**Solid Freeform Fabrication 2018: Proceedings of the 29th Annual International Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference Reviewed Paper**

# CONTROL SYSTEM FRAMEWORK FOR USING G-CODE-BASED 3D PRINTING PATHS ON A MULTI-DEGREE OF FREEDOM ROBOTIC ARM

Andrzej Nycz*, Mark W. Noakes*, Christopher J. Masuo*, and Lonnie J. Love*

*Oak Ridge National Laboratory, Manufacturing Demonstration Facility, 2370 Cherahala Blvd, Knoxville, TN 37932

## Abstract

This paper describes a control system framework using G-Code-based 3D printing paths on a serial link robot manipulator with multiple degrees of freedom. Usually, G-Code is created by a software application, commonly referred to as a *slicer,* meant for gantry systems. However, G-Code does not address the kinematic complexity nor take advantage of the flexibility available in serial link robot manipulators. This paper provides an overview of the additive manufacturing process and G-Code, types of additive manufacturing deposition movements, common terminology used, the roles of parsers and translators, step-by-step instructions on how to implement this control system, and results and findings from this research. The presented framework can be used for a number of additive manufacturing methods, hybrid solutions, or applications not directly related to additive manufacturing. The implementation was successfully tested on a manipulator with seven degrees of freedom that successfully performed hundreds of hours of large-scale wire arc metal deposition.

## Background

Additive manufacturing (AM) or three-dimensional (3D) printing is a production process relying on 100% digital and automatic path formulation [1]. The process usually starts with a 3D part model created in a computer aided design (CAD) program. Then, the model is saved in a standard tessellation format or Stereo Lithography (STL) file and processed through a 2.5D path generator. The path generation is commonly called *slicing,* and the software used to generate these paths is typically called a *slicer*. Slicing divides a part into vertical layers and beads, which makes going from layer to layer the only vertical movement. A bead is the extruded and then deposited material that forms lines and layers of a part as it is printed. There is usually a start and stop process between each bead as well as between each layer. The output of this process is a G-Code file describing the motion of the printing head and the necessary process settings.

G-Code [2] is a computer numerical control (CNC) machine motion programming language. Its roots go back to early 1950s machine tool control and was meant for simple motion control. Therefore, it does not take advantage of advanced features available in modern programming languages. Among many of its drawbacks, it lacks looping and variable paradigms. In addition, most of the physical controllers are not designed to execute closed loop control. For most 3D printing processes, the G-Code file is the final path description, and there is no process feedback involved.

In AM, the main generator of G-Code is the s*licer*. The majority of available slicers [3] produce a geometry in Cartesian space that is compatible with gantry systems. However, the drawback of using a gantry system in AM is that it limits movements. G-Code generated by slicers gives no consideration for the complexity of kinematics required to control serial link manipulators. This makes G-Code-based control of AM manipulators naturally incompatible.

A solution to a gantry system's limited movements is a robotic arm carrying the AM printing head because it offers increased complexity of motion. However, if the process requires feedback or advanced controls, the G-Code path and process description are inadequate, and an additional translator or post-processing agent is necessary. Additional complexity may occur if the AM process is not omnidirectional. This happens when the printing head cannot be considered a geometric point but is a two or more-dimensional object requiring a specific orientation along the printing path. The majority of fused deposition modeling processes are omnidirectional (Fig. 1), whereas most laser-based and some arc-based wire fed systems are directional and require advanced motion path handling or reduction in the path complexity (Fig. 2).

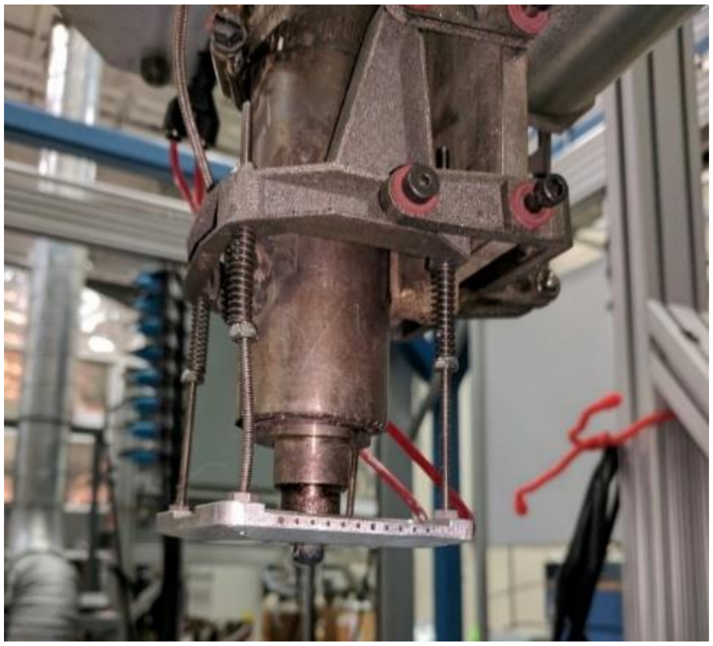


Fig 1. Omnidirectional BAAM-type (Big Area Additive Manufacturing) polymer printing head.

## Introduction

This paper describes a framework for translating and executing a G-Code generated path using a seven degree-of-freedom (DOF) Mitsubishi manipulator and advanced tungsten inert gas (TIG)-based wire fed metal arc deposition process as an example.

Control of print head orientation is required for proper execution of the build. This paper presents a complete sequence of implementation steps from G-Code reading through translation and execution. The human machine interface (HMI) and a data logger used for post-process analysis are also presented. The principal structure of the described system can be used for any AM process, hybrid additive-subtractive methods, and non-AM applications.

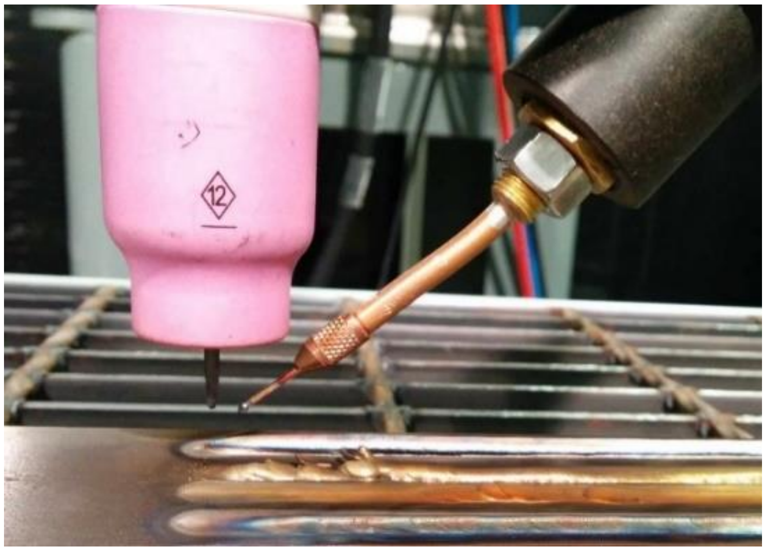


Fig. 2 Directional GTAW (TIG) deposition head with right side forward feeding wire.

There have been a few other implementations of G-Code-based execution for serial link robot manipulators; however, none of them are similar to the work presented in this paper. Dritsas and Goulthorpe used computer-aided manufacturing (CAM) software to use a Kuka six DOF robot manipulator to execute milling operations [4].They found the use of G-Code to be quite tedious though it initially appeared to be a simple translation. Human intervention was necessary via an interactive simulation program to modify geometry and machine paths to compensate for the information that G-Code generation lacked. End-effecter orientation and manipulator-specific problems, such as wrapping the arm cables around the arm, were just two of many challenges.

Surdilovic *et al.* [5] similarly used CAM software to generate G-Code to execute milling operations using a Comau robot manipulator. Like Dritsas and Goulthorpe, they used a simulation program to sort out the problems found in the G-Code that did not adequately describe motion necessary for the manipulator. The simulation software was used to fix collisions, bad manipulability, and joint singularities. The G-Code generation also did not adequately address motion profiles and rapid changes in end-effector orientation.

Slavkovic *et al.* used a five DOF robot manipulator to execute modified G-Code for milling operations [6]. The authors also found that post-processing of the G-Code was necessary to support operation on a robot manipulator. The authors chose to execute an off-line compensation by correcting the G-Code by predicting static tool tip displacements calculated based on a robot compliance model.
In summary, it appears that all prior work in the use of G-Code for tool path execution on serial link robot manipulators requires some version of post-processing of the generated G-Code to address the unique needs of each manipulator and the process that is being executed. The work presented in this paper discusses a more general scheme of G-Code post-processing to support both the manipulator and the tooling (AM or machining) processes task that make up operation.

## General System Design

A standard slicer produces toolpath commands controlling the motion of the print head and/or the base table. The base table is a platform onto which the material is deposited and can move in one or more directions in some systems. To make the algorithm more generic for presentation here, the motion of the table is not considered. However, it can be extended without changing the basic structure. Deposition in layer by layer mode is considered here.

## Types of Deposition Moves

*Deposition Moves*

A deposition move is a move of the printing head from the deposition start point to the end point while continuously printing (depositing material). Traditionally, it is a horizontal path during which the printing head creates a bead (Fig. 3).

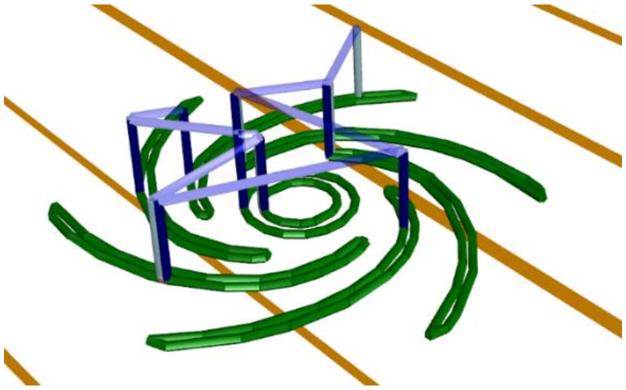


Fig 3. Deposition moves/beads (green) and travel moves (purple) in a layer of sliced impeller part.

Travel Moves

A travel move is a type of motion that the printing head takes to move from the end of the just executed bead to the beginning of the next bead without printing. Consecutive beads typically don’t have adjacent starts and stops. Therefore, they require this additional, nonprinting motion. It is usually comprised of a sequence of up, transverse, and down motions (Fig. 3).

Unwind Moves

An unwinding move is necessary when a non-omnidirectional deposition head or a robot manipulator has external cables routed down the length of the arm to prevent hardware damage.

Maintenance Moves

A maintenance move can be any move required by the printing process such as head cleaning, conditioning, and filament or wire cutting etc.

Utility Moves

A utility move can be necessary for complex processes and can include scanning, preheating, surface conditioning, or machining.

## System Conditions, States, and Assumptions

The system shown in Fig. 4, relies on the capability of the programming environment to run many threads at the same time and to provide proper inter-process communication tools such as queues, notifiers, and shared memory [7] . The implementation code for this project is written in LabView and relies heavily on action engines [8], a native LabView feature. Action engines combine the idea of shared memory, semaphores, object programming, and subroutines into one elegant, powerful, and easy to use concept. The presented algorithm can be implemented using different approaches; however, action engines provide a powerful tool for inter-process communication and synchronization.

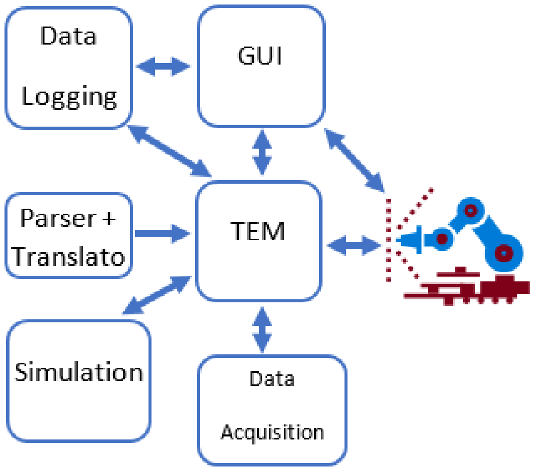
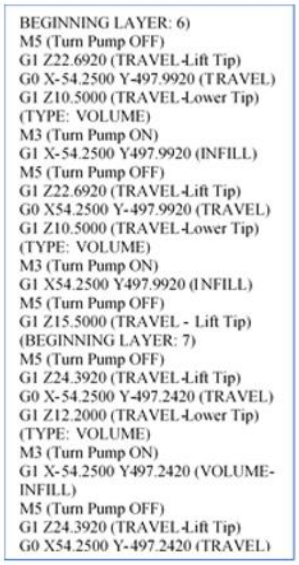


Fig. 4 System diagram.

The usual slicer output is a text file (shown in Fig. 5) with a combination of deposition and travel moves organized in layers. Each line represents a single atomic command (Fig. 3 and Fig 5). This text file is the input to the toolpath translator. Depending on the slicer and G-Code generator, the layer's end beads are defined by key words (*G1*, *G0*, etc.) either as part of G-Code language or as custom defined labels. For example, this could be extrusion on/off commands. Unless slicer settings are specifically defined for a process from the start, its output might contain redundant information that is often included just to improve human readability.



```
BEGINNING LAYER: 6)
M5 (Turn Pump OFF)
G1 Z22.6920 (TRAVEL-Lift Tip)
G0 X-54.2500 Y-497.9920 (TRAVEL)
G1 Z10.5000 (TRAVEL-Lower Tip)
(TYPE: VOLUME)
M3 (Turn Pump ON)
G1 X-54.2500 Y497.9920 (INFILL)
M5 (Turn Pump OFF)
G1 Z22.6920 (TRAVEL-Lift Tip)
G0 X54.2500 Y-497.9920 (TRAVEL)
G1 Z10.5000 (TRAVEL-Lower Tip)
(TYPE: VOLUME)
M3 (Turn Pump ON)
G1 X54.2500 Y497.9920 (INFILL)
M5 (Turn Pump OFF)
G1 Z15.5000 (TRAVEL - Lift Tip)
(BEGINNING LAYER: 7)
M5 (Turn Pump OFF)
G1 Z24.3920 (TRAVEL-Lift Tip)
G0 X-54.2500 Y-497.2420 (TRAVEL)
G1 Z12.2000 (TRAVEL-Lower Tip)
(TYPE: VOLUME)
M3 (Turn Pump ON)
G1 X-54.2500 Y497.2420 (VOLUME-
INFILL)
M5 (Turn Pump OFF)
G1 Z24.3920 (TRAVEL-Lift Tip)
G0 X54.2500 Y-497.2420 (TRAVEL)
```

Fig. 5 G-Code example, which is slicer output.

A good practice before moving the arm to an automatically generated path is to have it parked at a known position and configuration such that the next destination is achievable. This means there are no kinematic singularities or obstacles. For the sake of generality, this location is called *Work Home.*

Unless specified otherwise, the system might require a geometric or process calibration update. This can be either an automatic or manual process and should be done prior to the start of the deposition process.

In the case of large-scale deposition, without going into the specifics of the process, the slicer will not be able to easily produce all the steps and parameters necessary to execute a single node. This is caused by the process's dynamic nature, arm structure, or closed loop requirement. The alternative is allowing a large number of parts to fail.

## Terminology and Definitions

Task (Job)

A task or job is the complete set of actions that must occur to build an object/part.

Node

A node is a motion destination point described by X, Y, and Z coordinates. The process usually changes at node points in a form of new tool orientation, speed, or deposition. A current execution node is a node to which the execution takes the end effector. A previous node is the last node that the robotic arm reached and executed.

Node Property

The purpose of the node property is to describe specific actions taking place between the previous and current nodes.

Process

A process is a description of the deposition method. It may dictate a specific arm orientation, hardware use, or software interaction.

System Calibration

Depending on the type of task and hardware used, the system might have different end effector or process calibrations. In this work, it was the location and orientation of the print table.

Layer

A layer is a physical layer of print. On the code level, it is a set of nodes with all corresponding properties necessary to build a physical layer.

Timers

Timers are important from a maintenance, testing, and development point of view. It is important to know the amount of time it takes for layers, beads, or non-deposition movements to be completed. It also directly affects the economy of the process [9]. Hence, three basic timers are proposed here: task timer, layer timer, and deposition pause timer. It is also important to note that support code must be concise so that it does not impact execution time.

Step

A step is an atomic execution set consisting of one node, its properties, and all necessary instructions to accomplish it. The system adds additional motion or procedures based on an analysis of the step and the arm location as needed.

Parser

The parser's main role is to read the slicer text output and recognize the basic structure elements such as layers and beads. The parser ignores any commands that are not related to the specific robot manipulator or printing process.

Translator

The translator receives the data from the parser and packs it into a task consisting of layers and nodes with assigned properties. In addition, each node carries information of its current layer, the total number of nodes for that layer, and the total number of layers for that task. As necessary, the translator converts the slicer coordinate frame to the manipulator coordinate frame. Applying base height of the manipulator is a good example of such an action. The Parser and Translator could be designed as one entity, but for elegance of design and modularity, it is preferable to split them into two modules.

Simulation Module

An optional module can be used to run a full kinematics simulation to check if the motion is within the arm's workspace and to ensure that singularities and link constraints are avoided.

Task Execution Module

The Task Execution Module (TEM) interprets the newly created structure node by node, applies user interface settings, monitors the process, provides user feedback, executes control loop strategies, and provides input to the data logger.

Data Acquisition Module (DAQ)

To execute G-Code commands for the given process, it is often important to perform data acquisition on the robot and process parameters. For example, data acquisition can be used for closed loop control. This process should be implemented in such a way that it does not interfere with the primary control system while offering adequate acquisition rates.

Data Logger

A data logger is a software agent that collects all chosen system settings in a text file with a time stamp. It is used for part certification and build analysis.

Graphical User Interface (GUI)

The graphical user interface is a critical component of the system that must be addressed when creating a control system for advanced AM. GUI enables the user to import geometry, to set proper parameters, and to notify the user of build progress throughout the build. In a case of research and development projects, it is a valuable tool in diagnosing, understanding, and solving problems. Like the data acquisition module, it should be implemented in a separate thread so that the main control systems are not affected by user interface delays.

## System Structure Details

Parser - The parser data flow is as follows (Fig. 6):

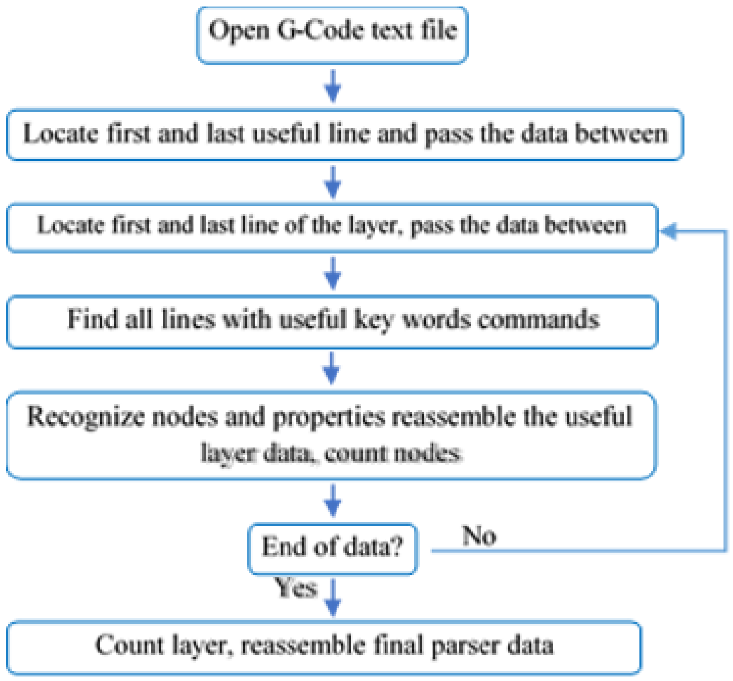
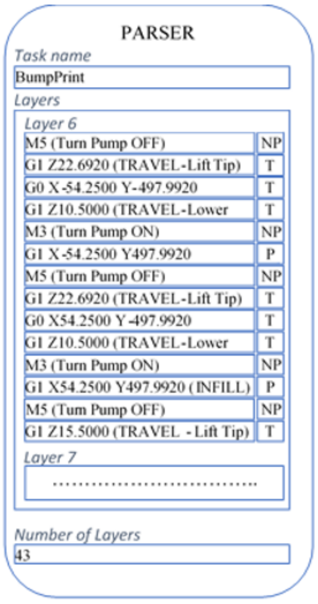

Open G-Code text file
Locate first and last useful line and pass the data between
Locate first and last line of the layer, pass the data between
Find all lines with useful key words commands
Recognize nodes and properties reassemble the useful layer data, count nodes
End of data?
No
Yes
Count layer, reassemble final parser data

Fig. 6 Parser flow diagram.

1. Read/load G-Code file.
2. Find the start and the end of the useful portion of the file. Not everything in the header and the end of the file is useful for robotic arm motion.
3. Identify the first and last line of the layer.
4. Beginning at the start line of the layer, identify all the code lines important for the process using their G-Code command descriptors.
5. Identify the move/process or property for each filtered line.
6. Repeat the process until all layers and lines of G-Code are analyzed.
7. Insert the information into newly defined structure containing: task name, lines of useful G-Code, and number of layers. The final form for the parser data is presented in Fig. 7



PARSER

Task name

BumpPrint

Layers

Layer 6

| | |
|---|---|
| M5 (Turn Pump OFF) | NP |
| G1 Z22.6920 (TRAVEL-Lift Tip) | T |
| G0 X-54.2500 Y-497.9920 | T |
| G1 Z10.5000 (TRAVEL-Lower | T |
| M3 (Turn Pump ON) | NP |
| G1 X-54.2500 Y497.9920 | P |
| M5 (Turn Pump OFF) | NP |
| G1 Z22.6920 (TRAVEL-Lift Tip) | T |
| G0 X54.2500 Y-497.9920 | T |
| G1 Z10.5000 (TRAVEL-Lower | T |
| M3 (Turn Pump ON) | NP |
| G1 X54.2500 Y497.9920 (INFILL) | P |
| M5 (Tum Pump OFF) | NP |
| G1 Z15.5000 (TRAVEL - Lift Tip) | T |

Layer 7

………………………………

Number of Layers

43

Fig. 7 Parser processed G-Code.

Translator - The translator has the following flow steps (Fig. 8):

1. Read the current system calibration.
2. For every layer
    a. Identify each node X, Y, and Z values.
    b. Apply part calibration.
    c. Apply default arm orientation. It is usually an orientation with the last joint vertical and pointing down toward the print table
    d. If the process is directional, calculate the azimuth angle for the direction.

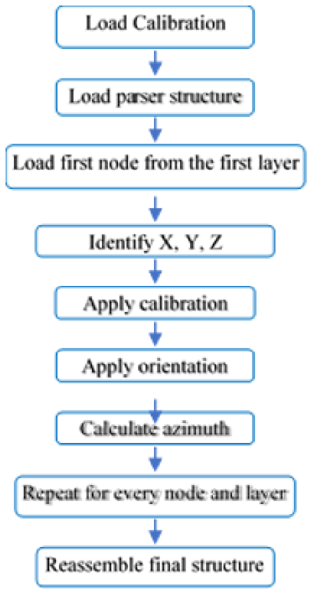
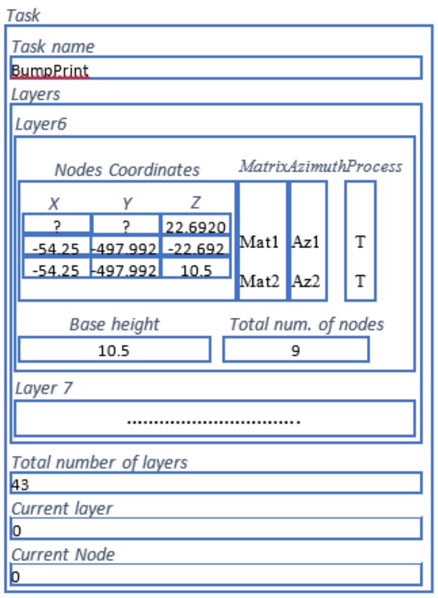


Fig. 8 Translator flow diagram.

3. Create the final task structure in a form following: task name, layers (node vectors: (XYZ), orientation matrix vectors, direction angle vectors, process vector, base height, and total number of nodes in the layers), total layer number, current layer, and current node. This form is presented in Fig. 9. The current node and current layer fields are used during the execution process to keep track of the progress and inform the operator via GUI.



Task name: BumpPrint

| X | Y | Z | Matrix | Azimuth | Process |
|---|---|---|---|---|---|
| ? | ? | 22.6920 | | | |
| -54.25 | -497.992 | -22.692 | Mat1 | Az1 | T |
| -54.25 | -497.992 | 10.5 | Mat2 | Az2 | T |

Base height: 10.5
Total num. of nodes: 9

Total number of layers: 43
Current layer: 0
Current Node: 0

Fig. 9 Translator data structure and transformed data.

TEM Structure- This module takes one step at a time from the task and executes it in a following manner (Fig. 10):

1. Start new data records (only the first time) with the data logger and data acquisition module.
2. Start task timer. Other timers are handled on an as-needed basis. The most standard timers are the pause timer and layer timer. Their time data is stored in the data logger and used in the GUI as well as for process analysis.
3. Check for *Step* completion. This portion of the algorithm runs in a logical loop. It means the execution comes back to this point at every step cycle. A new task starts with this flag set to "YES."
4. Check for pause or stop conditions. If the system has pause capability, this portion of the algorithm is used to stop both the motion and the deposition and allow for part inspection while keeping all the system states in the memory. It also allows for clean reentry back into deposition.
5. Load the printing speeds and deposition rates from the GUI.
6. Load a *Step* from the *Task*. It pulls a single node from the current layer along with its orientation matrix azimuth angle process type and base height.
7. Update the GUI, data logger, and data acquisition. All the user-available data gets updated at this step. Data logging and data acquisition receive their start triggers.

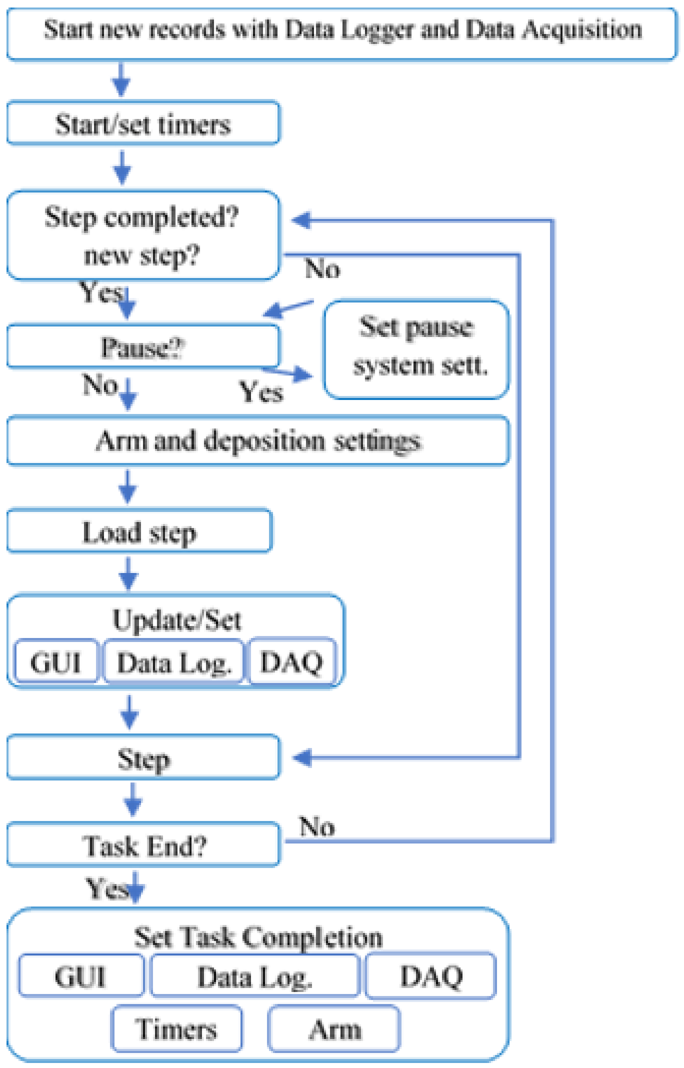


Fig. 10 Task execution module diagram.

8. Step execution (Fig. 11). Step execution starts with the application of a variable height. Variable height is optional and advantageous if the process requires manually settable height correction. Next, the system checks for cable unwinding and node properties to determine if immediate actions are needed before the start. If the previous node was in deposition mode and unwinding is necessary, the first action will be to stop deposition. The next step after unwinding is print head (end-effector) orientation. In this part and the position block, the inverse kinematics for the specific manipulator is performed. The final part of the sequence is the position command and deposition start sequence. Depending on the process, this can be a series of process-specific steps. If closed loop, real-time control is required for the process or any geometry command over-write, this is the place in the algorithm where it should be applied.
9. If this was the last *Step*, the system moves to final task completion.
10. Next, the algorithm goes back to point 3 and continues until the *Step* is completed.

11. After all the steps are completed, the system performs a graceful shut down. This includes parking the arm in the *Work Home* position. If not possible, the arm needs to lift up and clear the created part. In addition, all the timers need to be reset, data logging files can now be closed, and the arm should be set to normal settings.

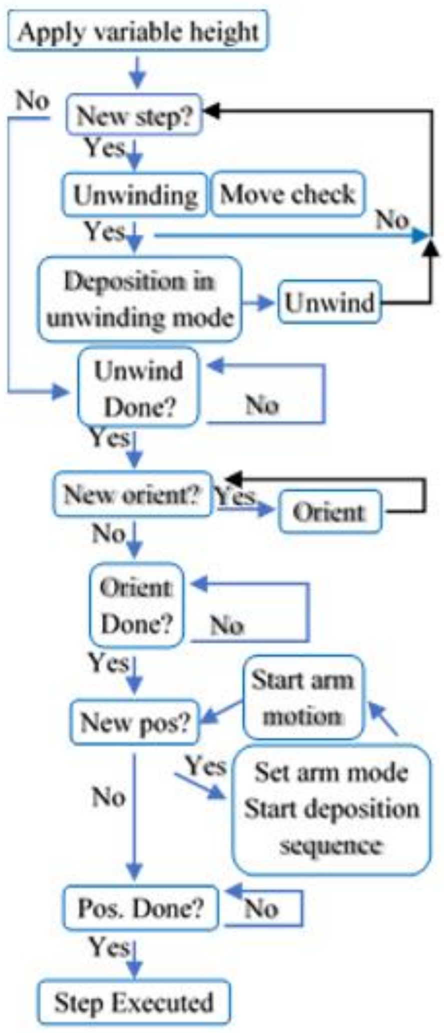


Fig. 11 Step flow diagram.

## System Implementation and Results

The presented algorithm was implemented and used to build a wire-arc deposition system. It consisted of a seven DOF Mitsubishi PA10 arm, Miller Dynasty 350 TIG welder, CK Worldwide WF-5 wire feeder, and pyrometers. A TIG torch was mounted on the end effector and used a deposition head. This setup was used to do wire deposition testing to build simple geometric shapes including single and multi-bead walls that were up to 1 ft tall, square tubes, hexagon tubes, and cubes. These geometries are the standard building blocks of any directed energy deposition system. The goal in printing these geometries was to test the feasibly of using this system. Eventually, a 1m. long replica of a Willys Jeep bumper (Fig. 12) was printed using this system. The Willys Jeep bumper was chosen to see if it was possible to print something as large as three to four ft. Further mechanical testing of the bumper showed that the isotropic mechanical properties and strength equal a fully solid block of material. With the use of high strength wire, the properties can be as good as or better than original legacy parts.

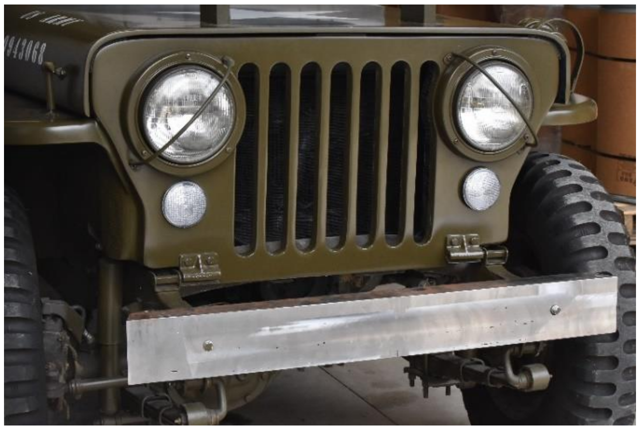


Fig. 12 Willys Jeep with a 3D printed bumper.

The practical software implementation was done using LabView (Fig. 13) with before mentioned hardware. The arm-specific control commands were included in the manufacturer supplied DLL (dynamic linked library). A LabView wrapper was created to call these functions and take advantage of the inverse kinematic algorithms.

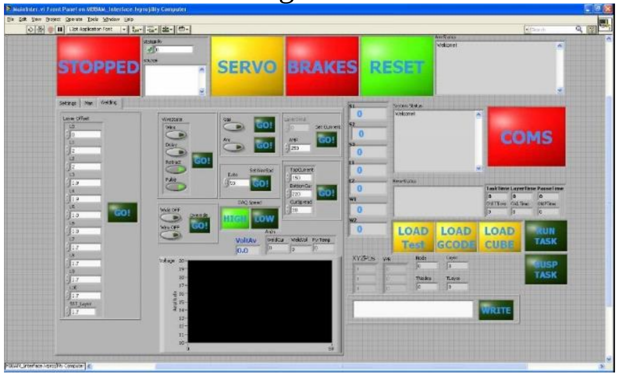


Fig. 13 GUI LabView interface with the algorithm implementation.

The GUI interface was updated at a maximum of 4Hz, whereas the TEM had 3ms forced delays between the cycles with a maximum theoretical rate of 333Hz. The remaining threads were run on demand.  Since the system did not require time resolution in micro second range, the implementation was done on a Windows based system with two Xeon CPUs. The low-level commands were executed through Mitsubishi-supplied hardware controllers. Therefore, to take full advantage of real-time capability and closed loop control, a high-level system with 500Hz or better with a hardware clock would be preferable. TEM can be implemented in both software and real-time based systems, but a real-time system is preferred for arc interpolation or complex moves and good process timing.

## Conclusion

This paper presents a system non-specific step-by-step procedure to use slicer-generated G-Code toolpaths to control robotic manipulators. The presented framework can be used for several additive manufacturing (AM) methods, hybrid solutions, or applications not directly related to AM. The implementation was successfully tested on a seven DOF robotic manipulator that successfully performed hundreds of hours of large-scale wire arc metal deposition.

## Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Energy Efficiency & Renewable Energy, Advanced Manufacturing Office, under contract number DE-AC05-00OR22725.

## References

1. Roebuck, K., *3D Printing: High-impact Emerging Technology-What You Need to Know Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. 2011: Tebbo.
2. Oberg, E., et al., *Machinery's handbook*. Vol. 200. 2004: Industrial Press New York.
3. Standardization, I.O.f., *ISO/ASTM 52900:2015 (ASTM F2792),* in *Additive manufacturing -- General principles -- Terminology*. 2015. p. 19.
4. Dritsas, S. and M. Goulthorpe. *An Automated Robotic Manufacturing Process: For the Thermoplastic Panel Building Technology*. in *International Conference on Computer-Aided Architectural Design Futures*. 2013. Springer.
5. Surdilovic, D., et al. *Advanced methods for small batch robotic machining of hard materials*. in *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*. 2012. VDE.
6. Slavkovic, N., D. Milutinovic, and M. Glavonjic, *A method for off-line compensation of cutting force-induced errors in robotic machining by tool path modification.* International Journal of Advanced Manufacturing Technology, 2014. **70**.
7. Mitchell, M., J. Oldham, and A. Samuel, *Advanced linux programming*. 2001: New Riders Publishing.
8. Bress, T., *Effective LabVIEW Programming:(* new file uploaded 02/19/15)*. 2013: Nts Press.
9. Atzeni, E. and A. Salmi, *Economics of additive manufacturing for end-usable metal parts.* The International Journal of Advanced Manufacturing Technology, 2012. **62**(9-12): p. 1147-1155.