



**Michigan
Technological
University**

Michigan Technological University
Digital Commons @ Michigan Tech

Department of Materials Science and
Engineering Publications

Department of Materials Science and
Engineering

1-27-2016

Free and open-source control software for 3-D motion and processing

Bas Wijnen

Michigan Technological University

G. C. Anzalone

Michigan Technological University

Amberlee S. Haselhuhn

Michigan Technological University

Paul G. Sanders

Michigan Technological University

Joshua M. Pearce

Michigan Technological University

Follow this and additional works at: https://digitalcommons.mtu.edu/materials_fp

 Part of the [Materials Science and Engineering Commons](#)


Recommended Citation

Wijnen, B., Anzalone, G. C., Haselhuhn, A. S., Sanders, P. G., & Pearce, J. M. (2016). Free and open-source control software for 3-D motion and processing. *Journal of Open Research Software*, 4(1).

<http://dx.doi.org/10.5334/jors.78>

Retrieved from: https://digitalcommons.mtu.edu/materials_fp/122

Follow this and additional works at: https://digitalcommons.mtu.edu/materials_fp

 Part of the [Materials Science and Engineering Commons](#)

SOFTWARE METAPAPER

Free and Open-source Control Software for 3-D Motion and Processing

Bas Wijnen¹, Gerald C. Anzalone¹, Amberlee S. Haselhuhn¹, P. G. Sanders¹ and Joshua M. Pearce²

¹ Department of Materials Science & Engineering, Michigan Technological University, Houghton, MI, 49931, US
bwijnen@mtu.edu, gcanzalo@mtu.edu, aslifer@mtu.edu, sanders@mtu.edu

² Department of Materials Science & Engineering, Michigan Technological University, Houghton, MI, 49931, US,
Department of Electrical & Computer Engineering, Michigan Technological University, Houghton, MI, 49931, US
pearce@mtu.edu

Corresponding author: Joshua M. Pearce

RepRap 3-D printers and their derivatives using conventional firmware are limited by: 1) requiring technical knowledge, 2) poor resilience with unreliable hardware, and 3) poor integration in complicated systems. In this paper, a new control system called Franklin, for CNC machines in general and 3-D printers specifically, is presented that enables web-based three dimensional control of additive, subtractive and analytical tools from any Internet connected device. Franklin can be set up and controlled entirely from a web interface; it uses a custom protocol which allows it to continue printing when the connection is temporarily lost, and allows communication with scripts.

Keywords: 3-D printing; additive manufacturing; distributed manufacturing; firmware; free and open source software; FOSS, open-source; open-source electronics; open-source hardware; personal fabrication; printing; RepRap; rapid prototyping

Funding statement: This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8650-12-2-7230. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government.

(1) Overview

Introduction

Since Bowyer's open source release of the self-replicating rapid prototyper (RepRap) [1, 2], people from all over the world have been building and improving RepRap 3-D printers [3]. The existing control system consists of two parts: firmware on the 3-D printer to send the signals through to individual components, and a program on a desktop computer to send instructions (G-code) to the firmware. Current firmware options (**Table 1**) use the same interface with the host, and suffer from two fundamental problems: 1) they are not robust and 2) they require a weak microcontroller to do significant work, thus limiting their functionality. Some firmware solve the latter challenge by requiring a more powerful microcontroller, which limits its applicability.

As there is a standard communication format between existing firmware and hosts, there are also many different

options for host software including Pronterface [4], Cura [5] and OctoPrint [6]. In terms of ease of use, OctoPrint stands out: it provides a web interface to control the printer over a network from any device with a browser. OctoPrint supports monitoring the printer with a webcam, and makes recordings of the print. Another project that is worth noting in detail is Pacemaker [7]. Pacemaker defines a new protocol that tries to move most computationally intensive tasks from the firmware to the host. This protocol replaces G-code and requires firmware that can handle it. The Pacemaker program itself runs on a host computer, converting G-code into the format understood by the firmware. The format is designed to be backwards compatible, so different firmware and hosts can work together. This only works if new configurations are tested and validated, requiring many users to maintain a unique configuration. At this point, it is unclear whether it will gain the critical mass necessary for success.

Name	Hardware	Comments
RepRap Firmware (original)	ARM	
LinuxCNC	PC	Uses the parallel port, or GPIO pins on a Raspberry Pi or BeagleBone
RepRap Firmware (fork by dc42)	ARM	Based on RepRap Firmware
Marlin	AVR	Adaptation of Grbl for 3-D printers
Repetier	AVR, ARM	Based on Marlin
Teacup	AVR, ARM	Runs on AVRs with low memory, such as Arduino Uno
aprinter	AVR, ARM	
Smoothie	ARM	

Table 1: Current RepRap Firmware.

Reliability has also been identified as a core challenge for low-cost 3-D printers [8]. For a system to work reliably, data sent between components must not be corrupted and as this is not always avoidable, error checking is required. Unfortunately, ubiquitous computer numerical control (CNC) tools of all types most commonly use G-code, which provides a weak checksum for data sent to the device and no protection for replies. For unreliable connections, the host computer reports a lost serial port, almost immediately followed by a newly discovered port. For all of the current control systems summarized in **Table 1**, when this occurs mid-print, the print fails. This can only be fixed by changing the protocol between firmware and host software, and then both the firmware and the host software need to be changed to implement the new protocol. Recovering from such connection issues is becoming increasingly important as printers become larger (e.g. Gigabot), work on more sophisticated designs such as scientific [9, 10] and medical equipment [11, 12, 13], and solar-powered printers are used to accelerate sustainable development [14, 15, 16].

CNC machines are powerful general purpose motion controllers that are capable of integrating with a system, for example to perform a constant criterion experiment such as recording the adjustments made to an actuator required to keep a measured value constant. While the hardware is capable, the G-code protocol provides no means for performing such functions. Similarly G-code does not integrate well with other programs, which prevents one click print functionality. To solve these two problems a scripted interface is necessary.

In this paper a new control system called Franklin is presented to be used with CNC machines in general and 3-D printers specifically. It was developed while exploring modifications to RepRap 3-D printers, in order to make them more powerful and more user friendly. It solves the problems identified above and allows CNC machines to be more productive and valuable. Franklin was tested on the following RepRap devices: Mendel RepRap, Delta RepRap, OS laser welder, PCB micromill, and the open source metal 3-D printer. The results are presented and discussed.

Implementation and Architecture

Franklin's purpose is to: 1) drive CNC hardware including motors, switches, and a tool head and 2) integrate well with the remaining software tool chain, such as 3-D modeling programs and slicers. Slicers translate a 3-D shape described by an STL file and slice it into consecutive thin layers in the z-direction (vertically) as g-code. The full code of Franklin is available on Github under the GNU Affero General Public License [17].

System Description

Reprap 3-D printers have a dedicated real time control board, which is normally based on the Arduino [18] prototyping platform. When using Franklin, the control board contains *Franklin Firmware*, which handles low-level control and communicates with a more powerful host computer using a serial interface. On the host computer, *Franklin Server* provides a web interface that can be used from any device on the same network, including the Internet (**Figure 1**). Franklin uses an encrypted (HTTPS) connection and allows restricting access with a password to prevent unauthorized users from controlling the device.

The Franklin Server can handle multiple devices simultaneously using a dedicated driver for each device. This driver consists of two processes: one handling high speed and time sensitive functions written in C++ (the *Franklin C Driver*), and one handling the rest written in Python (the *Franklin Python Driver*). In addition to communicating with each other, the *Franklin C Driver* communicates with the *Franklin Firmware*, while the *Franklin Python Driver* communicates with *Franklin Server*. During device discovery, before the driver is started, the *Franklin Server* also communicates directly with the *Franklin Firmware*.

For Franklin to be considered acceptable as a firmware solution, tests were run using the following: Mendel RepRap [19], Delta RepRap [20], quad delta RepRap (4 vertically stacked quad MOST delta RepRap heads run on a single microcontroller and set of three position stepper motors), OS laser welder [10], PCB micromill [21] and the open source metal 3-D printer [22]. The test data obtained from the latter was utilized to further develop the metal

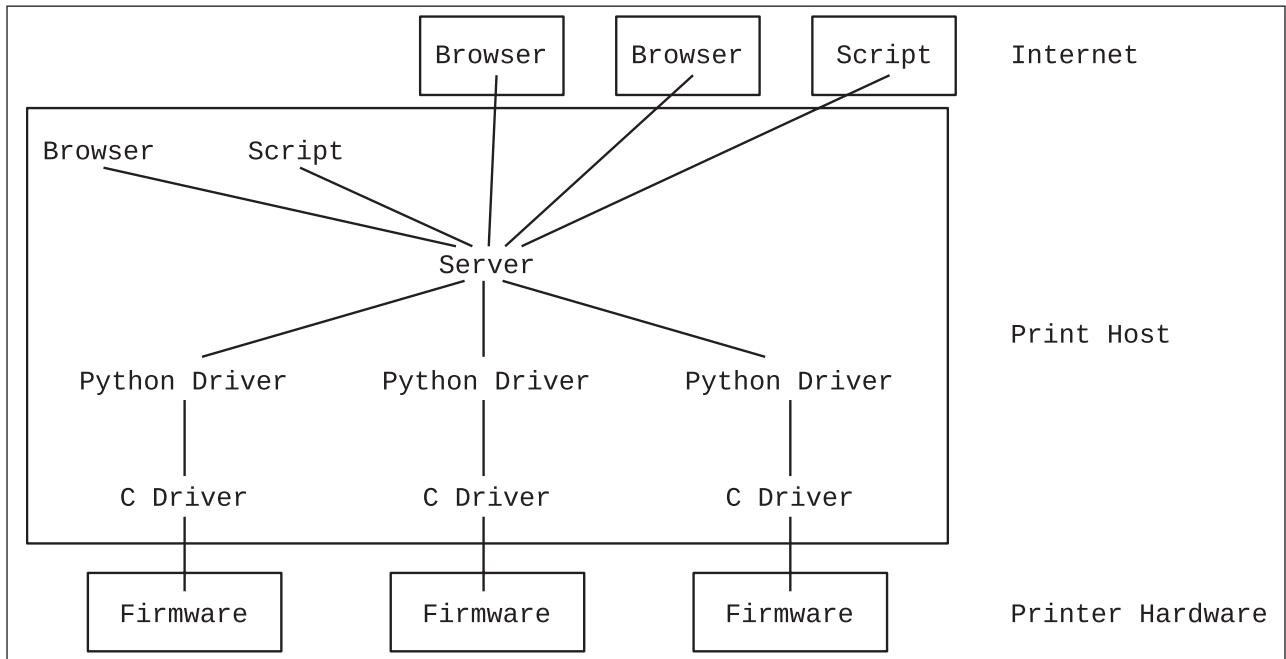


Figure 1: Schematic of the workflow of creating a 3-D printed object using Franklin.

3-D printer design [23]. Rather than including parts produced via fused filament fabrication, the updated 3-D printer design was constructed entirely from metal parts to reduce damage caused by weld splatter. Franklin Firmware and the improved metal 3-D printer were used to establish low-cost substrate release mechanisms that allow metal 3-D printed objects to be removed from a metal print substrate with minimal force [23].

Franklin Firmware

Franklin Firmware only handles tasks that cannot be performed well by the host computer as the microcontroller lacks processing power and memory (8 bit, 16 MHz, no hardware assisted floating point operations) compared to the host computer's processor. Fewer microcontroller tasks ensures more time to complete them, improving overall system performance. Franklin goes further than Pacemaker in this regard. Pacemaker assumes all machines to be Cartesian, and approximates moves on any machine that is not. Alternatively, Franklin allows the host to compute exactly when to execute a step with each motor, and the firmware will do them at the requested time, regardless of mechanical design. However, due to limits on the bandwidth of the serial port, this ideal situation only occurs at very low speeds (below 200 steps per second, which equates to around 4 mm per second, depending on the printer design). At higher speeds, the moves are interpolated in the same way as Pacemaker and most other firmwares handle non-Cartesian configurations. Franklin Firmware handles temperature controls, general purpose input/output (GPIO) pins, and control of stepper motors.

Temperature Controls

The Arduino has an analog-to-digital converter (ADC) multiplexed to several pins. On one or more of those pins, a voltage divider with a thermistor is normally used to

measure the temperature of extruders and the heated bed of conventional RepRaps. Franklin Firmware continuously reads all analog inputs that are set up as temperature controls and controls two GPIO (general-purpose input/output) pins based on the measured value. One of the controlled GPIOs is connected to the heater and the other controls a cooling fan. The heater and fan controls each have set points and the state of the GPIOs upon reaching set point is configurable. It is also possible to simply measure temperature without any control activity. The temperature reading is communicated to the host, so it can be used by scripts and shown in a browser.

Updating the heater and fan is not time sensitive so it is reasonable to take that part of the controls out of the firmware and implement it in the driver instead. However, it is important that these controls never fail: if a heater remains switched on while the extruder is very hot, it will damage the machine and can be a fire hazard. Were temperature control implemented in the Franklin Driver instead of Franklin Firmware, this scenario could occur when the serial connection is lost. Therefore, a simple heater control system is integrated into the Franklin Firmware. A more complicated system, such as a PID controller, could still be implemented in the Franklin C Driver. In that case, the system in the Firmware would serve as a protection against overheating.

GPIO Controls

Some printers and CNC machines have a need for controlling extra components. In most cases, they need simple digital signals, which can be either on or off. For example, the open source metal RepRap 3-D printer needs to switch a metal inert gas (MIG) welder (also known as gas metal arc welding or GMAW) on and off. Every microcontroller has a number of digital GPIO pins that can be used for this purpose. Franklin Firmware provides a simple interface to

use them: it allows the pins to be set in any of four states: output low, output high, input, or input with pull up resistor. Scripts can read the values from input pins. In addition to storing a current state of each pin, it also stores a reset state. When a reset is requested, or when the connection is lost for some time, the pins are all changed to their reset state to avoid problems (for example, the welder will be switched off upon permanent loss of communication with the host).

Stepper Motor Control

The hardware may have any number of stepper motors to move the tool in space, and to activate the tool (such as to extrude material from the nozzle in fused filament fabrication). However, the motors do not always correspond directly to a simple direction in Cartesian space. For example, on a delta bot, all three positioning motors need to move simultaneously for the tool to move in any straight line. As the conversion from tool position to motor positions can be complex, this conversion is performed in the Franklin C Driver.

Franklin Firmware only has a concept of motor positions. The Franklin C Driver sends it a list of numbers of steps to take, which can be positive or negative. Franklin Firmware steps through this list at a constant speed and sends the steps to the motors at the requested times. ADC readings are performed in the background, leaving the microcontroller to focus resources on motor control so that steps can be very well timed leading to smoother movement.

Franklin C Driver

The computationally intensive conversion from tool position to motor position is implemented in C++ as Python is not fast enough. It groups motors into “spaces”. Every space has a Cartesian coordinate system where the tool can be positioned, and a certain geometry. Currently, Cartesian and delta geometries are implemented, while other geometries are left for future work. 3-D printers will normally consist of two spaces: one for moving the 3-dimensional tool and one for the 1-dimensional extruder. A 3-D printer with multiple extruders has a higher dimensionality for its extruder space.

The Franklin C Driver receives instructions similar to G-code from the Franklin Python Driver. It converts those into the step maps that the Franklin Firmware needs, and sends them to the printer controller. In addition, it passes GPIO and ADC commands between the Franklin Firmware and the Franklin Python Driver.

Franklin Python Driver

As many tasks as possible are implemented in the Franklin Python driver, including interface provision with both scripts and browsers. The Franklin Python driver informs interfaces of changes, parses G-code they send, and stores settings. All settings, including pin assignments, can be changed without rewriting the firmware to the microcontroller. In addition, settings can be written to hard disk as machine profiles to make sure they are available for reloading the next time the machine starts. Any number of profiles are supported permitting settings for different

purposes. For example, if a machine can accommodate different tools, a different profile may be used for each tool.

Franklin Server

Multiple machines are handled concurrently by the Franklin Server. It detects devices when they are connected and starts the Franklin Python Driver for them, which in turn starts its own Franklin C driver. If a device is detected for which a Franklin Python Driver had already been started (that is, one that had lost its connection), the Franklin Server informs this driver of the reconnected machine and it will resume its operation. If this happens before the buffer in the Franklin Firmware runs out, it has no effect on the currently running job.

The Franklin Server provides a web interface, similar to Octoprint. (but without a webcam, although this is easily added). This means that any device with a browser, including a tablet or smartphone, can be used to control it. The web interface makes heavy use of Javascript to present all the changes that are reported by the Franklin Server to the user. It communicates with the Franklin Server using websockets. This same websockets interface is also exposed to other scripts. The web site allows manual control of the machine while scripts can integrate it with any automated or manual system.

The web interface allows all settings to be changed, including pin assignments, thermistor values and printer geometry (**Figure 2**). These changes take effect immediately. For example, if a request to double the number of steps per millimeter for the extruder drive is received while printing a segment, the remainder of that segment outputs twice as much filament per millimeter than before the change.

Additionally, if a change in machine geometry is made while the motors are enabled and idle, the tool is moved to the current position according to the settings describing the new geometry. This allows for easy machine calibration, including calibration types that are challenging with traditional systems such as the printer radius of a delta. Franklin instructs the printer to move to a height of 0. Then the limit switch positions are changed and the nozzle will move to reflect that change. When the nozzle touches the build platform, the limit switch position is correct. Then the nozzle is instructed to move horizontally to the edge of the build platform. Now the radius is changed and the nozzle will again move to reflect that change. When the nozzle touches the build platform again, the calibration is complete. The whole procedure can be completed within one minute with no additional sensors.

Communication

There are four communication interfaces in Franklin. The first is a websockets connection to the browser or script clients. Every packet on this connection is a JSON array, containing a numerical ID chosen by the client, the name of the function to call, a list of regular arguments and a list of keyword arguments. The server calls the requested function with the given arguments and returns either a value or an error, also encoded as a JSON array. This array contains the ID that was sent with the request, so

Spaces:	2					
Temps:	1					
Gpios:	1					
Spaces	Name	Type	Max Deviation (mm)			
Position		Cartesian ▾ Set Delta	Max Deviation 0.00			
Extruders		Cartesian ▾ Set Extruder	Max Deviation 0.00			
All Spaces		Cartesian ▾ Set	Max Deviation			
Cartesian/Extruder	Number of axes					
Extruders	1					
Axes	Name	Park pos (mm)	Park order	Max v (mm/s)	Min (mm)	Max (mm)
X		0.0	1	150	-200.0	200.0
Y		125.0	1	150	-200.0	200.0
Z		160.0	0	100	0.0	160.0
All axes 0						
E0		NaN	0	100	-Infinity	Infinity
Motor Settings	Name	Home order	Limit v (mm/s)	Limit a (mm/s²)		
U		0	150	4000.0		
V		0	150	4000.0		
W		0	150	4000.0		
All motors 0						
E		0	100	10000.0		
Motor Hardware	Coupling (steps/mm)	Microsteps	Switch pos (mm)			
U	53.333	16	241.100			
V	53.333	16	241.900			
W	53.333	16	241.700			
All motors 0						
E	90.000	16	0.000			
Delta	Min Axis Distance (mm)	Max Axis Distance (mm)	Rod Length (mm)	Radius (mm)		
U	0.0	Infinity	250.243	125.100		
V	0.0	Infinity	250.243	125.100		
W	0.0	Infinity	250.243	125.100		
All motors 0						
Delta	Angle					
Position	0.00					
Extruder	Offset X	Offset Y	Offset Z			
E0	0.0	0.0	0.0			
Temp Settings	Name	Fan temp (°C)	Bed			
E		50				
Temp Hardware	R0 (kΩ) or a (1000)	R1 (kΩ) or b (1000)	Rc (kΩ)	Tc (°C)	β (1) or NaN	
E	10.0	Infinity	100.0	20	3885	
Gpio	Name	Reset state	Fan	Spindle		
Fan		Off ▾	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
LED	D28 ▾	<input checked="" type="checkbox"/> Valid	<input type="checkbox"/> Inverted	Set		
Probe	D27 ▾	<input checked="" type="checkbox"/> Valid	<input checked="" type="checkbox"/> Inverted	Set		
U						
Step	D15 ▾	<input checked="" type="checkbox"/> Valid	<input checked="" type="checkbox"/> Inverted	Set		

Figure 2: Franklin web interface.

the client can send asynchronous messages. A list of supported functions is given in **Table 2**.

The second interface is between the Franklin Server and the Franklin Python Driver. The commands are sent over standard input and standard output of the Franklin Python Driver. The protocol is identical to the one described above, except that the JSON packets are not wrapped in a websocket before being sent. When the functions in **Table 2** are requested from the Franklin Server, it will pass them through to the Franklin Python Driver. The exception are the functions marked “server only”; those are handled by the server and not valid for this interface.

The third interface is between the Franklin Python Driver and the Franklin C Driver, and is different from the others. The Franklin C Driver should not be burdened with parsing JSON packets, so instead it accepts binary data which is more easily extracted from the packet. Every packet starts with a byte containing the length of the packet, followed by a command byte. Each command has its own arguments. A list of all commands that the Franklin C Driver understands is given in **Table 3**.

Commands from the Franklin C Driver to the Franklin Python Driver also start with a length byte and a command byte. Then follow three 32 bit integers and a 32 bit

floating point value. For most commands, 18 bytes are all that is required. The commands that may need more bytes (DATA and RUN_FILE) have a larger length value and follow those values with a list of bytes.

When a GOTO or PROBE command is received, a reply is sent, which is either OK or WAIT. In the latter case the queue is full and no further GOTO or PROBE commands can be sent until the Franklin C Driver sends a CONTINUE command.

The last interface is between the Franklin C Driver and the Franklin Firmware. This interface has unique qualities because it is implemented over a physical serial line that is much slower and unreliable. Because of the large amount of data that needs to be transmitted over this slow interface, the length byte is not sent, and the length of packets must be determined from their content. Because reliability is important, a checksum is added to maintain packet integrity.

Data is either a single byte command or a packet. The single byte commands all have their highest bit set, and are otherwise chosen to have a maximum distance between them in terms of bit flips required to go from one to the other. A packet starts with a command, none of which has the highest bit set. The command byte is followed by arguments. One or more checksum bytes are added at the end of the packet.

Command	Function
upload	Flash new firmware to an Arduino. (server only)
find_printer	Search for a machine by UUID or port. (server only)
set_autodetect, get_autodetect	Whether the server tries to detect a machine on newly discovered ports. (server only)
disable	Deactivate the machine. (server only)
detect, detect_all	Detect a machine, or all printers. (server only)
add_port, remove_port	Notify server of port discovery. (server only, called from kernel signals)
get_ports	Get list of available ports. (server only)
set_default_printer, get_default_printer	Which machine is used by new connections. (server only)
set_printer, get_printer	Which machine is used by this connection. (server only)
set_monitor, get_monitor	Whether this connection should be informed of changes in settings. (server only)
reset	Reset the Arduino.
die	Close the Python and C driver.
flush	Wait for queue to be empty.
probe	Map an area using a probe signal or manual feedback.
goto	Move motors to a position.
gotocb	Move motors to a position and wait for it to arrive.
sleep	Change sleep state of the motors.
settemp	Change the set point of a temperature control.
waittemp	Signal an alarm when a temperature enters a specified range.
readtemp	Request current temperature.
readpin	Request current value of a GPIO pin.
load, save, list_profiles, remove_profile, set_default_profile	Profile management.
abort	Disable heaters, sleep motors, reset GPIO pins and stop any running G-code.
pause	Pause or resume the currently running G-code.
queued	Request length of the queue.
home	Recalibrate the machine with limit switches.
park	Home if required, then go to park position.
wait_for_temp	Wait until an alarm from waittemp is triggered.
clear_alarm	Clear the waittemp alarms.
export_settings	Retrieve all settings for storing in a text file.
import_settings	Change settings from a text file.
gcode_run	Run (parsed) G-code.
request_confirmation	Wait for the user to press a button or abort.
confirm	Signal confirmation.
queue_add	Parse G-code and add it to the queue.
queue_remove	Remove an entry from the queue.
gcode_parse	Parse G-code and return the result.
gcode_bbox	Find the bounding box of parsed G-code.
queue_print	Send one or more queue entries to the machine.
queue_probe	Probe the combined bounding box of one or more entries, then send them to the machine.
get_globals, set_globals	Manage global settings
get_axis_pos	Get current position of an axis.
set_axis_pos	Set current position of an axis, without moving the motors.
get_space, get_axis, get_motor, set_space, set_axis, set_motor	Manage space settings.
get_temp, set_temp	Manage temp settings.
get_gpio, set_gpio	Manage GPIO settings.
send_printer	Request current state of a machine.

Table 2: Supported functions.

Command	Function	Response
RESET	Reset the machine.	
GET_UUID	Get universally unique identifier for the machine.	UUID
GOTO	Add segment to move queue.	MOVECB
RUN_FILE	Run a parsed G-Code file from disk.	UPDATE_TEMP, UPDATE_PIN, CONFIRM, FILE_DONE
PROBE	Like goto, and monitor probe pin to abort move and notify Python Driver about position.	MOVECB, LIMIT
SLEEP	Enable or disable the motors.	
SETTEMP	Set a temperature target.	
WAITTEMP	Set an alarm.	TEMPCB
READTEMP	Read current temperature.	TEMP
SETPOS	Set current axis position.	
GETPOS	Get current axis position.	POS
READ_GLOBALS	Get global settings.	DATA
WRITE_GLOBALS	Set global settings.	
READ_SPACE_INFO, READ_SPACE_AXIS, READ_SPACE_MOTOR	Get space settings.	DATA
WRITE_SPACE_INFO, WRITE_SPACE_AXIS, WRITE_SPACE_MOTOR	Set space settings.	
READ_TEMP	Get temp settings.	DATA
WRITE_TEMP	Set temp settings.	
READ_GPIO	Get GPIO settings.	DATA
WRITE_GPIO	Set GPIO settings.	
QUEUED	Request queue length and optionally abort move.	QUEUE
READPIN	Get GPIO pin state.	PIN
HOME	Move away from limit switches until it no longer hits them.	HOMED
RECONNECT	Machine has reconnected at this port.	
RESUME	Resume running a file that was previously paused.	
GETTIME	Get time estimates about the current job.	TIME
(asynchronous)	Notify that the state of an input GPIO has changed.	PINCHANGE
(asynchronous)	A limit switch was triggered during a move.	LIMIT
(asynchronous)	The machine was disabled due to a timeout.	TIMEOUT
(asynchronous)	The connection to the machine was lost.	DISCONNECT

Table 3: Commands for the Franklin C Driver.

In G-code, the checksum is computed by summing all bytes of the packet and using the lowest 8 bits of the result. This is very weak, and two flipped bits have a large chance of resulting in a valid checksum, even though the packet is incorrect.

Franklin uses a Hamming code [24] with one parity byte for every three bytes of data. That byte contains five parity bits, each of which set the parity of a selected group of bits to be even. The groups are carefully chosen to maximize the distance between valid packets. Two bit flips can never result in another valid packet, and more random flips are very unlikely to do so.

Detection of corrupt packets is required but is not sufficient for a reliable connection. Corrupt and lost packets must also be properly handled. When it is detected that a packet did not arrive, it is sent again. This means that if the packet was received but the acknowledgment was not, duplicate packets may be received. This must also be handled. The method for this has been copied from the USB standard: Each packet has one bit which indicates if it is an even or odd packet. Even packets are acknowledged with ACK0, odd packets with ACK1. If an even packet is received after an even packet, then the original ACK must have been lost. In that case, another ACK0 is

sent, but the packet is ignored, because the original even packet has already been handled. Odd packets are handled similarly.

This system allows any amount of lost or corrupted packets with no effect on the reliability of the link. Because the Franklin Server is capable of detecting that a new printer is the same as one that has previously disconnected, it can pick up the connection and continue as if nothing happened. If this happens in the middle of a print, it will only pause for a moment. If the connection is restored before the queue in Franklin firmware is empty, no pause will occur. **Table 4** lists all the commands that are supported.

Controlling Movements

A movement request to the C Driver consists of two speeds, F0 and F1, and a target position for each dimension in each space. F0 is the requested starting speed, while F1 is the requested finishing speed. The tool must move on a linear path and accelerate at a constant rate during the segment.

F0 and F1 are limited to maximum values which are set for each axis. Because of this, the common way to move fast is to request a speed of infinity. That will make the tool move at the maximum allowed speed configured in settings.

While moving, the speed and acceleration of every motor is limited as well. On a Cartesian system, the relationship between motor speed and axis speed is linear and simple. But this is not true for other machine geometries. For example, on a delta system motors are changing speed constantly to keep the nozzle moving in a horizontal line at a constant speed. Franklin Firmware limits the motor speed and acceleration so it is able to avoid missing steps even when the tool is at the edge of the build volume where one motor must move a large distance to effect a small amount of distance by the tool. Franklin Firmware accomplishes this without slowing the system down when it is more near the center.

Setting limits on acceleration has a very negative influence on print speed, especially if there are many short segments, such as in curves with small radii. This is due to a discontinuity in the direction of the path that has an infinitely large acceleration for any speed other than zero. To solve this, Franklin Firmware has a setting for how much to deviate from the requested path. Franklin Firmware will cut the corners by that amount, and it will use this curve to gradually change the speed of all the motors.

Figure 3 schematically shows how the move comprising a segment is prepared. This occurs when the previous move has completely finished; in the case of a deviation from the path, the move has already started at this point (**Figure 3**).

Command	Function	Response
BEGIN	Handshake; send version and receive capabilities.	READY
PING	Handshake.	PONG
RESET	Reset the Arduino.	
SETUP	Set up globals.	
CONTROL	Manage GPIO pin states.	
MSETUP	Set up motor settings.	
ASETUP	Set up ADC (temp) settings.	
HOME	Move away from limit switches.	HOMED
START_MOVE	Begin sending movement buffers.	
START_PROBE	Begin sending movement buffers for probing.	
MOVE	Send movement buffer for one motor.	
START	Begin moving.	
STOP	Stop moving and discard buffer.	STOPPED
ABORT	Stop moving, disable all motors, reset all heaters and GPIO.	STOPPED
DISCARD	Discard a part of the queued buffers without stopping the current move.	
GETPIN	Read current state of a GPIO pin.	PIN
(asynchronous)	Buffer has been completed.	DONE
(asynchronous)	Buffer has been completed and no next buffer is available.	UNDERRUN
(asynchronous)	ADC has been measured.	ADC
(asynchronous)	Limit switch has been triggered.	LIMIT
(asynchronous)	Machine has been deactivated due to timeout.	TIMEOUT
(asynchronous)	The value of an input pin has changed.	PINCHANGE

Table 4: Supported Commands, Function and Response.

For that reason, every axis stores the distance to move for the current segment, and for the next segment. After filling those values, the speeds F0 and F1 are limited to what the axes are set to allow. Then the position where the curved connection should start is computed and finally all variables are filled with their values.

Probing

For 3-D printing and most other applications, the machine can be calibrated once with the assumption that the surface is flat and horizontal and it does not require recalibration. However, for milling PCB circuits, it is very important that the distance into the surface is tightly controlled. For this purpose, Franklin supports probing the surface before running a job, and using the measured probe map to correct for deviations from the horizontal flat ideal. This can also be used for example to print on top of complex geometries or to repair products.

Scripting

The above describes how Franklin is useful for controlling a standalone 3-D printer or other CNC machine. However, in a scientific environment a machine is normally part of a larger setup, rather than only a standalone 3-D printer or CNC machine. Franklin allows tight integration with the rest of the setup by supporting an extension to G-code, which allows running system commands. This can be used to control other parts of the setup, for example to record an image with a camera. Because running system commands is a security risk, this feature is disabled by default and it cannot be enabled from scripts or browsers, only at startup using a configuration file or commandline switch.

G-code does not support input other than waiting for a button. Instead, the websockets interface allows direct control over the printer without using G-code. Using this interface, a script can generate a movement pattern in real time. For example, when a camera is connected as a tool, a script could use the images from it and continuously move it to keep a moving specimen centered in the field of view and in focus.

Quality control

All compiler warnings are enabled for both Franklin Firmware and the Franklin C Driver. Stack protection was disabled because the AVR platform does not support it. Valgrind [25] was used to find buffer overflows and use of uninitialized variables in the Franklin C Driver.

Franklin has been used extensively on a variety of devices, ranging from a 1-dimensional syringe pump to 3-dimensional printers. Firebug [26] was used to find errors in the HTML and Javascript. Franklin Server delivers web pages written in HTML 5 and Javascript (Ecma 262). Care has been taken to follow the standards; no browser extensions have been used.

Franklin has been used successfully on a variety of different machines listed above. In addition, Franklin has been used in conjunction with a metal inert gas (MIG) weld-based 3-D printer to develop substrate release mechanisms for 3-D printed parts [23]. Traditionally, metal 3-D printed parts must be removed from a print substrate

with the use of saws or other machining equipment. This removal step is undesirable as it results in excessive material waste and has additional associated time and cost requirements. In order to evaluate possible substrate release mechanisms, Franklin was used to print aluminum lap shear test specimens on aluminum and steel print substrates. This study observed that low-cost options, such as boron nitride coatings, and no-cost options, such as printing aluminum on steel substrates, minimized the amount of force required to remove metal specimens from a print substrate.

(2) Availability

Operating system

Franklin has been designed for and tested on Debian GNU/Linux [27], on i386, amd64 (PC) and armhf (Raspberry Pi [28] and Beaglebone Black [29]) architectures. It should work on any other GNU/Linux system and possibly other Unix-based systems. The packages have been built on the latest stable (jessie, or Debian 8) and unstable versions, and using backports [30] with some packages from jessie, also on oldstable (wheezy, or Debian 7).

Additional system requirements

Franklin is only useful if there is hardware connected to control. It does not have any other requirements.

Dependencies

- Python-fhs [31]: module for reading and writing files in the proper place according to the Filesystem Hierarchy Standard [32]
- Python-network [33]: module for using network connections, including SSL.
- Python-websockets [34]: module for hosting a web server that can communicate using websockets.

List of contributors

Bas Wijnen (code development).

Software location

Archive

Name: Purl.org

Persistent identifier: <http://purl.org/NET/mtu-most/franklin>

License: GNU Affero General Public License, version 3 or later.

Publisher: Michigan Technological University.

Date published: 09/06/2013.

Code repository

Name: GitHub

Identifier: <https://github.com/mtu-most/franklin>

License: GNU Affero General Public License, version 3 or later.

Date published: 09/06/2013.

Language

All code, comments and documentation are in American English.

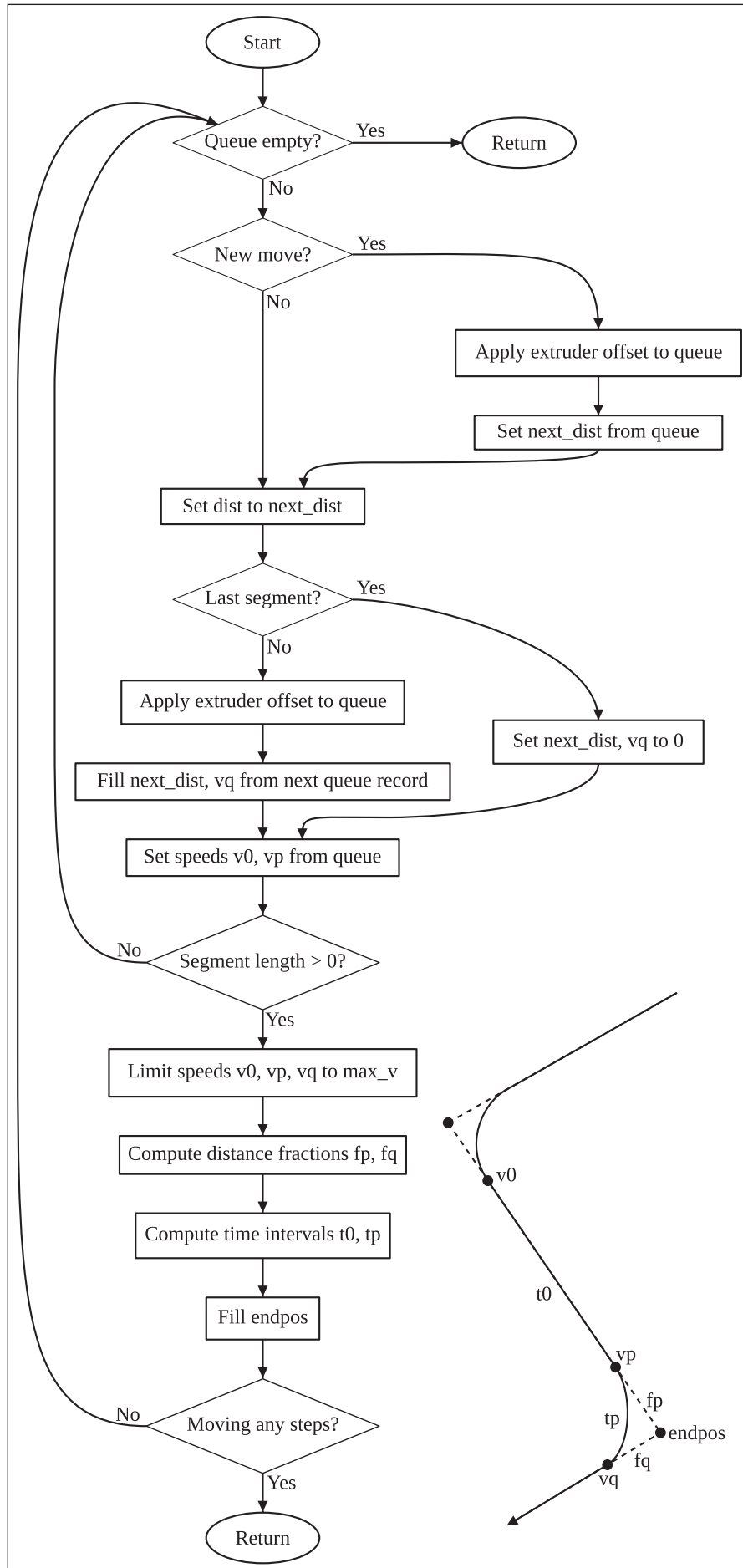


Figure 3: Schematic representation of how the move comprising a segment is prepared in Franklin.

(3) Reuse potential

In its current state, Franklin can be used for controlling 3-D printers and other manufacturing machines to produce research-grade equipment. Such equipment can also be controlled with Franklin.

While the web interface is targeted at 3-D printers and similar machines, the code is written in a way that makes it easy to design an alternative interface. For example, a syringe pump could be controlled with an interface that allows programming a sequence of flows in the way that users of syringe pumps expect. Such a program would use the web-sockets interface of Franklin to control the hardware, while presenting the user with an interface that is more appropriate for the application than the default CNC interface.

Because it is free software users can improve Franklin and the hardware to fit their requirements. Franklin is in active development and the developer can be contacted through github.

Possible improvements, which are considered or being worked on include:

- G-code parsing is presently time consuming and processor intensive. Performance can be improved through the use of a compiled language such as C++ instead of Python.
- One-click printing support like that common for 2-D printers would vastly improve ease-of-use. This functionality is a logical extension to common 3-D modeling packages (Blender, OpenSCAD, FreeCAD, etc.) and even web browsers e.g. when browsing objects hosted on 3-D printer aggregate sites such as Youmagine.com, 3dprint.nih.gov or Thingiverse.com.
- Running a G-code converter (a slicer for 3-D models) as part of Franklin would allow users to send model files directly to Franklin, again improving ease-of-use. (This is a requirement for the previous point, but is also a useful feature in itself.)
- Handling a microcontroller reset or loss of power on the entire system, including the host computer would be useful for those using 3-D printers in the developing world, where power is less reliable. It would also be useful when using very large 3-D printers, such as the Gigabot, which print for many hours to produce a single object.
- Because Franklin parses G-code before running it, and because it does this on a relatively powerful computer, it has the option to thoroughly analyze it and take action based on the relatively distant future.
- The BeagleBone contains all the features that are needed for the microcontroller and the host. It would be possible to use just the BeagleBone for the entire control system.

Conclusions

Firmware known as Franklin was developed to mitigate limitations associated with other CNC and 3-D printer control systems. Franklin was successfully demonstrated on a wide range of RepRap-derived devices: Mendel RepRap, Delta RepRap, Quad Delta RepRap, OS laser welder, PCB

micromill and the open source metal 3-D printer. Franklin demonstrated the ability to support the maker movement with low-cost open-source control of three dimensional additive and subtractive fabrication as well as scientific analytical equipment. Low-cost RepRap 3-D printers are being taken more seriously in the scientific and industrial worlds. Franklin improves upon this by allowing more functionality and better integration with new or existing systems.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The authors would like to acknowledge valuable discussions with Lars Pötter, and Markus Hitter on the RepRap mailing list, reppap-dev@lists.reppap.org.

References

1. **Jones, R, Haufe, P, Sells, E, Irvani, P, Olliver, V, Palmer, C and Bowyer, A** 2011 RepRap—the replicating rapid prototyper. *Robotica*, 29(01): 177–191. DOI: <http://dx.doi.org/10.1017/S026357471000069X>
2. **Sells, E, Smith, Z, Bailard, S, Bowyer, A and Olliver, V** 2009 RepRap: The Replicating Rapid Prototyper: Maximizing Customizability by Breeding the Means of Production. *Handbook of Research in Mass Customization and Personalization*.
3. **Bowyer, A** 2014 3D Printing and Humanity's First Imperfect Replicator. *3D Printing and Additive Manufacturing*, 1: 4–5.
4. **Pronterface, Pronsole, and Printcore—Pure Python 3d printing host software**, <https://github.com/kliment/Printrun>
5. **Cura download page**, <http://software.ultimaker.com>
6. **Octoprint 3-D printer controller**, <http://octoprint.org>
7. **Pacemaker, A host-client-system to control 3d printers, CNC milling or laser cutters**, <https://github.com/JustAnother1/Pacemaker>
8. **Wohlers, T T** 2013 Wohlers Report 2013: *Additive Manufacturing and 3D Printing State of the Industry: Annual Worldwide Progress Report*.
9. **Pearce, J M** 2012 Building research equipment with free, open-source hardware. *Science*, 6100: 1303–1304. DOI: <http://dx.doi.org/10.1126/science.1228183>. PMID: 22984059.
10. **Pearce, J M** 2014 *Open-Source Lab: How to Build Your Own Hardware and Reduce Research Costs*. Elsevier.
11. **Ventola, C L** 2014 Medical Applications for 3D Printing: Current and Projected Uses. *Pharmacy and Therapeutics*, 10: 704.
12. **Wijnen, B, Hunt, E J, Anzalone, G C and Pearce, J M** Open-source Syringe Pump Library. *PLoS ONE*, 9: e107216. DOI: <http://dx.doi.org/10.1371/journal.pone.0107216>. PMID: 25229451; PMCID: PMC4167991.
13. **Drescher, P, Spath, S and Seitz, H** 2014 Fabrication of biodegradable, porous scaffolds using a low-cost 3D printer. *International Journal of Rapid Manufacturing*, 2: 140–147. DOI: <http://dx.doi.org/10.1504/IJRAPIDM.2014.066035>

14. **King, D L, Babasola, A, Rozario, J and Pearce, J M** 2014 Mobile Open-Source Solar-Powered 3-D Printers for Distributed Manufacturing in Off-Grid Communities. *Challenges in Sustainability*, 1: 18–27. DOI: <http://dx.doi.org/10.12924/cis2014.02010018>
15. **Pearce, J M, Blair, C M, Laciak, K J, Andrews, R, Nosrat, A and Zelenika-Zovko, I** 2010 3-D printing of open source appropriate technologies for self-directed sustainable development. *Journal of Sustainable Development*, 4: 17. DOI: <http://dx.doi.org/10.5539/jsd.v3n4p17>
16. **Birtchnell, T and Hoyle, W** 2014 3D Printing for Development in the Global South: The 3D4D Challenge, Palgrave Macmillan. DOI: <http://dx.doi.org/10.1057/9781137365668>
17. **Franklin, 3-D printer and CNC controller**, <https://github.com/mtu-most/franklin>
18. **Arduino prototyping platform**, <http://www.arduino.cc/>
19. **Wittbrodt, B T, Glover, A G, Laureto, J, Anzalone, G C, Oppliger, D, Irwin, J L and Pearce, J M** 2013 Lifecycle economic analysis of distributed manufacturing with open-source 3-D printers. *Mechatronics*, 6: 713–726. DOI: <http://dx.doi.org/10.1016/j.mechatronics.2013.06.002>
20. **Irwin, J L, Pearce, J M, Oppliger, D and Anzalone, G C** 2014 The RepRap 3-D Printer Revolution in STEM Education. *121st ASEE Annual Conference and Exposition*, Indianapolis, IN. Paper ID #8696. Available at http://www.asee.org/file_server/papers/attachment/file/0004/4989/asee_reprap_paper_final1.pdf
21. **Anzalone, G C, Wijnen, B and Pearce, J M** 2015 Multi-Material Additive and Subtractive Prosumer Digital Fabrication with a Free and Open-source Convertible Delta RepRap 3-D Printer. *Rapid Prototyping* (in press).
22. **Anzalone, G C, Zhang, C, Wijnen, B, Sanders, P G and Pearce, J M** 2013 A low-cost open-source metal 3-D printer. *IEEE Access*, 1: 803–810. DOI: <http://dx.doi.org/10.1109/ACCESS.2013.2293018>
23. **Haselhuhn, A S, Gooding, E J, Glover, A G, Anzalone, G C, Wijnen, B, Sanders, P G and Pearce, J M** 2014 Substrate release mechanisms for gas metal arc weld 3-D aluminum metal printing. *3-D Printing and Additive Manufacturing*, 1(4): 204–209.
24. **Hamming, R W** 1950 Error detecting and error correcting codes. *Bell System technical journal*, 2: 147–160. DOI: <http://dx.doi.org/10.1002/j.1538-7305.1950.tb00463.x>
25. **Valgrind instrumentation framework for building dynamic analysis tools**, <http://valgrind.org>
26. **Firebug web development tool**, <http://getfirebug.com>
27. **Debian – The universal operating system**, <https://www.debian.org>
28. **Raspberry Pi**, <http://www.raspberrypi.org>
29. **BeagleBone Black**, <http://beagleboard.org>
30. **Debian Backports**, <http://backports.debian.org>
31. **Python-fhs**, <https://github.com/wijnen/python-fhs>
32. **Filesystem Hierarchy Standard**, <http://www.linuxfoundation.org/tags/filesystem-hierarchy-standard>
33. **Python-network**, <https://github.com/wijnen/python-network>
34. **Python-websockets**, <https://github.com/wijnen/python-websockets>

How to cite this article: Wijnen, B, Anzalone, G C, Haselhuhn, A S, Sanders P G and Pearce, J M 2016 Free and Open-source Control Software for 3-D Motion and Processing. *Journal of Open Research Software*, 4: e2, DOI: <http://dx.doi.org/10.5334/jors.78>

Submitted: 12 May 2015 **Accepted:** 06 January 2016 **Published:** 27 January 2016

Copyright: © 2016 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <http://creativecommons.org/licenses/by/4.0/>.