The International Journal of Advanced Manufacturing Technology (2020) 106:3609–3623
https://doi.org/10.1007/s00170-019-04713-y

ORIGINAL ARTICLE

FIBR3DEmul—an open-access simulation solution for 3D printing processes of FDM machines with 3+ actuated axes

Carlos Faria[1] · Jaime Fonseca[1] · Estela Bicho[1]

Received: 5 June 2019 / Accepted: 14 November 2019 / Published online: 7 January 2020

Abstract

This paper introduces a virtual emulator software for additive manufacturing (AM) processes based on filament deposition, the FIBR3DEmul. The presented software is capable of reading and parsing a G-Code file (ISO/DIN 66025), and realistically emulating a custom-designed 5-axis printer or a standard 3-axis Cartesian printer. The FIBR3DEmul was designed and implemented in two separate applications for reusability and scalability. First, the G-Code Interpreter is responsible for parsing the g-code script, controlling the flow of its execution, and notifying the user about detected printer-printer or printer-workpiece collisions. The second application involves the robotics simulator tool V-Rep. A custom plugin was implemented to mediate the communication with the Interpreter application, to generate the tool trajectories, to emulate the extrusion process, and to handle motion execution and collision detection. The process of designing and implementing a custom-printer control and motion execution in these two software is described. The performance of the virtual 5-axis printer was compared with the real machine in terms of position and velocity profiles. Results show a tight match between virtual and real printer-generated plots. The presented solution can also be extrapolated to CNC machines or WHASPs. The FIBR3DEmul source code is publicly available.

Keywords Additive manufacturing · Virtual simulation · G-Code interpretation · 5-axis printer

1 Introduction

Fused deposition modeling (FDM) is arguably the most popular additive manufacturing technology. Its applicability reaches well beyond the manufacturing industry, to most R&D laboratories as well as the general public. There are few limits to the versatility of conceptualizing and fabricating parts using an FDM process [7].

Following CNC-based technology, the generic FDM printer usually counts with the extrusion head, the print plate, multiple controlled axes, and the central control unit. Commonalities aside, different machine types are currently applied for 3D printing: Cartesian, Delta, Scara, or Polar. The vast majority, however, operates solely with 3 degrees of freedom (DoF) depositing consecutive thin layers of extruded material from the bottom-up.

Conceptually simple, this FDM process has known shortcomings such as limited motions, anisotropic material properties, stair-stepping, and the need to use supporting material to print structures with overhangs [4, 6]. One obvious solution to these shortcomings involves multi-axis deposition, where the actuated axes that control the tool or print table that holds the workpiece change orientation to permit material deposition in new planes.

Different architectures have been proposed for multi-axis deposition, 6-DoF Stewart mechanisms [4], industrial robotic arms with 6-DoF [5, 8], or adapted Cartesian systems with a rotatory print table [1, 3]. Independent of the followed approach, there is a significant shift in the 3D printing paradigm. First, the 3D model slicing algorithm needs to account for material deposition in different deposition planes; second, a new motion script protocol—typically a G-Code file—is required to address simultaneously every degree of freedom; and third, the printer should generate a coordinated motion between all

✉ Carlos Faria
cfaria@dei.uminho.pt

Jaime Fonseca
jaime@dei.uminho.pt

Estela Bicho
estela.bicho@dei.uminho.pt

[1] Centre Algorithmi and Department of Industrial Electronics, University of Minho, Guimarães, Portugal

actuated axes, in order to guarantee a correct position and orientation of the tool relative to the printing bed. Concurrently, the collision detection problem becomes significantly more complex. In a similar topic, Lauwers et al. [2] elaborate on how the tight integration between the development of new control algorithms for 5-axis milling machines and machine simulation permit optimization and collision avoidance.

FDM solutions with more than 3 DoF are still relatively rare, and consecutively there are limited resources to assist the development and testing of new solutions, which brings us to the main contribution of the FIBR3DEmul. The FIBR3DEmul is a software package that permits the test and validation of custom FDM systems with simultaneously controlled axes using a robotics simulation environment. The software package is divided into two parts: (i) an application to interpret the new G-Code standard (based on ISO/DIN 66025) to control up to 5-axis and (ii) a plugin developed for the robotics simulator—V-REP—to receive the commands from the parser tool and generate the motion of each individual axis accordingly.

Several advantages arise from splitting the simulation software into a two-part application:

- selection of better-fitting tools/frameworks for both G-Code parsing and Motion control or 3D virtualization;
- scalability and reusability of the developed solution, for example, a solution that can directly communicate with either the virtual or the real machine controller;
- allow the user to control the flow of G-Code reading and motion execution and still implementing a virtual controller "closer" to the emulated printer with reduced delay.

This solution can realistically emulate the behavior of the real machine at any workstation running the robotics simulator and permits testing and adjusting the control parameters without risking any damage to the material. It facilitates the G-Code introspection by enabling the user to run line-by-line, a specific block of lines, or the whole script at once. At the same time, the application tracks the line that is currently being executed and keeps track of commands that generate collisions between printer/printer or printer/work piece.

The FIBR3DEmul source code is available at: https://github.com/neuebot/FIBR3DEmul. It contains the code for the G-Code Interpretation application, as well as the code for the plugin developed to control the 5-axis printer and the virtual scene.

2 G-Code Interpreter

The "G-Code Interpreter" is a software application developed in C# (.NET Framework) to parse and interpret G-Code commands and forward the motion/actuation commands and respective parameters to a real or virtual controller. The interface permits the user to inspect the G-Code

Table 1 Supported G-Code commands and coordinate identifiers

G-Commands (modal and non-modal)	Coordinate identifiers
G00 - Rapid motion	X - *x*-axis position
G01 - Linear motion	Y - *y*-axis position
G02 - Arc motion clockwise	Z - *z*-axis position
G03 - Arc motion counter clockwise	I - center *x*-axis position
G04 - Dwell	J - center *y*-axis position
G08/09 - Acceleration/deceleration at block end	K - center *z*-axis position
G17/18/19 - Working plane *XY*/*ZX*/*YZ*	R - radius of circle
G53 - Deselection of current offset	A - filament advance
G54 to **G59** - Selection of 1st to 6th offset	B - *y*-axis rotation
G60 - Exact stop	C - *z*-axis rotation
G70/71 - Units inch/millimeters	F - advance velocity
G74 - Homing	
G90/91 - Absolute/incremental dimension	
G92 - Coordinate preset	
G161/162 - Circle center point absolute/relative	
G193/293 - Path/time interpolation	
G130 - Acceleration weight specific axis	
G131 - Acceleration weight all axes	
G231 - Acceleration weight all axes specific **G00**	

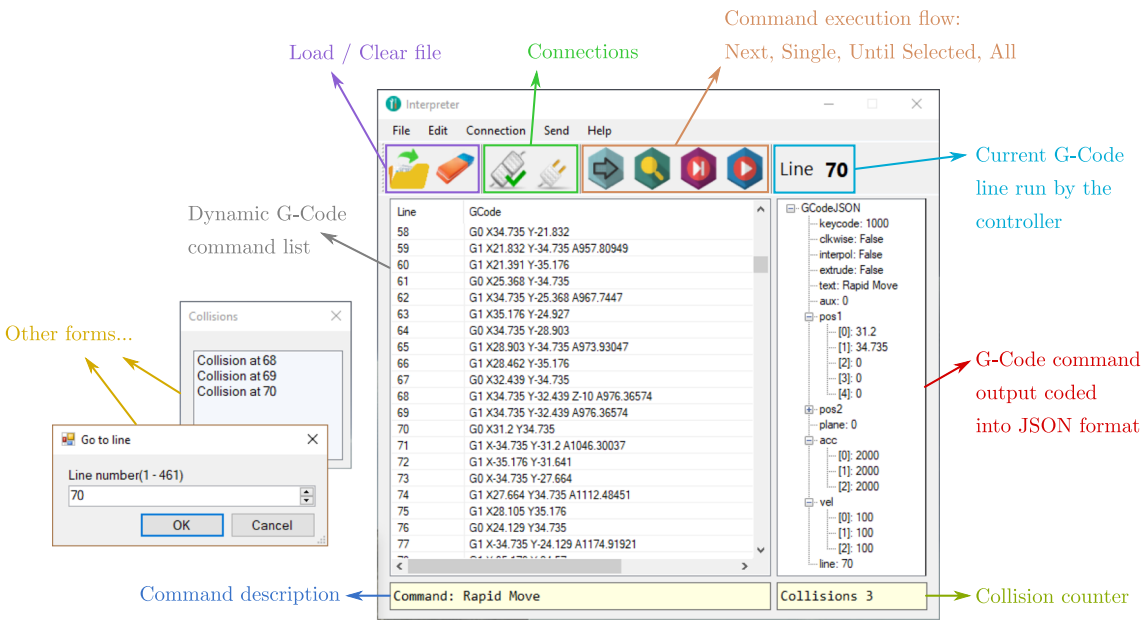


Fig. 1 Graphical user interface of the G-Code Interpreter application

line-by-line and selectively forward the commands to the controller. The ability to execute single commands or a block of commands is of invaluable help when testing or developing new routines. To avoid ambiguities with different G-Code standards, it only operates with G-Code files that comply with the norm ISO / DIN 66025.

The G-Code is a readable language commonly applied in computer-aided manufacturing. From tool path and velocity control, to coordinate system settings, the G-Code explicitly or implicitly defines the settings under which the machine operates and how it executes each command from the start to the end of a cycle. Commands are generally contained in a single line and preceded by a G- or M- character and a number identifier. The remaining characters code the parameters. The application parses each line from the G-Code file individually using string manipulation to isolate *command* and *parameter* "words". Then, depending on the command parsed, the read parameters are forwarded to a specific function that handles and repacks the information for the printer controller. A list of the implemented G-Code commands and parameters supported are presented in Table 1.

Two important details of G-Code parsing are embedded in the application: the *default* and the *modal* commands. G-Code has a rather vague specification of "grammatical rules", which makes the task of the interpreter more difficult. The G-Code may explicitly include every command detail—making each line extensive but self-contained—as opposed to relying on implicit information to reduce "wording" and include only the new details. This behavior is called *modality*. The context of a non-modal command is limited to its line, while a modal command is in effect until a new modal command of a related topic is read. Not only commands but also variables may be modal; i.e., by omission, the previously defined parameters are true until redefined (this is especially useful to avoid constantly defining feed rates). If a command for a specific topic is never explicitly defined, the default command is considered by omission [1].

To simplify the virtual controller logic, the implicit and explicit information contained in a G-Code line is translated into a single and unambiguous command sent to the printer controller. Messages are serialized into the JSON format (lightweight and readable) and exchanged with the virtual controller. The G-Code Interpreter works as the client in a client-server communication model based on TCP/IP sockets (.NET Berkeley socket implementation). The interpreter and the controller communicate asynchronously and bi-directionally. The applications may be launched from the same host or across the network.

The G-Code Interpreter has a graphical user interface (GUI) to interact with the G-Code file and send commands to the controller (see Fig. 1).

Initially, the user is required to provide a G-Code file, which can be opened through the menu or simply by a drag-and-drop operation. Once loaded, the dynamic list container displays the G-Code file in separate selectable lines. When a line of the list is selected, it is parsed and the interpreted

[1] For example, if no command relative to the actuation plane is specified: **G17***, **G18**, or **G19**, the printer defaults to the **G17** command (*X*-*Y* plane).

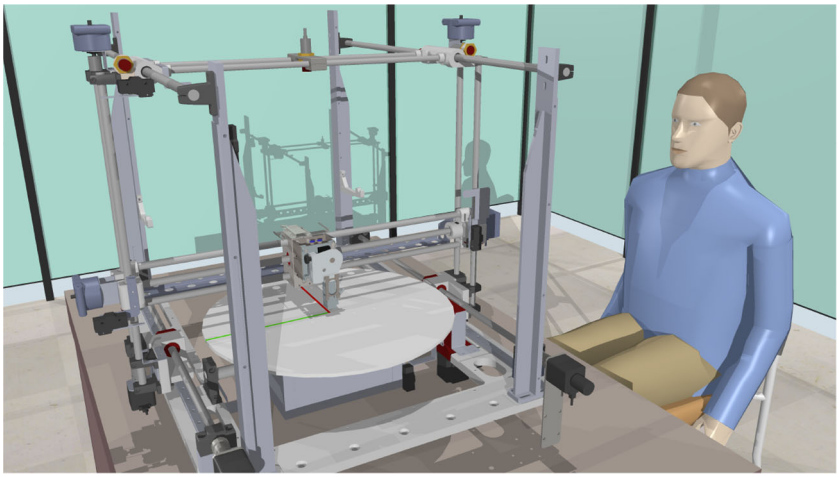


Fig. 2 V-Rep scene with imported and assembled model of the 3D printer

information is displayed as a JSON message in the view to the right. Before sending G-Code commands, the user is prompted to connect to the controller. Afterward, the user may control the G-Code command execution with the possibility to send:

- the next G-Code command in the list;
- a single selected command;
- all commands from the current until the selected line;
- all commands from the current line until the end of the file.

At any moment, the current line being executed by the controller is displayed. The application also keeps track of G-Code lines that results in collisions between the printer parts, or between the printer and the workpiece. These collisions are detected when the physics engine of the V-Rep simulator registers the intersection of physical bodies. These physical bodies are designed to mimic the graphical model of the printer or to represent the extruded material (refer to Section 3.2.1).

3 Virtual scenario

The second part of the FIBR3DEmul concerns the virtualization tool to emulate the 3D printer movements and the deposition process. Although these points are related, they concern two dissimilar problems: on one side, the goal is to simulate coordinated movements of the printer parts and tool path/velocity or check for possible collisions; and on the other side, the objective is to replicate the material deposition process. With no software solution capable of realistically simulating both processes, we set for the compromise of having a robotics simulator tool to replicate the behavior of the 3D printer, and to develop a process to visualize the material deposition.

The V-Rep (Coppelia Robotics, Zürich, Switzerland) software was selected as the robotics simulator. It offers a wide range of actuators, sensors, scenarios, and functionalities to facilitate the development and testing of new robotic platform solutions, such as the 5-axis printer. Moreover, V-Rep is a cross-platform and supports multiple programming languages and approaches, ranging from embedded scripts to remote APIs or client plugins.

3.1 Models

The proposed system was tested with the 5-axis "C3DPrinter" developed on the scope of the FIBR3D project (POCI-01-0145-FEDER-016414). It is based on the typical 3 DoF Cartesian printer with an additional 2 DoF rotating print table (see Fig. 2).

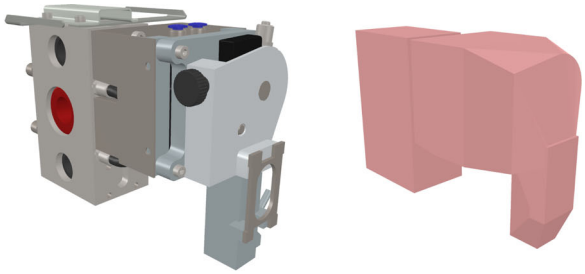
The 3D printer model designed in CAD software was imported as a set of independent meshes into a V-Rep scene. Physical parts of the models that are tracked for collisions are appended to the graphical counterpart, either by using an agglomerate of simple shapes (cubes, cylinders, and spheres) or by convex hull/decomposition (see Fig. 3). The printer parts are scaled to the dimensions of the real 3D printer, and prismatic/rotational joints are assigned between moving parts. Each joint is modeled after the hardware of the reference printer including the mechanical limits[2] and the control loop.

3.2 Extruder tool

The V-Rep robotics simulator lacks a feature to directly emulate the extrusion of material as required for FDM applications. Without the ability to accurately reproduce fluid mechanics or material interface phenomena, the

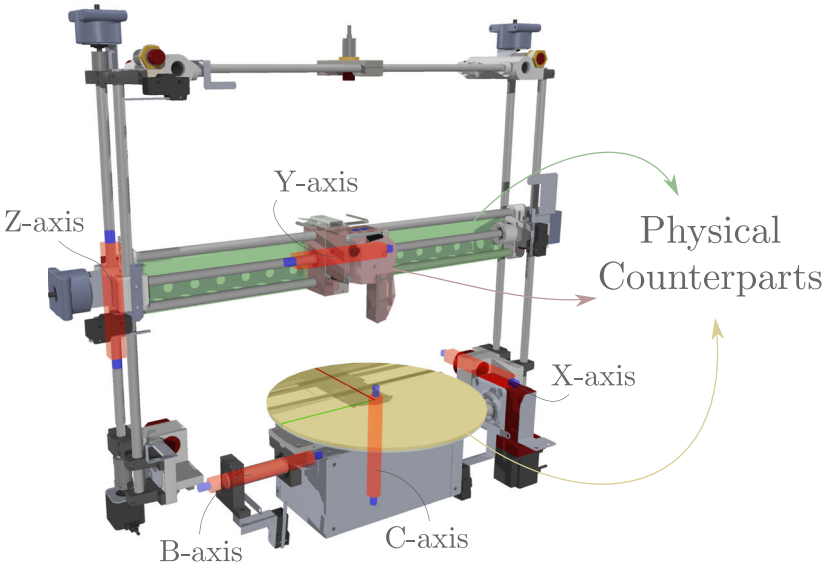
[2]No limits are specified for joint C as it is cyclic.

Fig. 3 **a**–**c** Extruder model graphical and physical part



(a) Extruder Graphical mesh imported from the CAD software.

(b) Extruder Physical part computed from a convex decomposition from its mesh.



(c) C3DPrinter moving parts. Graphical and physical parts represented, as well as the position of each of the 5-axis.

proposed application settles by representing the filament deposition as it follows the tool path.

The "extruder tool" was developed based on the drawing object functionality from V-Rep that permits painting shapes in runtime. The process of extruding the fused filament is replicated by continuously drawing shapes as

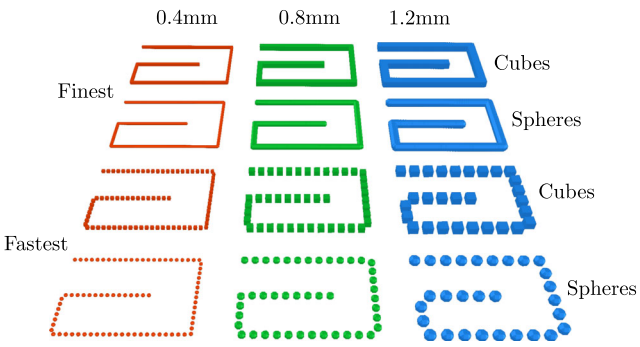


Fig. 4 Filament variations

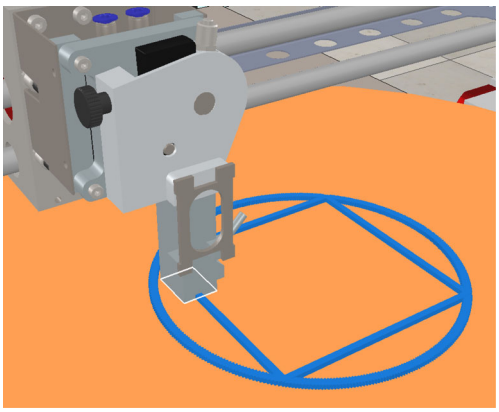
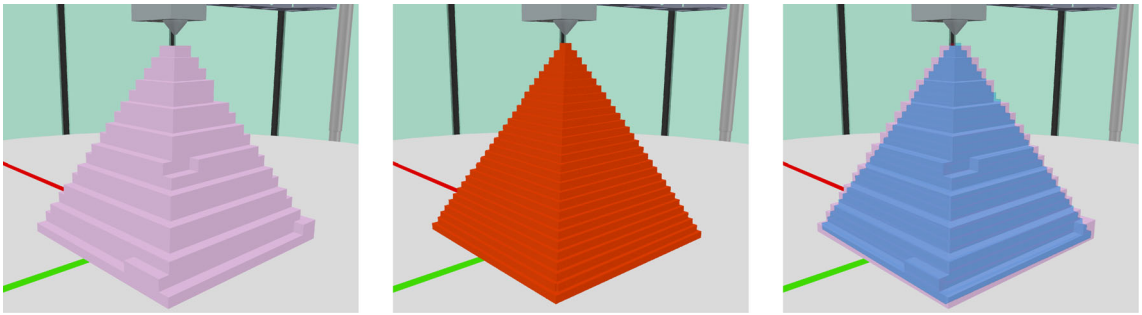


Fig. 5 Collision detected between the extruder and the printer table, which changed color to denote the occurrence



(a) Octree. (b) Filament. (c) Overlap.

Fig. 6 **a**–**c** Graphical and physical representation of the extruded filament

the tool moves relative to the print table.[3] New shapes are inserted into the scene at each simulation step to replicate the process of extruding the filament.[4] The simulator requires the position, orientation, and reference system where each shape is to be inserted—more about this process in Section 4.3.

The user is given the option to customize the filament in terms of shape type, color, size, and sparsity (see Fig. 4).

The type and the sparsity of the drawn shapes impact the simulation performance, especially noted in larger printed objects. A sphere-type filament more closely resembles the extruded filament form but it is 3 times more time consuming to draw as each sphere-shape is a 36-faced mesh as opposed to the 12 faces of a cube mesh.

3.2.1 Collision detection

One of the principal contributions of the proposed application is the ability to detect and notify about possible collisions between machine parts, or between machine and the workpiece. This proposition becomes significantly more relevant for prototype technology such as the "C3DPrinter", which involves simultaneous motion of 3+2 axes.

The V-Rep collision detection module checks at each simulation step for intersections between pairs of physical bodies. For the *printer/printer* collisions, the physics engine checks two collision pairs: between the print table and the extruder body, and between the printer table and the *Y*-axis bar (see Fig. 3c). When a collision occurs, the collided part changes color, and an outline is drawn to demarcate the intersection region, Fig. 5.

To detect *printer/workpiece* collisions, one has to attribute a physical form to the extruded filament, since the drawn shapes (Fig. 4) are only graphical objects. Endowing the filament with a physical body was achieved using octree objects. Octrees are feature objects provided with V-Rep that permit dynamically shaping a physical body in runtime by adding or subtracting "leaves" to the tree data structure. Leaves are isotropic voxels that occupy a partitioned space. The user may adjust the space partitioning dimensions to obtain a finer match between the octree shape and the filament shape at a performance cost (see Fig. 6). During filament extrusion, the extruder tool path relative to the printer table is tracked and these position coordinates are sent, at each simulation step, to the octree object handler. Internally, a function checks for the octree point occupancy. If the received position fits inside an existing octree voxel, nothing happens; else, a voxel is inserted in the new spatial partitioned cell. Two instances of *printer/workpiece* collisions are routinely checked: between the octree and the extruder, and between the octree and the *Y*-axis bar.

4 Controller plugin

A plugin for the V-Rep simulator was developed to communicate with the G-Code Interpreter and interact with the virtual 3D printer (see Fig. 7). The choice of using a plugin over other programming approaches was primarily weighted on the completeness of the available API and the fact that it interacts directly with the simulator's execution thread resulting in a *no-delay* communication between controller and actuators.

The plugin was written in C++ according to the V-Rep plugin guidelines and is loaded by the main client application at start-up. It includes three base objects:

1. *Printer* - encapsulates data and methods relative to the virtual machine joints and collision events;
2. *Extruder* - stores information about the filament properties and octrees, and provides methods that determine how to draw the filament shapes;

[3] Note that both the extruder (moves in the *X*-, *Y*-, and *Z*-axis) and the print table (moves in the *B*- and *C*-axis) move relative to one another

[4] Parameters such as layer height and tool path velocity affect the real filament deposition process. These effects will be addressed by modeling the properties of the generated particles in a future version of the proposed software.

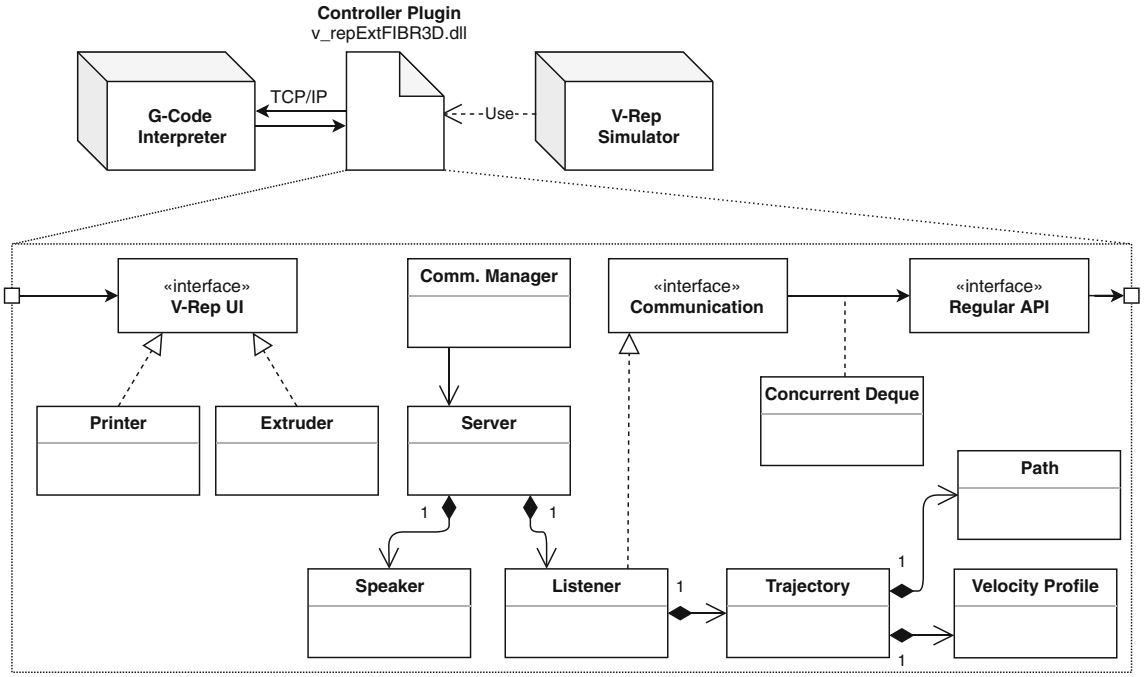


Fig. 7 UML diagram of the controller plugin

3. *Communication Manager* - handles the launch and termination of the server to communicate with the G-Code Interpreter application.

4.1 Communication

As aforementioned, the connection with the G-Code Interpreter application is based on the TCP/IP model. In the controller plugin, the communication layer was implemented based on Boost.Asio network libraries and provides bilateral asynchronous message transmission. On simulation start-up, the Communication Manager launches a Server in a new execution thread. This server is responsible for accepting incoming client connections and initiating the Sessions (sockets) where the messages are sent or received. Two types of sessions are initialized per-connection: an outbound connection to forward messages from the plugin to the interpreter, and one inbound that accepts and acknowledges incoming messages. Exchanged messages follow the JSON format and require marshalling/unmarshalling previous to being sent or upon reception. This is the task of the *Speaker* and *Listener* objects.

The *Speaker* formulates an outbound message with two fields, an integer notifying about the current G-Code line being processed and a Boolean variable coding whether the current command registers any collision. A new outbound message is sent whenever the G-Code line being processed advances or whenever a new collision is detected. The *Listener* processes inbound messages from the G-Code Interpreter. Each message codes a new command that is identified by the unique keycode field and parameterized by the remaining fields, refer to Fig. 1. After deserializing the incoming message, the command is forwarded to the Trajectory object.

4.2 Motion control

Messages deserialized are categorized into different types of movements, which are processed by the *Trajectory* object. The type of G-Code command dictates how the trajectory will be handled out of the five possible categories: (i) rapid movement, (ii) linear, (iii) arc, (iv) full circle, or a (v) dwell (halt motion during a determined time).

With the exception to the dwell trajectory, which only requires the initial joint positions and the time to halt the movement, other trajectories expect the following parameters: (a) initial joint positions, (b) target joint positions, (c) trajectory velocities, (d) maximum acceleration, (e) type of interpolation, and (f) other parameters. By other parameters, understand the center position/radius of an arc trajectory, the option to generate a clockwise or counterclockwise arc, and the plane of reference. Since the program is emulating a 5-joint machine, the initial and target joint positions expect coordinates for every actuated axis.

For clarity, the concept of a trajectory is broken down into two ideas: the *Path* and the *Velocity Profile*, also known as “feed rates”. The path represents the geometric shape of the trajectory, while the velocity profile codes the time law—how the extruder moves through the geometric path. Logically, the type of trajectory affects primarily the path,

since the velocity profile only depends on the type of path interpolation.

4.2.1 Path

Considering the 5-axis machine, we split the final path into the two constituents: the extruder path from *X*-, *Y*-, and *Z*-joints and the printer table path from the *B*- and *C*-joints (see Fig. 3c). The extruder path follows the movement command specifications from the ISO/DIN 66025 norm, while the print table joints (*B* and *C*) execute a joint interpolation from the initial to the reference value. The objective of *Path* object is to parametrize the geometric shape of the tool path.

For an extruder linear path, it means to determine the unit vector $\vec{v}_{xyz}$ that points from the initial to the final joint positions (*X*, *Y*, and *Z*). The same applies to the print table path (*B* and *C*), $\vec{v}_{bc}$. Since the C-joint is cyclic, its path needs to be adjusted to the $]-180°, 180°]$ interval without breaking the movement continuity. Note also that the C-joint always executes the shortest path motion between the current and target joint coordinate; e.g., if the motion command requires a movement from $-170°$ to $170°$, the joint executes a $-20°$ movement rather than a $+340°$ displacement.

To generate an arc or circle path, it is required that the plane where the motion is inscribed (*XY*, *ZX*, or *YZ*),[5] the direction of the arc (clockwise ↻ or counterclockwise ↺), the initial and final joint positions, and either the center position of the arc or its radius; and the output is the starting angle of the arc (α), the arc angle (γ), and the center point or the radius. If the radius (r) is provided, the first step is to determine the arc center position. Since the arc is inscribed in a plane, the problem is defined in $\mathbb{R}^2$. Let $\mathbf{p}_i$ and $\mathbf{p}_f$ denote the initial and final arc points, from which it is determined the Euclidean distance in-between, d, as well as the middle point $\mathbf{p}_m$. The arc center point $\mathbf{p}_{\text{cen}}$ is determined as follows,

$$\mathbf{p}_{\text{cen}} = \begin{cases} \begin{bmatrix} \mathbf{p}_{m,X} + \left(\frac{\mathbf{p}_{i,Y} - \mathbf{p}_{f,Y}}{d}\right)\sqrt{r^2 - \left(\frac{d}{2}\right)^2} \\ \mathbf{p}_{m,Y} + \left(\frac{\mathbf{p}_{i,X} - \mathbf{p}_{f,X}}{d}\right)\sqrt{r^2 - \left(\frac{d}{2}\right)^2} \end{bmatrix}, & \text{if } (r < 0 \wedge \circlearrowright) \vee (r > 0 \wedge \circlearrowleft) \\ \begin{bmatrix} \mathbf{p}_{m,X} - \left(\frac{\mathbf{p}_{i,Y} - \mathbf{p}_{f,Y}}{d}\right)\sqrt{r^2 - \left(\frac{d}{2}\right)^2} \\ \mathbf{p}_{m,Y} - \left(\frac{\mathbf{p}_{i,X} - \mathbf{p}_{f,X}}{d}\right)\sqrt{r^2 - \left(\frac{d}{2}\right)^2} \end{bmatrix}, & \text{if } (r > 0 \wedge \circlearrowright) \vee (r < 0 \wedge \circlearrowleft) \end{cases}. \tag{1}$$

Once the arc center point is known, one might determine the arc start (α) and final angle (β),

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \text{atan2}(\mathbf{p}_{i,Y} - \mathbf{p}_{\text{cen,Y}}, \mathbf{p}_{i,X} - \mathbf{p}_{\text{cen,X}}) \\ \text{atan2}(\mathbf{p}_{f,Y} - \mathbf{p}_{\text{cen,Y}}, \mathbf{p}_{f,X} - \mathbf{p}_{\text{cen,X}}) \end{bmatrix} \tag{2}$$

The arc angle γ is the angle between α and β. Finally, the extruder and the table path lengths are determined, the Euclidean distance of the *X*, *Y*, and *Z* movement in millimeters and the spherical distance of the *B* and *C* movement in degrees respectively.

4.2.2 Velocity profile

To determine the form of the velocity profile for a given command, the following parameters are required: (i) the start, travel, and end-velocity, (ii) the maximum allowed acceleration, and (iii) the path length (S)—more about this in Section 4.2.3. Two types of velocity profiles are implemented according to the supported G-Code commands (Table 1): the linear interpolation profile and the trapezoidal profile.

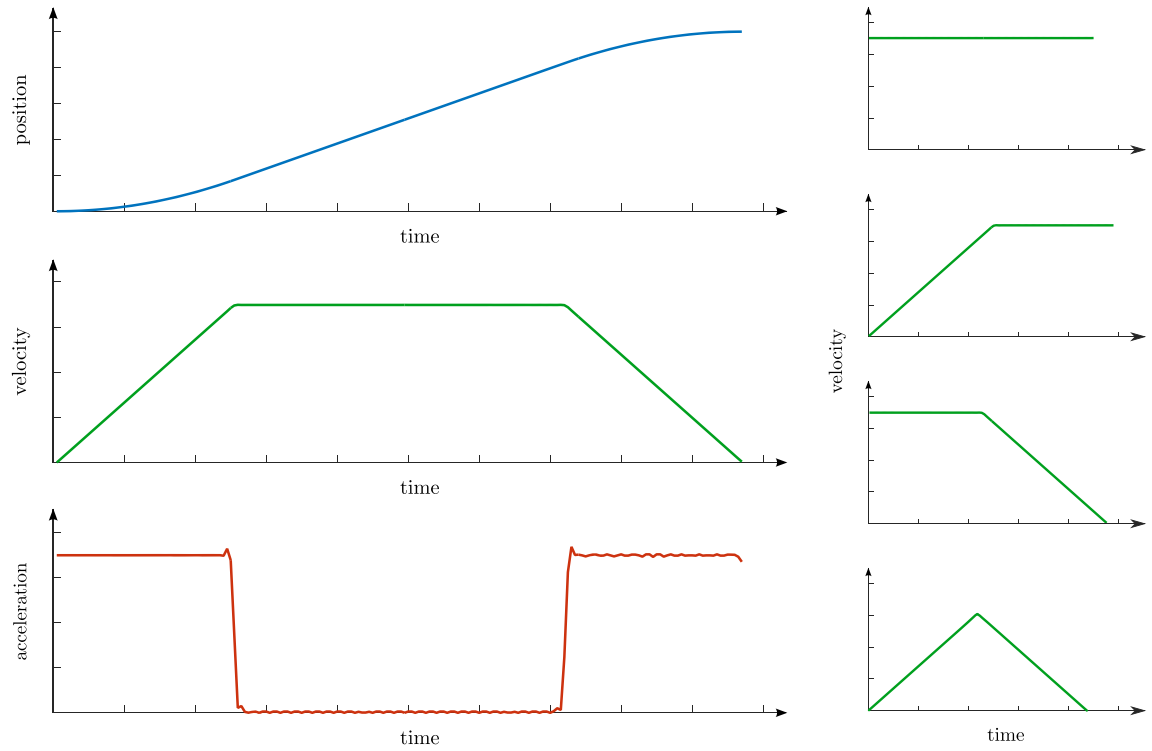
If the commanded movement is specified according to the linear interpolation profile, the velocity profile is described by the *first* and *second equations of motion* with a single acceleration slope transitioning from the initial to the final velocity. Let the path be $\{s \in \mathbb{R}, s \in [0, S]\}$, the initial and final velocities (v_i and v_f) and accelerations (a_i and a_f) be known. The trajectory total time (T) is,

$$T = \frac{2S}{v_i + v_f}, \tag{3}$$

and the acceleration that is constant in this case ($a_i = a_f$)

$$a_i = \frac{v_f - v_i}{T}. \tag{4}$$

[5] For conciseness, the position notation ($\mathbf{p}_\bullet$) is presented for the XY-Plane case.



(a) Block with different start and end velocities. Complete trapezium velocity profile shape.

(b) Alternative velocity profile shapes.

Fig. 8 **a**, **b** The five different possible trapezoidal velocity profile shapes. For simplification, the initial ramp is depicted as a positive acceleration and the final ramp as a negative acceleration (deceleration) although the opposite cases are also possible

According to the actuators' time-step (t_s), the velocity profile determines the progression along the path ($s(t)$) from $s_i = 0$ to $s_f = S$.

By default, the movement is configured to the trapezoidal velocity profile. Depending on the motion command properties (whether **G08/09** are set), the feed rate may accelerate/decelerate at the beginning and/or at the end of the block (see Fig. 8).

If the motion command involves a change in the feed rate at the start and/or end of a block, the module and direction of the acceleration vector(s) are first determined. From the *first* and *second equations of motion*, one calculates the time required to change the velocity (start to travel and/or travel to end) and consecutively the length of path covered during the acceleration phase(s).[6] If the path length traveled during the acceleration phase is inferior to the total path length, the remaining path is covered at constant velocity (travel velocity).

On the other hand, if the traveled distance during the acceleration phase exceeds the total path length, the time of the acceleration phase is recalculated to guarantee that the end velocity, as well as the path length, is met, at the cost of not reaching intermediate velocities. This velocity profile is characterized by its "triangular shape" (Fig. 8b) as there is a time spent moving at an initial acceleration and the remaining time at the final acceleration. The path point that marks the transition from the initial to the final acceleration profile is referred to as s_m. From the second equation of motion, we gather that,

$$(s_i - s_m) + v_i t_i + \frac{1}{2} a_i {t_i}^2 = 0. \tag{5}$$

The velocity profile is easily determined after calculating t_i that stands for the time of motion during the first acceleration phase, which is solved as a quadratic polynomial equation. As boundary condition, we know that $s_i = 0$ and $s_f = S$, and s_m can be determined using the third equation of motion,

$$s_m = \frac{{v_f}^2 - {v_i}^2 - 2a_f S}{2(a_i - a_f)}. \tag{6}$$

[6] To avoid repetition, henceforth consider the "acceleration phase" as the start, the end and/or both acceleration phases.

It logically follows that the total trajectory time (T) is

$$T = t_i + t_f = t_i + \frac{v_f - (v_i + a_i t_i)}{a_f} \tag{7}$$

Since the quadratic polynomial problem might return up to two solutions, it is selected the solution that permits the smallest and possible total time.

Similarly to the linear interpolation profile case, path progression for the trapezoidal profile is also discretized according to the time step.

4.2.3 Trajectory

The trajectory object is finally responsible for combining the geometric path with the velocity profile timing law. The joint positions during the path progression are determined using the geometric path parameters.

As referred in Section 4.2.1, the extruder and print table paths are handled separately. The velocity profile, however, codes the combined movement of the extruder and the print table. To achieve this behavior, we determine a combined path length that results from the extruder path distance and the print table path distance. The combined path length is passed as a parameter to the velocity profile calculation.

To guarantee synchronous motion of the extruder and the print table, their movements are adjusted to the corresponding ratio of the combined length. A set of the 5-axis joint positions is determined for each time step, Algorithm 1. Once computed, the motion sums up to a vector of joint positions, which is pushed into a concurrent queue to be processed by the V-Rep execution thread.

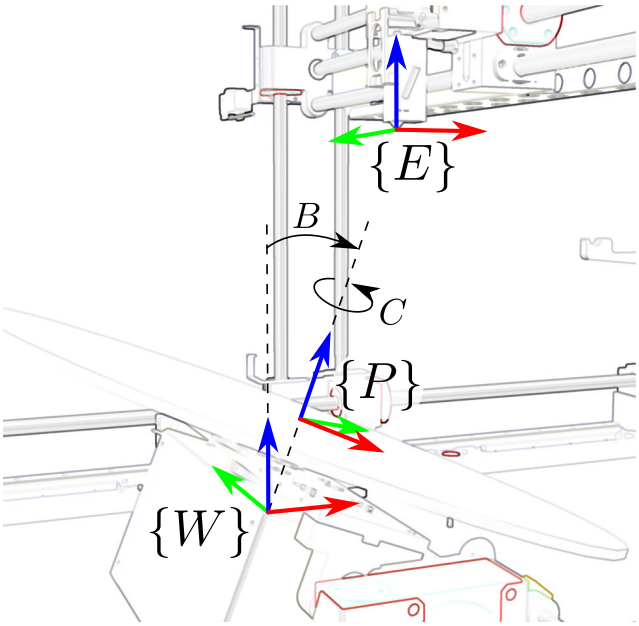


Fig. 9 Reference frames of the $\{W\}$ world, $\{E\}$ extruder, and $\{P\}$ print table

Algorithm 1 Trajectory generation algorithm.

```
/* Inputs depend on the type of Path
   and VelocityProfile requested   */
```
Input:
Path: $\mathbf{p}_i, \mathbf{p}_f, \vec{v}_{xyz}, \vec{v}_{bc}, \alpha, \gamma, \mathbf{p}_{cen}$
Velocity Profile: $s(k), \forall k \in \{1, \cdots, N\}$
```
/* Matrix with 5-axis positions for
   each path iteration             */
/* Let l= 1,2,3 represent X, Y, Z  */
/* Let l = 4,5 represent B, C      */
```
Output: $\mathbf{J} = (j_{k,l}) \in \mathbb{R}^{N\times 5}$

1 $N := \lceil T/t_s \rceil$ `// number of iterations`
2 $S_{1:xyz} := S_{xyz}/S$ `// ratio of the extruder path`
3 $S_{1:bc} := S_{bc}/S$ `// ratio of the print table path`
4 **for** *k := 1, N-1, step 1* **do**
5 $\quad a_{k,4} = s(k) \times \vec{v}_{bc,B} \times S_{1:bc} + \mathbf{p}_{i,B}$
6 $\quad a_{k,5} = s(k) \times \vec{v}_{bc,C} \times S_{1:bc} + \mathbf{p}_{i,C}$
$\quad$ `/* Indices 1,2,3 represent X,Y,Z for the default 'XY-Plane', and may represent other coordinates for alternative Planes (XZ or YZ)  */`
7 $\quad$ **if** *Linear Trajectory* **then**
8 $\quad\quad a_{k,1} = s(k) \times \vec{v}_{xyz,1} \times S_{1:xyz} + \mathbf{p}_{i,1}$
9 $\quad\quad a_{k,2} = s(k) \times \vec{v}_{xyz,2} \times S_{1:xyz} + \mathbf{p}_{i,2}$
10 $\quad\quad a_{k,3} = s(k) \times \vec{v}_{xyz,3} \times S_{1:xyz} + \mathbf{p}_{i,3}$
11 $\quad$ **else if** *Arc / Circle Trajectory* **then**
12 $\quad\quad \tau = \alpha + \frac{s(k)}{S}\gamma$
13 $\quad\quad a_{k,1} = r \times \cos\tau \times S_{1:xyz} + \mathbf{p}_{i,1}$
14 $\quad\quad a_{k,2} = r \times \sin\tau \times S_{1:xyz} + \mathbf{p}_{i,2}$
15 $\quad\quad a_{k,3} = \mathbf{p}_{i,3}$
16 $\quad$ **else**
$\quad\quad$ `/* Dwell */`
17 $\quad\quad a_{k,[1,\cdots,5]} = \mathbf{p}_i$
18 $a_{N,[1,\cdots,5]} = \mathbf{p}_f$

4.3 Extrusion mechanic

If the received motion command includes the variable that codes the filament advance (A—refer to Table 1), the virtual extrusion process is initiated. A new *Drawing Object* is created based on the type, color, size of the filament, and base entity with a reference frame. As the extrusion

Table 2 Standard Denavit-Hartenberg parameters of the print table mechanism. Z_0 amounts to the distance from $\{W\}$ to $\{P\}$

i	a_i	α_i	d_i	θ_i
$W \rightarrow 0$	0	−90	0	0
$0 \rightarrow 1$	0	90	0	B
$1 \rightarrow 2$	0	0	Z_0	C

proceeds, new filament shapes (Fig. 4) are concatenated to the drawing object item list, which is processed by the simulator graphics renderer at each cycle. If the base entity, or in this case, the print table moves, the drawn shapes will also move accordingly. However, the position of the new shapes only depends on the position of the extruder relative to the world. The implications of this statement are explained next.

To simulate the deposition of a filament, two points are considered in each simulation step: the last position[7] of the extruder relative to the world—transformed of the differential print table movement in respect to the world ${}^{W}\mathbf{p}_{E,t-1}$—and the current position of the extruder relative to the world ${}^{W}\mathbf{p}_{E,t}$ (see Fig. 9).

In each cycle, the extrusion handler will attempt to draw shapes between both points. The number of items to be added depends on the ratio between the Euclidean distance and the extrusion resolution.

At the simulation start-up, the initial extruder transformation relative to the world reference frame is saved as ${}^{W}\mathbf{T}_{E,0}$. Thus, the current position of the extruder relative to the world at each simulation cycle can be directly deduced from the current X, Y, and Z joint positions (${}^{E,0}\mathbf{p}_{E,t} = [X_t, Y_t, Z_t]$),

$$ {}^{W}\mathbf{p}_{E,t} = {}^{W}\mathbf{T}_{E,0}\, {}^{E,0}\mathbf{p}_{E,t} \tag{8} $$

Determining ${}^{W}\mathbf{p}_{E,t-1}$ requires additional steps, as it depends on the differential transformation of the print table from the previous iteration. The transformation from the world reference frame to the print table can be determined using the Denavit-Hartenberg notation (Table 2), and the corresponding "relative transformation matrix",

$$ {}^{i-1}\mathbf{T}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{9} $$

[7] Let ${}^{A}\mathbf{p}_{B,\bullet}$ represent the position of B relative to A at the time instant $\bullet$. Similarly, let ${}^{A}\mathbf{T}_{B,\bullet}$ represent the transformation of B relative to A in the time instant $\bullet$.

The first DH parameter relative matrix (${}^{W}\mathbf{T}_0$) is required to account for the orientation of the B-axis relative to the world reference frame z-axis ($\{W\}_z$). The print table and the extruder transformations relative to the world reference frame can now be calculated,

$$ {}^{W}\mathbf{T}_P = {}^{W}\mathbf{T}_1\, {}^{1}\mathbf{T}_2 \tag{10} $$

$$ {}^{W}\mathbf{T}_E = \begin{bmatrix} 1 & 0 & 0 & X_t \\ 0 & 1 & 0 & Y_t \\ 0 & 0 & 1 & Z_t + Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{11} $$

The transformation of the extruder frame relative to the print table is calculated as,

$$ {}^{P}\mathbf{T}_E = ({}^{W}\mathbf{T}_P)^{-1}\, {}^{W}\mathbf{T}_E, \tag{12} $$

and consequently the last position of the extruder relative to the last position of the print table,

$$ {}^{P,t-1}\mathbf{p}_{E,t-1} = ({}^{W}\mathbf{T}_{P,t-1})^{-1}\, {}^{W}\mathbf{p}_{E,t-1}. \tag{13} $$

Finally, ${}^{W}\mathbf{p}_{E,t-1}$ can be calculated as,

$$ {}^{W}\mathbf{p}_{E,t-1} = {}^{W}\mathbf{T}_{P,t-1}\, {}^{P,t-1}\mathbf{p}_{E,t-1}. \tag{14} $$

If the Euclidean distance between ${}^{W}\mathbf{p}_{E,t-1}$ and ${}^{W}\mathbf{p}_{E,t}$ is smaller than the resolution step, no item is drawn, and the value of the last extruder position not updated. Else, shapes are drawn between the points at equally spaced intervals.

5 Results and discussion

The FIBR3DEmul was tested with 3- and 5-axis G-Code files (Figs. 10 and 11). By adjusting the virtual filament dimension (3 to 0.4 mm), it is possible to produce with pieces with higher resolution. While it is possible to create shapes smaller than 0.4 mm, the impact in software performance from the higher shape density would make the simulation impractical.

To validate the FIBR3DEmul simulation tool in terms of position and time precision, a G-Code script was created and ran simultaneously in the virtual and in the real "C3DPrinter". The test G-Code includes commands to move the extruder, the print table, and both parts synchronously at with different motion commands and feed rates.

A real-time logger, running in both virtual and real controllers, collected the position and velocity of each of the 5 axes at each 10 ms. The comparison of the virtual and real printer movements can be visualized at https://www.youtube.com/watch?v=VFvln1082aQ. These results are presented in Figs. 12 and 13. With respect to the extruder axes (Fig. 12), the virtual X- and Y-axis positions as a

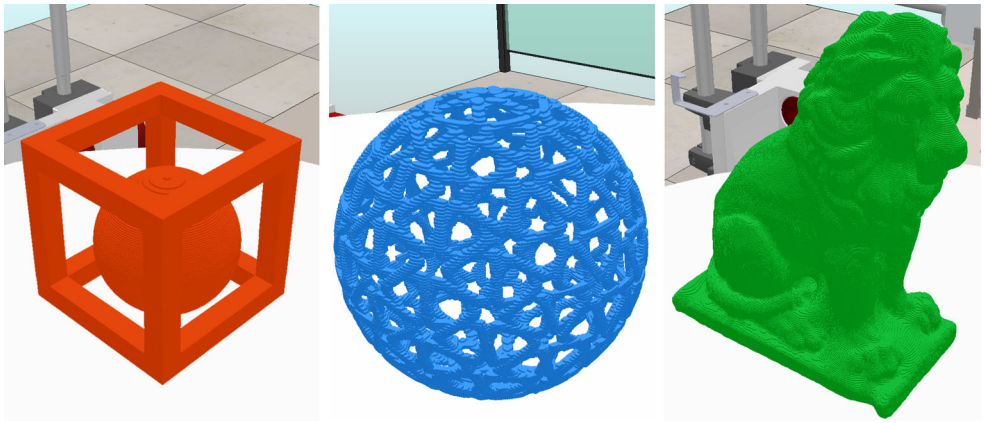


Fig. 10 Examples of detailed printed 3D models using FIBR3DEmul. The cube printed with a filament width of 0.8 mm, the other two with filament width of 0.4 mm

function of time are identical to the values found for the actual machine. The *Z*-axis curves depict a difference of 100 mm throughout the trajectory. This discrepancy is due to the fact that the virtual printer is unable to account for the offsets that are locally set in the real machine controller. The velocity curves of the virtual and real extruder axes mostly overlap. Upon close inspection, it is noticeable in the final part of the trajectory (around 150 s of the *Z*-axis velocity plot) that the virtual printer has a delay of about 50 ms when compared with the real printer. This delay might be attributed to the mechanism of receiving and stacking new joint trajectories as discrete vectors of joint positions (see Section 4.2).

The *B*-axis position plots for the virtual and real printer (Fig. 13) are mostly overlapping. Around the 70-s mark, there is a mismatch between the values of the *C*-axis. This is an expected behavior and is attributed to how the simulator and the real controller handle cyclic rotational joint limits. In the simulator, the value of a rotational joint is comprehended between [−180, 180] degrees, whereas in the real controller, this interval is [0, 360]. The time delay is also noticeable in the velocity plots of the print table axes.

It should be noted that as of this moment, the proposed solution does not emulate filament deposition dynamics. As explained in Section 3.2, the FIBR3DEmul generates a string of particles to emulate the filament considering only the tool path. Deposition parameters such as layer height and nozzle velocity, which affect the profile of the filament, are currently not accounted for. The proposed FIBR3DEmul does allow for the customization of the

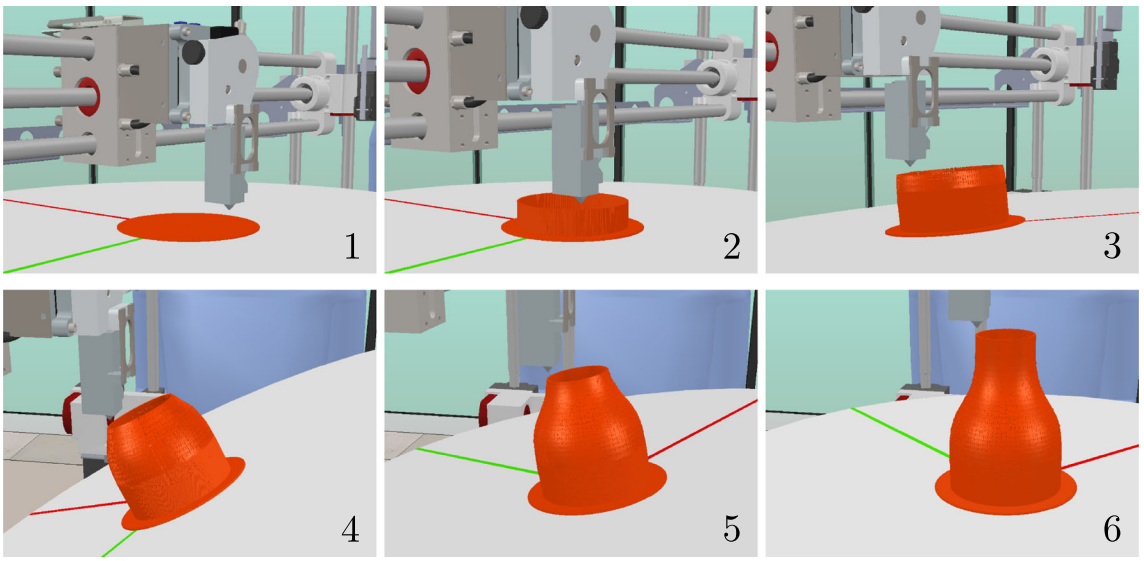


Fig. 11 Example of a 3D-printed model with 5 simultaneously actuated axes

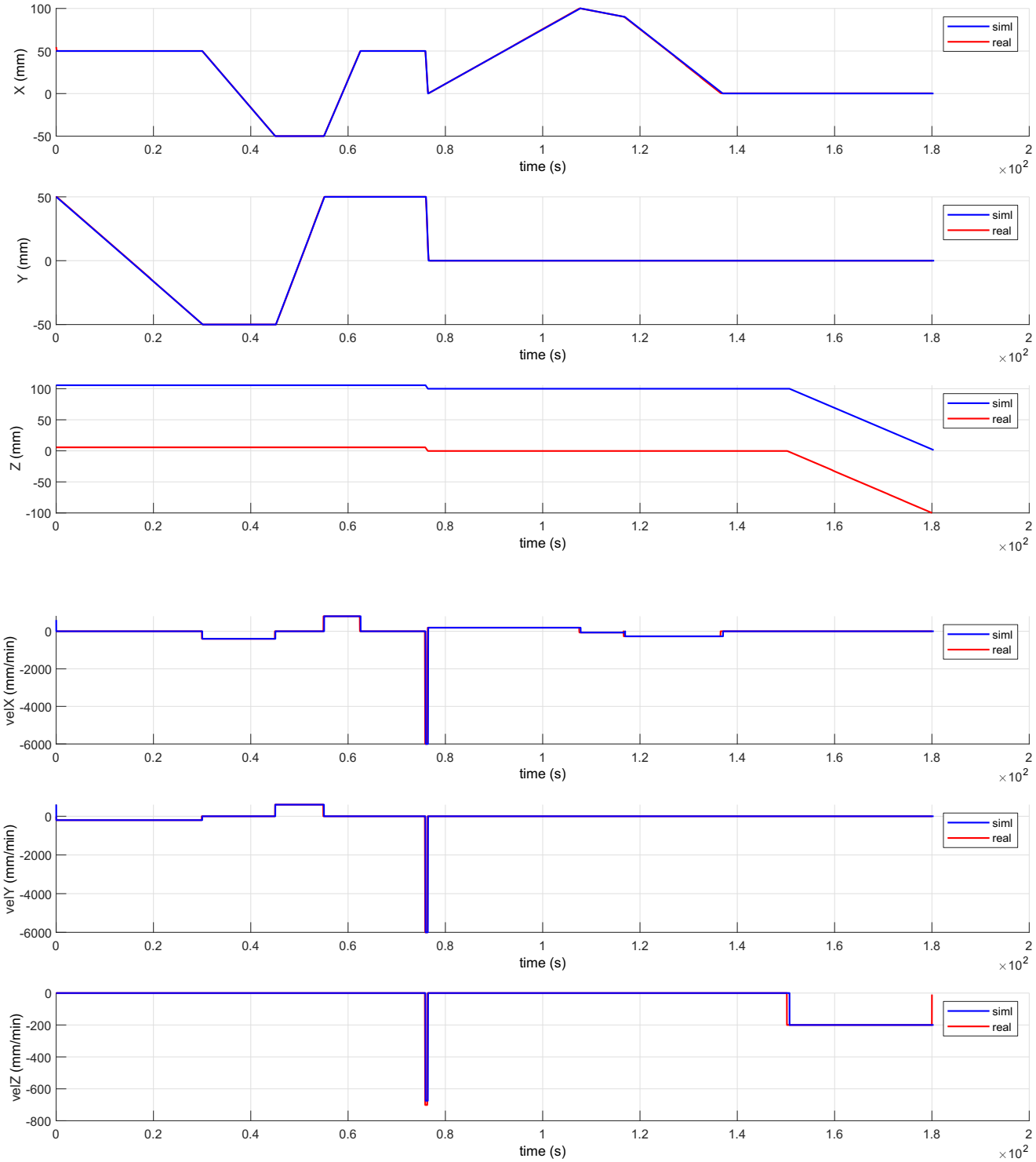


Fig. 12 Position and velocity plots of the extruder axis of the virtual and the real "C3DPrinter"

particles in run-time, which means that one can model the generated filament in terms of position, size, sparsity, shape, and color. In future work, the extrusion mechanism (refer to Section 4.3) will be adjusted to model the characteristics of the generated filament—especially size and position—based on the layer height and nozzle velocity. While the

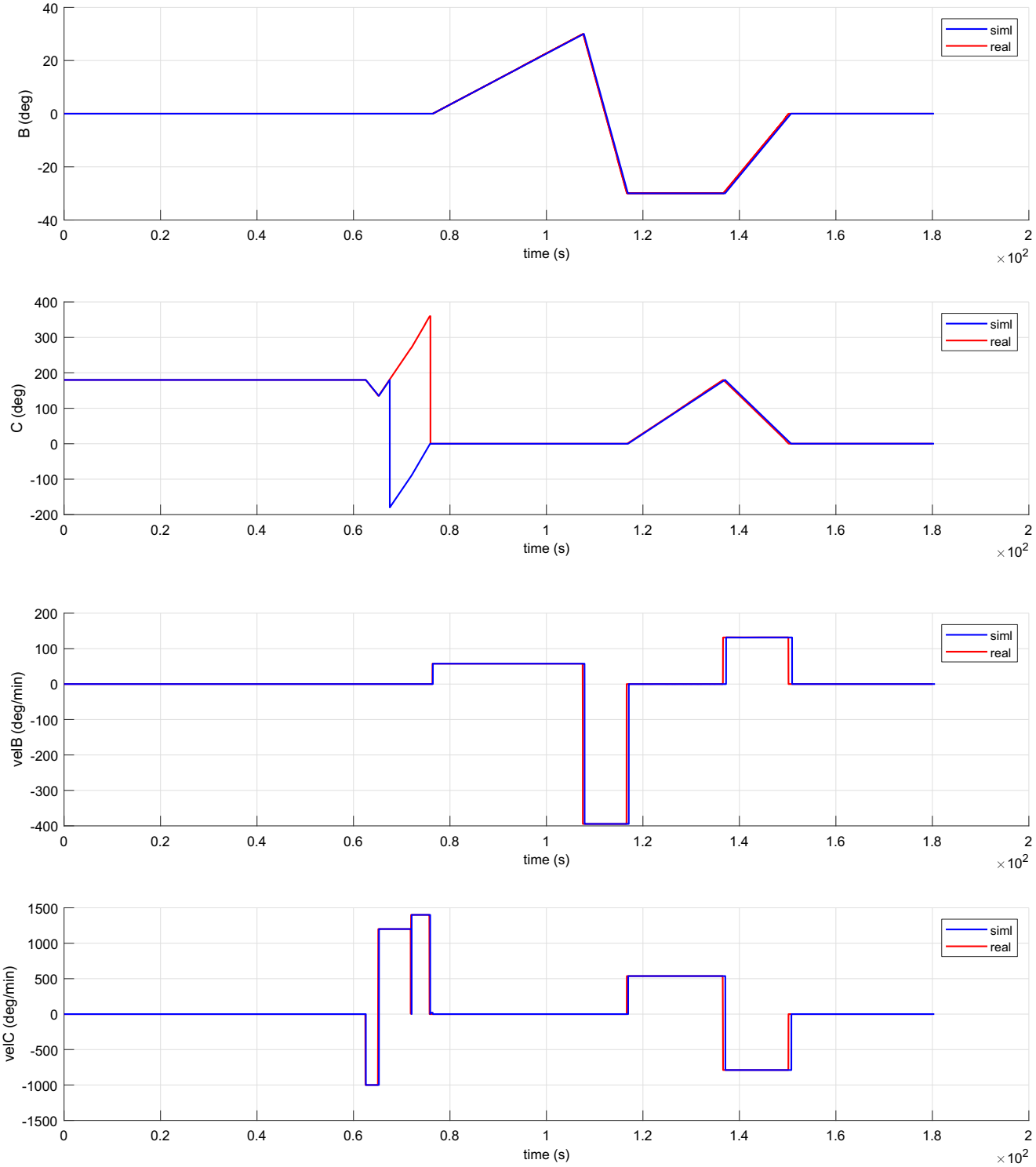


Fig. 13 Position and velocity plots of the print table axis of the virtual and the real "C3DPrinter"

relation between nozzle velocity and the size of the thinning of the filament might be straightforward to implement programmatically, the impact of the layer height on the flattening of the filament might involve adjusting the size, shape, and position of the particles that form the filament.

6 Conclusion

This paper presents the FIBR3DEmul software, a simulation tool capable of interpreting ISO/DIN 66025 G-Code and translating the commands to a robotics simulator to virtually

emulate the real machine operation. At the moment, the FIBR3DEmul software permits testing FDM processes for the standard 3-axis architecture, as well as for a new standard with 5-axes simultaneously actuated (Cartesian-type actuator controlling the extruder and a 2-axis rotating table).

It is explained how the simulation solution was designed and implemented in two separate applications, the G-Code Interpreter and the V-Rep simulator with a custom-developed controller plugin. It is described how a custom printer is introduced and controlled in the virtual scenario, as well as the simulation process.

It is expected that the presented solution will have more impact next to developers of new FDM machine architectures or typologies. The presented solution allows the developers to control the G-Code execution flow, check for printer-printer or printer-workpiece collisions, and, most importantly, permits testing G-Code scripts from any workstation without the risk of damaging equipment and with very comparable results to the real machinery.

The simulation software permits adjusting the filament color, size, and sparsity in runtime. This flexibility can be explored in the future to tackle some of the limitations of the proposed solution, i.e., to emulate deposition parameters such as temperature, velocity, or layer height, which vary according to the material being deposited. Adjusting these parameters to test the machine output becomes particularly relevant when we consider operating with less common and more expensive materials as is the case of PEEK (polyether ether ketone), which is 7 to 10 times more expensive than the more common ABS or PLA.

Funding information This work has been financed by the FIBR3D Project (POCI-01-0145-FEDER-016414).

References

1. Kallevik G (2015) 5-axis 3D Printer-Designing a 5-axis 3D printer. PhD thesis, University of Oslo
2. Lauwers B, Dejonghe P, Kruth JP (2003) Optimal and collision free tool posture in five-axis machining through the tight integration of tool path generation and machine simulation. CAD Computer Aided Design 35(5):421–432
3. Lee WC, Wei CC, Chung SC (2014) Development of a hybrid rapid prototyping system using low-cost fused deposition modeling and five-axis machining. J Mater Process Technol 214(11):2366–2374
4. Song X, Pan Y, Chen Y (2014) Development of a low-cost parallel kinematic machine for multidirectional additive manufacturing. J Manuf Sci Eng 137(2):021005
5. Wu C, Dai C, Fang G, Liu YJ, Wang CCL (2017) RoboFDM a robotic system for support-free fabrication using FDM. Proceedings - IEEE international conference on robotics and automation: 1175–1180
6. Wulle F, Coupek D, Schäffner F, Verl A, Oberhofer F, Maier T (2017) Workpiece and machine design in additive manufacturing for multi-axis fused deposition modeling. Procedia CIRP 60:229–234
7. Zhang GQ, Li X, Boca R, Newkirk J, Zhang B, Fuhlbrigge TA, Feng HK, Hunt NJ (2014) Use of industrial robots in additive manufacturing - a survey and feasibility study. Proceedings of 41st international symposium on robotics: 512–517
8. Zhang GQ, Spaak A, Martinez C, Lasko DT, Zhang B, Fuhlbrigge TA (2016) Robotic additive manufacturing process simulation-towards design and analysis with building parameter in consideration. IEEE Int Conf Auto Sci Engi 2016-Novem:609–613