DANMARKS TEKNISKE UNIVERSITET

FINAL PROJECT

62607 - Applied Industrial Robotics

TEAM 4

Lukáš Málek s212074 _____

Martin Miksik s212075 _____

Jan Heimsoth s212169 _____

December 2021
Technical university of Denmark

Contents

Introduction

This assignment aims to implement 3D Printer using KUKA KR 6 R700 sixx and provided extruder tool. The project consists of the following tasks, where each task is answered in the separated section.

1. Choose suitable 3D model to be printed, slice the model using third party software and export the model as a gcode file.
2. Make a python parser script, extracting usable data from the gcode file.
3. Use RoboDK for full simulation of the robot movements.
4. Write a python script, translating position commands as in a kuka source and data files.
5. Make a 3D printed mounting tool for PLA hotend extruder.
6. Load our Kuka program into the robot controller with a USB-stick
7. Execute the program and make a 3D print with the robot

It's forbidden to use third party libraries for parser and export scripts. Also, each line of the code should be commented.

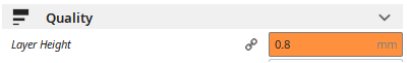
Every member of the team contributed to the project equally and tried his own method of implementing the problem. Some were more successful than others, but each gave us different point of view as of how to solve the problems of this project.

1 Slicing

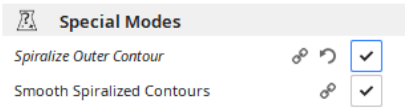
For exporting gcode, our team used Ultimaker Cura (4.10.0). The decision for choosing this slicing program was mostly based on previous experience with the interface, resulting in faster task completion.

We will go through these steps to slice the model:

1. Download Ultimaker Cura
2. Create or download your desired model
3. In Ultimaker, use suitable printer template. The Ultimaker 3 Extended was chosen.
4. Move the model close to the origin.
5. We will work and tune several parameters for the best performing print:
   - LAYER HEIGHT: 4-8 [mm]

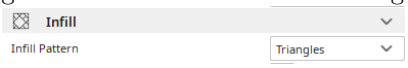
   - SPIRALIZE MODE: Printing in one continous line without the need of switching on and off the extruder

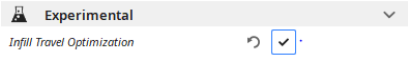
   - BUILD PLATE: As the bed isn't heated, most suitable setting is as 'None'

   - INFILL PATTERN: 'Triangles' works the best with our design.

   - TRAVEL OPTIMIZATION: Turn on for optimized printing path

- COASTING: Enable, to reduce stringing

6. Slice the model using *Slice* button in the right bottom corner
7. After slicing is finished, save the generated gcode usign the button *Save to Disk*

The profile changes mentioned above are the most important as they directly affect the printing quality. Other settings, such as printing temperature, prime blob, cooling or printing speed are already set by the provided extruder.

2 Gcode Parser

We will introduce two ways of programming the parser to extract the usable data from Gcode. However, the general idea of the parser's functionality remains the same; read the gcode file that is supposed to give information to a commercial printer, extract values that are valuable for the project and store them by a suitable method to be later used in the program.

Three versions of the 3D Printer were implemented, where both parser approaches were used. Therefore it is crucial to include them both in the report. Trying more versions of how to operate the robot deepened our understanding of the problematics. Also, now it is possible to create a more accurate conclusion and compare the different methods.

2.1 Method 1

The parsing function is called at the beginning of the printing program and processes the whole gcode at once. The parsed values are stored in a list[]. Each item in the list is in the following format [X axis, Y axis, layer, printing status].

As the robot operates with respect to the task frame and we are only utilising the first quadrant of the XY plane, the X and Y coordinates will have only positive values.The Z value can be incrementally added with the reference of currently printed layer. This gives us an opportunity of further testing and tuning the layer height.

- X value = list[0]
- Y value = list[1]
- Z value = list[2] * layer height

Below, you can see the Python function, which is called from the main program.

```
def parser(filename):
    parsed_file = list()                                        # Initialize the list variable parsed_file
    layer = 0                                                   # Initialize the lazer variable
    with open(filename) as gcode:                               # Oepns the gcode file under variable gcode
        for line in gcode:                                      # Starts reading each line in gcode
            line = line.strip()                                 # Formats each line correctly
            if re.findall("LAYER:", line):                      # Find every line that contains word "layer: "
                layer += 1                                      # Add 1 to variable layer
                continue                                        # Skips the rest of the cycle
            if re.findall("G1", line):                          # Find every keyword g1 that specifies print path
                coord = re.findall(r'[XY].?\d+.\d+', line)      # Use regex expression to store x and y values
                if len(coord) == 2:                             # Check if the line mentioned both x and y
                    X = re.findall('\d*\.?\d+', coord[0])[0]    # Store only the number without x into var
                    Y = re.findall('\d*\.?\d+', coord[1])[0]    # Do the same but for y
                    parsed_file.append([float(X), float(Y), layer, True])   # Append list in format [x,y,layer,print status]
            if re.findall("G0", line):                          # Find each movement path when dont extrude
                coord = re.findall(r'[XY].?\d+.\d+', line)      # Use regex to store x and y coordinate
                if len(coord) == 2:                             # Check if we found both x and y values
                    X = re.findall('\d*\.?\d+', coord[0])[0]    # Store only the number for X
                    Y = re.findall('\d*\.?\d+', coord[1])[0]    # Store only the number for y
                    parsed_file.append([float(X), float(Y), layer, False])  # Append list in format [x,y,layer,print status]
    return parsed_file                                          # Return the list with all added coordinates
```

Example: We are looking only for lines beginning with G0 or G1, which are the printing commands. We will store G1 as boolean 'True' and G0 as 'False' as the last value in our list. These characters inform the printer whether it should only move it's end-effector or printing is in process simultaneously. The boolean value will

be useful in determining whether the extruder should be turned on or off. Consecutive to these characters, we will extract X, Y coordinates of the printing position. The third value is the Z coordinate, which suits as n-th printing layer. Together, line *[G1 X76.699 Y32.267 E3.04345]* will be stored in a list as [76.699, 32.267, 1, True].

2.2 Method 2

This parser Method stores all the important commands and coordinates from the .gcode file into a list of lists. Each list in this list contains 4 numbers: [Index of the G-command, x-value, y-value, z-value]. The storing of the index behind the G-command is important in order to know when to extrude material and when to stop.

When the function `gcode_to_list()` is called on the path of a .gcode file it gets opened as `gc` in the read only mode.[1] After that the lines are separated, and the empty lines are filtered out.Then every line in `gc` is processed into a list of numbers as described before by `calc_new_pos()` from the last position. This is done by comparing the last extracted position with the new one and updating the appropriate positions.

```
1  # Define the re pattern that is used to find the coordinates from the
2  # gcode lines, such that every number following the letters 'G', 'X',
3  # 'Y' and 'Z' are included.
4  RE_ARG = re.compile(r"[GXYZ]\d+[.]?\d*")
5
6  def gcode_to_list(gcode):
7
8      # Open the .gcode file as gc and read the lines to use
9      # calc_new_pos() on each line.
10     with open(gcode, 'r') as gc:
11
12         # Read all the lines in a list of list.
13         gc = [re.findall(RE_ARG, l) for l in gc.readlines()]
14
15         # Filter out all empty elements of the list.
16         gc = list(filter(None, gc))
17
18         positions = []
19
20         # Calculate first position and append it to 'positions'.
21         last_pos = calc_new_pos(np.zeros((4, 1)), gc[0])
22         positions.append([item for sublist in last_pos.tolist()
23                       for item in sublist])
24
25         # Iterate through the rest of the lines from the gcode and
26         # calculate the new position to append it to 'positions'.
27         for i in gc[1:]:
28             last_pos = calc_new_pos(last_pos,i)
29             positions.append([item for sublist in last_pos.tolist()
30                           for item in sublist])
31
32     return positions    # return the positions list
33
34 def calc_new_pos(last_pos, line):
35
36     # Retrive the new position and the valid array from
37     # get_pos_from_line()
38     pos_cmd, pos_cmd_valid = get_pos_from_line(line)
39
40     new_pos = np.zeros((4, 1))  # Initialise array to store the pos.
41     last_pos[pos_cmd_valid] = 0 # Set all valid positions to zero.
42
43     # Add the new valid positions to the old position, where the valid
44     # positions have be set to zero.
```

[1] Compare to line 11

```
45      new_pos = last_pos + pos_cmd
46
47      return new_pos  # return the new calculated position as np.array
48
49  def get_pos_from_line(line):
50      # Initialisation of arrays for storage of the returns.
51      pos = np.zeros((4, 1))
52      pos_valid = np.zeros((4, 1), dtype=np.bool_)
53
54      # Dictionary to define which letter is followed by which position
55      # it belongs to.
56      COORD_DICT = {'G': 0, 'X': 1, 'Y': 2, 'Z': 3}
57
58      # Checking whether each element i of line mathces RE_ARG and
59      # if its updated by the gcode or not.
60      if 'G' in line[0][0]:
61          for i in line:
62              if RE_ARG.match(i) is not None:
63                  pos[COORD_DICT[i[0]]] = float(i[1:])
64                  pos_valid[COORD_DICT[i[0]]] = True
65      return pos, pos_valid
```

Example: A line 'G0 F6666.7 X21.431 Y21.035 Z2.4' is firstly separated into an array ['G0', 'X21.431', 'Y21.035', 'Z2.4']. Then a position array [[0.],[21.431],[21.035],[2.4]] and a boolean array [[True][True][True][True]] will be returned. now all values of the last position, where the boolean array is 'True' will be updated into the values from the positions array and then appended to the list of lists, which then is the output of gcode_to_list().

3 Simulation

3.1 Method 1: Online Programming

This method solves the project, including parsing, simulation and printing, by the Online-Programming approach using RoboDK's Robolink.py libraries and general inverse and forward kinematics for task-frame coordinate transformations. This method was later abandoned, as components in the Robolink library assuring the project to work in accordance to our design were missing. One of the problems encountered was the lack of driver support for setting the robot's end-effector speed to a linear velocity, which is crucial for the correct performance of the 3D printer. There were attempts of fitting the end-effector's path with additional points for smoother linear movement or injecting instructions for linear movement directly to our robot. However, by this time, the team has agreed that a more suitable method should be used, even if it would cause the inconvenience of Offline-Programming's slow debugging and operation.

Even with the knowledge of the prohibited use of Online-Programming for this Project, the team has concluded that the insights of the method were positive enough to include them in this report.

3.1.1 User manual

1. Open RoboDK
2. Open robot base in RoboDK
3. Define task frame in RoboDK:
   (a) Create a new frame
   (b) Drag and drop the frame over the robot to initialise parent
   (c) Rename frame to 'task_frame'
   (d) Manually move the robot to 3 points that will define the task frame. The first and second points must lie on X-axis, the third point must lie on Y-axis. Save the joint angles of these points.
   (e) click Define Frame in RoboDK

(f) Insert obtained angles

(g) Click Update

4. Connect to the robot by plugging RJ45 cable into your computer and set ethernet IPv4 as '172.31.1.147'
5. open 3Dprint_online.py
6. Adjust values flagged as 'ADJUSTABLE', include a path to your sliced gcode and use one of the obtained P points as reference in 'home_angles'.
7. Run the listening script on the Kuka controller
8. Before continuing, check all of the safety measures as you are about to initialize the robot movement
9. Run the python script
10. Adjust 'ADJUSTABLE' parameters if needed and repeat running the script

3.1.2 Code Description

Used files:

- functions.py
- 3Dprint_online.py

Gcode model
This is the chosen sliced model to print. As the controller is in blocking mode while listening to position instructions, it is no longer possible to send I/O commands. This problem could be solved by adding an I/O listener to the KUKA controller program to change the state of the extruder. Therefore it is highly recommended to use spiralized slicing, as it utilises the print in one continuous line without the need of extrusion stops.

Functions.py

```
import re                                                   #standard library providing regular expression facilities
import math                                                 #standard library providing mathematical operations

def move(joint, client):
    '''converts the output (joint angles) as commands to the robot controller'''
    string = '{A1 ' + str(list(joint)[0][0]) + ', A2 ' + str(list(joint)[0][1]) + ', A3 '\
        + str(list(joint)[0][2]) + ', A4 ' + str(list(joint)[0][3]) + ', A5 '\
        + str(list(joint)[0][4]) + ', A6 ' + str(list(joint)[0][5]) + '}'
    client.write('MYAXIS', string)


def extruder_online(client, arg):
    '''Converts parser's instruction to start/stop extruding to robot controller as ON/OFF on I/O
        and prints the change of extruding state'''
    if arg == 'START':
        client.write('$OUT[16]', 'True')
    elif arg == 'STOP':
        client.write('$OUT[16]', 'False')
    else:
        print('The input of extruder() has to be either START or STOP')


def extruder_offline(robot, arg):
    '''Same as extruder(), just for offline usage '''
    if arg == 'START':
        robot.setDO('$OUT[16]', 'True')
    elif arg == 'STOP':
        robot.setDO('$OUT[16]', 'False')
    else:
        print('The input of extruder() has to be either START or STOP')


def parser(filename):
    '''parse gcode commands to be used in our code'''
    parsed_file = list()                                    # create a list
    layer = 0                                               # z-axis, or current height of the print
    with open(filename) as gcode:                           # open gcode path and use as variable
        for line in gcode:                                  # for each line in gcode:
            line = line.strip()                             # delete all unnecessary ASCII characters
            if re.findall("LAYER:", line):                  # if new layer found, update z-axes
                layer += 1
```

```
                continue                                                    # similar to elif
            if re.findall("G1", line):                                      # check if line begins with "G1"
                coord = re.findall(r'[XY].?\d+.\d+', line)                  # assign coord values
                if len(coord) == 2:                                         # if both coords were assigned
                    X = re.findall('\d*\.?\d+', coord[0])[0]                # assign X-value
                    Y = re.findall('\d*\.?\d+', coord[1])[0]                # assign Y-value
                    parsed_file.append([float(X), float(Y), layer, True])   # append our desired output for use in the project
    return parsed_file                                                      # return parsed list


def interpolate(path, detail):
    '''Interpolates end-effector's path for smooth and linear velocity.
       Not used, hence not commented.'''
    extra = 0                                                   # inserted value, colide with indexing
    for i in range(len(path)-1):                                # go through all points
        number_of_cuts = int(distance(path[i+extra], path[i+extra+1])//detail) #calculates how many times we fit distance 1.45
        if number_of_cuts > 0:                                  # interpolate the 2 points only if the distance is > 1.45
            dist_x, dist_y = distance_x_y(
                path[i+extra], path[i+extra+1], number_of_cuts+1)   # return the distance to be added to each coordinate
            for k in range(number_of_cuts):                     # for each index k depending on value of number_of_cuts
                item = list()                                   # each iteration create a new list and discard the last
                item.append((k+1)*dist_x + path[i+extra][0])    # append new interpolated x coordinate to the path
                item.append((k+1)*dist_y + path[i+extra][1])    # append new interpolated y coordinate to the path
                item.append(path[i+extra][2])                   # append which lazer it is
                item.append(path[i+extra][3])                   # append the true or false value whether we are extruding
                path.insert(i+k+extra+1, item)                  # insert the created list above to the path on the correct index
        extra += number_of_cuts                                 # store how many items were added for correct indexing


def distance(point1, point2):
    '''computes distance between points for use in interpolate()'''
    # index 0 is X, index 1 is Y
    distance = math.sqrt((point1[0]-point2[0])**2 + (point1[1]-point2[1])**2) # calculates the eular distance between 2 points
    return distance


def distance_x_y(point1, point2, number_of_cuts):
    '''return the distance which is added to each points so that the points are equally distanced'''
    distance_x = (point2[0] - point1[0])/(number_of_cuts)       # calculate the distance for x coord depending on number_of_cuts
    distance_y = (point2[1] - point1[1])/(number_of_cuts)       # calculate the distance for y coord depending on number_of_cuts
    return distance_x, distance_y
```

3Dprint_online.py

```
# Type help("robolink") or help("robodk") for more information
# Documentation: https://robodk.com/doc/en/RoboDK-API.html
# Reference:     https://robodk.com/doc/en/PythonAPI/index.html
from robolink import *                                  # RoboDK API
from robodk import *                                    # Robot toolbox
import time                                             # Standard library for representing time
from py_openshowvar import openshowvar                  # Python port of KUKA VarProxy client
from functions import *                                 # Our functions

# ROBODK INITIALIZATION
RDK = Robolink()                                        # Initialize Robolink as RDK
robot = RDK.Item('KUKA KR 6 R700 sixx')                 # Retrieve robot from RDK
RDK.setSimulationSpeed(1)                               # ADJUSTABLE speed which corelates with speed of robot
reference = robot.Parent()                              # Retrieve the robot reference frame
robot.setPoseFrame(reference)                           # Use the robot base frame as active reference
home = [0, -90, 90, 0, 0, 0]                            # Joint angles for default home position
robot.MoveJ(home)                                       # Move to default home position in RoboDK
robot.setSpeed(speed_linear=20, speed_joints=20,        # ADJUSTABLE
               accel_linear=3000, accel_joints=20)      # Robot speed together w/ line 11
layer_height = 0.55                                     # ADJUSTABLE layer height


# PARSER
path = parser('desired/path/to/gcode')                  # ADJUSTABLE: Parse the gcode

#INTERPOLATION CHECK
detail = 1.4                                            # Everything over this distance is interpolated
interpolate(path, detail)                               # New interpolated path
error = False                                           # Initialize error status
for i in range(len(path)-1):                            # Additional check, loop through parsed list
    if distance(path[i], path[i+1]) > detail:           # If consequent point are far apart
        error = True                                    # Error has occured, debug.
print('Smoothing Error') if error else print('Smoothing OK') # Print status validation


# CONNECT TO ROBOT
# client = openshowvar('172.31.1.147', 7000)              # I.P adress + port 7000
# client.write('$OUT[16]', 'FALSE')                       # Turn off extruder if some before left it on
# client.write(
```

```
40  #   'XHOME', '{A1 0.0, A2 -90, A3 90, A4 0, A5 -0, A6 0.0,\
41  #         E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0}')    # Set home command position to the robot
42  # client.write(
43  #   'MYAXIS', '{A1 0.0, A2 -90, A3 90, A4 0, A5 -0, A6 0.0,\
44  #        E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0}')     # Command robot to move home as well
45  # time.sleep(3)                                                 # Wait for safety reasons
46  print('To run the simulation on real robot, uncomment all lines with keyword "client"')
47
48
49  #TASK FRAME INIT
50  # 1) move robot to points P1, P2 on X-Axis and P3 on Y-Axis
51  # 2) define frame by algebraic properties and basis transformation or in RoboDK
52  # 2) move to one of the P and retrieve end-effector angle
53  item_frame = RDK.Item('task_frame')                           # ADJUSTABLE retrieve task_frame from RoboDK
54  home_joints = [-5.520000, -107.550000, 115.910000, 1.620000,
55                 37.390000, -0.2400006]                         # ADJUSTABLE Orient end-effector wrt task_frame angle
56  robot.setFrame(item_frame)                                    # Use newly obtained frame as reference
57  robot.MoveJ(home_joints)                                      # Move to home, first move must be joint move to avoid singularities
58  # move(robot.Joints(), client)
59
60  #INVERSE KINEMATICS
61  # For use with general starting position of the robot: compute
62  # future movement's joint angles by inverse kinematics transformation used on
63  # desired destination point in space, which is obtained by forward kinematics of
64  # current robot pose.
65  orient_frame2tool = invH(item_frame.Pose())*robot.SolveFK(home_joints)
66  # remove unnecessary position vector of homogeneous transformation matrix
67  # -> only rotational matrix is persieved
68  orient_frame2tool[0:3, 3] = Mat([0, 0, 0])
69
70
71  # GET RID OF EXTRA FILAMENT (CAN BE REMOVED)
72  robot.MoveL(transl([0, 0, 0])*orient_frame2tool)              # Printing blob
73  # move(robot.Joints(), client)
74  time.sleep(2)                                                 # Used to leave the blob
75  robot.MoveL(transl([0, 0, 3])*orient_frame2tool)              # Move up to get rid of the filament
76
77  # ADJUSTABLE SAFETY CHECK of task_frame margins
78  robot.MoveL(transl([0, 0, 0])*orient_frame2tool)              # Move to the first corner
79  # move(robot.Joints(), client)
80  robot.MoveL(transl([0, 70, 0])*orient_frame2tool)             # Move to the second corner
81  # move(robot.Joints(), client)
82  robot.MoveL(transl([70, 70, 0])*orient_frame2tool)            # Move to the third corner
83  # move(robot.Joints(), client)
84  robot.MoveL(transl([70, 0, 0])*orient_frame2tool)             # Move to the forth corner
85  # move(robot.Joints(), client)
86  robot.MoveL(transl([0, 0, 0])*orient_frame2tool)              # Move back to the start
87  # move(robot.Joints(), client)
88
89  # PRINTING
90  RDK.setSimulationSpeed(4)                                     # ADJUSTABLE set sim speed
91  for item in path:                                             # For each line of parsed file:
92      # inject parsed coordinates and calculate Z-axis (based on layer height, and adjustable 0.15 offset for first layer)
93      target_point = [item[0], item[1], layer_height*item[2]+0.15]
94      target0 = transl(target_point)*orient_frame2tool          # Compute translation with respect to task_frame
95      robot.MoveL(target0, blocking=False)                      # move to the target in linear manner
96      # move(robot.Joints(), client)
97  robot.MoveJ(home)                                             # When done, move to home joint position
```

Thanks to the general transformation approach, this method can initialise the robot movement from any initial pose. However, since there are no workspace constraints introduced, it is highly recommended to start from the home position in order not to hit the working table. This safety measure is already implemented in the code on line 17.

The parser used, is implemented at Method 1 and is directly applicable on line 24, with only minor adjustments to the layer height.

In order to change the robot's position with respect to our task frame, we will need to establish such a frame. Due to a prohibition of nonstandard python libraries (NumPy), we have chosen to compute the inverse and forward kinematics with dedicated Robolink library functions, rather than algebraic computations. The future movement position must be associated with robot joint angles. We will compute them on line 65: Desired joint angles are computed by inverse kinematics transform used on desired destination point in space, which is obtained by forward kinematics of current robot pose. As DH-notation is already set by RoboDK, we don't need to go through the struggle of defining the robot arms' specifications. We are therefore given a homogeneous transformation matrix and as we are interested in joint angles only, we will remove the position column vector from the function matrix on line 68.

As an additional safety validation, we introduced a task frame sweep check on lines 77-87, where the robot

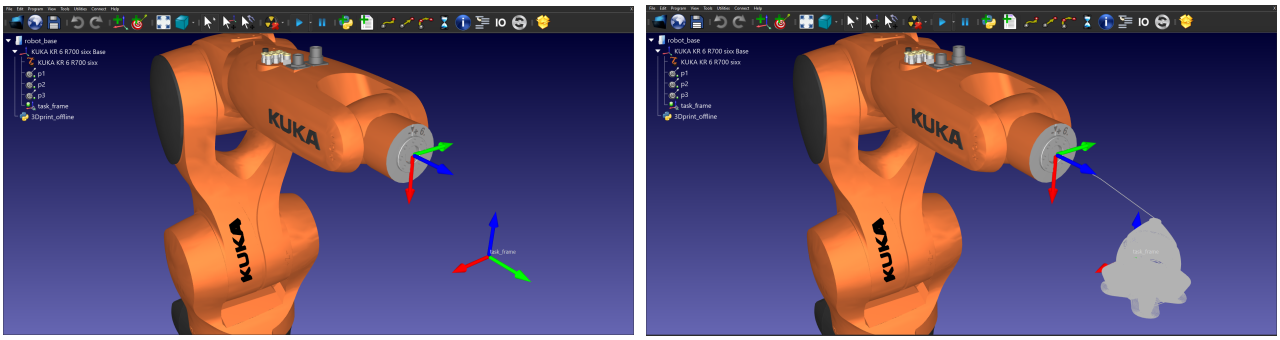
moves through defined task frame margins at such a slow speed, that the operator has a chance of stopping any misconduct behaviour.

In order to start robot movements, only client() functions must conviniently be uncommented.

After that, the robot moves into task frame origin and initialises the printing loop. For each line of the parsed list, the target point is established by coordinates [X, Y, Z] and translation of two concurrent points is introduced with respect to the task frame. Note that no rotation is needed, as the robot has already positioned its end-effector into the correct angle with respect to the task frame hyperplane.

The operator has still a possibility of choosing the speed at which the instructions are executed on the controller throughout the whole print.

Even though this method was later abandoned due to the assignment requirements, the procedure gave us many insights and challenges of working with RoboDK middleware, physical robot and debugging on-the-go.



(a) Running simulation (b) Finished Simulation

Figure 1: Robot Simulation inside RoboDK with the line trace activated.

3.1.3 Method Summary

This method has several advantages, as well as disadvantages:

- First initialization is slow
- Operator must have an RJ45 connector or appropriate reduction
- High dependency on RoboDK platform
- Listener blocking extruder commands.
+ Simulating the process in RoboDK is as easy as commenting out 'client()' parts of the program
+ Quick debugging
+ Quick changes of the printing parameters
+ General approach of computing task frame by algebraic theory and properties
+ Simulation availability without the need for a physical robot

3.2 Method 2 - Offline Programming v1

In this method, we run and generate the .src file directly in the RoboDK using parser Method 1. The reason for choosing this approach was the end effector's constant speed, resulting in smooth movements. The mathematical theory behind the program remains the same as in Method 1: Online Programming. Still, minor changes were made to the program to fit the use in RoboDK better.

Setting up the task frame in RoboDK instead of the KUKA controller empowers the possibility to simulate the exact movements in RoboDK.

The method was tested and was proven to be working as expected. The biggest drawback of this method is that travelling movements without extrusion is not working as expected. The delay after changing the output

value is around 2 seconds, increasing the printing time rapidly and needing further tuning. More information regarding this issue is described in the following method.

3.2.1 User manual

1. Open robot base in RoboDK
2. Define task frame in RoboDK:
   (a) Create a new frame
   (b) Drag and drop the frame over the robot to initialise parent
   (c) Rename frame to 'task_frame'
   (d) Manually move the robot to 3 points that will define the task frame. The first and second points must lie on X-axis, the third point must lie on Y-axis. Save the joint angles of these points. The points can be modified by clicking on point and then in menu bar Program → Modify Target, then you can change the angles of each joint.
   (e) Click Define Frame in RoboDK
   (f) Insert obtained angles
   (g) Click Update
3. Go to Tools→Options→Program and limit the maximum number of lines to 1000000
4. Right click on the python script in RoboDK → Edit and change the path of the GCode (search for ADJUSTABLE comments)
5. Double click the Python script to run the simulation in RoboDK
6. If simulation runs smoothly, right click on the script → Generate program
7. Copy paste the generated file to USB Disk
8. Plug the USB Stick in KUKA controller
9. Switch to Expert user profile
10. Copy-paste the code from USB Driver to memory of KUKA Controller
11. Run the air movement test (without printing)
12. Check the safety measures and close the doors
13. Turn the key to change the mode to AUT at the KUKA controller
14. Run the program

During the simulation in RoboDK App, it is highly recommended to turn on the trace to track if the path is correct. That can be done by clicking Tools → Trace → Activate.

3.2.2 Code Description

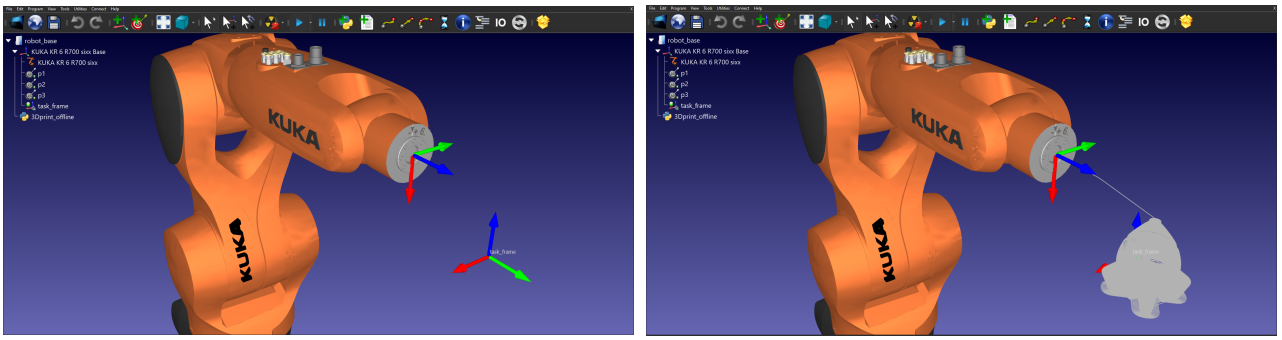
Used files:

- robot_base.rdk
- 3Dprint_offline.py (included in robot_base.rdk file)

3Dprint_offline.py

```
# Type help("robolink") or help("robodk") for more information
# Documentation: https://robodk.com/doc/en/RoboDK-API.html
# Reference:     https://robodk.com/doc/en/PythonAPI/index.html
# Note: It is not required to keep a copy of this file, your python script is saved with the station
from robolink import *                                  # RoboDK API
from robodk import *                                    # Robot toolbox
import time                                             # Standard time library
import re                                               # Imports the regex expressions
```

```
9
10  ######################################################## FUNCTIONS ###################################################################
11
12
13  def extruder_offline(robot, arg):
14      '''Turns on and off the extrusion '''                          # Not used in code for reasons described above
15      if arg == 'START':                                             # User sets the arg to start printing
16          robot.setDO('$OUT[16]', 'True')                            # Set the output value to start printing
17      elif arg == 'STOP':                                            # User sets the arg to stop printing
18          robot.setDO('$OUT[16]', 'False')                           # Set the output value to stop printing
19      else:
20          print('The input of extruder() has to be either START or STOP')
21
22  def parser(filename):
23      '''Parse gcode commands to be used in our code'''
24      parsed_file = list()                                           # Create a list
25      layer = 0                                                      # Z-axis, or current height of the print
26      with open(filename) as gcode:                                  # Open gcode path and use as variable
27          for line in gcode:                                         # For each line in gcode:
28              line = line.strip()                                    # Delete all unnecessary ASCII characters
29              if re.findall("LAYER:", line):                         # If new layer found, update z-axes
30                  layer += 1                                         # Add the layer value
31                  continue                                           # Skip the next if
32              if re.findall("G1", line):                             # Check if line begins with "G1"
33                  coord = re.findall(r'[XY].?\d+.\d+', line)         # Assign coord values
34                  if len(coord) == 2:                                # If both coords were assigned
35                      X = re.findall('\d*\.?\d+', coord[0])[0]       # Assign X-value
36                      Y = re.findall('\d*\.?\d+', coord[1])[0]       # Assign Y-value
37                      parsed_file.append([float(X), float(Y), layer, True])   # Append our desired output for use in the project
38      return parsed_file
39
40
41  ########################################################## MAIN ######################################################################
42
43  # ROBODK INITIALIZATION
44  RDK = Robolink()                                                   # sys.path.insert(0, "C:\ProgramFiles\RoboDK\Python")
45  robot = RDK.Item('KUKA KR 6 R700 sixx')                            # Create the robot instance
46  RDK.setSimulationSpeed(1)                                          # ADJUSTABLE: Simulation speed (1=default)
47  reference = robot.Parent()                                         # Retrieve the robot reference frame
48  robot.setPoseFrame(reference)                                      # Use the robot base frame as active reference
49  home = [0, -90, 90, 0, 0, 0]                                       # Setup the home joint position
50  robot.MoveJ(home)                                                  # Move to home in RoboDK
51  robot.setSpeed(speed_linear=100)                                   # ADJUSTABLE: Change the printing speed here
52  layer_height = 0.55                                                # ADJUSTABLE: edit the layer height
53  path = parser('desired/path/to/gcode') # ADJUSTABLE: Parse the gcode
54  item_frame = RDK.Item('task_frame')                                # ADJUSTABLE Load the task_frame from RoboDK
55  home_joints = [-5.520000, -107.550000, 115.910000, 1.620000,
56                 37.390000, -0.2400006]                              # ADJUSTABLE Orient robot with joints to P1
57  robot.setFrame(item_frame)                                         # Set the robot frame
58  robot.MoveJ(home_joints)                                           # Move to home position (First move must be in joints)
59
60
61  # TOOL ORIENTATION + INVERSE KINEMATICS
62  orient_frame2tool = invH(item_frame.Pose())*robot.SolveFK(home_joints)  # Homogeneous matrix * joint orientation
63  orient_frame2tool[0:3, 3] = Mat([0, 0, 0])                             # Remove the last column of homogeneous matrix
64
65
66  # TASK_FRAME CHECK
67  robot.MoveL(transl([0, 0, 0])*orient_frame2tool)                   # Move to the first corner
68  robot.MoveL(transl([0, 70, 0])*orient_frame2tool)                  # Move to the second corner
69  robot.MoveL(transl([70, 70, 0])*orient_frame2tool)                 # Move to the third corner
70  robot.MoveL(transl([70, 0, 0])*orient_frame2tool)                  # Move to the forth corner
71  robot.MoveL(transl([0, 0, 0])*orient_frame2tool)                   # Move back to the start
72
73
74  # GET RID OF EXTRA FILAMENT (CAN BE REMOVED)
75  robot.MoveL(transl([0, 0, 0])*orient_frame2tool)                   # Printing blob
76  time.sleep(2)                                                      # Used to leave the blob
77  robot.MoveL(transl([0, 0, 3])*orient_frame2tool)                   # Move up to get rid of the filament
78
79
80  # PRINTING
81  RDK.setSimulationSpeed(1)                                          # ADJUSTABLE, set to default speed
82  printing_status = False                                            # Default value for printing status
83  for item in path:                                                  # Go through each coordinate from the parsed GCode file
84      if item[3] == False and printing_status == True:               # If we want to stop extrusion which is running right now
85          extruder_offline(robot, "STOP")                            # Function to stop extrusion is called
86          printing_status = False                                    # Set the printing status to false
87      elif item[3] == True and printing_status == False:             # If we want to start printing and we are not
88          extruder_offline(robot, "START")                           # Call the function to start extruding
89          printing_status = True                                     # Set the printing status to true
90      target_point = [item[0], item[1], layer_height*item[2]+0.15]   # ADJSUTABLE, call the coord to go to, 0.55 is the layer heigh
91      target0 = transl(target_point)*orient_frame2tool               # Set the target point with correct rotation of the extruder
92      robot.MoveL(target0, blocking=False)                           # Move the robot to target
93  robot.MoveJ(home)                                                  # Move back home at the end of program
```



(a) Running simulation (b) Finished Simulation

Figure 2: Robot Simulation inside RoboDK with the line trace activated.

3.2.3 Method Summary

- First initialization is slow
- Slow parameters changes as we have to generate and copy paste new file each time
- High dependency on RoboDK platform
+ Constant movement of the end-effector
+ Speed of the robot can be changed very easily in the Python script as robot.setSpeed
+ General approach of computing task frame by algebraic theory and properties
+ Simulation availability without the need for a physical robot
+ The script can simulate the cooperation between multiple robots

3.3 Method 3: Offline Programming v2

This method is the best performing one and is subjected to be primarily used. This method and the previous one are both using the offline programming approach. That means that a USB Driver must be plugged into a KUKA controller to execute the program at the real robot.

While the first method calibrates the task frame in the RoboDK, in this method, the task frame is calibrated using the RoboDK controller. Moreover, the graphic user interface and the export function to create the file executable on the KUKA robot were added. All wrapped up into a single file, which does every task in one run.

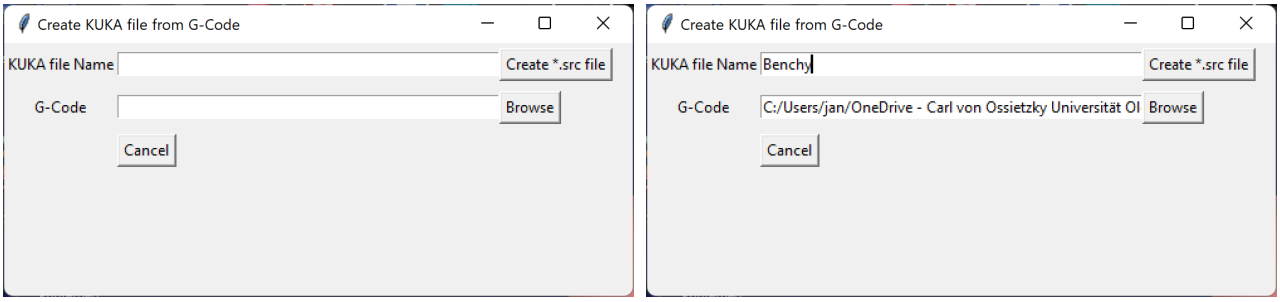
3.3.1 User manual

1. Open KUKA_3D_Printing_Simulation.rdk in RoboDK
2. Double-click on gcode_parser
3. Fill the GUI as stated below
4. Wait for the RoboDk simulation to finish and inspect the result
5. Copy paste the generated file to USB Disk
6. Plug the USB Stick in KUKA controller
7. Switch to Expert user profile
8. Copy-paste the code from USB Driver to memory of KUKA Controller
9. Run the TCP calibration using the 4 points method [2]
10. Setup the task frame by running 3 point method

[2]For more info visit the Section Running KUKA Program

11. Run the air movement test (without printing)
12. Check the safety measures and close the doors
13. Turn the key to change the mode to AUT at the KUKA controller
14. Run the program

If the entire script is run from RoboDK, a window for .gcode file selection and .src file creation opens. The window can be seen in Figure 3. Firstly, by pressing the '*Browse*' button, the .gcode file can be selected using File Manager. After naming the KUKA file, the script is executed by pressing '*Create *.src file*'. Once the simulation has run, the .src file will be saved as the given name or as '`myfile.src`' if no filename was given.



(a) Empty GUI Window (b) Filled GUI Window

Figure 3: The GUI Window for the file selection and creation

3.3.2 Code Description

Used files:

- KUKA_3D_Printing_Simulation.rdk
- gcode_parser.py (included in KUKA_3D_Printing_Simulation.rdk file and described by Method 2)

The simulation in RoboDK is based on the results from section 2.2. For this file the `robolink` and `robodk` libraries from the `Python` API are used. The simulation is then started by the code below, where lines 185, 188, 192 and 196 define the robot and its home position. Line 197 preallocates the current position being the home position as `pose` for later manipulation. In lines 202-206, the script loops through coordinates called `p_code`, which were created by calling the parser from Section 2.2.

Firstly we check whether the current coordinate represents the print or travel movement. Only the coordinates where the extrusion should be on are simulated [3]. Then the X, Y and Z coordinates get stored in `pose`, using the `pose.setPos()` method. Finally, the position of the robot is reset to the `home` position after the loop is finished.

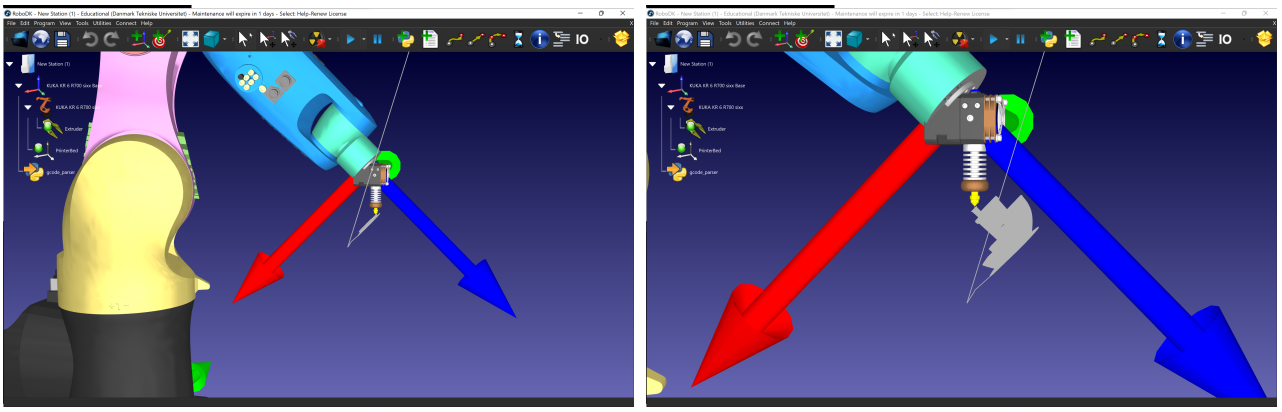
```
184 # Creation of the Robolink() object 'RDK'
185 RDK = Robolink()
186
187 # Define which robot is being used
188 robot = RDK.Item('KUKA KR 6 R700 sixx')
189
190 # set the home position of the robot
191 robot.setJoints([0.000000, -90.000000, 90.000000,
192                  0.000000, 45.000000, 0.000000])
193 home = robot.Joints()   # Save the home position
194 pose = robot.Pose()     # Preallocate the pose
195
```

[3] Compare to line 20

```
196 # Move to all the extruding coordinated in the simulation for a ...
197 # ...clearer visual of the path in RoboDK. The actual path will include ...
198 # ...waiting positions G0 from the gcode
199 for i in p_Code:
200     if i[0]:
201         pos = [i[1], i[2], i[3]]
202         pose.setPos(pos)
203         robot.MoveJ(pose)
204 robot.MoveJ(home)
```

A tool and a reference frame have to be set up inside RoboDK. In this simulation, the tool called '*Extruder*' and the reference frame called '*PrinterBed*' is chosen arbitrarily since the simulation will not be linked to the KUKA robot.

While the simulation is running, the line trace should be activated inside RoboDK to track the progress of the simulation and visualise the print. The progress of the simulation is shown in Figure 4.



(a) Running simulation (b) Finished Simulation

Figure 4: Robot Simulation inside RoboDK with the line trace activated.

3.3.3 Method Summary

- The correct task frame in RoboDK must be specified in extra steps
- Slow parameters changes as we have to generate and copy paste new file each time
+ Low dependency on RoboDK platform
+ Constant movement of the end-effector
+ Everything can be run from only one file
+ Generating the .src file is based on user export function
+ Method is well tested and working for a large scale models

4 Export KRL Script

As already covered in Section 2.2 and 3.3, the script `gcode_parser` also creates and stores a .src file which then can be transferred to the actual robot and be used to 3D print with it. Before a file can be created however, it has to be opened using the `'w'` argument for 'write'. This is implemented on lines 214 and 215, which open a file named `file_name` in the current working directory `os.getcwd()`.

4.1 The Header

The header of the source code stores essential parameters for the robot, and it takes care of the initialisation of the KUKA robot. The header is a combination of needed parameters in the code below, inspired by KUKA robot controller source codes.

On the line 218, the .src file gets defined by its name, `file_name`. Lines 220-228 initialise the robot. '`$ADVANCE=5`' in line 230 defines how many moves upfront the robot is supposed to consider while interpolating. Then, the extruder is switched off for safety reasons. Then, the robot moves to its home position with 100% velocity. Within the `;FOLD PTP HOME Vel= 100% DEFAULT` the tool and base, which are supposed to be used for the rest of the program, are defined in line 235: '`FDAT_ACT= TOOL_NO 11, BASE_NO 1, IPO_FRAME #BASE`'. This is crucial, as the robot will transform the gcode coordinates with respect to the task frame by itself.

After the setup, the robot shall move out of the possible forward kinematics singularity, the home position, to the first position of the gcode. There is 20mm space above the task frame in this movement, in order to allow for later slower and more precise movements. Once in position, the robot waits for the extruder to be ready, and it gets switched on in line 249.

```
212 # Open a new file with the name file_name in order to write the KRL
213 # script into, as my_file.
214 with open(os.getcwd()+file_name + file_name,
215           'w') as my_file:
216     # Write the header into the file.
217     my_file.write(
218         'DEF ' + file_name.replace('.src','') + ' ( )\n'
219         '\n'
220         ';FOLD INI;%{PE}\n'
221         ';FOLD BASISTECH INI\n'
222         'GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )\n'
223         'INTERRUPT ON 3 \n'
224         'BAS (#INITMOV,0 )\n'
225         ';ENDFOLD (BASISTECH INI)\n'
226         ';FOLD USER INI\n'
227         ';ENDFOLD (USER INI)\n'
228         ';ENDFOLD (INI)\n'
229         '\n'
230         '$ADVANCE=5'
231         '\n'
232         '$OUT[16]=FALSE\n'
233         ';FOLD PTP HOME  Vel= 100% DEFAULT\n'
234         '$BWDSTART= FALSE\n'
235         'FDAT_ACT= {TOOL_NO 11, BASE_NO 1, IPO_FRAME #BASE}\n'
236         'BAS(#PTP_PARAMS,100 )\n'
237         '$H_POS=XHOME\n'
238         'PTP XHOME\n'
239         ';ENDFOLD\n'
240         '\n'
241         'PTP {X ' + str(p_Code[1][1]) + ',Y ' + str(p_Code[1][2]) + ',Z '
242         + str(p_Code[1][3]+20) + ',A 0,B 135,C 0}\n'
243         '$VEL.CP = 0.07\n'
244     )
245
246     my_file.write('WAIT FOR ($IN[10])\n'    # wait for
247                               # the hotend to heat up and start the
```

```
248                             # extrusion.
249                     '$OUT[16]=TRUE\n' # Start the extrusion.
250                 )
```

4.2 The Movements

In the next part, the movements are written to the .src file as a `LIN` movement command, which means the robot will move from one point to another on a direct line. The `my_file.write('LIN X '+str(i[1])+',Y '+str(i[2])+',Z ' + str(i[3])+' C_VEL \n')` command defines the linear movement to the next position, while `C_VEL` interpolates the robots velocity to be constant. This is important, because many close points will slow down the movement to a significant degree otherwise.

Since there are `G0` Instructions in many gcodes, a way must be found to deal with the interruption of extrusion from the extruders. Due to the nature of this assignment the extruders and their behaviour were not to be altered. When the extruder is switched off by '`$OUT[16]=FALSE`' it not only stops extruding, but also retracts the filament by roughly 2cm. The problem this poses is not too significant, but has to be considered.

The extruders have a relatively slow extrusion rate, when compared to the retraction rate, also the retraction of both extruders is different. This means every time the filament was retracted the robot has to wait about 2.2s before moving again, or there will be no material in the nozzle while it tries to print. This problem gives rise to *three* solutions for handeling `G0` instructions:

Waiting for the filament to reach the nozzle after retraction is the first most proper option that can be considered. For this it is needed to know whether there is extrusion right now or not, so a variable `extruding = 0` is preallocated.[4] Whenever a line is written to the .src file it is checked, whether extrusion should happen, and if it is happening right now.[5] In the case that nothing should change only the coordinate is passed to the file in the `LIN` instruction. If extrusion is supposed to stop the output 16 is written to `FALSE` by line 276. If extrusion was to restart again the robot waits for the signal, that the extruder is ready to print, as in line 257: '`WAIT FOR ($IN[10])`', then switches on the extruder and waits another 2.1s for the filament to reach the nozzle again.

In the case of our standard test model the '3D_Benchy' from Thingiverse this results in 2668 stops, where the robot just waits 2.1s resulting in a total waiting time of 5602.8s = 93:22min, which is roughly 3 times the printing time without any pauses, which was 33min.

```
251     extruding = 0   # Establish a variable, to check whether extrusion
252                     # is happening or not.
253
254     for i in p_Code:    # Iterate through the positions list of lists
255         if i[0] == 1:   # If extrusion should happen
256             if extruding == 0:  # and it is not happening right now,
257                 my_file.write('WAIT FOR ($IN[10])\n'    # wait for
258                               # the hotend to heat up and start the
259                               # extrusion.
260                               '$OUT[16]=TRUE\n'
261                               'WAIT SEC 2.1\n'
262                               )
263                 extruding = 1   # Set the extrusion variable to 1.
264
265             # Write the according line for a liner move to the next
266             # postition into the .src file.
267             my_file.write('LIN {X '+str(i[1])+',Y '+str(i[2])+',Z '
268                           + str(i[3])+'} C_VEL\n')  # C_VEL inter-
269                 # polates for constand velocity, while C_DIS tries
```

[4] compare to line 251
[5] compare to lines 256, 257, 274 and 275

```
270                 # to achieve eqal distance between path points
271
272
273         # if extrusion is supposed to stop, it will be stopped here.
274         elif i[0] == 0:
275             if extruding == 1:
276                 my_file.write('$OUT[16]=FALSE\n')
277                 my_file.write('LIN {X '+str(i[1])+',Y '+str(i[2])+',Z '
278                             + str(i[3])+'} C_VEL\n')
279                 extruding = 0
280             elif extruding == 0:
281             # Then the move will be written to the .src file.
282                 my_file.write('LIN {X '+str(i[1])+',Y '+str(i[2])+',Z '
283                           + str(i[3])+'} C_VEL\n')
284     # In the end of the program stop the extruder and return to the
285     # home position.
286     my_file.write('$OUT[16]=FALSE\n'
287                   'PTP XHOME\n'
288                   'END')
289
290 print('Done')
```

Accelerating whenever there is not supposed to be a line printed, because of a `G0` instruction in the gcode, it could also be a viable option to change the movement speed to a higher value in order to achieve a thinner stream of molten filament, or to rip off the filament stream. This could be used to print very thinly while there should not be a print as a compromise between solid strings and a proper print.

For this instead of switching off the extruder the speed gets set to 100% by '`$VEL.CP = 1`' from line 271 for `G0` and back to 10% in line 257.

This option also proofed impractical, as there still is enough material dripping out of the nozzle, such that it can bunch up and harden in places it is not supposed to be. By this unforeseen elevations in the print occur and the nozzle gets caught in those causing the robot to remove the print from the printer bed. Thus the prints do not turn out well.

```
251     extruding = 0   # Establish a variable, to check whether extrusion
252                     # is happening or not.
253
254     for i in p_Code:    # Iterate through the positions list of lists
255         if i[0] == 1:   # If extrusion should happen
256             if extruding == 0:  # and it is not happening right now,
257                 my_file.write('$VEL.CP = 0.07\n')
258                 extruding = 1   # Set the extrusion variable to 1.
259
260             # Write the according line for a liner move to the next
261             # postition into the .src file.
262             my_file.write('LIN {X '+str(i[1])+',Y '+str(i[2])+',Z '
263                           + str(i[3])+'} C_VEL\n')  # C_VEL inter-
264             # polates for constand velocity, while C_DIS tries
265             # to achieve eqal distance between path points
266
267
268         # if extrusion is supposed to stop, it will be stopped here.
269         elif i[0] == 0:
```

```
            if extruding == 1:
                my_file.write('$VEL.CP = 1\n')
                my_file.write('LIN {X '+str(i[1])+',Y '+str(i[2])+',Z '
                            + str(i[3])+'} C_VEL\n')
                extruding = 0
            elif extruding == 0:
            # Then the move will be written to the .src file.
                my_file.write('LIN {X '+str(i[1])+',Y '+str(i[2])+',Z '
                          + str(i[3])+'} C_VEL\n')
    # In the end of the program stop the extruder and return to the
    # home position.
    my_file.write('$OUT[16]=FALSE\n'
                  'PTP XHOME\n'
                  'END')

print('Done')
```

Ignoring the `G0` Instructions all together proofed to be the most reliable and satisfactory solution. The extruder gets switched on in the beginning of the program and switched off in the end. Although this does not result in the cleanest print for some models, it does the most reliable job in printing with acceptable quality, speed and precision. In this approach the only thing that needs to be checked is if the first component of each line is equal to 1 and otherwise ignore it.

```
    my_file.write('WAIT FOR ($IN[10])\n'    # wait for
                              # the hotend to heat up and start the
                              # extrusion.
                              '$OUT[16]=TRUE\n'
                              )

    for i in p_Code:    # Iterate through the positions list of lists
        if i[0] == 1:   # If extrusion should happen

            # Write the according line for a liner move to the next
            # postition into the .src file.
            my_file.write('LIN {X '+str(i[1])+',Y '+str(i[2])+',Z '
                          + str(i[3])+'} C_VEL\n')  # C_VEL inter-
                # polates for constand velocity, while C_DIS tries
                # to achieve eqal distance between path points

    # In the end of the program stop the extruder and return to the
    # home position.
    my_file.write('$OUT[16]=FALSE\n'
                  'PTP XHOME\n'
                  'END')

print('Done')
```

4.3 Gcode to KRL

In this section, the exact translation from a few lines of gcode into a few lines of the .src file written in the KRL shall be shown. Below is the beginning of a chosen gcode file. When the file is read by the `Python` code lines 1-27 get completely ignored, because they do not contain any of the letters 'G', 'X', 'Y' or 'Z' immediately followed by a number including its decimal places. Also, all other lines not fitting this criterion get ignored.

```
;START_OF_HEADER
;HEADER_VERSION:0.1
;FLAVOR:Griffin
;GENERATOR.NAME:Cura_SteamEngine
;GENERATOR.VERSION:4.12.0
;GENERATOR.BUILD_DATE:2021-11-09
;TARGET_MACHINE.NAME:Ultimaker 3 Extended
;EXTRUDER_TRAIN.0.INITIAL_TEMPERATURE:215
;EXTRUDER_TRAIN.0.MATERIAL.VOLUME_USED:78266
;EXTRUDER_TRAIN.0.MATERIAL.GUID:506c9f0d-e3aa-4bd4-b2d2-23e2425b1aa9
;EXTRUDER_TRAIN.0.NOZZLE.DIAMETER:0.8
;EXTRUDER_TRAIN.0.NOZZLE.NAME:AA 0.8
;BUILD_PLATE.TYPE:glass
;BUILD_PLATE.INITIAL_TEMPERATURE:60
;PRINT.TIME:7451
;PRINT.GROUPS:1
;PRINT.SIZE.MIN.X:9
;PRINT.SIZE.MIN.Y:6
;PRINT.SIZE.MIN.Z:0.4
;PRINT.SIZE.MAX.X:97.132
;PRINT.SIZE.MAX.Y:107.695
;PRINT.SIZE.MAX.Z:70
;END_OF_HEADER
;Generated with Cura_SteamEngine 4.12.0
T0
M82 ;absolute extrusion mode

G92 E0
M109 S215
G0 F15000 X9 Y6 Z2
G280
G1 F1500 E-5
;LAYER_COUNT:176
;LAYER:0
M107
M204 S625
M205 X30 Y30
G1 F600 Z2.4
G0 F6666.7 X30.635 Y33.996 Z2.4
M204 S500
M205 X25 Y25
;TYPE:SKIRT
G1 F600 Z0.4
G1 F900 E0
G1 F1200 X31.24 Y33.539 E0.04564
G1 X31.89 Y33.148 E0.0913
...
```

The lines below are the lines left from this first filter. After this the coordinates get extracted from these lines and stored in a list of lists.

```
G92 E0
G0 F15000 X9 Y6 Z2
```

```
G280
G1 F1500 E-5
G1 F600 Z2.4
G0 F6666.7 X30.635 Y33.996 Z2.4
G1 F600 Z0.4
G1 F900 E0
G1 F1200 X31.24 Y33.539 E0.04564
G1 X31.89 Y33.148 E0.0913
...
```

The coordinates are stored in a list of lists of strings as shown below, from this coordinates, another array containing all coordinates is formed, such that if one value is `None` it will be taken from the last time it was defined. Also all entries, where the first value is not 0 or 1 will be removed.

```
pcode = [['92'],    [''],       [''],       ['']
         ['0'],     ['9'],      ['6'],      ['2']
         ['280'],   [''],       [''],       ['']
         ['1'],     [''],       [''],       ['']
         ['1'],     [''],       [''],       ['0.4']
         ['0'],     ['30.635'], ['33.996'], ['2.4']
         ['1'],     [''],       [''],       ['0.4']
         ['1'],     [''],       [''],       ['']
         ['1'],     ['31.24'],  ['33.539'], ['']
         ['1'],     ['31.89'],  ['33.148'], ['']
         ...
        ]
```

The result of this extrusion of coordinates is a list as shown below:

```
pcode = [[0.0],     [9.0],      [6.0],      [2.0]
         [1.0],     [9.0],      [6.0],      [2.0]
         [1.0],     [9.0],      [6.0],      [0.4]
         [0.0],     [30.635],   [33.996],   [2.4]
         [1.0],     [30.635],   [33.996],   [0.4]
         [1.0],     [30.635],   [33.996],   [0.4]
         [1.0],     [31.24],    [33.539],   [0.4]
         [1.0],     [31.89],    [33.148],   [0.4]
         ...
        ]
```

From this, all the lines with a 0 value in the first place will also be excluded since all travel paths (`G0` instructions) will be ignored due to the problems covered above. These remaining coordinates will be written into the `LINX ,Y ,Z C_VEL` command of the .src file as follows:

```
LIN{X 9.0,Y 9.0,Z 2.0} C\_VEL
LIN{X 9.0,Y 9.0,Z 0.4} C\_VEL
LIN{X 30.635,Y 33.996,Z 0.4} C\_VEL
LIN{X 30.635,Y 33.996,Z 0.4} C\_VEL
LIN{X 30.635,Y 33.996,Z 0.4} C\_VEL
LIN{X 31.24,Y 33.539,Z 0.4} C\_VEL
LIN{X 31.89,Y 33.148,Z 0.4} C\
...
```

The `LIN` command instructs the robot to move to the next point on a line in Cartesian space, as opposed to the `PTP` command, which operates in joint space. The addition of `C_VEL` makes sure that the velocity is constant while printing because the path gets interpolated based on a constant speed.

5 Tool Modelling

When designing the tool holder, many things such as the shape, strength, durability and ease of manufacturing were considered. When choosing the angle between the robot's flange and the extruder, a 45° angle was chosen to maximise the robot's working space, ensuring that the robot arm does not collide with the table, printer bed, or print itself.

All needed dimensions of the tool holder were measured before the modelling. The final design is shown in Figure 5. The connection between the base and holder is thicker for better stabilisation.

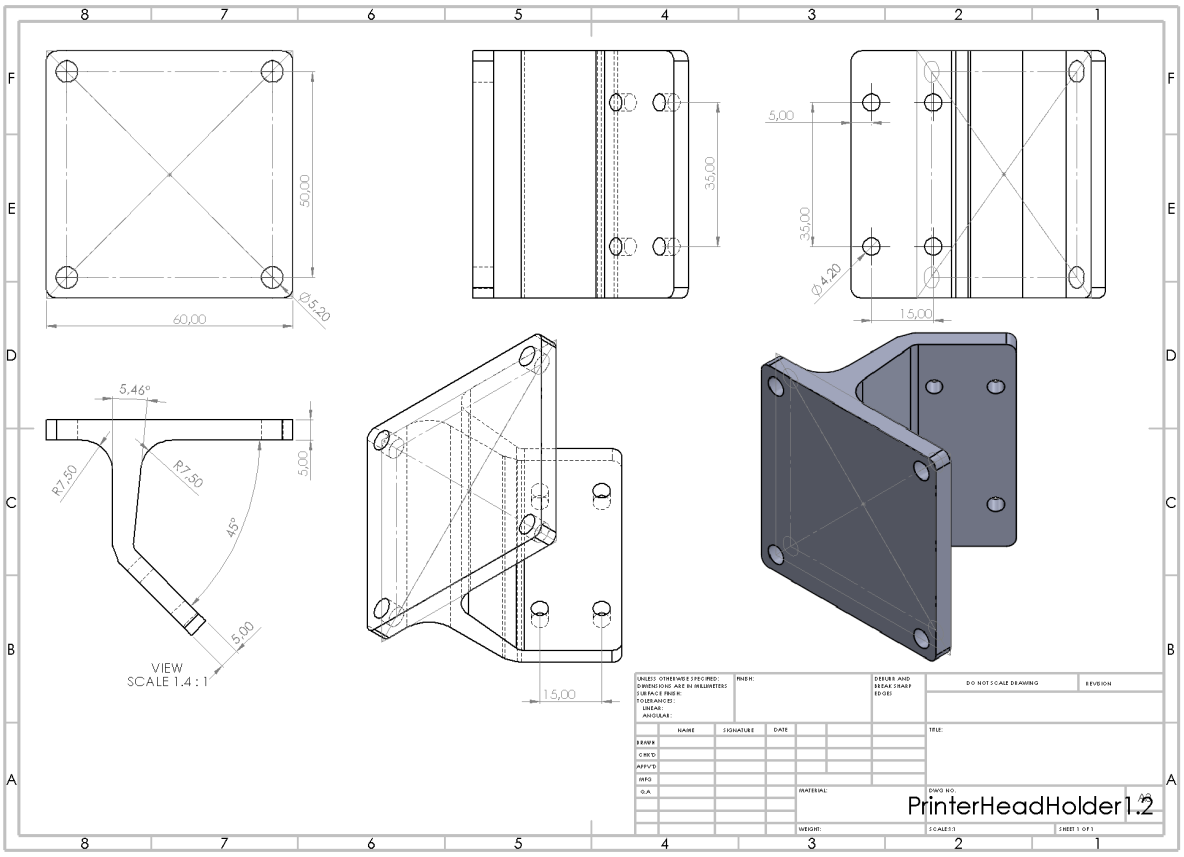


Figure 5: Drawing of the tool holder Design.

The final 3D print was on the Ultimaker Cura 3 Extended 3D printer. Multiple layer heights were tried, and the layer height around 0.15mm with an infill density of 60% provides the best time/quality ratio. The tool was mounted using the eight screws and TCP calibrated before the print.

6 Running KUKA Program

Once the KRL code is written into the .src file correctly, the robot has to be set into expert mode using the panel and the password: *kuka*. After that, the .src file can be copy-pasted off the USB drive onto the robots system. The program cannot be executed without moving the file from the USB drive to the controller memory.

Before running the script, it is highly recommended to calibrate the tool inside the KUKA using the TCP 4 point calibration. Also, the base has to be defined using the 3 point method in the KUKA controller. In this assignment base 1 and tool 11 have been used to store the data.

The first try should always be run as the Air movement test using the settings T0 or T1 with a languid speed movement and with the extruder turned off. If this test is run correctly, we can close the doors, check all the security terms, turn the key to switch into AUT mode, select the program and finally run the script.

7 3D Printing

If the program is run as described in Section 6 and the robot starts printing. Two main parameters are important for printing: the printing speed and the layer height. The layer height can be set during the slicing or inside the file. The printing speed is adjusted through trial and error in the lab. Running multiple tests and considering not only the printing speed but also the layer height, we concluded that speeds of `VEL.CP = 0.08` to `VEL.CP = 0.07` are feasible, and a layer height of 0.4mm has been proven to be a good setting. Since only the method ignoring all the `G0` instructions is consistent and acceptable in results, there is extra material on the print, which can be removed, depending on the 3D model that is being printed. The development of the print is shown in Figure 6.

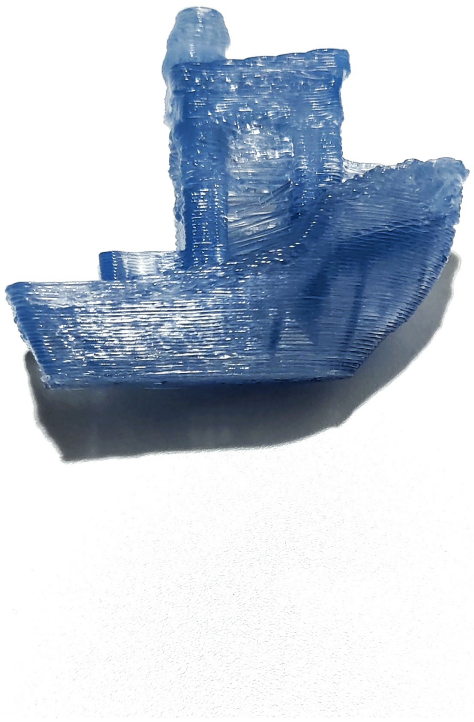


Figure 6: Resulting 3D print

Figure 7: Resulting 3D print

8 Conclusion

By several methods, we have described the possibility of a KUKA robot printing 3D objects and tested the process with a great success. We have however found the limitations and obstacles associated with using an industrial robot for such a task.

Firstly, the robot must be manually calibrated and working frame must be created within every slight change of the work space. Secondly, the starting and stopping extrusion command is executed slowly, resulting in a time inefficient printing process. The limitations in print quality is moreover determined mainly by the extruder design, that could be impoved by using a narrower nozzle and better retraction and extrusion control. Last but not least, the online programming approach is widely limited by third party software and it's API. Using RoboDK works poorly for 3D printing, as the implementation of end-effector's constant speed is missing.

On the other hand, using a six joint industrial robot opens up new opportunities, such as printing on uneven surfaces or on the walls of parts that have been printed earlier. Also, completely new objects, which were rather complicated or even impossible to print, could be printable using an industrial robot such as the KUKA instead of a traditional CNC 3 axis extruder. With more time and a slicer made solely for this purpose, prints' stability and surface quality could also be improved.