LAB University of Applied Sciences

Industrial 3D printer farm automatization: a proof of concept

LAB University of Applied Sciences
Bachelor of Information Technology 2022
Thibault Dekock

Abstract

Author(s)	Publication type	Published
Dekock Thibault	Thesis, UAS	2022
	Number of pages	
	47	

Title of Publication

Industrial 3D printer farm automatization: a proof of concept

Degree and field of study

Bachelor of Information Technology

Abstract

The objective of the thesis was to prove that the process of industrial 3D printer farms can be improved by boosting their efficiency, speed, customer experience, and maintenance. The aim was to find out the current problems with these farms and to explore theoretical solutions for these problems. This thesis was also meant to show whether or not a collaboration between industrial robotic arms and 3D printing technology can help improve the efficiency of current 3D print farms.

The author also conducted a proof of concept using a desktop application, a Raspberry Pi to control a 3D printer remotely, and a robotic arm to harvest the printer after it is done printing. This proved that the fully developed system would definitely work.

The author found that the development of this kind of system is the future of schools/ companies with multiple 3D printers. Although this system would require a high step-in price, it can be seen as a return on investment (ROI) since it would require fewer employees and would increase the production rate by three-four times. This increase in production rate means that companies would require fewer 3D printers

Keywords

3D printing, Robotics, Slicers

Contents

Appendices

1 Introduction

Industrial automatization is the past, present and future of the industry. For this thesis, we focus on two rapidly developing areas. At first, industrial robotic arms are being used to execute specific tasks. When these work together, they can easily take over the tasks of humans in the production process. These robots can work, day and night, to do dirty or dangerous work, as well as repetitive work that is too boring for a human being. A big plus is that they are much cheaper than a regular employee with the bonus of eliminating human error.

The second area is the rapidly developing field is 3D printing. This is the process of making a three-dimensional object from a 3D model. It allows the user to create objects, that are extremely expensive and difficult to manufacture without 3D printing technology. The number of companies that provide 3D printing services is growing rapidly. This way, the relatively ‘new’ technology is reaching many people very fast. It is a golden age for both 3D printer manufacturers and service providers.

The primary aim of this thesis is to prove that the efficiency of 3D printer farms can be increased by combining both industrial robotic arms and 3D printing technologies. It will also show how both customer and producer experience can be improved.

This thesis will explain how 3D- printers work besides briefly discussing the different kinds of robotic arms. It will also show the deficiency of current print farms and how the production rate can be improved. Finally, a proof of concept is made to show a feasible solution.

2 3D printing

2.1 Intro

3D printing or additive manufacturing is the process of creating solid three-dimensional objects from a digital file. It allows you to create shapes, which are typically very difficult to manufacture, in a fast, precise, and easy way (Figure 1). It also uses less material than subtractive manufacturing.

(3DPrinting.com, 2022)

Figure 1. 3D printed object (Mindworks, 2022)

2.2 Applications

There are many applications for 3D printing that reach both professionals and hobbyists. It has the power to create things that were thought to be impossible to make twenty years ago.

A few examples of applications:

- robotics
- architectural scale models & maquettes
- prosthetics
- industrial products
- dental products
- reconstructing fossils
- replicating ancient artefacts
- building houses
- education
- aerospace
- movie props
- cosplay
- …

(3DPrinting.com, 2022)

Many of the examples mentioned above can correlate to one aspect of the production process: rapid prototyping.
Rapid prototyping is a technique used during the production process to create a very cost-effective version of the finished product, possibly with higher tolerances. This allows for more margin of mistake and the ability to make changes or improvements earlier in the design process (Figure 2).
Apart from the advantages mentioned above, there is also a significant benefit of presenting the products. The visualization of the product allows both colleagues and customers to give better feedback to the designers.
(Sabhadiya, 2022)

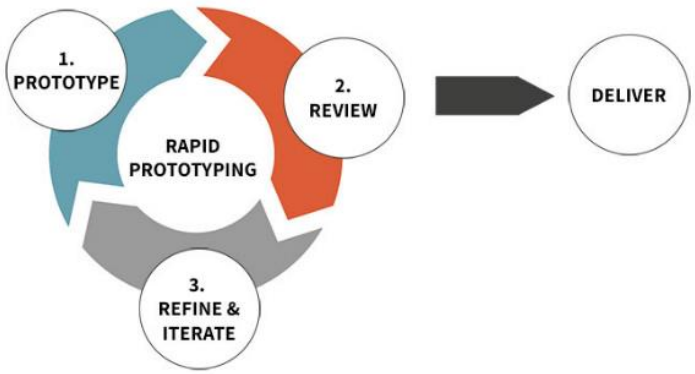

Figure 2. Rapid prototyping (Sabhadiya, 2022)

2.3 3D printing processes

There are many types of 3D printing processes. Each one is more suited for the application than the other. The choice of the process depends on several factors, including the volume or size of the part, the surface finish, the material (mechanical/ cosmetic requirements), the amount of post-processing, and the budget of the project. The most popular processes are FDM and SLA. (Ahart, 2022)

The 3D printing processes include

- Stereolithography (SLA)
- Selective Laser Sintering (SLS)
- Fused Deposition Modeling (FDM)
- Digital Light Process (DLP)
- Multi Jet Fusion (MJF)
- PolyJet
- Direct Metal Laser Sintering (DMLS)
- Electron Beam Melting (EBM)

(Ahart, 2022)

2.4 Fused Deposition Modeling

Fused Deposition Modeling or FDM printing is the most popular process at the consumer level. Through the use of heated nozzles, thermoplastics such as ABS, PLA, PETG, etc, can be extruded (Figure 3). The melted material is then applied layer by layer until the whole part is finished. In recent years, this technology can also be used with more exotic materials such as chocolate, pastes, concrete, or metal/ wood-infused thermoplastic. (Grames, 2022)

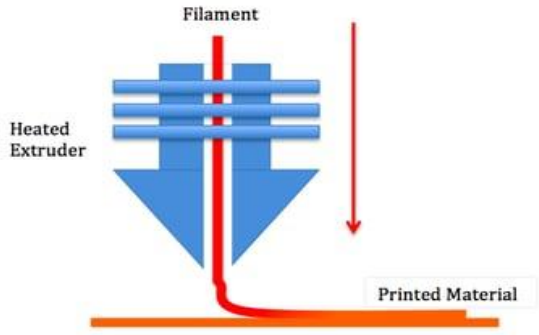

Figure 3. FDM 3D printing (Grames, 2022)

2.5 Types of FDM printers

There are four different types of FDM 3D printers: Cartesian, Delta, Polar, and robotic arm. Robotic arm printers are mainly used for extremely large printing applications such as printing rocket parts or printing houses. The most popular types on the customer level are cartesian and delta printers. (M., 2022)

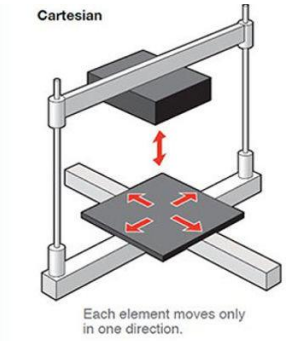

Figure 4. Cartesian printer (M., 2022)

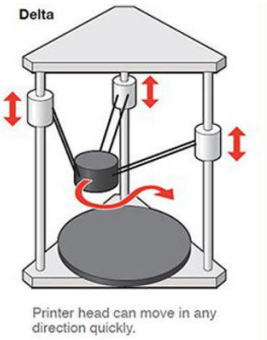

Figure 5. Delta printer (M., 2022)

Cartesian printers are the most common type of 3D printer on the market. It is based on the cartesian coordinate system which uses three-axis (X, Y, Z) to position the printhead. This system allows for bigger prints and excels in horizontal prints.

Delta printers also use the cartesian coordinate system but instead of using linear motions, they use three arms to move the printhead around. Although the build size is limited, it excels in big vertical prints. (M., 2022)

It is important to choose the correct type of printer for your needs. This depends on different factors (Tractus3D, 2022):

- Specifications
    - Max extruder(s) temperature and enclosure

      The extruder must be able to print the materials you want. Some materials require a temperature that is a lot higher than others. This means that not all printers are equipped with the correct hardware to print these materials. The higher the temperature goes, the greater the need for an enclosure. This keeps the heat from going away from the print and provides better results.
    - Resolution

      The quality of the output is affected by the thickness of the layers. The higher the resolution, the thinner the layers are. Thinner layers mean better quality. For example, the standard resolution for high-quality prints is between 50 and 100 microns.
    - Speed

      Travel and printing speed are the two different speeds that should be considered. Travel speed is the speed at which the printhead moves between points whilst it is not printing. Printing speed however is the speed at which each layer is printed. Changing the speed of the printer will drastically influence the quality of the result. Prints with a lot of detail or small tolerances should be printed slower and will take more time.
    - Build volume

      It is essential to consider both the size of your designs and whether or not you will be 3D printing in batches of items. These two factors affect the build volume you need.
- Customer service

  Every 3D printer can experience issues due to malfunctions or parts that get deteriorated over time. That is why it is important to invest in a company width great support and warranty terms.

2.6 Pros & cons FDM printing

As with many technologies, there are always pros and cons to the system. These include (Grames, 2022):

Pro's	Cons
Easy to scale to any size	Part quality
Great cost-to-size ratio	Details = lots of post-processing
Variety of material choice	'Weak' parts due to layer adhesion
Creates objects relatively fast	Needs a lot of maintenance & calibration

2.7 Slicers

Slicing is the process of turning 3D models into specific instructions. This is a necessary step in the 3D printing process since 3D printers cannot translate CAD drawings by themselves. These instructions are called G-code. There are numerous slicers available, some are free, and some are very expensive.

G-code is easy to write but since the layer height of the print is very small, and the shapes of the models can be quite complex, the number of G-code instructions can add up to thousands of instructions. That is why slicing software is used to generate the G-code commands. This software divides the model into multiple layers, determined by the layer height (Figure 6). After this, it creates G-code commands for the linear movements of the extruder. These are all added into one file which the printer uses to print the object

(Carolo, 2022).

Figure 6. Slicing process (Spiegel, 2022)

Currently, the most used file format in 3D printing is STL files. As this file format only recreates the surface of the 3D model, an infill is added whilst printing to improve the strength and quality of the print (Figure 7).

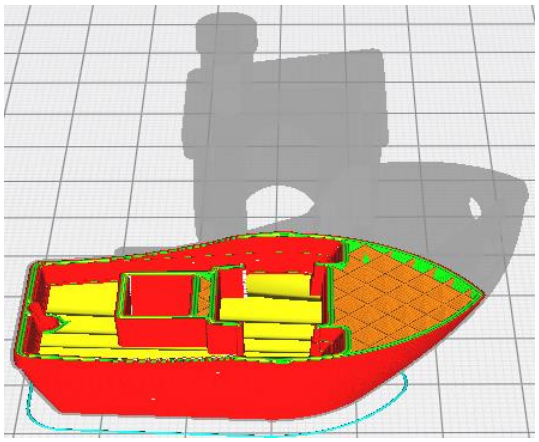

Figure 7. Sliced object

Unfortunately, STL files have their limitations, the fact that they only understand the surface of the model is one of them. Another limitation is the lack of support for manifold edges. Also, the information about the model such as the colour, texture, … is not included in the file. Finally, the quality of the STL files depends on multiple settings when saving the file. The angle control, chordal deviation, or the way the information is stored (binary or ASCII) all impact the quality of the model. (Chen, 2022)

These limitations all affect the slicing of the object. Which indirectly affects the outcome of the print.

3 Industrial robot arms

3.1 Usage

Robotic arms are machines that are programmed to execute specific tasks quickly, more efficiently, and with extreme precision. Therefore, they are very popular in industrial production, assembly lines, and machine tooling, … They are usually used to carry out fast, consistent, heavy, or repetitive procedures. Typically, an industrial robot arm has a series of joints controlled by an electric motor. These joints then work together to perform as many movements as possible, mimicking (or improving) the movements of humans.

(RS, 2022)

Current advances in technology have greatly reduced the cost of robotic arms. Thus, they are now more available to smaller companies or even schools. Of course, they are still far too expensive for personal use.

Companies could buy robot arms from €50,000 to €250,000 and it would still be cheaper than paying a human worker:

Estimated gross income employee: 2,500 euro/ month

250,000 / 2,500 = 100 months

100 / 12 = ~ 8 years & 4 months.

So, in just over eight years, the robotic arm has already paid for itself. This is roughly calculated with one employee while the robot arm can work 24/7. Since the employee will only be working eight hours a day, we can again divide this period by three. This gives us a total of two years and seven months that the company must pay off before it can make a pure profit in the coming years. So, it is a good deal.

When the right kind of robot is chosen for a particular task, the possibilities are endless. This mechanism, combined with computer vision or simultaneous localization and mapping, can achieve anything.

3.2 Types of robot arms

There are numerous kinds of robot arms. Every robot arm has its own purpose and excels at its own application.

3.2.1 Cartesian/ gantry robots

This robot (Figure 8) moves in a two-dimensional plane and uses the Cartesian coordinate system. This means that the head is moved into the correct position (coordinate) by two (or more) motors that determine the position on the X and Y-axis. (Brookins, 2022)

Figure 8. Gantry robot (Collins, 2022)

Because they are easy to configure, they are used in many applications such as

- pick & place
- milling machines
- sorting
- labelling

(Brookins, 2022)

There are two types of designs (Figure 9). Each design has its perks, and the application differs in the function and the load that the robot arm has. Since the Y-axis is supported by the X-axis, gantry robots can support larger payloads. (Brookins, 2022)

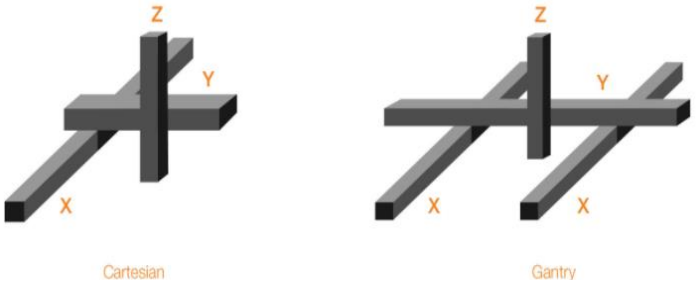

Figure 9. Cartesian vs gantry designs (Brookins,

3.2.2 Articulated robotic arms

This is probably the most popular and recognizable type of robot arm (Figure 10). These are fixed to a base with a rotating joint. Typically, it is made with 4 to 6 joints, but this number can go up to 10.

This type of robot arm is ideal for complex movements as it moves more like human arms. The reach and capacity of the arm are determined during its development. These robots are more difficult to control and cost more because they have more expensive parts than other types of robot arms such as the Cartesian robot arm. (Brookins, 2022)

Figure 10. Articulated robotic arm (D.S.C., 2022)

This type of robotic arm is used in the following applications:

- arc welding
- assembly
- packaging
- palletizing
- …

3.2.3 Scara

Scara stands for ‘selective compliance assembly robot arm’ or ‘selective compliance articulated robot arm’. This robot arm has three axes (X, Y, Z) to perform basic movements (Figure 11). It is ideal for vertical assembly lines, biomedical applications, and palletizing.

This machine is very similar to the robotic arm. Here too, the range and capacity are determined during the development of the arm. They can also perform complex tasks but are limited by their three axes. In addition to performing these complex tasks, they may be more expensive due to their expensive features. A plus side to this robot is that they take up less space and weigh less than other types of robots. (Brookins, 2022)

Figure 11. Scara robot (Brookins, 2022)

3.2.4 Delta

Delta robots or parallel/ spider robots use three arms to move a toolhead around. Because of their small work area and low payload capacity compared to other robots, the base of the robot has to be mounted above the working area. This results in the ability to move at high speeds (~ 200 picks per minute) which makes it perfect for lightweight pick and place tasks. (Brookins, 2022)

Figure 12. Delta robots (FANUC, 2022)

3.3 Choosing the correct robot arm

It is important to link the correct robot arm to the desired application. After all, a company would not like to pay tens of thousands of euros too much for a robot arm that is overkill or buy one that breaks down after a month. Here are some elements that influence this choice:

- Load
  The maximum load capacity has to be larger than the total weight of the payload at the end of the arm. Sensors, grippers, or any other tool must be added to this. If this total weight is too close to the maximum load capacity, the arm may lose its precision.

- Orientation
  Is the working area limited? Is the robot stationary on the ground, on the ceiling, ...

- Speed & travel
  How fast does the robot need to move? Cartesian robots, for example, have acceleration up to 5 m/sec. This is much larger than a robotic arm or scara robot.

- Precision
  Some robots require more precision than others. In some cases, this can be changed after purchasing the robot by installing different actuators.

- Environment
  Is there dust or dirt around the motor? Then everything needs to be properly sealed so that dust cannot get in.

- Duty cycle
  How long does the robot need to work? If it must work 24/7, the robot will not last as long, ...

(Brookins, 2022)

3.4 Axis of Articulated robotic arms

The articulated robotic arm is a suitable candidate to harvest the print beds. This robot is able to execute the complex movements that are necessary to lift the print bed off the printer and put another one in the correct spot. Every axis has its name and function. This can be seen here (Figure 13).

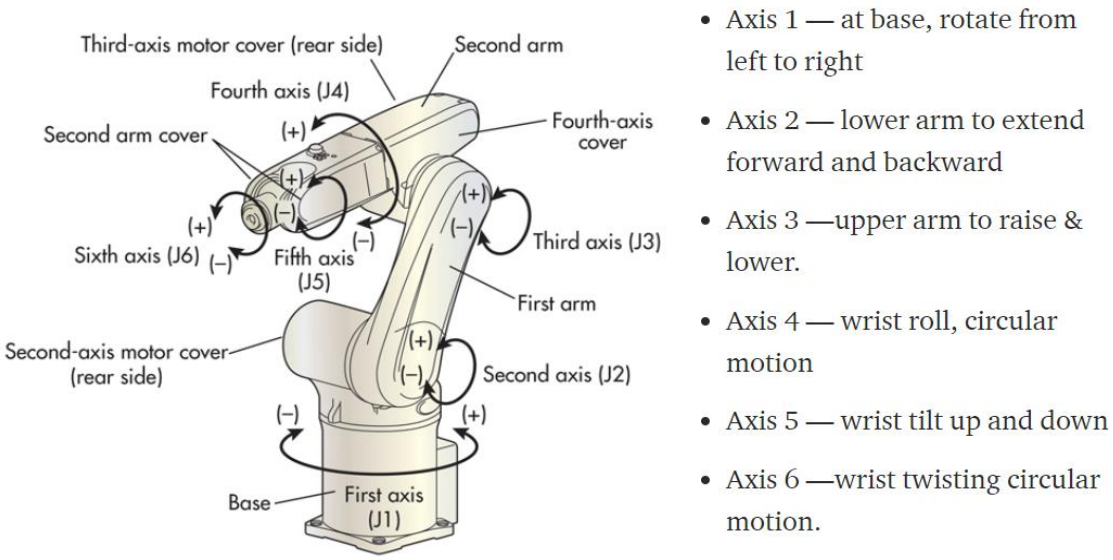

Figure 13. Axes robotic arm (adapted from Bland, 2022)

3.5 Forward/ Inverse kinematics

Both forward and inverse kinematics can be used to control the movement of a robotic arm. Forward kinematics is mostly used for small, simple projects where you can program each joint manually. This takes the joint angles as input and calculates the (Cartesian) position and orientation of the end-effector (gripper/ nozzle/ …).

For robots, such as the articulated robotic arm, it is a lot easier than using inverse kinematics because inverse problems are usually much harder to solve. However, if you use it in a parallel arm (Delta robot), the difficulty is reversed.

Unfortunately, in forward kinematics, programming movements can be a lot of work and is not sufficient for controlling robotic arms. That is where inverse kinematics save the day.

Calculating inverse kinematics for robotic arms is hard. This is mainly because sometimes, there are multiple solutions to the equation and sometimes there are no solutions. Instead of taking the joint angles as an input like forward kinematics, it receives the cartesian position and orientation of the end effector as input and calculates the joint angles. (Elre, 2022)

3.6 Calculating inverse kinematics

Calculating inverse kinematics can be tricky. It is a lot more difficult than forward kinematics but ends up being a lot more efficient and easier to use in the long term. That is why it is used in most robotic applications. As mentioned before, in inverse kinematics, the angles are calculated from the coordinates of the end-effectors. The calculation goes like this:

Granted (Figure 14): t, b, a, p, E

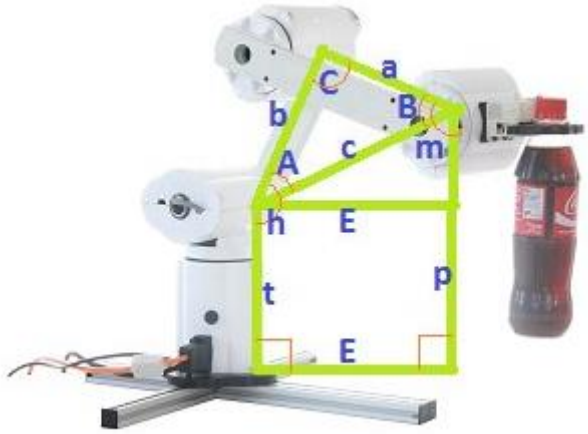

Figure 14. Calculation inverse kinematics (Elre, 2022)

$$c = \sqrt{E^2 + (p - t)^2}$$

$$C = cos^{-1}(\frac{a^2 + b^2 + c^2}{2ab})$$

$$\frac{a}{sinA} = \frac{b}{sinB} = \frac{c}{sinC}$$

$$A = sin^{-1}(\frac{a}{c} * sinC)$$

$$B = sin^{-1}(\frac{b}{c} * sinC)$$

$$m = sin^{-1}\left(\frac{E}{c}\right) + sin^{-1}\left(\frac{b}{c} * sinC\right)$$

$$h = cos^{-1}\left(\frac{E}{c}\right) + sin^{-1}\left(\frac{a}{c} * sinC\right) + 90°$$

(Elre, 2022)

4 Automating 3D printing processes

4.1 Problems of running 3D printer farms

Having to run a company with a large amount of 3D printers is a very time-consuming job. Unfortunately, there still is a large amount of human interference which leads to the 3D printer not working at its full potential. Because of this, a company could miss out on a lot of income and would have to pay too many expenses such as the unnecessary jobs that are required to keep the printers running.

There are two steps that can be automated such as

- Client-side communication
- Managing the print farm

4.2 Automating client-side communication

When ordering a print, there is usually a lot of unnecessary contact between the customer and producer before the printing can start. The customer must contact the producer to share the model and the required settings to ensure the print ends up being what the client wants. Depending on the knowledge of the customer, this process could add up to an estimated total of 3 to 5 emails. After that, the producer must present the bill and the client must partially/ fully pay for his order. Only after the order was paid for, can the producer start printing. This process takes a lot of time, and this is not well-received on both sides. That is why every company should invest in a system that eliminates this part of the job. Clients should be able to order their parts on a website and immediately pay for them with the correct tool.

This tool would have the following steps:

- Upload the model
- Select a few settings for the slicer
- Send the file to a slicer via an API which will respond with the estimated print time and price, …
- After this, the client can either change a few settings or pay for the order
- The order arrives at the producer, and they can start printing without any effort.

4.3 Automating print farm management

Multiple people are needed to run a 3D printer farm. These people are constantly harvesting printers and starting new prints. This process can be very time-consuming and depending on the size of the print farm, it can require multiple people to maintain. This makes it an expensive option.

Another problem with having employees taking care of the printing farm is that they are limited to the number of hours they work every day. This can be compensated by having the longer prints run overnight but even that does not always assure the optimal result.

The solution to this problem can be divided into two main parts:

- Remotely controlling 3D printers
- Automatically harvesting the 3D printers

By remotely controlling the 3D printers, employees no longer have to hassle with SD cards anymore. These cards contain the G-code files that the printer needs. Every time you would want to print something, you would have to take the card and get it to the computer that will use the slicer to obtain the G-code file. A master application that can divide all files between printers and a slave application to control each printer would be the perfect solution to this problem. In addition to this, the master application could also automatically schedule the prints in the queue. This could either be first come first served or a metaheuristic algorithm that schedules smaller prints during the day and the larger prints during the night to increase production during the day. Cameras can also be added to detect print failures using image recognition which would save both time and materials.

The solution mentioned above works perfectly to control the printers for one time since the software cannot harvest the physical prints of the build plates. That is why there also has to be something to harvest the build plates to ensure the printers can work without any delay. This could be done with robotic arms that exchange the used build plates with empty ones.

4.4 Applications

Automating the full process of 3D printing is especially attractive to schools/ companies with multiple 3D printers. This could go from a small number of printers like 5 to a bigger number like 20, 50, 100 or more. The more printers you have, the more interesting it is. One big problem is the budget of the user. Although not all parts of the automatization would be expensive, the price of a good robotic arm is quite high depending on which one you use.

5 Case: 3D printer automatization

5.1 Introduction

As previously mentioned in chapter 4.3, there are two sides to automizing the 3D printing process: controlling and harvesting the 3D printer(s). A robotic arm can be used to harvest the 3D printers' build plates. By using a personalised gripper, it could be able to take the used print bed off the printer and replace it with an empty bed. That way, the printer can start a new print without the need of cooling down or heat up first. This will automatically save a lot of time. To be able to control when and which printer has to be harvested, the printer cannot just work with SD cards (what most printers work with). That is why an application is required to control the printer. This application would have to constantly take care of instructions that will control every movement or action the printer takes. Both sides of this atomization process rely on one main factor: communication. This is the main bottleneck in the project. If a valid connection between the application and both the robotic arm and the printer can be established, it would be enough to prove that this system would work. This case only aims to control one printer. This means that there is no need for the master application discussed in chapter 4.3. As can be seen in the scheme below (Figure 15) the case will focus on creating the slave application which will control the 3D printer and the robotic arm.

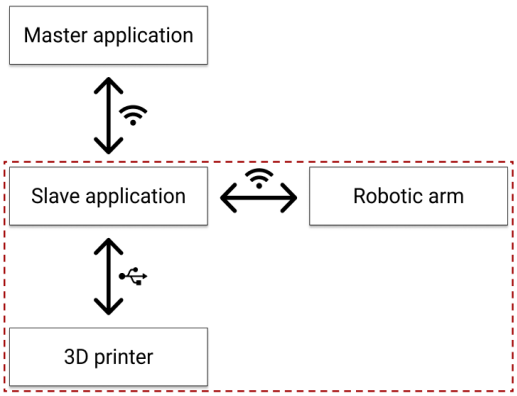

Figure 15. Case scheme

The printer that will be used in this case is a Prusa MK3S from the company Prusa Research. It is an ideal printer because of its open design which allows the robotic arm to grab the print bed without the need of opening a chamber first. The printer will be mounted near a robotic arm. This arm is the UR5 from Universal robots. The UR5 is well suited for the application because of its size, power and number of axes.

Code fragments are added in the case explanation because the proof of concept is to create the communications and show how these will work and how they can be implemented.

5.2 Controlling a 3D printer with a Java application

The first part of the case is controlling a 3D printer. Here, human interaction is needed to guide the program. The following interactions will be included:

- Connecting to the printer (This includes selecting the correct COM ports of the serial connection)
- Starting a print
- Pausing or resuming a print
- Stopping a print
- Harvesting build plate
- Quitting the application

In a finished product, these interactions will no longer require human interference since the commands will be sent from the master application instead. But in this case, it has to be added to preserve the good functionality of the application since the slave will not know what to do without it.

Whether or not it is the user or the main application that sends these commands, they always have to be available. This means that the program cannot be doing something else when a command is chosen. That is why the main thread of the program constantly asks for an action. The other tasks have to be run in another thread. Because this is the slave application, there is no need for a UI. That is why this is purely a console application.

As can be seen here (Figure 17), the main thread runs a loop in which the application is expecting a command to execute. The list of all available COM ports is requested directly when starting the application because it takes a few seconds before these are available. This way, the program does not have to wait every time a connection is being made with the printer. The result of this code can be seen here (Figure 16).

```
Starting application...
Select which action you want to execute:
0. Connect to printer
1. Start printing
2. Pause/ resume printing
3. Stop printing
4. Harvest print bed
5. Exit application

Input:
```

Figure 16. Start menu application

```java
public class Main_App {
    static boolean printerOccupied = false;
    static boolean connectionInitialised = false;
    static Connection connection = new Connection();
    static Printer printer = new Printer();
    static Universal_Connection universal_connection = new Universal_Connection();
    public static void main(String[] args) throws InterruptedException, IOException {
        System.out.println("Starting application...");
        Scanner sc= new Scanner(System.in);
        // Get a list of the com ports
        SerialPort[] ports = SerialPort.getCommPorts();
        PrintMenu();
        int input = 0;
        while(input != 5){
            // Wait for commands
            if(sc.hasNext()){
                input = sc.nextInt();
                switch (input){
                    case 0:
                        // Connect to the printer
                        break;
                    case 1:
                        // Start printing
                        break;
                    case 2:
                        // Pause or resume printing
                        break;
                    case 3:
                        // Stop printing
                        break;
                    case 4:
                        // Harvest build plate
                        break;
                    case 5:
                        // Quit application
                        break;
                    default:
                        System.out.println("Given input is not an option.");
                        break;
                }
                PrintMenu();
            }
        }
    }
```

Figure 17. Main thread application

5.2.1 Initialising a connection with the printer

Before a connection can be established with the printer, the correct COM port has to be selected. This part is located in the main thread (Figure 18). Since the ports have already been requested at the start-up of the program, the delay that comes with it is eliminated.

When choosing the option to connect with the printer, the ports are printed, and the program waits until the user selects the correct port. After a validation to make sure the port is not wrong, a new connection object is created which takes the COM port as a parameter. After this, a new thread is started. In this thread, the connection itself will be made.

```
23      case 0:
24          System.out.println("\nSelect the correct COM port and connect to the printer!");
25          System.out.println("Available COM ports: ");
26          for (int i = 0; i < ports.length; i++){
27              System.out.println(i + ". " + ports[i]);
28          }
29          System.out.print("\nCOM port or quit (q): ");
30          while (!connectionInitialised){
31              if(sc.hasNext()){
32                  String val = sc.nextLine();
33                  if(!val.equals("")){
34                      if (val.equals("q")) {
35                          System.out.println("\nStopped.\n");
36                          break;
37                      }
38                      // Validating given input
39                      try{
40                          int portNumber = Integer.parseInt(val);
41                          if(portNumber < ports.length){
42                              System.out.println("\nConnecting to the printer...\n");
43                              // When the input is correct, start a new thread
44                              connection = new Connection(ports[portNumber]);
45                              Thread connect = new Thread(connection);
46                              connect.start();
47                              connectionInitialised = true;
48                          }else{
49                              throw new NumberFormatException();
50                          }
51                      }catch (NumberFormatException e){
52                          System.out.println("Given input is not an option.");
53                          System.out.print("COM port or quit (q): ");
54                      }
55                  }
56              }
57          }
58          break;
```

Figure 18. Starting the connection thread

In the connection class, the actual connection with the printer is made. Because this part is separate from the code that sends the commands to the printer, a connection only has to be made once. This way, prints can be started again and again without the need of reconnecting to the printer. The connection is made using the 'jSerialComm' library. This is a library that allows the user to use the serial port.

The 3D printer has a baud rate of 115200, eight data bits, one stop bit and no parity. These parameters must be equal to each other to sustain a good connection without data loss. After the connection is made, the 3D printer restarts and sends a couple of lines to indicate the process (Figure 19). The thread waits in a while loop until all these lines are received to make sure the printer has started correctly.

```java
import com.fazecast.jSerialComm.*;
import java.util.Arrays;

public class Connection implements Runnable {
    public SerialPort comPort;
    public boolean connected = false;
    public Connection(){}
    public Connection(SerialPort port) { this.comPort = port; }
    @Override
    public void run() {
        comPort.setComPortParameters( newBaudRate: 115200, newDataBits: 8, newStopBits: 1, newParity: 0);
        comPort.openPort();
        try{
            while(!connected){
                while(comPort.bytesAvailable() == 0){
                    Thread.sleep( millis: 20);
                }
                // Reading input send by the printer
                String input = "";
                byte[] readBuffer = new byte[comPort.bytesAvailable()];
                int numRead = comPort.readBytes(readBuffer, readBuffer.length);
                input = new String(Arrays.copyOfRange(readBuffer, from: 0, to: readBuffer.length-1));
                //System.out.println("Read: " + input);
                if(input.endsWith("DISABLED")){
                    this.connected = true;
                }
            }
        }catch (Exception e){
            e.printStackTrace();
            comPort.closePort();
        }
    }
}
```

Figure 19. Initialising a connection with the 3D printer

5.2.2 Starting a print on the application

Before starting the printer thread, the program checks whether or not it is connected to a printer, if the printer is printing something already or if it is currently still connecting. If it is the last of these options, it will wait until it is connected after which it will start the printer thread (Figure 20).

```
59          case 1:
60              if(printerOccupied){
61                  System.out.println("\nAlready printing something!\n");
62                  break;
63              } else if(!connection.connected && !connectionInitialised){
64                  System.out.println("\nNot connected to a printer yet. \nExecute step 0 first.\n");
65                  break;
66              }else if(!connection.connected && connectionInitialised){
67                  System.out.print("Still connecting to printer");
68                  while(connection.connected==false){
69                      Thread.sleep( millis: 500);
70                      System.out.print(".");
71                  }
72                  System.out.println("\nConnected!\n");
73              }
74              System.out.println("\nStarting printing process!\n");
75              printer = new Printer(connection.comPort);
76              Thread print = new Thread(printer);
77              print.start();
78              printerOccupied = true;
79              break;
```

Figure 20. Starting the printing thread

Every interaction with the 3D printer happens in the ‘Printer’ class. As said earlier, this class may only work in another thread. This means that the class must implement the Runnable interface. With this interface, the ‘run’ method can be overwritten with the actual code that has to be executed.

As can be seen in Figure 21, there are a lot of methods regarding multithreading. The function of these methods is to pause, resume or stop the thread and with this also the printing process.

```
1  import com.fazecast.jSerialComm.*;
2  import java.io.*;
3  import java.io.PrintWriter;
4  import java.util.Arrays;
5  import java.util.Scanner;
6
7  public class Printer implements Runnable {
8      private SerialPort comPort;
9      private boolean stop_requested = false;
10     private boolean pause_requested = false;
11     public boolean finished = false;
12     public Printer(){}
13     public Printer(SerialPort port) { this.comPort = port; }
16     public void request_stop() { this.stop_requested = true; }
19     public void request_pause() { this.pause_requested = true; }
22     public synchronized boolean isPause_requested() { return pause_requested; }
25
26     public void request_start() { this.pause_requested = false; }
29
30     public synchronized boolean isStop_requested() { return this.stop_requested; }
33
34     private void check_pause() throws InterruptedException {
35         while(this.isPause_requested()){
36             Thread.sleep( millis: 100);
37         }
38     }
```

Figure 21. Multithreading interaction methods

When the thread is started, the printer can start printing. In order to print with the application, a G-code file is required. This file is then read line per line with an input stream (Figure 22). After slicing, the file can include comments or unnecessary backspaces which are not required for the printer to execute commands. Sending this along with the commands only increases the risk of data loss. So to increase the accuracy of the data transfers, these parts are filtered out. The simplified G-code commands are then sent to the printer. After this, the program waits on the reply. The software of the 3D printer usually replies with ‘ok’ after a command is executed. It is however possible that, due to data loss, the program only receives ‘o’ or ‘k’ which is why these chars also approve sending the next command.

```
@Override
public void run(){
    PrintWriter out = new PrintWriter(comPort.getOutputStream(), autoFlush: true);
    try
    {
        // The file to be opened for reading
        FileInputStream fis = new FileInputStream( name: "C:\\Users\\Thiba\\SchoolProjecten\\Thesis\\G-code Files\\3DBenchy.gcode");
        Scanner sc = new Scanner(fis);    // File to be scanned
        // Returns true if there is another line to read
        int i = 0;
        finished = false;
        while(sc.hasNextLine() && !isStop_requested())
        {
            check_pause();
            String line = sc.nextLine();
            if(!line.startsWith(";") && !line.equals("")){
                boolean executed = false;
                if(line.contains(";")){
                    line = line.substring(0,line.indexOf(';')-1);
                }
                //System.out.println("Command: " + line + " was sent.");      //returns the line that was skipped
                out.println(line);
                Thread.sleep( millis: 20);
                while(!executed && !isStop_requested()){
                    check_pause();
                    while(comPort.bytesAvailable() == 0){
                        Thread.sleep( millis: 10);
                        check_pause();
                    }
                    String input = "";
                    byte[] readBuffer = new byte[comPort.bytesAvailable()];
                    int numRead = comPort.readBytes(readBuffer, readBuffer.length);
                    input = new String(Arrays.copyOfRange(readBuffer, from: 0, to: readBuffer.length-1));
                    //System.out.println("Read: " + input);
                    if(input.endsWith("ok") || input.equals("ok") || ( input.length() <= 3 && (input.contains("o") || input.contains("k"))))
                        executed = true;
                }
                i++;
                Thread.sleep( millis: 20);
            }
        }
        sc.close();     // Closes the scanner
        finished = true;
        //System.out.println("Print finished.\n");
    }
    catch(IOException e)
    {
        e.printStackTrace();
    }
```

Figure 22. Serial communication with printer

5.2.3 Pause, resume or stop printing

This is where the methods in the multithreading part of the serial communication with the printers are used. The pause method keeps the thread busy until it is resumed or terminated. Before doing this, the program has to check whether or not the printer is printing.

```
80      case 2:
81          if (!printerOccupied){
82              System.out.println("No prints started yet...\n");
83              break;
84          }
85          if(printer.isPause_requested())
86              printer.request_start();
87          else
88              printer.request_pause();
89          break;
90      case 3:
91          if (!printerOccupied){
92              System.out.println("No prints started yet...\n");
93              break;
94          }
95          printer.request_stop();
96          printerOccupied = false;
97          break;
```

Figure 23. Pause, resume and stop printing

5.2.4 Serial connection with the UR-5

Finally, the application also needs to connect to the UR5. How this works will be explained in the next chapter. This part of the application should be implemented in the master application that will divide all prints between the slaves. This way, only one connection has to be made to the robotic arm and a queue can be used to determine which printer will be harvested first. Since the case only focuses on the slave application, the connection is also made here.

Before the harvesting process starts, the program checks whether or not the print bed is empty or if the printer is still printing (Figure 24). Harvesting the build plate is not allowed in either one of those scenarios. If all is good, a new thread is started, and the connection is created. Since the application cannot execute anything while the bed is being changed, it will wait until the process is finished.

```
98                  case 4:
99                      if (!printerOccupied){
100                         System.out.println("No prints started yet...\n");
101                         break;
102                     }else if(!printer.finished){
103                         System.out.println("\nHarvesting is not allowed while the printer is printing.\n");
104                         break;
105                     }
106                     universal_connection = new Universal_Connection( ip: "10.100.100.16", port: 29999);
107                     Thread harvest = new Thread(universal_connection);
108                     harvest.start();
109                     while (!universal_connection.harvested){
110                         Thread.sleep( millis: 500);
111                     }
112                     System.out.println("\nHarvesting cycle completed.\n");
113                     printerOccupied = false;
114                     break;
```

Figure 24. Starting the harvesting process

Creating a connection with the UR5 can be done with the java.io and java.net libraries. Using these libraries, a socket can be created that connects to the dashboard server of the UR5 (Figure 25). The printwriter and bufferedreader are used to communicate over this socket. These are implemented in the ‘sendMessage’ method.

Finally, there is also a disconnect method that will disconnect all services.

```java
import java.io.*;
import java.net.*;

public class Universal_Connection implements Runnable {
    private Socket client;
    private PrintWriter out;
    private BufferedReader in;
    public boolean harvested = false;
    private String ip;
    private int port;

    public Universal_Connection(){}
    public Universal_Connection(String ip, int port){
        this.ip = ip;
        this.port = port;
    }

    private void startConnection() throws IOException {
        client = new Socket(this.ip, this.port);
        out = new PrintWriter(client.getOutputStream(), autoFlush: true);
        in = new BufferedReader(new InputStreamReader(client.getInputStream()));
    }

    private String sendMessage(String message) throws IOException {
        out.println(message);
        String response = in.readLine();
        return response;
    }

    private void Disconnect() throws IOException {
        out.close();
        in.close();
        client.close();
    }
```

Figure 25. Connection methods UR5

All the pieces fall together in the run method (Figure 26). Here the connection is initialised after which the program waits until it is made. If a valid connection is established, commands can be sent to the printer.

The first command ensures that the program is loaded. If this program is loaded, it gets started immediately. A delay of 500ms is followed to make sure the ‘running’ command has been updated within the UR5 programming. If this delay is not added, it will return false which will end the process before it had even started.

Finally, polling is used to check whether or not the robotic arm has finished the harvesting process. Here, the program sends the command ‘running’ every two seconds. This will return true if the robot is still harvesting.

After this, the program will disconnect from the UR5, and a new print can be started.

```
36      @Override
37      public void run() {
38          try{
39              this.harvested = false;
40              startConnection();
41              System.out.print("Connecting to UR5");
42              while(!in.readLine().contains("Connected: Universal Robots Dashboard Server")){
43                  System.out.print(".");
44                  Thread.sleep( millis: 250);
45              };
46              System.out.print("\n");
47              System.out.println(sendMessage("load /programs/remotetest.urp"));
48              System.out.println(sendMessage("play"));
49              Thread.sleep( millis: 500);
50
51              String a;
52              System.out.print("Harvesting build plate");
53              while((a = sendMessage("running")).contains("true"))
54              {
55                  System.out.print(".");
56                  Thread.sleep( millis: 2000);
57              }
58              System.out.print("\n");
59              Disconnect();
60              this.harvested = true;
61          }catch (Exception e){
62              System.out.println("Harvest failed.");
63          }
64      }
65  }
```

Figure 26. Executing the harvesting process

5.3 Controlling the UR5

The UR5 (Figure 27), developed by the company Universal Robots is an industrial robotic arm that is designed to do repetitive tasks with a maximum payload of up to 5 kg with very good precision (WiredWorkers, 2022). This six-axis robot arm is ideal for tasks such as packing, assembly or testing. Because of his six axes, it is a perfect fit for this case. The robot has a maximum radius of 850 mm which does not make this an ideal robot for bigger print farms. Either way, this does not pose a problem for this case because only one printer will be used here.

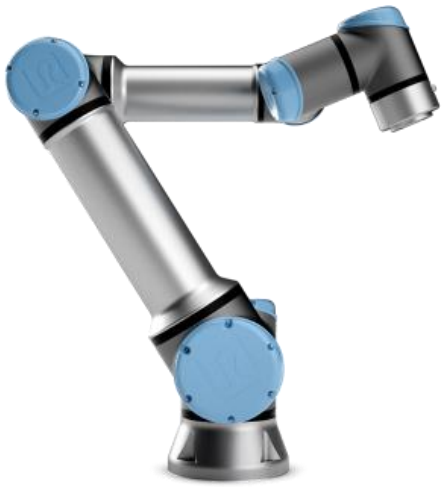

Figure 27. UR5 (Universal Robots, 2022)

5.3.1 Hardware

To harvest the printer, a personalised gripper is needed. This gripper has to be able to hold the build plate as good as possible while it is moving.

The gripper that is used in this project is the ‘Robotiq Two Finger Adaptive Gripper’. It has a grip force of 10 to 125 N which will be needed to firmly grip the print bed in order to be able to lift it off the surface.

As can be seen in the image below (Figure 29), this gripper is used with two 3D printed parts with embedded rubber in order to have a better grip on the build plate. Not only the gripper has to be personalised, but also the build plate.

The build plate needs to have a place where the robot can hold it. In order to be able to do this, it needs an extra place in the front of the build plate. This is displayed in Figure 29.

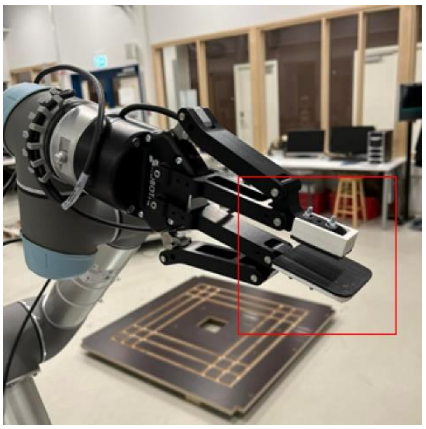

Figure 29. Modified gripper

Figure 29. Ideal layout build plate

5.3.2 Software

The UR5 comes with a built-in display that can be used to control the robot arm, and create programs, … This makes programming the robot very easy which implies that the robot can start working relatively fast.

The following commands will be used in the program:

- Move

  This command allows the robot to move. Here, acceleration and speed can also be defined. Multiple waypoints can be added to specify a certain path that should be followed while executing the move action. These waypoints describe the location and orientation inside the Cartesian coordinate system.

- Gripper Setup

  At the start-up of the program, the gripper has to be initialised. After this, the gripper can be used in action. You can eighter control how far it opens, with what speed it operates or with how much torque it closes.

(UNSW Making Sydney, 2022)

5.3.3 Harvesting a 3D printer

Printer specifications

Not every printer is able to use a system that can harvest the print bed. The printer itself also needs to fulfil some requirements such as:

- The print bed must have space for the robot arm to grab it strong.

  See 5.3.1

- The magnetic pull cannot be too strong.

  The magnetic beds cannot stick too hard to the print bed otherwise the stepper motors of the printer cannot hold the bed in place, and the printer would also be lifted, …

  An ideal print bed for a system like this would be either with a small amount of magnetic pull or with electrical magnets that can be turned on or off.

- Preferably an open chamber.

  Although this specification is not mandatory, it certainly makes things a lot easier. If the printer does not have an open chamber (Figure 31) the doors that access the printer have to be opened first. This will make the system more complicated. It can be done with servo motors that are connected to the raspberry pi that controls the printer.

  It would be easier with printers that are not enclosed (Figure 31). Although an enclosure is good for a printer, the printers can be in a room that is completely sealed off from any drafts which eliminates the need for the enclosure. This way, the robot arm can reach the bed without any problems.

Figure 31. Creality CR-5 Pro
(3D-CADSOLUTIONS, 2022)

Figure 31. Prusa MK3S
(Prusa Research, 2022)

Executing programs on demand

The UR5 is equipped with a dashboard server that allows the user to remotely control the robot arm. This is perfect for applications in industrial environments where they control all their robotic applications from one room. This also allows this case to achieve a fully automated harvesting process. As shown earlier in chapter 5.2.4, the application will connect via a TCP/IP socket and will then use certain commands to start the correct harvesting process. A full list of these commands can be found in **Fout! Verwijzingsbron niet gevonden.**.

In a normal application, the robotic arm would be connected to a network but this wasn't possible on the campus due to security reasons. That is why the arm is directly connected to the device on which the application is running. To achieve a valid connection, the IP address of this device needs to be in the same range as the one of the robotic arm. This can be seen in Figure 33. If these IP addresses are valid, the connection can be created.

To test this functionality, the open-source software 'SocketTest' is used. The client connects to the dashboard server on port 29999 after which it will be able to send commands to the UR5. As can be seen in Figure 33, the robot can load a program, start a program and tell you whether or not this is still running or not. By polling the 'running' command, the program can know when the robot is done harvesting (Figure 33).

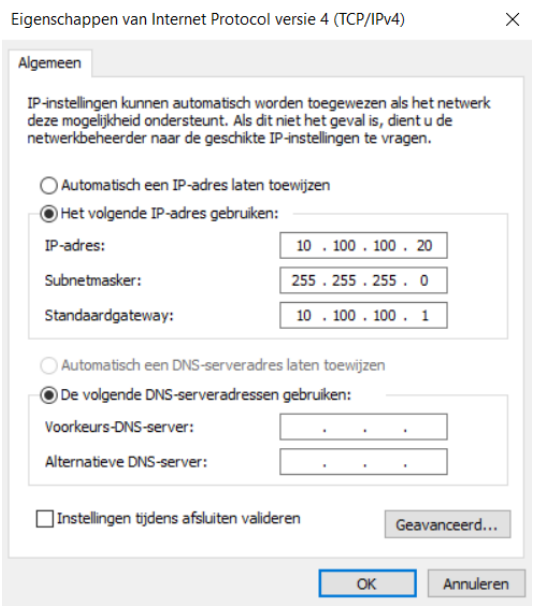

Figure 33. Setup TCP/IPv4

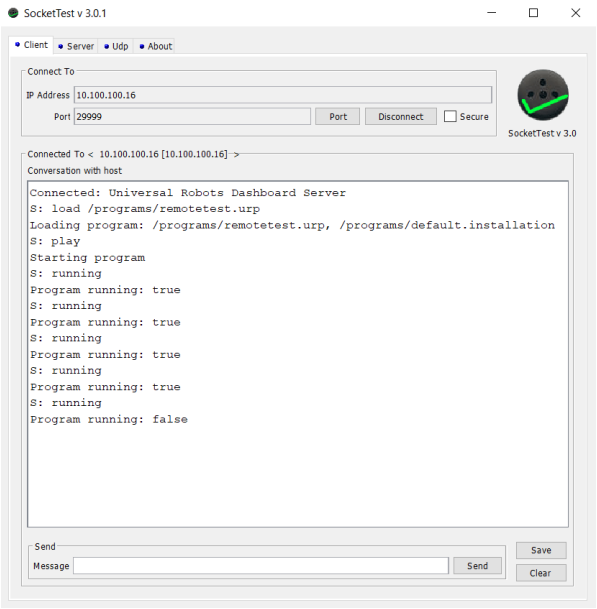

Figure 33. Functionality commands

Programming the UR5 for harvesting print bed

The program structure of the UR5 can be seen in Figure 34 and Figure 35. As earlier stated in chapter 5.3.2, the majority of the commands are controlling the position and the gripper of the UR5. These commands follow the following path:

1. Home position
2. Quick move to a safe place in front of the print bed
3. Slow move to the holding position
4. Close gripper
5. Slowly go up and take the bed off the printer
6. Quickly move to the place where the harvested beds go
7. Slowly put the bed in the correct position
8. Open gripper
9. Slowly move away from the print bed
10. Home position
11. Quick move to a safe place in front of the new print bed
12. Slowly go to the precise location
13. Close gripper
14. Slowly take the new bed up
15. Quickly move to a safe place in front of the print bed
16. Slowly place the new bed on the printer
17. Slowly move away from the print bed
18. Home position

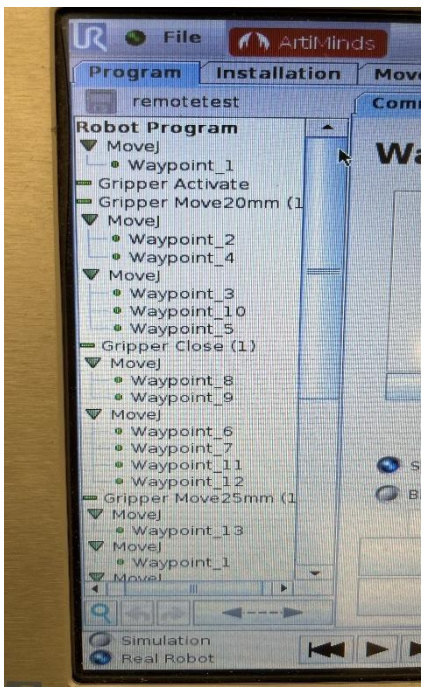

Figure 35. UR5 program structure part 1

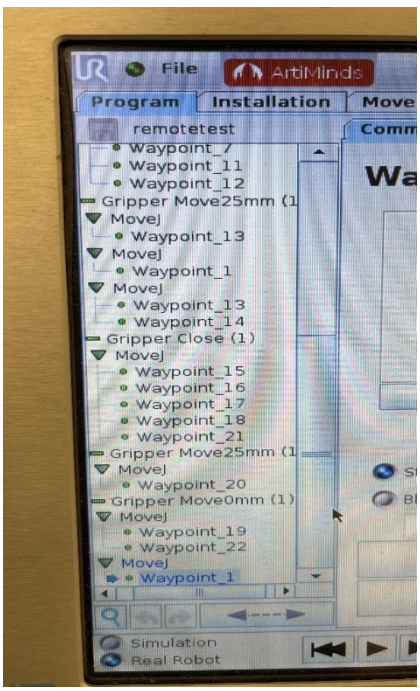

Figure 35. UR5 program structure part 2

6 Industrial application

6.1 Setup

In an industrial environment, a larger arm should be used. The UR10e, made by Universal Robots is a solid option for this. It has a standard range of 1,300 mm and shares the same design as the UR5 that is used in this case. There are multiple ways of setting up the robot arm. This can eighter be done with a fixed position as illustrated on the left side of Figure 36, or with a variable position where the UR10 moves as can be seen on the right side of Figure 36.

Both systems could stack up to 3 to 4 printers on top of each other which would bring the total amount of printers handled by the fixed system to a range of 36 to 48 printers in a very compact space. The system where the robot arm can move could theoretically hold as many printers as you want although it would start losing performance since there would be too many printers for the arm to handle.

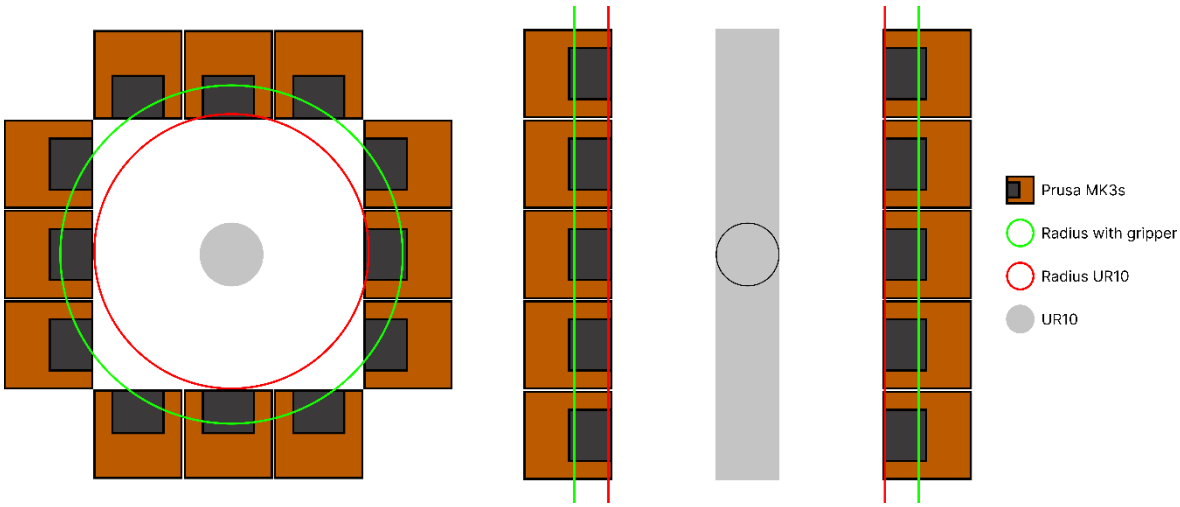

Figure 36. Possible industrial setups UR10

6.2 Price calculation

Both systems mentioned earlier would come at a hefty price tag. Since the system, with the UR10 in a fixed position, is the least variable, this will be used to do the price calculation.

The system would include

- 48 Prusa MK3S+ 3D printers (€ 769 apiece)
- 1 UR10e (€ 35,500)
- 48 Raspberry Pi 3b+ (€ 28 apiece)

The total price of this hardware amounts to 73,756 euros. This can be rounded up to 80,000 euros for the shelves, extra print beds, and a conveyer belt to move harvested print beds, …

This might seem like a hefty price tag, but the system pays for itself.

At least one employee is needed to harvest all print beds and start new prints. He or she works 8 hours a day, 5 days a week for an estimated gross salary of € 30,000 a year. The robot arm on the other hand can work 24 hours a day and does this 7 days a week. This is 4.2 times as much as the employee without keeping the vacation/sick days of the employees in mind. In other words, the application saves an estimated amount of about ~ €126,000 a year which means that the system pays for itself in less than 8 months.

6.3 Pros / Cons

Using a system like this will of course have pros and cons. These include

Pro's	**Cons**
Faster print retrieval	Setup time
More output since all printers work 24/7	Not ideal for small robotic arms
A lot cheaper in the long run	High entry price
Does not need a lot of maintenance	Maintenance only by certified people

7 Summary and discussion

This thesis proves that collaboration between 3D printers and industrial robotic arms can be a viable solution for big 3D printer farms. Although the full application was not created in this thesis, the proof of concept indicates that a system like this will work extremely well. This does not mean that the theoretical solution would be perfect. The solution would perform extraordinarily on harvesting and starting prints automatically which will save a lot of time, but it would not work well on print failures. This is because there is not a solution for this yet. This system could in addition implement more safety features like using cameras and image recognition to detect whether or not a print is failing or succeeding. The application would also benefit from automatic print scheduling. This would add a nondeterministic polynomial-time complete (NP-complete) problem to the case which can be solved with a metaheuristic algorithm. In this algorithm, soft and hard constraints can be implemented. A soft constraint could be that the shorter prints are prioritised during the day to be able to have more shipments out and more work for the employees who do the quality control. A hard constraint can be that a print cannot have a longer waiting period than a certain number of days and that a printer can only print one thing at a time.

With all these safety features installed, the system would be very promising. No human interaction would be needed at all to handle the print farm. That is if no printers break. The system allows companies with a small number of employees to still have a large production rate.

The main obstacle for companies or schools that are interested in this system is the entry price. As shown in chapter 6.2, this would be an estimated price of at least 80 000 euros. This is a very high price tag but since the farm could work 24/7 and would eliminate at least one employee, it would pay for itself in less than nine months, if not less.

Unfortunately, the client-side application was not physically created. The case only focussed on improving the producer experience, specifically maintaining the printers.

It can be concluded that this concept is very promising and a viable, realistic solution for schools or companies. It is estimated to increase the production rate up to three to four times with the same number of printers. This increase in production would also mean that companies with 100 printers divided into multiple sectors could instead use this system in one small workspace resulting in a cheaper, more efficient use of fewer printers.

Although this thesis proved that this system would work, it is still a large project that could benefit from many design iterations before going into production. Given the price tag of the system, its future development would require company funding from interested companies.

References

3D-CADSOLUTIONS. (2022, 04 23). *Webshop*. Opgehaald van 3D-CADSOLUTIONS: https://www.an-cadsolutions.fi/tuote/creality-cr-5-pro-h-3d-tulostin/?gclid=CjwKCAjwx46TBhBhEiwArA_DjNAgLWe_14fAT6GK91bIUoj5KPRZAdL-sf2Klm9cKR7kxONQQ5L1NRoCansQAvD_BwE

3DPrinting.com. (2022, 02 14). *What is 3D printing? How does a 3D printer work? Learn 3D printing*. Retrieved from 3D Printing: https://3dprinting.com/what-is-3d-printing/

Ahart, M. (2022, 02 14). *Types of 3D Printing Technology*. Retrieved from Protolabs.com: https://www.protolabs.com/resources/blog/types-of-3d-printing/

Bland, D. (2022, 22 02). *The Business Model for Robot Food - Precoil.* Retrieved from Medium: https://medium.com/precoil/the-business-model-for-robot-food-136d461fbfb0

Brookins, A. (2022, 02 15). *What are the different types of industrial robots?* Retrieved from Igus Engineer's Toolbox: https://toolbox.igus.com/motion-plastics-blog/what-are-the-different-types-of-industrial-robots

Carolo, L. (2022, 02 15). *What Is a 3D Slicer? – Simply Explained*. Retrieved from All3DP: https://all3dp.com/2/what-is-a-3d-slicer-simply-explained/

Chen, A. (2022, 03 31). *Cmac*. Opgehaald van 5 vital things to know about STL file format in 3D printing: https://www.cmac.com.au/blog/5-vital-things-about-stl-file-format-3d-printing

Collins, D. (2022, 02 15). *When Do You Need a Gantry Robot?* Retrieved from Linear Motion Tips: https://www.linearmotiontips.com/when-do-you-need-a-gantry-robot/

D.S.C. (2022, 02 22). *R17 low-cost benchtop articulated robot arm.* Retrieved from robotics: https://strobotics.com/articulated-robot-arm.htm

Elre, C. (2022, 02 22). *What is the difference between inverse kinematics approach and forward kinematics approach?* Opgehaald van Quora: https://www.quora.com/What-is-the-difference-between-inverse-kinematics-approach-and-forward-kinematics-approach#:%7E:text=unique%20solution%2C%20i-,Forward%20kinematics%20(for%20a%20robot%20arm)%20takes%20as%20input%20joint,is%20used%20for%20trajectory%2

FANUC. (2022, 02 22). *FANUC M-1 industrial delta robot series - Fanuc.* Opgehaald van FANUC: https://www.fanuc.eu/pl/en/robots/robot-filter-page/delta-robots/m1-series

Grames, E. (2022, 02 14). *What is FDM 3D Printing? – Simply Explained*. Retrieved from All3DP: https://all3dp.com/2/fused-deposition-modeling-fdm-3d-printing-simply-explained/

M., A. (2022, 02 15). *The 4 Types of FFF / FDM 3D Printer Explained (Cartesian, Delta, Polar)*. Retrieved from 3Dnatives: https://www.3dnatives.com/en/four-types-fdm-3d-printers140620174/#!

Mindworks. (2022, 02 14). *3D Printing*. Retrieved from Mindworks: http://mindworks.shoutwiki.com/wiki/3D_Printing

Prusa Research. (2022, 04 23). *Original Prusa i3 MK3S+ | Original Prusa 3D printers directly from Josef Prusa*. Opgehaald van Prusa Research: https://www.prusa3d.com/category/original-prusa-i3-mk3s/

RS. (2022, 04 7). *What are robotic arms*. Opgehaald van RS: https://uk.rs-online.com/web/generalDisplay.html?id=ideas-and-advice/robotic-arms-guide

Sabhadiya, J. (2022, 02 15). *What Is Rapid Prototyping?- Types, And Working*. Retrieved from Engineering Choice: https://www.engineeringchoice.com/rapid-prototyping/

Spiegel, G. (2022, 03 31). *IV. Slicing - 3D Printing in O&P*. Opgehaald van Medium: https://medium.com/3d-printing-in-o-p/iv-slicing-72a9515f44bc

Tractus3D. (2022, 02 15). *FDM 3D printing - Fused Deposition Modeling also known as FFF*. Retrieved from Tractus3D: https://tractus3d.com/knowledge/learn-3d-printing/fdm-3d-printing/

Universal Robots. (2022, 04 20). *Collaborative robots from UR | Start your automation journey*. Opgehaald van COLLABORATIVE ROBOTS FROM UNIVERSAL ROBOTS: https://www.universal-robots.com/products/

UNSW Making Sydney. (2022, 04 25). *UR5 Basics Learn Module | UNSW Making*. Opgehaald van UNSW Making: https://www.making.unsw.edu.au/learn/ur5-basics-learn-module/

WiredWorkers. (2022, 04 20). *Universal Robots UR5 | Tough, flexible collaborative robot arm.* Opgehaald van WiredWorkers: https://wiredworkers.io/universal-robots-ur5/

Appendix 1. Dashboard server commands

Command	Return value	Description	Supported from version
load <program.urp>	On success: • "Loading program: <program.urp>" On Failure: • "File not found: <program.urp>" • "Error while loading program: <program.urp>"	Returns when both program and associated installation has loaded (or failed). The load command fails if the associated installation requires confirmation of safety. The return value in this case will be 'Error while loading program'.	v1.4
play	On success: • "Starting program" On failure: • "Failed to execute: play"	Returns failure if the program fails to start. In previous versions this did not happen in all cases.	v1.4
stop	On success: • "Stopped" On failure: • "Failed to execute: stop"	Returns failure if the program fails to stop. In previous versions this did not happen in all cases..	v1.4
pause	On success: • "Pausing program" On failure: • "Failed to execute: pause"	Returns failure if the program fails to pause. In previous versions this did not happen in all cases.	v1.4
quit	"Disconnected"	Closes connection	v1.4
shutdown	"Shutting down"	Shuts down and turns off robot and controller	v1.4
running	"Program running: True" OR "Program running: False"	Execution state enquiry	v1.6

Command	Return value	Description	Supported from version
robotmode	CB2: integer is returned • NO_CONTROLLER_MODE = -1 • ROBOT_RUNNING_MODE = 0 (This is "normal" mode) • ROBOT_FREEDRIVE_MODE = 1 • ROBOT_READY_MODE = 2 • ROBOT_INITIALIZING_MODE = 3 • ROBOT_SECURITY_STOPPED_MODE = 4 • ROBOT_EMERGENCY_STOPPED_MODE = 5 • ROBOT_FAULT_MODE = 6 • ROBOT_NO_POWER_MODE = 7 • ROBOT_NOT_CONNECTED_MODE = 8 • ROBOT_SHUTDOWN_MODE = 9 CB3: text is returned "Robotmode: <mode>", where <mode> is • NO_CONTROLLER • DISCONNECTED • CONFIRM_SAFETY • BOOTING • POWER_OFF • POWER_ON • IDLE • BACKDRIVE • RUNNING	Robot mode enquiry	v1.6
get loaded program	"Loaded program: <path to loaded program file>" OR "No program loaded"	Which program is loaded	v1.6
popup <popup-text>	"showing popup"	The popup-text will be translated to the selected language, if the text exists in the language file	v1.6
close popup	"closing popup"	Closes the popup	v1.6

Command	Return value	Description	Supported from version
addToLog <log-message>	"Added log message" Or "No log message to add"	Adds log-message to the Log history	v1.8
isProgramSaved	"True" OR "False"	Returns the save state of the active program and path to loaded program file.	v1.8
programState	"STOPPED" if no program is running "PLAYING" if program is running "PAUSED" if program is paused(CB3/CB3.1 only)	Returns the state of the active program and path to loaded program file, or STOPPED if no program is loaded	v1.8
PolyscopeVersion	version number, like "3.0.15547"	Returns the version of the Polyscope software	v1.8
setUserRole <role> where <role> can be: **programmer** **operator** **none** **locked**	"Setting user role: <role>" OR "Failed setting user role: <role>" • *programmer* = In Setup Robot, buttons "Update", "Set Password", "Network", "Time" and "URCaps" are disabled, "Expert Mode" is available (if correct password is supplied) • *operator* = Only "RUN Program" and "SHUTDOWN Robot" buttons are enabled, "Expert Mode" cannot be activated • *none* (*or send setUserRole*) = All buttons enabled, "Expert Mode" is available (if correct password is supplied) • *locked* = All buttons disabled and "Expert Mode" cannot be activated	Control of user privileges: controls the available options on the Welcome screen. Note: If the Welcome screen is not active when the command is sent, the user privileges defined by the new user role will not be effective until the user switches to the Welcome screen.	v1.8
setUserRole <role> extended with new role: **restricted**	"Restricted" works like "operator" but does not give access to the move tab.		v3.1

Command	Return value	Description	Supported from version
getUserRole	PROGRAMMER, OPERATOR, NONE, LOCKED or RESTRICTED	Returns the user role	
power on	"Powering on"	Powers on the robot arm	v3.0
power off	"Powering off"	Powers off the robot arm	v3.0
brake release	"Brake releasing"	Releases the brakes	v3.0
safetymode	"Safetymode: <mode>", where <mode> is • NORMAL • REDUCED • PROTECTIVE_STOP • RECOVERY • SAFEGUARD_STOP • SYSTEM_EMERGENCY_STOP • ROBOT_EMERGENCY_STOP • VIOLATION • FAULT	Safety mode enquiry	v3.0
unlock protective stop	On success: • "Protective stop releasing" On failure: • "Cannot unlock protective stop until 5s after occurrence. Always inspect cause of protective stop before unlocking"	Closes the current popup and unlocks protective stop The unlock protective stop command fails if less than 5 seconds has passed since the protective stop occurred.	v3.1
close safety popup	"closing safety popup"	Closes a safety popup	v3.1

Command	Return value	Description	Supported from version
load installation <default.installation>	On success: • "Loading installation: <default.installation>" On failure: • "File not found: <default.installation>" • "Failed to load installation: <default.installation>"	Loads the specified installation file but does not return until the load has completed (or failed). The load command fails if the associated installation requires confirmation of safety. The return value will be 'Failed to load installation'.	v3.2
restart safety	Restarting safety	Used when robot gets a safety fault or violation to restart the safety. After safety has been rebooted the robot will be in Power Off. IMPORTANT: You should always ensure it is okay to restart the system. It is highly recommended to check the error log before using this command (either via PolyScope or e.g. ssh connection).	v3.7
get serial number	Serial number like "2017351234"	Returns serial number of the robot.	v3.12
get robot model	UR3, UR5, UR10	Returns the robot model	v3.12
generate flight report <report type> where possible report types are: **controller** **software** **system** Default Type is 'system' if no option is specified	On success report id is printed. Error Message on a failure. Command can take few minutes to complete.	Triggers a Flight Report of the following type: Controller - report with information specific for diagnosing controller errors. For example, in case of protective stops, faults or violations. Software - report with information specific for polyscope software failures. System - report with information about robot configuration, programs, installations etc. It is required to wait at least 30 seconds between triggering software or controller reports.	v3.13

Command	Return value	Description	Supported from version
load installation <default.installation>	On success: • "Loading installation: <default.installation>" On failure: • "File not found: <default.installation>" • "Failed to load installation: <default.installation>"	Loads the specified installation file but does not return until the load has completed (or failed). The load command fails if the associated installation requires confirmation of safety. The return value will be 'Failed to load installation'.	v3.2
restart safety	Restarting safety	Used when robot gets a safety fault or violation to restart the safety. After safety has been rebooted the robot will be in Power Off. IMPORTANT: You should always ensure it is okay to restart the system. It is highly recommended to check the error log before using this command (either via PolyScope or e.g. ssh connection).	v3.7
get serial number	Serial number like "2017351234"	Returns serial number of the robot.	v3.12
get robot model	UR3, UR5, UR10	Returns the robot model	v3.12
generate flight report <report type> where possible report types are: **controller** **software** **system** Default Type is 'system' if no option is specified	On success report id is printed. Error Message on a failure. Command can take few minutes to complete.	Triggers a Flight Report of the following type: Controller - report with information specific for diagnosing controller errors. For example, in case of protective stops, faults or violations. Software - report with information specific for polyscope software failures. System - report with information about robot configuration, programs, installations etc. It is required to wait at least 30 seconds between triggering software or controller reports.	v3.13

Command	Return value	Description	Supported from version
generate support file <Directory path> where <Directory path> represents path to an already existing directory location inside the programs directory. In particular path can point to special usbdisk subfolders inside programs folder.	On success "Completed successfully: <result file name>" is printed otherwise an error message with possible cause of the error is shown. Command can take up to 10 minutes to complete.	Generates a flight report of the type "System" and creates a compressed collection of all the existing flight reports on the robot along with the generated flight report. Result file *ur_[robot serial number]_YYYY-MM-DD_HH-MM-SS.zip* is saved inside <Directory path>	v3.13