Implementation of a robot control architecture for additive manufacturing applications

Filipe Monteiro Ribeiro and J. Norberto Pires
Universidade de Coimbra Departamento de Engenharia Mecânica, Coimbra, Portugal, and
Amin S. Azar
SINTEF Industry, Oslo, Oslo, Norway

Abstract

Purpose – Additive manufacturing (AM) technologies have recently turned into a mainstream production method in many industries. The adoption of new manufacturing scenarios led to the necessity of cross-disciplinary developments by combining several fields such as materials, robotics and computer programming. This paper aims to describe an innovative solution for implementing robotic simulation for AM experiments using a robot cell, which is controlled through a system control application (SCA).

Design/methodology/approach – For this purpose, the emulation of the AM tasks was executed by creating a robot working station in RoboDK software, which is responsible for the automatic administration of additive tasks. This is done by interpreting gcode from the Slic3r software environment. Posteriorly, all the SCA and relevant graphical user interface (GUI) were developed in Python to control the AM tasks from the RoboDK software environment. As an extra feature, Slic3r was embedded in the SCA to enable the generation of gcode automatically, without using the original user interface of the software. To sum up, this paper adds a new insight in the field of AM as it demonstrates the possibility of simulating and controlling AM tasks into a robot station.

Findings – The purpose of this paper is to contribute to the AM field by introducing and implementing an SCA capable of executing/simulating robotic AM tasks. It also shows how an advanced user can integrate advanced simulation technologies with a real AM system, creating in this way a powerful system for R&D and operational manufacturing tasks. As demonstrated, the creation of the AM environment was only possible by using the RoboDk software that allows the creation of a robot working station and its main operations.

Originality/value – Although the AM simulation was satisfactory, it was necessary to develop an SCA capable of controlling the whole simulation through simple commands instructed by users. As described in this work, the development of SCA was entirely implemented in Python by using official libraries. The solution was presented in the form of an application capable of controlling the AM operation through a server/client socket connection. In summary, a system architecture that is capable of controlling an AM simulation was presented. Moreover, implementation of commands in a simple GUI was shown as a step forward in implementation of modern AM process controls.

Keywords Robotics, Additive manufacturing, Additive manufacturing simulation

Paper type Research paper

1. Introduction

Nowadays, the industry is constantly searching for modern approaches (Pires, 2007), as it is necessary to develop new solutions and formulas for classical problems that may not introduce new sets of challenges (Pires *et al.*, 2006). This paradigm shift happens in virtue of previous production methods, which were based on application of a machine for each operation. Such approaches are becoming outdated due to many reasons, including but not limited to economic and environmental reasons.

The adoption of new solutions can be achieved by using non-conventional working scenarios packed with machines, which are capable of executing a wide range of operations. The main goal of applying this concept is to decrease the production costs, through the increase of the system efficiency, toward saving resources such as prime materials and energy. Although such scenarios are based on the economical revenue, the quality should almost never be compromised.

There are several international initiatives (Pires and Azar, 2017, 2018a, 2018b) that are expected to make a significant impact on industrial variables; examples are as follows:

- *Production time*: A decrease of at least 20 per cent is estimated as the operations are optimized and fully integrated into only one cell.
- *Production costs*: A decrease of at least 25 per cent is estimated as all the former process integration into the single working cell reduced the capex and machinery costs.
- *Resources costs*: A decrease of at least 50 per cent is estimated due to the technologies and architecture in this process, in which it is important to highlight the usage of hybrid manufacturing process, resulting in reduced lead-time between operations.

The current issue and full text archive of this journal is available on Emerald Insight at: **www.emeraldinsight.com/0143-991X.htm**

Industrial Robot: the international journal of robotics research and application
46/1 (2019) 73–82
 [ISSN 0143-991X]
[DOI 10.1108/IR-11-2018-0226]

Dr Albert Nubiola, CEO of *RoboDK*, is gratefully acknowledged for granting access to the software for the Hyroman initiative.

Received 5 November 2018
Revised 12 December 2018
6 January 2019
Accepted 8 January 2019

In this article, some considerations about the creation of a virtual robot working cell, currently referred as digital twin, will be presented. It is intended to simulate additive manufacturing (AM) tasks, and the system control application (SCA) responsible to control and adjust all the robot simulation parameters.

1.1 Additive manufacturing overview

In the 1980s, the rapid prototyping method was developed as a way to produce 3D objects, layer by layer, directly from CAD packages. It was a significant advance in industry as models and prototypes could be easily produced.

AM extended significantly this concept, just by introducing a set of technologies that allow a 3D object to be built by the simple process of adding layer upon layer of material until reaching the desired shape (ASTM-International, 2012). The main advantage of the AM is the versatility of the concept, which allows to create virtually any 3D shape as they are all bases in layer-by-layer manufacturing. Consequently, modeling and prototyping were the first applications. But quickly it started to be applied to manufacturing of small series, tricky parts, etc., and are now seriously considered for the production of large metal parts that are difficult to manufacture in traditional manufacturing plants, i.e. using traditional manufacturing technologies. Also, producing complex 3D parts, for several industries, including aeronautics and space, seems easier and more efficient if AM is included in the manufacturing process. In fact, AM has grown significantly in the past years, especially because of the industrial interest on the subject. However, it is important to clarify that AM is not just limited to the commonly used printers, optimized to build plastic parts. The ASTM committee (ASTM-International, 2012) published a set of standards which describe all the current seven classes of AM technology.

As in the conventional technologies, each AM technology uses specific materials as genetics of the processes does not allow a full material coverage. Consequently, as explained in the study conducted by Bourell *et al.* (2009), the materials used in AM technologies can be grouped into two categories; homogenous and heterogeneous materials. In addition, some studies about the practical application of these materials have been done, such as the ones in the following table (Table I).

Accordingly to Bourell *et al.* (2009) and Mueller (2012), it will be possible to see AM technologies in industries such as aerospace, military, automotive and motorsport, electronics, biomedical, jewelry, collectables, dentistry, food, education and toys in the near future.

According to Espacenet (Ménière *et al.*, 2017), the number of patented intellectual properties regarding AM are facing an exponential growth since a few years back, a behavior that is expected to keep its tendency justified by the increase of investment from big companies and countries, that are inserted in the program Industry 4.0.

AM promises to overcome many boundaries imposed by conventional manufacturing technologies such as the production limitations of components with complex geometries/shape, and the excessive waste of material due to excessive wall thickness or the type of the technological process.

Table I Important AM projects in each type of material

Type of materials	Important studies about applications which use AM technologies
Homogenous materials	
Polymers, such as epoxies and thermoplastics	Odom *et al.* (2017)
Natural materials, such as living tissues, paper/adhesive, starch	Melchels *et al.* (2012)
Metals such as alloy compositions	Tan *et al.* (2017)
Ceramics such as glasses, cement	Guo and Leu (2013)
Heterogeneous materials	
Polymeric matrix composites (PMC)	Tekinalp *et al.* (2014)
Metallic matrix composites (MMC)	Murr *et al.* (2012)
Ceramic matrix composites (CMC)	Eckel *et al.* (2016)

However, it is important to notice that in an industrial environment, all of these AM technologies are only possible due to the integration of multidisciplinary areas.

Industrial robot manipulators have been allied with AM technologies due to the fact that they are machines with a huge potential in this field as they have natural characteristics that make them enabled to perform AM tasks such as the ability to perform repetitive tasks, a high reliability and performance, easy to program and control, and, the ability to fabricate large components, an option that is challenging for common AM machines.

A recent study conducted by Evjemo *et al.* (2017) shows some projects that are impractical by using a traditional 3D-printing machine due to physical limitations Most of the common machines only have 3 degrees of freedom (DoF), which only allows translations along X, Y or Z. In the worst-case scenario, a robot manipulator has more than 5 DoF, allowing execution of rotational movements in addition. Such setups make difference in a 3D-printing task because it allows keeping the extruder nozzle correctly oriented during the movements.

Some of the recent crucial developments could be challenging to realize if the advanced robot manipulators were not in use; examples are as follows:

- *Concrete printing*: The aim is to produce full-scale constructions and architectural components such as walls, bridges and houses (Bos *et al.*, 2016).
- *MX3D bridge*: This project aims to build strong, complex and gracious structures in steel, by a combination of AM technologies and advanced robots.

State-of-the-art summaries about AM and related technologies can be found in Gardan (2016) and Lehmhus *et al.* (2018). These two papers introduce the various AM technologies currently available, namely, laser, flash, extrusion, jet, lamination & cutting, etc., and discuss their major advantages and drawbacks, including an overview of the manufacturing processes that can take advantage from their basic characteristics.

Incorporating simulation of the AM process into the system, improving in this way the quality of the printing process, the capacity to do it right the first time and achieve autonomy, was also proposed recently by several authors (Azar and Pires, 2018; Seidel *et al.*, 2014; Zadpoor, 2018; Witvrouw *et al.*,

2017). The idea is to fully transform the AM process, just by adding mechanisms to better plan the manufacturing process and to dynamically resolve the problems as they arise. This means creating a data-driven AM system with the capacity of dynamic trajectory planning, taking information from near real-time process data and updated simulations (Azar and Pires, 2018).

In summary, the robot-assisted AM technologies are paving the way to become an eminent part of the factories of the future. It is hard to ensure that by implementing these technologies, the challenges will be settled. Nevertheless, it is undeniable that it will help to reach a commonly acceptable resolution. In this line, the major contribution would be in terms of elevating the grade of these machinery to reach the level of plug-and-play and higher autonomy, with the objective of eliminating the inconveniences and challenges in programming of these advanced machines for executing a sophisticated printing task. In this article, an innovative method for creation of a virtual working cell using *RoboDK* and *Slic3r* software and a new design of SCA and interface using *Python* high-level programming is delineated.

1.2 Software

1.2.1 RoboDK

RoboDK is an industrial robot software which has the possibility to perform offline/online robot programming and simulation (Nubiola, 2015).

The following are the advantages of this software package for becoming the choice of this study:

- For sake of conceivable object-oriented programming, it is necessary to add visual perspective during the simulation of the process. *RoboDK* provides a graphical presentation of the cell and it is possible to perform offline robot programming.
- A wide range of robot manipulator brands and tools are available in *RoboDK*. Therefore, it is possible to create a generic task that can be executed on more than 200 types and brands of robots.
- *RoboDK* offers a built-in *gcode* interpreter for execution of tool-path related projects.

1.2.2 Slic3r

Slic3r is an open-source software launched in 2011 by the *RepRap community*, with the objective of making self-replicating machines freely available for the benefits of the users. Following the rules of this community, *Slic3r* is used to convert 3D models into printing instructions by generating *gcode*, which is a programming language used to create numeric instructions that make a machine moving in Cartesian coordination system.

The input of the program is a pre-designed *CAD* file and the output is the *gcode*, which is used for path generation in the robotic printing technology.

1.2.3 Python

Python is an *interpreted high-level programming language* created by Guido van Rossum in 1991, and designed to work in a wide range of domains *(General purpose programming language)*.

Python is known for its versatility, for being a fast development environment, an open-source software and also a "clean" and "clear" programming language (Peters, 2004).

In this research *Python 3.6* was used because of the following reasons:

- The *RoboDK* source code was developed in *Python* and it offers a specific library that allows controlling all the robot station operations called *Robolink*. This library s used for offline/online programming and simulation.
- *Python* is one of the most applied programming languages in the world, and it is also experiences one of the highest growth rate (StackOverFlow, 2018).
- *Python* has a wide range of libraries such as *Asyncio*, *threading*, *subprocess and kivy*, which allow the creation of a comprehensive SCA.

1.3 Additive manufacturing simulation

One of the goals of the developments in this study is the development of an AM simulation, and this section will reflect all the work done to achieve it by using *RoboDK* and *Slic3r*. Basically, the system starts with the CAD file of the part to 3D print and follows the following steps:

- The user must slice the part to be printed, i.e. generate the code containing the robot trajectories required to manufacture the part. Slic3r is used to generate *gcode*. A generic configuration file is given for each to work with each user application.
- Automatic robot code is obtained from the gcode taking in consideration the AM process selected, along with the type of AM process adopted.
- There is the option to upload the generated code to the robot controller and control the operation of the robot.
- There is also the possibility to include mechanisms to observe online the quality of the process under study and perform, also online and in real-time, the parameters of the AM process under study (Azar and Pires, 2018).

These aspects will be described in detail in the rest of the paper.

1.3.1 Robot offline programming

The robot offline program is an essential part of this research toward emulating the real working cell environment, as well as all the AM tasks (e.g. extrusion control and control of the heating systems). In addition, the advantages of using this way of programming are tremendous (Nubiola, 2015; Montaqim, 2015):

- There is no break or pause in the production flow.
- It is safer because there is no involved risk of damage to the working cell and its operator because of its virtual nature.
- All the simulated processes can be reproduced in the real working cell as the virtual environment replicated the real case scenario.

To make the virtual developments useful in the physical robot cell, the design has to follow some important considerations:

- The robot must move freely, without any restriction or obstacle, otherwise it will stop the operation with error messages (collision control).
- The robot must have a tool for simulating the material deposition that can be replaced by any other tool in a short instance.
- All the deposited parts are built on an appropriate region such as a substrate on a workbench.

Before presenting the working cell architecture, it is important to discuss some details in the process of creating the presented working cell. As a good practice, in *RoboDk*, the working station should always follow a hierarchy in terms of sequence of introducing tools. The reason for following the hierarchical sequence is to make a reference frame, which allows the objects and tools to be oriented correctly. This is a delicate subject as it is the only way to ensure that all the generated paths and simulated operations will keep the same reference frame and to avoid problems such as singularities or wrong tool orientation.

Hence, the robot base was adopted as a reference frame because all the subsequent operations will be executed by the robot.

The hierarchy construction rule implies that all the other frames are dependent on the reference frame, as can be seen in Figure 1.

After this brief and important consideration, the working cell was designed following the previously presented requisites. In terms of design, the final solution consists of a table which supports a robot equipped with an extruder and a bed plate. The selected robot model is *ABB IRB140*, which is available in the *RoboDK* library, as well as the extruder and the work bench. The substrate was built in a CAD editor with dimensions of 250 × 250 × 10 mm and imported to the software environment.

Table II lists the vectoral positions (x, y, z, θ_1, θ_2, θ_3) of the used objects. The position of the extruder is not mentioned as it is attached in the robot flange.

Figure 2 shows the designed virtual cell. The generation of the robot program for executing the AM simulation is presented and discussed in the next section.

1.3.2 Robot programs

An AM simulation *gcode* is composed of millions of points. Thus, the task of translating these connected points into a robot language is a critical step toward warranting the significance of all the movements.

Figure 1 Reference frame and all the dependencies

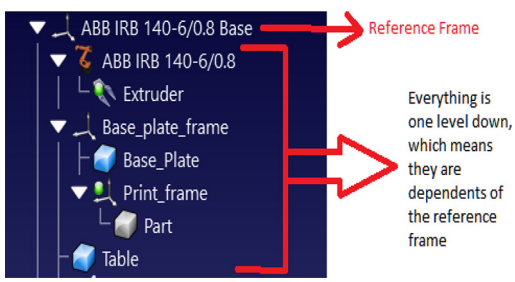
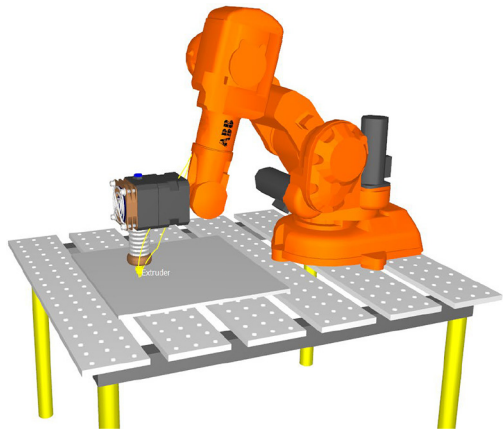


Table II Robot and objects position in the working cell

Object	Position
Robot base	(0,0,0,0,0,0)
Table	(310,0,0,0,0,0)
Plate	(600,0,0,0,0,0)

Figure 2 Working cell design



The *RoboDK* software offers possibility of compiling information about the simulation, such as the *robot flange velocities*, *paths*, *types of movements* and controlling the *active tool operation* from *gcode* into robot-specific program. In this research, programs created through both manual and automatic approaches as presented below.

1.3.3 Manual programming

Manual programming is similar to *teaching by showing* presented by Lozano-Peres, where targets and operations are defined by demonstration (Lozano-Perez, 1983).

Targets are the space coordinates of each position and orientation occupied by the robot flange, which will be interpolated into a *linear*, *joint or circular movement* in the backstage. In addition, parameters such as robot flange velocity or material deposition conditions are also defined manually.

In this research, manual programming is used to define the following AM operations:

- approaching the robot flange and the mounted tool, close to the substrate surface; and
- returning the robot to a home position, after the AM simulation.

Both approaching and homing operations were performed by joint movements only as a high movement pace was envisaged. The velocity was set to 150 mm/s and the precision was set to high. Figure 3 shows the two aforementioned targets.

1.3.4 Automatic programming

RoboDK can generate automatic programs using the incorporated projects. The presented automatic programming in this research, which is responsible to simulate an AM process for building a simple CAD model, uses a project called *machining/welding project*.

This project only requires a *gcode* file to generate the toolpath, extruder function controller and the robot flange velocities. To simulate the material deposition, a *Python script* was developed, that is executed once the robot program is called.

To generate the *gcode*, *Slic3r* software was used. As the AM task is assumed to be fused deposition modeling (FDM), the printer configurations from *Lulzbot*[1] was used to define the slicing conditions.

Figure 3 Robot flange approaching the plate surface through a joint movement

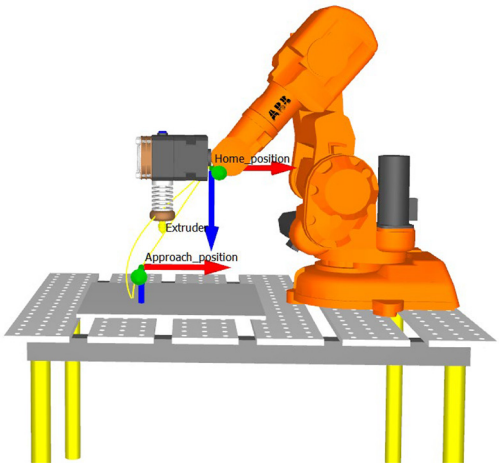


Figure 4 shows the milling project user interface, and the *toolpath* that will simulate the AM task.

At the end of this operation, all programs were grouped in an eventual solution that performs AM simulation on the robot. This program is responsible to make predefined sequential calls of the robot programs simulate all the essential robot movements/actions during simulation. The details about the sequence of command execution are as below:

- *Approach*: This module is the manually defined robot program that approaches the mounted deposition tool close to the substrate.
- *3D*: This module is the automatic robot program which contains all the information related to the toolpath.
- *Filament on/off*: This module is the *Python* program which simulates the material extrusion.
- *Return home*: This is a manual robot program module that moves the robot to the predefined home position at the end of the AM simulation (Figure 5).

2. System Control Application

SCA is a user-interface mechanism that enables the interaction between a user and the virtual/real robot station. The importance of allowing SCA in simulation environments, namely by providing interface mechanisms with the system, is related to the fact that it

Figure 4 Milling project user interface and the tool paths generated

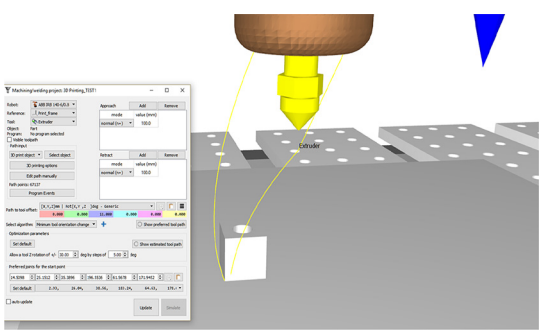
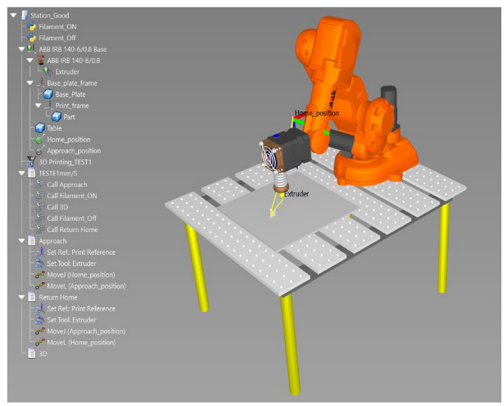


Figure 5 Final look of the working cell, which is ready to perform an AM simulation



allows users to control the operations through a previously designed and user-friendly application. The user-friendliness arises from a user interface that is intended to be intuitive and logical. These characteristics are achieved by creating the application with a simple design and clear commands, addressing the average factory floor staff independent of their experience.

As described previously, the SCA in this work was developed in *Python* through available libraries. Therefore, to understand the functionalities of the developed SCA, the applied libraries will be contextualized and explained accordingly to their objective of usage and the way they were combined to build a complete application.

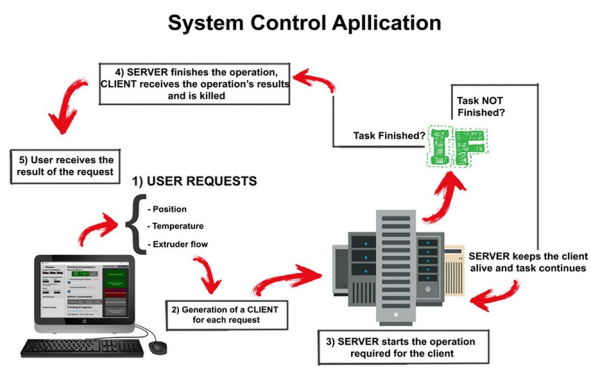
The basis for the developed SCA are (Figure 6):

- Asynchronous *transport control protocol/internet protocol (TCP/IP)* – Server/client.
- Robot Station Control.
- Generation of *gcode*.
- Graphical user interface, *GUI*.

2.1 Asynchronous transport control protocol/internet protocol – Server/client

Principally, it is important to recognize that the robot and the computer are two devices that do not perceive how to share and communicate information as they do not have an implemented native protocol, capable of being interpreted by both. Consequently, a *TCP* was implemented and combined with the

Figure 6 Flow of information in the developed SCA



IP, also known as *TCP/IP*, to establish the communication protocol.

The development of the SCA began with the creation of an *asynchronous server and client*, addressing the required SCA tasks in the AM simulation.

From a computer science perspective, the server and a client are part of a software architecture, where the objective is to establish and maintain inter-communication that the *clients always sent requests* while the server always responds the received requests. It is also responsible for *executing and scheduling the client's request* (Oluwatosin, 2014). Therefore, these two programs can be seen as a symbiosis case as they need each other to remain operational.

In the developed SCA, the clients represent all the operations requested by a user, while the server is the program responsible to receive the requests, and then, accessing the AM robot's station to set up the requested operation by the user, through *sockets* that are responsible for establishing the connection between the end nodes (Xue and Zhu, 2009) (Figure 7).

It is also important to ensure that the system is asynchronous, or else it is impossible to perform multiple simultaneous tasks (e.g. request position, extruder flow and bed temperature) as the system is blocked while it is performing an operation and it is only available at the end nodes.

The usage of asynchronous programming enables the system to interleave tasks, by suspending and returning through I/O control. This is called a *concurrent programming*, where all the tasks can start, run and complete at overlapping time periods. The biggest advantage of such strategy is the fact the system is always responsive and independent of the end of each operation to progress (Ghezzi, 1985) (Figure 8).

The implementation of an asynchronous *TCP/IP* server/client was implemented in *Python*, as it was mentioned before, using the *Asyncio* library (Rossum, 2012).

2.1.1 Asynchronous server creation

The first step in any *Asyncio* program is the creation of an event loop, which in this case should *run forever()* as the server should always be ready to perform any operation or to receive a new client. The usage of a loop that will run forever means that it will only stop when a *stop()* command is triggered. However, it is fundamental to have in mind that after stopping the loop it is impossible to start it again.

After implementing the loop, it is necessary to create a coroutine, which can be performed together with a task. The coroutine is responsible *to return* a tuple with two objects: a StreamReader, which is responsible to interpret the message received by the client, and a *StreamWriter*, which is responsible to send information to the client. One important characteristic about this coroutine is the fact that it only executes once a new connection is established as each client's connection requires a new socket or else, the application may be blocked or even result in mixed or missed information.

The final step was to *fire* the loop to start the server, making it able to receive the client requests. The main code of the server can be seen. Each line of the code is commented for clarification. Server code snippet:

```
import asyncio

class Server:

      def_init_(self):
            "This class is used to create
             the server"
             self.loop = asyncio.get_
             event_loop()
             self.loop.run_until_complete
             (self.coro) #runs everytime there
              is a client
             self.run_forever()

      async def_coro(self):
             "Asyncio Coroutine that defines the
             TCP/IP protocol to start the
             server"
             await asyncio.start_server
             (self.handle_echo, `localhost,
             port = 8888, loop = self.loop)
             #ensure
server is created

      async def handle_echo (self, reader,
      writer):
             "Asyncio coroutine which creates
             the StreamReader and StreamWriter
             self.reader = reader
             self.writer = writer

      del decisions(self):
             "Method used to decode the receives
             message and interact with the AM
             simulation" pass

if__name__ = `__main__':
      Server()
```

2.1.2 Asynchronous client creation

The client is not supposed to keep connections alive forever as it will lead to a *high consumption of the computer resources*, which affects parameters such as *connection velocity*, as well as

Figure 7 Schematic representation of a sock

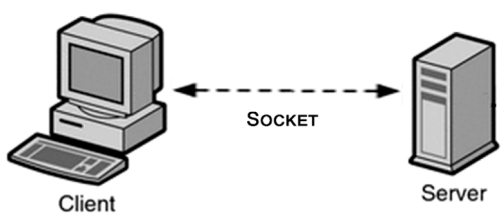
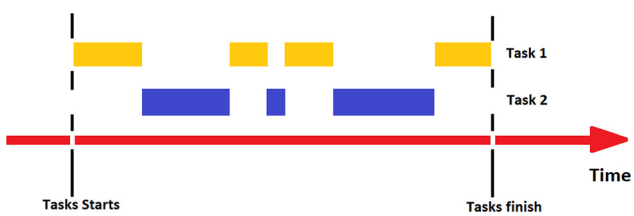


Figure 8 Illustration of how the tasks are interleaved in a concurrent programming



tasks and app performance. Problems can be easily avoided if the client is killed at the end of each *socket*. In other words, the client should *run until complete*, whereas the server will *run forever*.

It is important to note that if the server is asynchronous, the client also needs to be in the same scheme. Otherwise, the code will be blocked until the end of the client's work, due to the lack of *concurrent tasks*. The client was created with the *Asyncio* library, which has specific documentation about implementation of *TCP*/IP client (Rossum, 2012).

The first step toward creating a server is implementation of the *event loop*, which always needs to be prepared to generate a new client when sending a request to the server and also to kill it whenever the work is finalized.

After creating the loop, the next step is to enable the client generation without waiting until the end of the previous task. The objective of the latter approach is to eliminate the blocking risk and to allow parallel requests. The solution to these problems was the creation of a task that will run the *thread safe* coroutine. By running this coroutine, the loop will be able to perform the creation of a new client, because it will prepare the loop to start a coroutine, which was not scheduled before firing the loop, which has initial running conditions without breaking the loop.

The source code of the scrutinized client can be seen. Each line of the code is commented for clarification. Client code snippet:

```
import asyncio

class Client:

    def __init__(self):
            "This class is used to create the
            server"
            self.loop = asyncio.get_
            event_loop()
            self.message = None

    async def create_cleinte(self):
            "Asyncio Coroutine that defines the
            TCP/IP protocol to create a
            client"
            reader, writer = await
            asyncio.open_connection
            (`localhost, port = 8888,
            loop = self.loop)
# creates the client as also the StreamReader
and the StreamWriter
            self.reader = reader
            self.writer = writer
            await asyncio.ensure_future
            (self.connection) #waits until the
            end of the client

    async def connection(self):
            "Asyncio coroutine to send and
            receive information"
            self.writer, write(self.message)
            #Writes on server
            server_message = await
            self.reader.read(100) #Receives
            the server's message
            await self.writer.drain()

    def new_request(self, message):
            "this method creates a new client
            self.message = message.encode()
            asyncio.run_coroutine_threadsafe
            (coro = self.create_client(),
            loop = self.loop)
if__name__ = `__main__':
    client = Client()
    client.new_request(message = "Hello World")
```

2.2 Robot station control

The server needs to perform tasks in the robot station; yet the server does not conceive how it shall be done as the source code is only able to accept clients and manage requests.

Therefore, it is necessary to implement a solution that will be called each time the server receives a request related to the robot station. In such way, the server is capable of performing the station related tasks and translating and sending back the information from the robot station to the user.

This controller was implemented using *Python* and *RoboDK* library (RoboDK, 2018) as all the functionalities are well integrated.

The implementation of all the mentioned concept was realized in two main parts; the *station recognition* and the *station tasks*. Both parts are methods of the *Subclass ValidateRobodk (Robolink)*.

2.2.1 Robot station recognition

The station recognition is a method that is responsible to detect all the objects in the cell, such as the robot, the working tool, the working frame and the final robot program. The recognition of the robot station is very important as the SCA must have access to the information about all the cell objects or else they will be uncontrollable from the application. A snippet view of the station recognition is presented. Code snippet showing the robot station Detection:

```
from robolink import *

class ValidateRobodk(Robolink):
  """RobotDk Class that makes possible manage
  robotDK, for this we use the Parent Class
  RoboLink"""

  def __init__(self):
     self.Render(True)
     self.ShowRoboDK()
     self.robot = self.Item
     ('', ITEM_TYPE_ROBOT) # This detects
     the robot
     if not self.robot.Valid():
        raise Exception('No robot selected or
        available')
```

2.2.2 Robot station tasks

The station tasks are responsible for interacting with the robot station to perform all the user's requests. The following are a list of tasks:

- *Get_position()*: A method responsible to get the real position of the robot flange.

- *Go_home()*: A method that sends the robot to a predefined home position.
- *Start_printing()*: A method that starts the robot simulation.
- *Pause_printing()*: A method that pauses the robot simulation.
- *Stop_printing()*: A method that stops the robot simulation.
- *Check_collisions()*: A method that checks if there is any collision.
- *Calibrate_robot()*: A method that performs an automatic robot calibration.
- *Station_ready()*: A method which verifies if the station is ready to perform.
- *Simulation_time()*: A method that calculates the simulation time during the simulation.

A snippet view of the code, which has all the methods referred above, is illustrated. Code Snippet showing the robot station task:

```
from robolink import *

class ValidateRobodk(Robolink):
  """RobotDk Class that makes possible manage
  robotDK, for this we use the Parent Class
  RoboLink"""

  def get_position(self):
      """This is a method that gets the real
      position of the robot flange"""
      position = Pose_2_TxyzRxyz
      (self.robot.Pose())
      position_final = [round(position[elem],
      2) for elem in range(len(position))]
      return position_final[:3]

  def go_home(self):
      """This is a method that sends robot to
      home position"""
      Robolink().RunCode('Return Home', True)
      return str("Robot at Home position")
```

The listed methods represent all the client requests, which are performed by *RoboDK* and launched by the server. As it was mentioned before, such AM-specific class is crucial for enabling the system to collect information about the real-time status of the robot station and launching the required user operation.

2.3 Generation of gcode

The SCA should easily generate executable *gcode* that runs conditionally on the AM setup. When it comes to the *Slic3r's* software, it is possible to execute the slicing tasks from the command line, which makes it easy to implement it in another programming language platform (Gary, 2013). Therefore, by launching the *Slic3r* executable in the *Python shell*, generation of the *gcode* can be performed.

If the commands in the documentation of the *Slic3r* software is implemented with care, there should not be any difference between the *gcode* generated from the command prompt and the *gcode* generated in the user interface.

The slicing command implementation was done by *Python*, using the subprocess library as it is necessary to execute command lines on the shell.

2.4 Graphical User Interface

GUI allows users to configure the operations in the system as it is usually simple and intuitive. In the SCA, GUI was developed to control the robot station through a series of simple interactions with the application, such as clicking buttons or draging bars and intuitive generation of the *gcode*. The GUI was designed and constructed using *Python*'s *Kivy* library as it allows creation of elegant interfaces easier and faster than *Python's* native GUI developer called *TkInter*. Moreover, *Kivy* has a better compatibility with several operating systems.

2.4.1 Graphical user interface's development

The first step to develop the GUI is the *creation of the Kivy's loop*, and *KV file*. This was done simultaneously to visualize the construction of the GUI. It shows a piece of the *Kivy* class code. Code snippet of kivy class responsible to generate the loop:

```
class InterfaceApp(App):

    def build(self):

      "Method used to build the loop and the
      User interface"
      return GenerateSlide()

    if __name__ == '__main__':
      InterfaceApp().run()
```

In terms of design, the GUI was projected using straightforward features such as buttons, selectors and bars, which only require one touch to interact with the *EventDispatcher*.

As it was explained previously, the server and the client are asynchronous programs that require an *Asyncio Event loop* to be executed, which should not be suspended to keep the server and the client alive. This requirement entails a serious problem as it turns *Kivy* and *Asyncio* into incompatible libraries because they do not allow the execution of simultaneous looping.

The rationale behind this is related to the fact that both loops operate in the same thread, causing in suspension of one loop during the execution of another. This leads to lack of resources that are necessary to keep both sides alive. Therefore, the solution for this problem is launching the *Asyncio's event loop* in a *new thread*, using the *multithreading library*, which allows parallel execution without breaking (Noller and Oudkerk, 2008). In addition, the *Kivy's* loop will be launched in a lower-level thread, which represents the *mainthread* of execution.

However, the threads will have to communicate with each other, otherwise, controlling the AM simulation may not be feasible as the *mainthread* only triggers the *callback* that will imply a client generation and the respective messages in the upper-level thread.

To handle this situation, a *worker* was created to act as an intermediate character between each thread-lever. Its working principle is based on a *callback* that is triggered every time the *EventDispatcher* creates an *event*. A message will be sent after the worker has been created, containing the operation to be executed to the *Asyncio* thread. However, the opposite working

flow must happen as the GUI must be updated, according to the user requests in the server.

Finally, the last step in the GUI development was to enable the application to generate the *gcode*. This feature was also implemented by launching another thread as the *gcode* program blocks during operation when it is required to *waits()* until the generation of *gcode*.

A complete view and description of the GUI can be seen in Figure 9.

It should be clarified that with this approach the user can run the simulation in real-time, fully controlling its behavior, and access an instance of RobotDK that handles the simulation process. Other work, from the same authors, explored the possibility of using other platforms, like *ABB Robotstudio* (Pires and Azar, 2018a, 2018b), including the possibility to have simulation and real production in parallel. The authors show a comprehensive demonstration of that possibility in Pires and Azar (2017, 2018a, 2018b), using a real test-case and showing full implementation details.

3. Conclusion

AM technologies are seen as one of the most influential area in research and development because of their positive impact on the industrial routines. However, these technologies still have a long way to reach maturity as the field is highly multidisciplinary.

The purpose of this paper is to contribute to the AM field by introducing and implementing an SCA capable of executing/simulating robotic AM tasks. It also shows how an advanced user can integrate advanced simulation technologies with a real AM system, creating in this way a powerful system for R&D and operational manufacturing tasks.

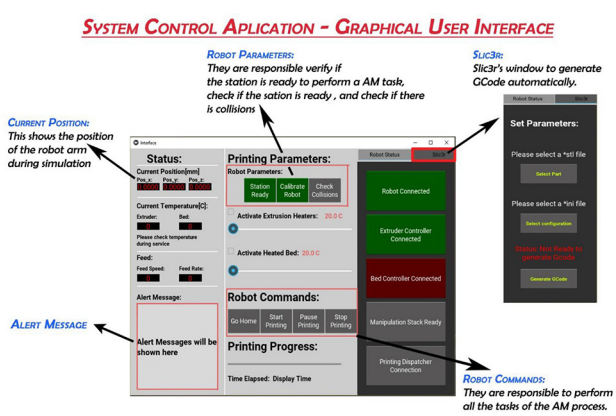
As demonstrated, the creation of the AM environment was only possible by using the *RoboDk* software that allows the creation of a robot working station and its main operations.

Although the AM simulation was satisfactory, it was necessary to develop a SCA capable of controlling the whole simulation through simple commands instructed by users. As described in this work, the development of SCA was entirely implemented in *Python*, using official libraries. The solution was presented in the form of an application, capable of controlling all the AM operation through a server/client socket connection.

In addition, the SCA is also capable of automatic generation of *gcode*, through as simple operation as clicking a button.

In summary, a system architecture was presented that is capable of controlling an AM simulation. Moreover, implementation of commands in a simple GUI was shown as a step forward in implementation of modern AM process controls.

Figure 9 Overview of the SCA



Note

1 TAZ PLA Profile – Medium PLA 0,35 mm Nozzle.

References

ASTM-International (2012), *Standard Terminology for Additive Manufacturing Technologies*, ASTM International, available at: www.astm.org/

Azar, A.S. and Pires, J.N. (2018), "Chapter 6.7: Impressão 3D de peças metálicas: método numérico Para seleção dos parâmetros do processo", *Robótica Industrial*, Lidel, Lisbon.

Bos, F., Wolfs, R., Ahmed, Z. and Salet, T. (2016), "Additive manufacturing of concrete in construction: potentials and challenges of 3D concrete printing", *Virtual Physical Prototyping*, Vol. 11 No. 3, pp. 209-225.

Bourell, D., Leu, M. and Rosen, D. (2009), *Identifying the Future of Freeform Processing*, Vol. 2009.

Eckel, Z.C., Zhou, C., Martin, J.H., Jacobsen, A.J., Carter, W. B. and Schaedler, T.A. (2016), "Additive manufacturing of polymer-derived ceramics", *Science*, Vol. 351 No. 6268, pp. 58-62.

Evjemo, L.D., Moe, S., Gravdahl, J.T., Roulet-Dubonnet, O. and Gellein, L.T. (2017), "Additive manufacturing by robot manipulator: an overview of the state-of-the-art and proof-of-concept results", *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, *IEEE*, pp. 1-8.

Gardan, J. (2016), "Additive manufacturing technologies: state of the art and trends", *International Journal of Production Research*, Vol. 54 No. 10, pp. 3118-3132.

Gary, H. (2013), *Slic3r User Manual*, available at: www.slic3r.com

Ghezzi, C. (1985), "Concurrency in programming languages: a survey", *Parallel Computing*, Vol. 2 No. 3, pp. 229-241.

Guo, N. and Leu, M.C. (2013), "Additive manufacturing: technology, applications and research needs", *Frontiers of Mechanical Engineering*, Vol. 8 No. 3, pp. 215-243.

Lehmhus, D., Busse, M., Von Hehl, A. and Jägle, E. (2018), "State of the art and emerging trends in additive manufacturing: from multi-material processes to 3D printed electronics", *MATEC Web of Conferences, 2018: EDP Sciences*, p. 03013.

Lozano-Perez, T. (1983), "Robot programming", *Proceedings of the IEEE*, Vol. 71 No. 7, pp. 821-841.

Melchels, F.P., Domingos, M.A., Klein, T.J., Malda, J., Bartolo, P.J. and Hutmacher, D.W. (2012), "Additive manufacturing of tissues and organs", *Progress in Polymer Science*, Vol. 37 No. 8, pp. 1079-1104.

Ménière, Y., Rudyk, I. and Valdes, J. (2017), "Patents and the fourth industrial revolution: the inventions behind the digital transformation", Report of the European Patent Office available at: www.epo.org, December, Munich.

Montaqim, A. (2015), "Offline programming software for industrial robots from RoboDK offers hundreds of virtual industrial robots from top robotics companies", *Robotic and Automation News*.

Mueller, B. (2012), "Additive manufacturing technologies: rapid prototyping to direct digital manufacturing", *Assembly Automation*, p. 32.

Murr, L.E., Gaytan, S.M., Ramirez, D.A., Martinez, E., Hernandez, J., Amato, K.N., Shindo, P.W., Medina, F.R. and Wicker, R.B. (2012), "Metal fabrication by additive manufacturing using laser and electron beam melting technologies", *Journal of Materials Science Technology*, Vol. 28 No. 1, pp. 1-14.

Noller, J. and Oudkerk, R. (2008), "PEP 371 – addition of the multiprocessing package to the standard library", Phyton developers guide, available at: https://staging2.python.org/dev/peps/pep-0371/

Nubiola, A. (2015), "The future of robot off-line programming", available at: http://coro.etsmtl.ca/blog/?p=529-2018

Odom, M.G., Sweeney, C.B., Parviz, D., Sill, L.P., Saed, M.A. and Green, M.J. (2017), "Rapid curing and additive manufacturing of thermoset systems using scanning microwave heating of carbon nanotube/epoxy composites", *Carbon*, Vol. 120, pp. 447-453.

Oluwatosin, H.S. (2014), "Client-server model", *IOSRJ Computational Engineering*, Vol. 16, pp. 2278-8727.

Peters, T. (2004), "PEP 20 – The zen of Python", Python developers guide, available at: www.python.org/dev/peps/pep-0020/

Pires, J.N. (2007), "Force control experiments for industrial applications: a test case using an industrial deburring example", *Assembly Automation*, Vol. 27 No. 2, pp. 148-156.

Pires, J.N. and Azar, A.S. (2017), "HYROMAN – Hybrid robotic additive HYROMAN – Hybrid robotic additive manufacturing platform for agile production of large multi-metal components", *Robótica*, p. 106.

Pires, J.N. and Azar, A.S. (2018a), "3D printing of large metallic parts", available at: www.youtube.com/watch?v=bUYLfFmXi_Y.-YouTube

Pires, J.N. and Azar, A.S. (2018b), "Advances in robotics for additive/hybrid manufacturing: robot control, speech interface and path planning", *Industrial Robot: An International Journal*, Vol. 45.

Pires, J.N., Loureiro, A. and Bomsjo, G. (2006), "Welding robots: technology, system issues and applications", Springer, London.

RoboDK (2018), "RobotDK API – robolink module", available at: www.robodk.com/download

Rossum, G.V. (2012), "PEP 3156 – asynchronous IO support rebooted", Python Developers Guide, available at: www.python.org/dev/peps/pep-3156/

Seidel, C., Zaeh, M., Wunderer, M., Weirather, J., Krol, T. and Ott, M. (2014), "Simulation of the laser beam melting process–approaches for an efficient modelling of the beam-material interaction", *Procedia CIRP*, Vol. 25, pp. 146-153.

Stackoverflow (2018), "Stack overflow developer survey 2018", available at: https://insights.stackoverflow.com/survey/2018/#technology (accessed 2018).

Tan, X., Tan, Y., Chow, C., Tor, S. and Yeong, W. (2017), "Metallic powder-bed based 3D printing of cellular scaffolds for orthopaedic implants: a state-of-the-art review on manufacturing, topological design, mechanical properties and biocompatibility", *Materials Science Engineering: C*, Vol. 76, pp. 1328-1343.

Tekinalp, H.L., Kunc, V., Velez-Garcia, G.M., Duty, C.E., Love, L.J., Naskar, A.K., Blue, C.A. and Ozcan, S. (2014), "Highly oriented carbon fiber–polymer composites via additive manufacturing", *Composites Science and Technology*, Vol. 105, pp. 144-150.

Witvrouw, A., Metelkova, J., Ranjan, R., Bayat, M., De, D., Baere, M.M. Tosello, G., Solheid, J., Charles, A. and Scholz, S. (2017), "Precision additive metal manufacturing", *Proceedings of the ASPE and euspen Summer Topical Meeting "Advancing Precision in Additive Manufacturing", Berkeley, CA*.

Xue, M. and Zhu, C. (2009), "The socket programming and software design for communication based on client/server", *Pacific-Asia Conference on Circuits, Communications and Systems, 2009. PACCS'09, IEEE*, pp. 775-777.

Zadpoor, A. (2018), *Frontiers of Additively Manufactured Metallic Materials*, Multidisciplinary Digital Publishing Institute, available at: www.mdpi.com/1996-1944/11/9/1566/htm

Corresponding author

J. Norberto Pires can be contacted at: norberto@uc.pt