to be assigned in MainForm to poll text
    private void btnApply_Click(object sender, EventArgs e)        // in MainForm: _text_form.btnApply.Click += getGCodeFromText;
    {
        getSettings();
        gcode.setup();
        GCodeFromFont.reset();

        GCodeFromFont.gcText = tBText.Text;
        GCodeFromFont.gcFont = GCodeFromFont.getFontIndex(cBFont.SelectedIndex);
```

```
            GCodeFromFont.gcFontName = cBFont.Items[cBFont.SelectedIndex].ToString();
            GCodeFromFont.gcHeight = (double)nUDFontSize.Value;
            GCodeFromFont.gcFontDistance = (double)nUDFontDistance.Value;
            GCodeFromFont.gcLineDistance = (double)(nUDFontLine.Value / nUDFontSize.Value);
            GCodeFromFont.gcSpacing = (double)(nUDFontLine.Value/ nUDFontSize.Value)/1.5;
            GCodeFromFont.gcPauseChar = cBPauseChar.Checked;
            GCodeFromFont.gcPauseWord = cBPauseWord.Checked;
            GCodeFromFont.gcPauseLine = cBPauseLine.Checked;

//          MessageBox.Show(cBFont.Items[cBFont.SelectedIndex].ToString());
            gcodeString.Clear();
            GCodeFromFont.getCode(gcodeString);

            if (gcodeUseSpindle) gcode.SpindleOff(gcodeString, "End");
            gcodeString.Replace(',', '.');
            string header = gcode.GetHeader("Text import");
            string footer = gcode.GetFooter();

            textgcode = header + gcodeString.ToString() + footer;
        }
```

Part of code – Font Changing

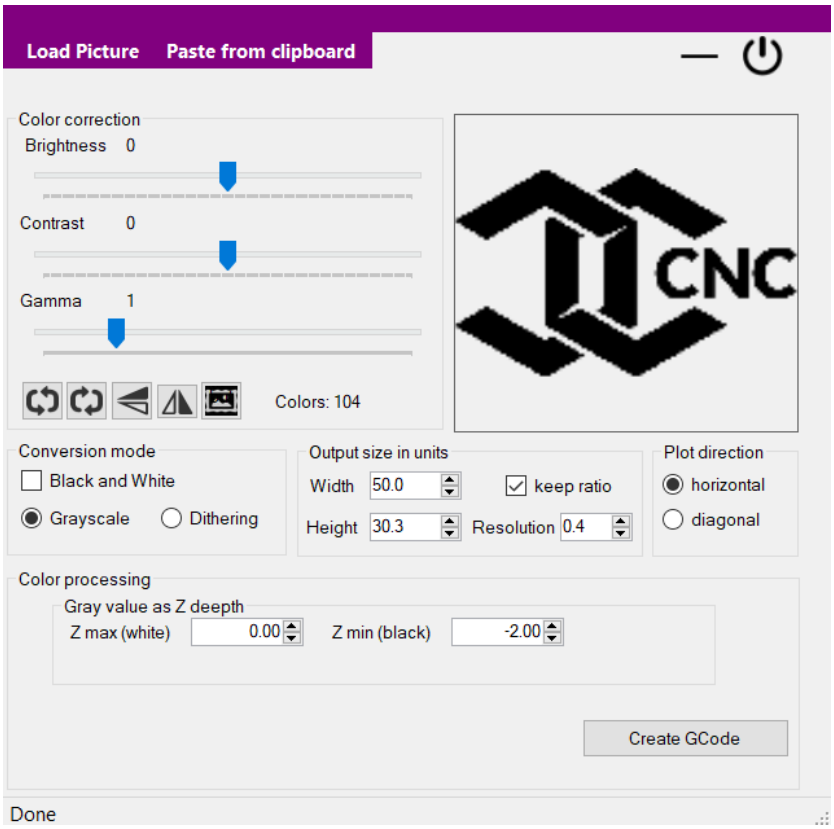
```
public static string[] fontFileName()
    {   if (Directory.Exists("fonts"))
            return Directory.GetFiles("fonts");
        return new string[0];
    }
    public static void reset()
    {
        gcFontName = "standard"; gcText = ""; gcFont = 0; gcAttachPoint = 7;
        gcHeight = 0; gcWidth = 0; gcAngle = 0; gcSpacing = 1; gcOffX = 0; gcOffY = 0;
        gcPauseLine = false; gcPauseWord = false; gcPauseChar = false; gcodePenIsUp = false;
        useLFF = false; gcLineDistance = 1.5; gcFontDistance = 0;
    }

    public static bool getCode(StringBuilder gcodeString)
    {
        double scale = gcHeight / 21;
        string tmp1 = gcText.Replace('\r', '|');
        gcodeString.AppendFormat("( Text: {0} )\r\n", tmp1.Replace('\n', ' '));
        string[] fileContent=new string[] { "" };

        string fileName = "";
        if (gcFontName.IndexOf(@"fonts\") >= 0)
            fileName = gcFontName;
        else
            fileName = @"fonts\" + gcFontName + ".lff";
        bool fontFound = false;
        if (gcFontName != "")
        {   if (File.Exists(fileName))
            {   fileContent = File.ReadAllLines(fileName);
```

```
            scale = gcHeight / 9;
            useLFF = true;
            offsetY = 0;
            gcLineDistance = 1.667 * gcSpacing;
            fontFound = true;
            foreach (string line in fileContent)
            {   if (line.IndexOf("LetterSpacing") >= 0)
                {   string[] tmp = line.Split(':');
                    gcLetterSpacing = Convert.ToDouble(tmp[1].Trim());
                }
                if (line.IndexOf("WordSpacing") >= 0)
                {
                    string[] tmp = line.Split(':');
                    gcWordSpacing = Convert.ToDouble(tmp[1].Trim());
                }
            }
        }
        else
        {   gcodeString.AppendFormat("( Font '{0}' not found )\r\n", gcFontName);
            gcodeString.Append("( Using alternative font )\r\n");
        }
    }
    if ((gcAttachPoint == 2) || (gcAttachPoint == 5) || (gcAttachPoint == 8))
        gcOffX -= gcWidth / 2;
    if ((gcAttachPoint == 3) || (gcAttachPoint == 6) || (gcAttachPoint == 9))
        gcOffX -= gcWidth;
    if ((gcAttachPoint == 4) || (gcAttachPoint == 5) || (gcAttachPoint == 6))
        gcOffY -= gcHeight / 2;
    if ((gcAttachPoint == 1) || (gcAttachPoint == 2) || (gcAttachPoint == 3))
        gcOffY -= gcHeight;
```

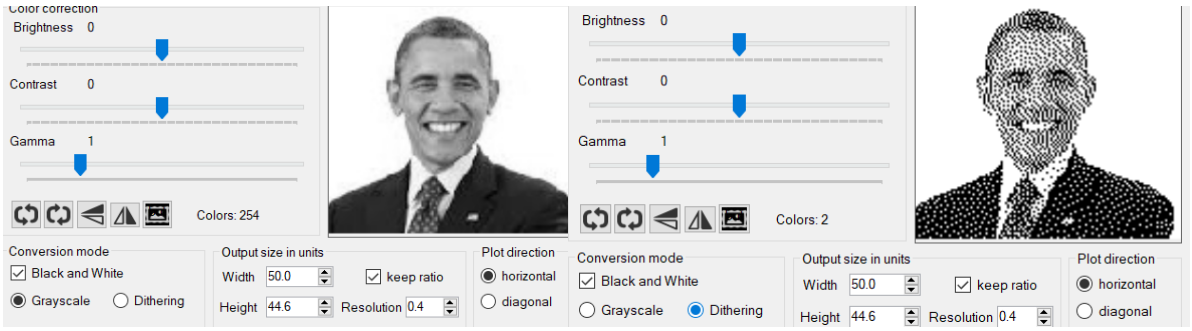
```
        string[] lines;
        if (gcText.IndexOf("\\P") >= 0)
        {   gcText = gcText.Replace("\\P", "\n"); }
        lines = gcText.Split('\n');
        offsetX = 0;
        offsetY = 9 * scale + ((double)lines.Length - 1) * gcHeight * gcLineDistance;// (double)nUDFontLine.Value;
        if (useLFF)
           offsetY = ((double)lines.Length - 1) * gcHeight * gcLineDistance;// (double)nUDFontLine.Value;


        for (int txtIndex = 0; txtIndex < gcText.Length; txtIndex++)
        {
           gcodePenUp(gcodeString);
           int chrIndex = (int)gcText[txtIndex] - 32;
           int chrIndexLFF = (int)gcText[txtIndex];


           if (gcText[txtIndex] == '\n')                // next line
           {
              offsetX = 0;
              offsetY -= gcHeight * gcLineDistance;


              if (gcPauseLine)
              {   gcode.Pause(gcodeString, "Pause before line");
              }
           }
           else if (useLFF)
              drawCharLFF(gcodeString, ref fileContent, chrIndexLFF, scale); // regular char
           else
```

```
    {   if ((chrIndex < 0) || (chrIndex > 95))     // no valid char
        {
            offsetX += 2 * gcSpacing;                  // apply space
            if (gcPauseWord)
            {   gcode.Pause(gcodeString, "Pause before word");
            }
        }
        else
        {
            gcodeString.AppendFormat("( Char: {0})\r\n", gcText[txtIndex]);
            if (gcPauseChar)
            {   gcode.Pause(gcodeString, "Pause before char");
            }
            if (gcPauseChar && (gcText[txtIndex] == ' '))
            {   gcode.Pause(gcodeString, "Pause before word");
            }
            drawChar(gcodeString, fontChar[gcFont][chrIndex], scale); // regular char
        }
      }
   }
   gcode.PenUp(gcodeString);
   return fontFound;
}
```

5.4. Image printing



To tune and print an image first we should load a picture from an open file dialog to the form or we can paste from clipboard. And from the color corrections we can adjust brightness, contrast and gamma. From the rotating buttons the image can rotate by 90 left or right. And also can change the vertical flip and the horizontal flip and can invert colors. The width, height and the resolution of the image can be changed by the form. For improve the drawing quality of the image we can convert the image into dither or grayscale. By dithering a normal image can be converted into doted form.



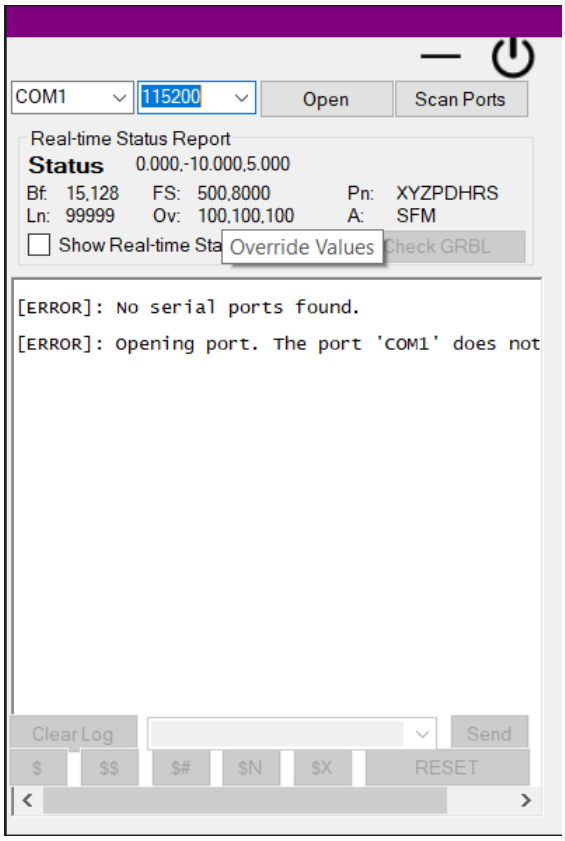
Part of code – Image printing

```
private void rbModeGray_CheckedChanged(object sender, EventArgs e)
    {
       if (adjustedImage == null) return;//if no image, do nothing
       if (rbModeDither.Checked)// cbDirthering.Text == "Dirthering FS 1 bit")
       {
          lblStatus.Text = "Dirtering...";
          adjustedImage = imgDirther(adjustedImage);
          pictureBox1.Image = adjustedImage;
          lblStatus.Text = "Done";
       }
       else
          userAdjust();
       updateLabelColor = true;


    }
 private Bitmap imgDirther(Bitmap input)
    {
       lblStatus.Text = "Dirthering...";
       Refresh();
       var masks = new byte[] { 0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01 };
       var output = new Bitmap(input.Width, input.Height, PixelFormat.Format1bppIndexed);
       var data = new sbyte[input.Width, input.Height];
       var inputData = input.LockBits(new Rectangle(0, 0, input.Width, input.Height),
ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);
       try
       {
          var scanLine = inputData.Scan0;
          var line = new byte[inputData.Stride];
          for (var y = 0; y < inputData.Height; y++, scanLine += inputData.Stride)
```

```
            {
                Marshal.Copy(scanLine, line, 0, line.Length);
                for (var x = 0; x < input.Width; x++)
                {
                    data[x, y] = (sbyte)(64 * (GetGreyLevel(line[x * 3 + 2], line[x * 3 + 1], line[x * 3 + 0]) - 0.5));
                }
            }
        }
        finally
        {
            input.UnlockBits(inputData);
        }
        var outputData = output.LockBits(new Rectangle(0, 0, output.Width, output.Height), ImageLockMode.WriteOnly, PixelFormat.Format1bppIndexed);
        try
        {
            var scanLine = outputData.Scan0;
            for (var y = 0; y < outputData.Height; y++, scanLine += outputData.Stride)
            {
                var line = new byte[outputData.Stride];
                for (var x = 0; x < input.Width; x++)
                {
                    var j = data[x, y] > 0;
                    if (j) line[x / 8] |= masks[x % 8];
                    var error = (sbyte)(data[x, y] - (j ? 32 : -32));
                    if (x < input.Width - 1) data[x + 1, y] += (sbyte)(7 * error / 16);
                    if (y < input.Height - 1)
                    {
                        if (x > 0) data[x - 1, y + 1] += (sbyte)(3 * error / 16);
```

```
                    data[x, y + 1] += (sbyte)(5 * error / 16);
                    if (x < input.Width - 1) data[x + 1, y + 1] += (sbyte)(1 * error / 16);
                }
            }
            Marshal.Copy(line, 0, scanLine, outputData.Stride);
        }
    }
    finally
    {
        output.UnlockBits(outputData);
    }
    lblStatus.Text = "Done";
    Refresh();
    return (output);
}
private static double GetGreyLevel(byte r, byte g, byte b)//aux for dirthering
{
    return (r * 0.299 + g * 0.587 + b * 0.114) / 255;
}
```

5.5. Serial communication interface



Serial communication form is used for send the g-code to the Uno board. In our program we cannot close the Serial communication form because it will cause to errors in sending values. When the form is loading, form load event will check whether if the last connected port is opened or not.(lastly connected port is saved in a data base) After 10 seconds of connecting to the port automatically serial form will be minimized. The connected board rate should be 115200 because serial communication speed of our gbrl v3 is 115200. By the real time status report that we can see the real time data transactions of values.

Buttons and features

scan port button - can search the available ports

show real time status report check box -we can view a special report such as last saved position.

clear log button - can clear the log history.

$ button - can get the all the g-codes (similar to help)

$$ button - can get the all the grbl settings and meanings

$# button - can get the all the g-code parameters (Ex: G54, G55)

$N button - can get the all the startup blocks (EX: $N, $N1)

$x button - kill alarm

Reset button - reset all the alarms and the current status (idle status)

Part of code – Serial Form

```
private void SerialForm_Load(object sender, EventArgs e)
    {
        Size desktopSize = System.Windows.Forms.SystemInformation.PrimaryMonitorSize;
        SerialForm_Resize(sender, e);
        refreshPorts();
        updateControls();
        loadSettings();
        openPort();
        machineState.Clear();
        if (iamSerial == 1)
        {   Location = Properties.Settings.Default.locationSerForm1;}
        else
        {   Location = Properties.Settings.Default.locationSerForm2;}
        Text = formTitle;
```

```
        if ((Location.X < -20) || (Location.X > (desktopSize.Width-100)) || (Location.Y < -20) ||
(Location.Y > (desktopSize.Height-100))) { Location = new Point(100, 100); }
        isLasermode = Properties.Settings.Default.ctrlLaserMode;
    }

  private void SerialForm_Resize(object sender, EventArgs e)
    {
        rtbLog.Width = this.Width - 20;
        rtbLog.Height = this.Height - 205;
        btnClear.Location = new Point(btnClear.Location.X, this.Height - 86);
        cBCommand.Location = new Point(cBCommand.Location.X, this.Height - 84);
        btnSend.Location = new Point(btnSend.Location.X, this.Height - 86);
        btnGRBLCommand0.Location = new Point(btnGRBLCommand0.Location.X, this.Height
- 62);
        btnGRBLCommand1.Location = new Point(btnGRBLCommand1.Location.X, this.Height
- 62);
        btnGRBLCommand2.Location = new Point(btnGRBLCommand2.Location.X, this.Height
- 62);
        btnGRBLCommand3.Location = new Point(btnGRBLCommand3.Location.X, this.Height
- 62);
        btnGRBLCommand4.Location = new Point(btnGRBLCommand4.Location.X, this.Height
- 62);
        btnGRBLReset.Location = new Point(btnGRBLReset.Location.X, this.Height - 62);
    }
 private void timerSerialEnable(bool value)
    {
        timerSerial.Enabled = value;
    }

    private void timerSerial_Tick(object sender, EventArgs e)
    {
        if (minimizeCount > 0)
```

```
        { minimizeCount--;
            if (minimizeCount == 0)
                this.WindowState = FormWindowState.Minimized;
        }

        if (serialPort.IsOpen)
        {
            try
            {   var dataArray = new byte[] { Convert.ToByte('?') };
                serialPort.Write(dataArray, 0, 1);
            }
            catch (Exception er)
            {   logError("Retrieving GRBL status", er);
                serialPort.Close();
            }
        }
        if (waitForIdle)
        { processSend(); addToLog("."); }
        if (isStreaming && !isStreamingRequestPause && !isStreamingPause)
            preProcessStreaming();
        if (callCheckGRBL > 0)
        {   callCheckGRBL--;
            if (callCheckGRBL == 0)
            { btnCheckGRBL_Click(sender, e); }
        }
    }
private void loadSettings()
    {
        try
        {
```

```
        if (iamSerial == 1)
        {
            cbPort.Text = Properties.Settings.Default.serialPort1;
            cbBaud.Text = Properties.Settings.Default.serialBaud1;
        }
        else
        {
            cbPort.Text = Properties.Settings.Default.serialPort2;
            cbBaud.Text = Properties.Settings.Default.serialBaud2;
        }
    }
    catch (Exception e)
    {
        logError("Loading settings", e);
    }
}
private void saveSettings()
{
    try
    {
        if (iamSerial == 1)
        {
            Properties.Settings.Default.serialPort1 = cbPort.Text;
            Properties.Settings.Default.serialBaud1 = cbBaud.Text;
            Properties.Settings.Default.ctrlLaserMode = isLasermode;
        }
        else
        {
            Properties.Settings.Default.serialPort2 = cbPort.Text;
            Properties.Settings.Default.serialBaud2 = cbBaud.Text;
```

```
            }

            Properties.Settings.Default.Save();
        }
        catch (Exception e)
        {
            logError("Saving settings", e);
        }
    }
    private void saveLastPos()
    {
        if (iamSerial == 1)
        {   rtbLog.AppendText("Save last pos.: "+posWorld.Print()+"\n");
            Properties.Settings.Default.lastOffsetX = Math.Round(posWorld.X, 3);
            Properties.Settings.Default.lastOffsetY = Math.Round(posWorld.Y, 3);
            Properties.Settings.Default.lastOffsetZ = Math.Round(posWorld.Z, 3);
            Properties.Settings.Default.lastOffsetA = Math.Round(posWorld.A, 3);
            int gNr = mParserState.coord_select;
            gNr = ((gNr >= 54) && (gNr <= 59)) ? gNr : 54;
            Properties.Settings.Default.lastOffsetCoord = gNr;
//global.grblParserState.coord_select;
            Properties.Settings.Default.Save();
        }
    }

    private void updateControls()
    {
        bool isConnected = serialPort.IsOpen;
        serialPortOpen = isConnected;
        bool isSensing = isStreaming;
```

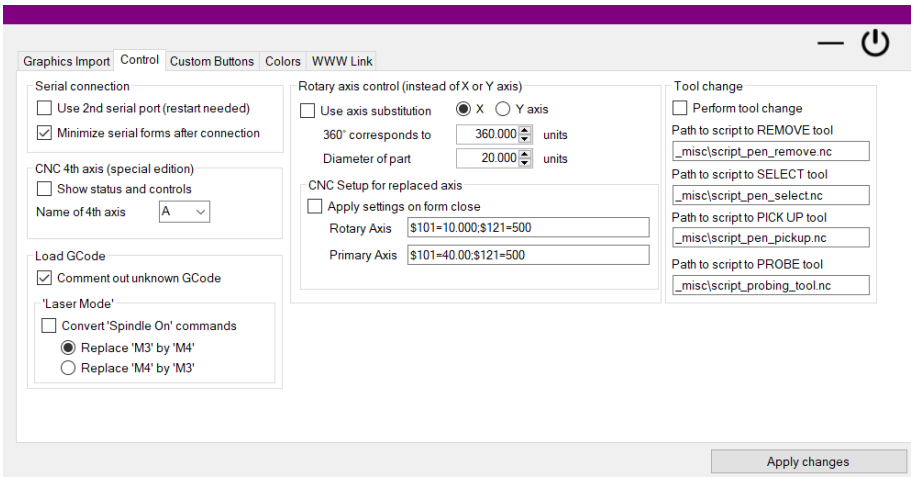
```
    cbPort.Enabled = !isConnected;
    cbBaud.Enabled = !isConnected;
    btnScanPort.Enabled = !isConnected;
    btnClear.Enabled = isConnected;
    cBCommand.Enabled = isConnected && !isSensing;
    btnSend.Enabled = isConnected && !isSensing;
    btnGRBLCommand0.Enabled = isConnected && !isSensing;
    btnGRBLCommand1.Enabled = isConnected && !isSensing;
    btnGRBLCommand2.Enabled = isConnected && !isSensing;
    btnGRBLCommand3.Enabled = isConnected && !isSensing;
    btnGRBLCommand4.Enabled = isConnected && !isSensing;
    btnCheckGRBL.Enabled = isConnected && !isSensing;// && !isGrblVers0;
    btnGRBLReset.Enabled = isConnected;// & !isSensing;
}

private void logError(string message, Exception err)
{
    string textmsg = "\r\n[ERROR]: " + message + ". ";
    if (err != null) textmsg += err.Message;
    textmsg += "\r\n";
    rtbLog.AppendText(textmsg);
    rtbLog.ScrollToCaret();
}
public void logErrorThr(object sender, EventArgs e)
{
    logError(mens, err);
    updateControls();
}
public void addToLog(string text)
{
```

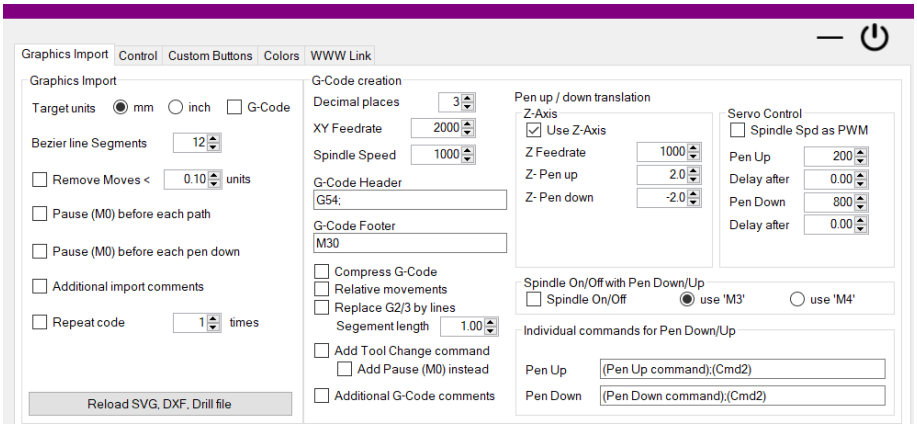
```
    rtbLog.AppendText(text + "\r");
    rtbLog.ScrollToCaret();
}

private void btnScanPort_Click(object sender, EventArgs e)
{ refreshPorts(); }
private void refreshPorts()
{
    List<String> tList = new List<String>();
    cbPort.Items.Clear();
    foreach (string s in System.IO.Ports.SerialPort.GetPortNames()) tList.Add(s);
    if (tList.Count < 1) logError("No serial ports found", null);
    else
    {
        tList.Sort();
        cbPort.Items.AddRange(tList.ToArray());
    }
}
private void btnOpenPort_Click(object sender, EventArgs e)
{
    if (serialPort.IsOpen)
        closePort();
    else
        openPort();
    updateControls();
}
private int minimizeCount = 0;
private bool openPort()
{
    try
```

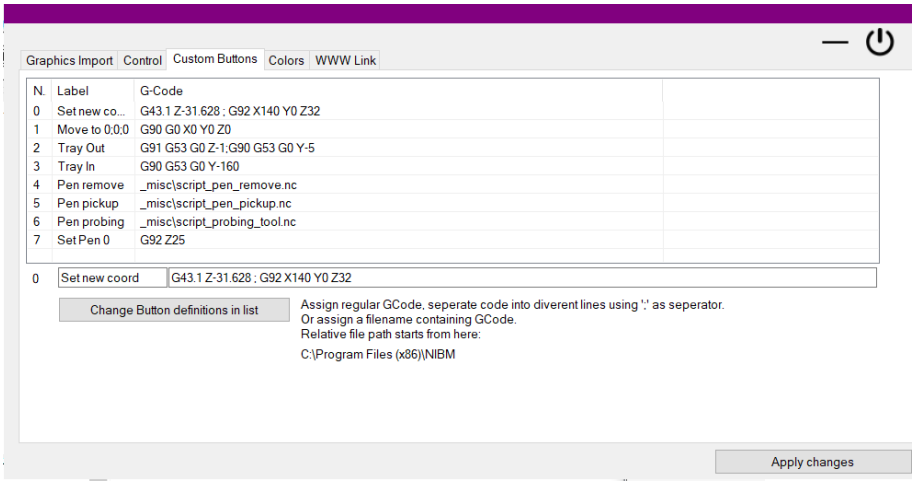
```
    {
        serialPort.PortName = cbPort.Text;
        serialPort.BaudRate = Convert.ToInt32(cbBaud.Text);
        serialPort.Open();
        rtbLog.Clear();
        rtbLog.AppendText("Open " + cbPort.Text + "\r\n");
        btnOpenPort.Text = "Close";
        isDataProcessing = true;
        grblReset(false);
        updateControls();
        if (Properties.Settings.Default.serialMinimize)
            minimizeCount = 10;     // minimize window after 10 timer ticks
        timerSerial.Interval = timerReload;
        return (true);
    }
    catch (Exception err)
    {
        minimizeCount = 0;
        logError("Opening port", err);
        updateControls();
        return (false);
    }
}
public bool closePort()
{
    try
    {               if (serialPort.IsOpen)
        {   serialPort.Close();
        }
        rtbLog.AppendText("Close " + cbPort.Text + "\r\n");
```

```
            btnOpenPort.Text = "Open";
            updateControls();
            timerSerial.Interval = 1000;
            return (true);
        }
        catch (Exception err)
        {
            logError("Closing port", err);
            updateControls();
            timerSerial.Enabled = false;
            return (false);
        }
    }
```

5.6. Setup Form







Part of code – Setup

```
private void SetupForm_Load(object sender, EventArgs e)
    {
        string[] parts;
        for (int i = 1; i <= 8; i++)
        {
            parts = Properties.Settings.Default["custom" + i.ToString()].ToString().Split('|');
            ListViewItem item = new ListViewItem((i-1).ToString());
            item.SubItems.AddRange(parts);
            lvCustomButtons.Items.Add(item);
        }
        lvCustomButtons.Items[0].Selected = true;
        setButtonColors(btnColorBackground,Properties.Settings.Default.colorBackground);
        setButtonColors(btnColorRuler,Properties.Settings.Default.colorRuler);
        setButtonColors(btnColorPenUp,Properties.Settings.Default.colorPenUp);
        setButtonColors(btnColorPenDown,Properties.Settings.Default.colorPenDown);
        setButtonColors(btnColorTool,Properties.Settings.Default.colorTool);
        setButtonColors(btnColorMarker, Properties.Settings.Default.colorMarker);

        nUDImportDecPlaces.Value = Properties.Settings.Default.importGCDecPlaces;

        Location = Properties.Settings.Default.locationSetForm;
        Size desktopSize = System.Windows.Forms.SystemInformation.PrimaryMonitorSize;
        if ((Location.X < -20) || (Location.X > (desktopSize.Width - 100)) || (Location.Y < -20) ||
(Location.Y > (desktopSize.Height - 100))) { Location = new Point(0, 0); }

        if (Properties.Settings.Default.importGCSpindleCmd)
            rBImportGCSpindleCmd1.Checked = true;
        else
            rBImportGCSpindleCmd2.Checked = true;
```

```
        if (Properties.Settings.Default.ctrlReplaceM3)
            rBCtrlReplaceM3.Checked = true;
        else
            rBCtrlReplaceM4.Checked = true;

        if (Properties.Settings.Default.rotarySubstitutionX)
            rBRotaryX.Checked = true;
        else
            rBRotaryY.Checked = true;

        if (Properties.Settings.Default.importUnitmm)
            rBImportUnitmm.Checked = true;
        else
            rBImportUnitInch.Checked = true;

        lblFilePath.Text = System.Windows.Forms.Application.StartupPath;



    }

    private void saveSettings()
    {
        for (int i = 1; i <= 8; i++)
        {
            ListViewItem item = lvCustomButtons.Items[i - 1];
            Properties.Settings.Default["custom" + i.ToString()] = item.SubItems[1].Text + "|" +
item.SubItems[2].Text;// + "|" + item.SubItems[3].Text;
        }
        Properties.Settings.Default.importGCDecPlaces = nUDImportDecPlaces.Value;
```

```
        Properties.Settings.Default.importGCSpindleCmd = rBImportGCSpindleCmd1.Checked;
        Properties.Settings.Default.ctrlReplaceM3 = rBCtrlReplaceM3.Checked;
        Properties.Settings.Default.rotarySubstitutionX = rBRotaryX.Checked;

        Properties.Settings.Default.Save();
      }

    int lastIndex = 0;
    private void lvCustomButtons_ItemSelectionChanged(object sender,
ListViewItemSelectionChangedEventArgs e)
    {
      if (lvCustomButtons != null)
      { //MessageBox.Show(e.ItemIndex.ToString());
        lastIndex = e.ItemIndex;
        ListViewItem item = lvCustomButtons.Items[lastIndex];
        textBox1.Text = item.SubItems[1].Text;
        textBox2.Text = item.SubItems[2].Text;
//        textBox3.Text = item.SubItems[3].Text;
        lblcbnr.Text = lastIndex.ToString();
      }
    }

    private void btnChangeDefinition_Click(object sender, EventArgs e)
    {
      if (lvCustomButtons != null)
      {
        ListViewItem item = lvCustomButtons.Items[lastIndex]; // SelectedItems[0];
        item.SubItems[1].Text = textBox1.Text;
        item.SubItems[2].Text = textBox2.Text;
//        item.SubItems[3].Text = textBox3.Text;
```

```
    }
}

private void btnApplyChangings_Click(object sender, EventArgs e)
{
    saveSettings();
}

private void btnColorBackground_Click(object sender, EventArgs e)
{ applyColor(btnColorBackground, "colorBackground"); }
private void btnColorRuler_Click(object sender, EventArgs e)
{ applyColor(btnColorRuler, "colorRuler"); }
private void btnColorPenUp_Click(object sender, EventArgs e)
{ applyColor(btnColorPenUp, "colorPenUp"); }
private void btnColorPenDown_Click(object sender, EventArgs e)
{ applyColor(btnColorPenDown, "colorPenDown"); }
private void btnColorTool_Click(object sender, EventArgs e)
{ applyColor(btnColorTool, "colorTool");        }
private void btnColorMarker_Click(object sender, EventArgs e)
{ applyColor(btnColorMarker, "colorMarker"); }

private void applyColor(Button btn,string settings)
{
    if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
        setButtonColors(btn, colorDialog1.Color);
        Properties.Settings.Default[settings] = colorDialog1.Color;
        saveSettings();
    }
}
```

```
private void setButtonColors(Button btn, Color col)
{
   btn.BackColor = col;
   btn.ForeColor = ContrastColor(col);
}
private Color ContrastColor(Color color)
{
   int d = 0;
   // Counting the perceptive luminance - human eye favors green color...
   double a = 1 - (0.299 * color.R + 0.587 * color.G + 0.114 * color.B) / 255;
   if (a < 0.5)
      d = 0; // bright colors - black font
   else
      d = 255; // dark colors - white font
   return Color.FromArgb(d, d, d);
}

private void btnReloadFile_Click(object sender, EventArgs e)
{
   saveSettings();
}

private bool isExpand = false;
private void expandForm_Click(object sender, EventArgs e)
{
   if (!isExpand)
   {
      this.Width = 730;
      this.Height = 350;
      btnResizeForm.Text = "reduce <";
```

```
        isExpand = true;
    }
}

private void btnResizeForm_Click(object sender, EventArgs e)
{
    if (!isExpand)
    {
        this.Width = 750;
        this.Height = 400;
        btnResizeForm.Text = "reduce <";
        isExpand = true;
    }
    else
    {
        this.Width = 260;
        this.Height = 365;
        btnResizeForm.Text = "expand >";
        isExpand = false;
    }
}
```

6.Implementations

- By adding a sensor to detect the paper quality of the drawing can be improved.
- If we add a camera to the tool, we can see the real time updates on drawing.
- We can develop this to a PCB drawer by adding a camera to the tool with shape recognition program.
- We can develop this machine to a 3D printer.

7.Conclusion

- CNC is a programmable automation printer which use G-code.
- It consists of three motors and their drivers with its basic circuit and body have three axis and pen/drill and the paper/wood we want to draw on it
- We tried to make a cheap, fast, safe CNC machine that can draw pictures, words, rigiform cutting and wood carving.

As a final product

8.Reference

- [1]"CNC Code Programming", Carlsonmfg.com, 2019. [Online]. Available: http://carlsonmfg.com/cnc-g-code-m-code-programming. [Accessed: 21- Feb- 2019].
- [2]"G-Code Tutorial - The 10 Most Common Commands for 3D Printing", Simplify3d.com, 2019. [Online]. Available: https://www.simplify3d.com/support/articles/3d-printing-gcode-tutorial/. [Accessed: 21- Feb- 2019].
- [3]B. Warfield, "CNC Programming with G Code: Definitive Free Tutorial [2019]", CNCCookbook: Be A Better CNC'er, 2019. [Online]. Available: https://www.cnccookbook.com/cnc-programming-g-code/. [Accessed: 21- Mar- 2019].
- [4]"Gcode", Duet3D, 2019. [Online]. Available: https://duet3d.dozuki.com/Wiki/Gcode. [Accessed: 21- Feb- 2019].