Expertise
and insight
for the future

Tom-Ilia Tsereh

3D Printing with a 6-axis Robot: Trajectory Generation and Printer Head Command Using ROS

Metropolia University of Applied Sciences

Bachelor of Engineering

Electrical and automation engineering

Bachelor's Thesis

25 May 2020

Author Title Number of Pages Date	Tom-Ilia Tsereh 3D Printing with a 6-axis Robot: Trajectory Generation and Printer Head Command Using ROS 54 pages + 6 appendices 25 May 2020
Degree	Bachelor of Engineering
Degree Programme	Electrical and automation engineering
Professional Major	Automation technology
Instructors	Marc Kunze, Professor Timo Tuominen, Senior Lecturer

To follow digitalization, small and medium sized enterprises are in need for affordable, multiprocess robotic solutions for manufacturing in reconfigurable workshop spaces. This project's long term goal is developing a mobile, self-referencing industrial robotic platform for additive/subtractive manufacturing of composites structures.

This project is an extension of a David Rüegg's bachelor thesis work and is a step with respect to the global project goal. It is based on ROS, an open-source Robot Operating System middleware. The robot used in this project is a Stäubli TX40, a 6 Degree Of Freedom (6-DOF) robotic arm. The attached tool is a custom made thermoplastic extractor. This thesis work has been made for Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD).

The goal of this thesis project was to develop forward what has been previously done, with the aim to make physical object printing possible using 6-DOF robotic arm. Foundation has been laid for making communication possible with the robot, using ROS middleware, and to be able to send few coordinate points to the robot for execution. Topic related to printing parts was not studied.

This thesis project dealt with researching a wide array of variables. Generating a G-code file which included commands for the robot and the custom printer from the CAD model. Creating and adjusting tools for the robot so it could execute trajectories properly in a defined place. Integrating previously unknown custom printer tools. Creating a program capable of communication via ROS-middleware, which commands the printer in synchronization with the robot.

The completed thesis work provides an opportunity for the future development, which is planned to be finished in September 2021.

Keywords	Additive Manufacturing, Robot Operating System, 6-DOF

Tekijä Otsikko Sivumäärä Aika	Tom-Ilia Tsereh 3D tulostaminen käyttäen 6-akselista käsivarsirobottia: liikeradan luominen ja tulostuspään käyttöönotto ROS ympäristössä. 54 sivua + 6 liitettä 25.5.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	sähkö- ja automaatiotekniikka
Ammatillinen pääaine	automaatiotekniikka
Ohjaajat	professori Marc Kunze lehtori Timo Tuominen

Pysyäkseen digitalisaation perässä pienet ja keskisuuret yritykset tarvitsevat edullisia, moniprosessisia robottiratkaisuja kappaleiden valmistukseen uudelleenkonfiguroitavissa työtiloissa. Tämän projektin pitkän aikavälin tavoitteena on kehittää mobiili teollisuusrobotti, joka pystyisi työstämään kappaleita käyttäen kolmiulotteista tulostusta tai lastuavia työstömenetelmiä. Opinnäytetyö on tehty HEIG-VD yliopistolle.

Opinnäytetyö oli jatko-osa David Rüeggin opinnäytetyölle ja on askel kohti suurempaa projektitavoitetta. Se perustui ROS:iin, avoimen lähdekoodin robottioperaatiojärjestelmään. Työssä käytettiin 6-akselista käsivarsirobottia, johon oli asennettu mukautettu työkalu kolmiulotteista tulostusta varten.

Opinnäytetyössä jatkettiin käsivarsirobotin konfigurointia, tutkittiin osien tulostamiseen liittyvää työkalua sekä luotiin sille ohjelma, jotta tulostin voisi vastaanottaa komentoja ROS-ympäristössä, sekä toimia käsivarsirobotin kanssa yhtäaikaisesti. Opinnäytetyössä käsiteltiin myös G-kooditiedoston luomista CAD-mallista, joka sisältää komentoja käsivarsirobotille ja kolmiulotteiselle tulostimelle. Lisäksi mukautettiin ohjelmistoa, jotta robotin liikeradat voidaan suorittaa oikein ja määrätyssä paikassa.

Opinnäytetyön tuloksena saatiin jatkokehitettyä tarvittavia työkaluja kolmiulotteista tulostusta varten, kerättiin lisää tietoa tulevaan kehitykseen sekä otettiin askel lähemmäs projektin lopputavoitetta. Valmis opinnäytetyö tarjoaa myös mahdollisuuden projektin tulevalle kehitykselle, jonka suunniteltu valmistumisaika on syyskuussa 2021.

Avainsanat	kolmiulotteinen tulostus, robottioperaatiojärjestelmä, 6-akselinen käsivarsirobotti

Contents

List of Abbreviations

1 Introduction 1

1.1 Background 1
1.2 Additive Manufacturing 2
1.3 Subtractive Manufacturing 2
1.4 Aim 3
1.5 Approach to the Problem 4

2 Hardware Introduction 6

2.1 Stäubli TX40 6
2.2 Thermoplastic Extractor 7
2.2.1 Controller 9

3 Software Introduction 10

3.1 Ubuntu 18.04 LTS Bionic Beaver 10
3.2 Robot Operating System (ROS) Melodic 10
3.2.1 ROS Computation Graph Level 11
3.2.1.1 Master 12
3.2.1.2 Node 12
3.2.1.3 Parameter Server 12
3.2.1.4 Messages 13
3.2.1.5 Topics 13
3.2.1.6 Services 13
3.2.1.7 Bags 14
3.2.2 ROS Filesystem Level 15
3.2.2.1 Workspace 15
3.2.2.2 Package 15
3.2.2.3 Metapackage 17

3.2.3 ROS Community Level 17
3.2.4 Graph and Package Resource Names 18
3.3 Pronterface 18
4 Implementation 19
4.1 Converting CAD Model to a G-code 20
4.2 Staubli TX40 22
4.2.1 Node descartes_manufacturing 22
4.2.2 Adjusting Generated G-code for the Robot 22
4.2.3 Setting Velocity Parameters 25
4.2.4 Understanding Unified Robot Description Format 27
4.2.4.1 Change Robot Model 30
4.2.4.2 Creating Reconfigurable Frame Where Trajectory Will Be Executed 31
4.2.5 Getting Robot Position Data for Defining a New Frame 35
4.3 Developing Thermoplastic Extractor 36
4.3.1 Marlin 36
4.3.2 Creating Printertool Node 37
4.3.2.1 Communication with the USB Port 38
4.3.2.2 Converting G-code Values 39
4.3.2.3 Synchronization with the Robot Trajectory and Sending G-code to the Printer 41
5 Application 42
6 Results, Future Development and Bugs 45
6.1 Results 45
6.2 Future Development 46
6.3 Bugs 47
7 Summary 49
References 51
Appendices
Appendix 1. Node Descartes_manufacturing
Appendix 2. Node staubli_tf_listener
Appendix 3. Node printer_printertool
Appendix 4. Source code mySerial.cpp

Appendix 5. Header file mySerial.h
Appendix 6. File tx40_macro.xacro

List of Abbreviations

6-DOF	Six degrees of freedom
AM	Additive manufacturing
CAD	Computer-aided-design
FDM	Fusion deposit modeling
RAMPS	RepRap Arduino Mega Pololu Shield
ROS	Robot operating system
SM	Subtractive manufacturing
TF	Package, that lets user keep track of multiple coordinate frames
URDF	Unified robot description format

1 Introduction

1.1 Background

To produce lightweight and high performance structures, it is popular to use composite materials. Usually manufacturing is time consuming, and many operations are performed manually. Automated composite manufacturing solutions already exists, but they are expensive because of their very high complexity and cost, and because of that they are only accessible to large companies. Purpose of this work is to develop affordable solution, which could be more accessible and flexible. This is a long term project, and previous research work has been done by David Ruegg. Ruegg's thesis dealt mainly with initializing robot to work with Robot Operating System, as well as he took the first steps related to the topic with the sending trajectory to the robot.

The subject of this thesis work was given by the School of Business and Engineering Vaud, HEIG-VD. It is a public university, located in Yverdon-les-Bains, Switzerland. It was created on August 1, 2004. It has around 2 000 students, and that is making HEIG-VD the largest branch of the University of Applied Sciences in Western Switzerland. HEIG-VD is actively participating in regional, national and international research and industry development covered by its teaching program. [1.]

1.2 Additive Manufacturing

Additive Manufacturing (AM), also known as 3D printing, is a process where three dimensional object are produced by adding material layer by layer. To reproduce physical object, AM uses computer-aided-design (CAD) model, and grow it one layer at a time. Each layer is built by melting or partially melting material on the surface. [2; 3.] AM process is shown in figure 1.

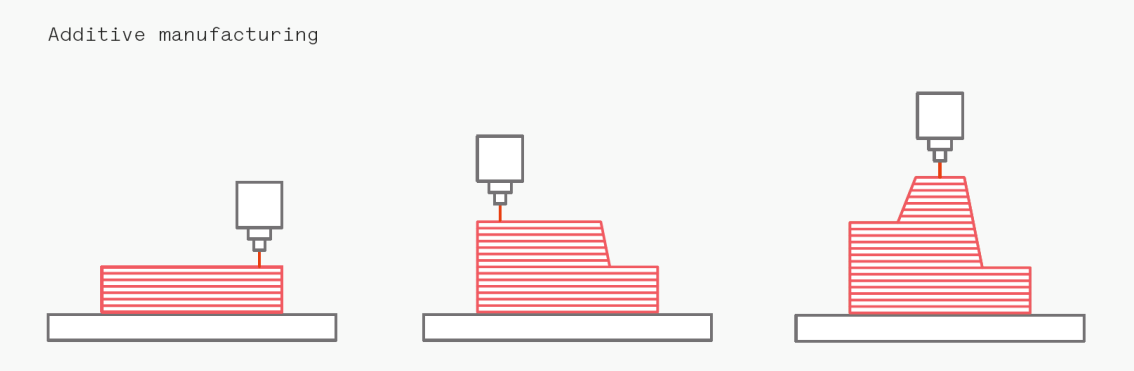


Figure 1. Pictorial representation of AM. [4.]

It is possible to use different types of material for layering, including metal, plastics, glass and even edibles. Commonly used materials for 3D printing are polylactic acid (PLA), polyethylene terephthalate (PET) and acrylonitrile butadiene styrene (ABS).

1.3 Subtractive Manufacturing

Subtractive manufacturing (SM) is a process, where three dimensional objects are constructed by successfully cutting material away from a solid block of material. SM is typically done with a CNC machine, but could be cut manually. [5.] SM process is shown in figure 2.

Figure 2. Pictorial representation of SM [6.]

One of the principal advantages of the SM over AM is ability to machine extremely thin piece of plastic.

1.4 Aim

The long term goal of this project is to develop more affordable and simple to use multi-process robotic solution for manufacturing in reconfigurable workshop spaces. Target is to create mobile, self-referencing industrial robotic platform, using 6-DOF robot with attached replaceable tool on it, mobile robotic platform, and optical positioning system in one machine. Proposed concept could be brought to the workshop floor and be integrated seamlessly into an existing workflow. Operators would be skilled workers, not necessarily engineers.

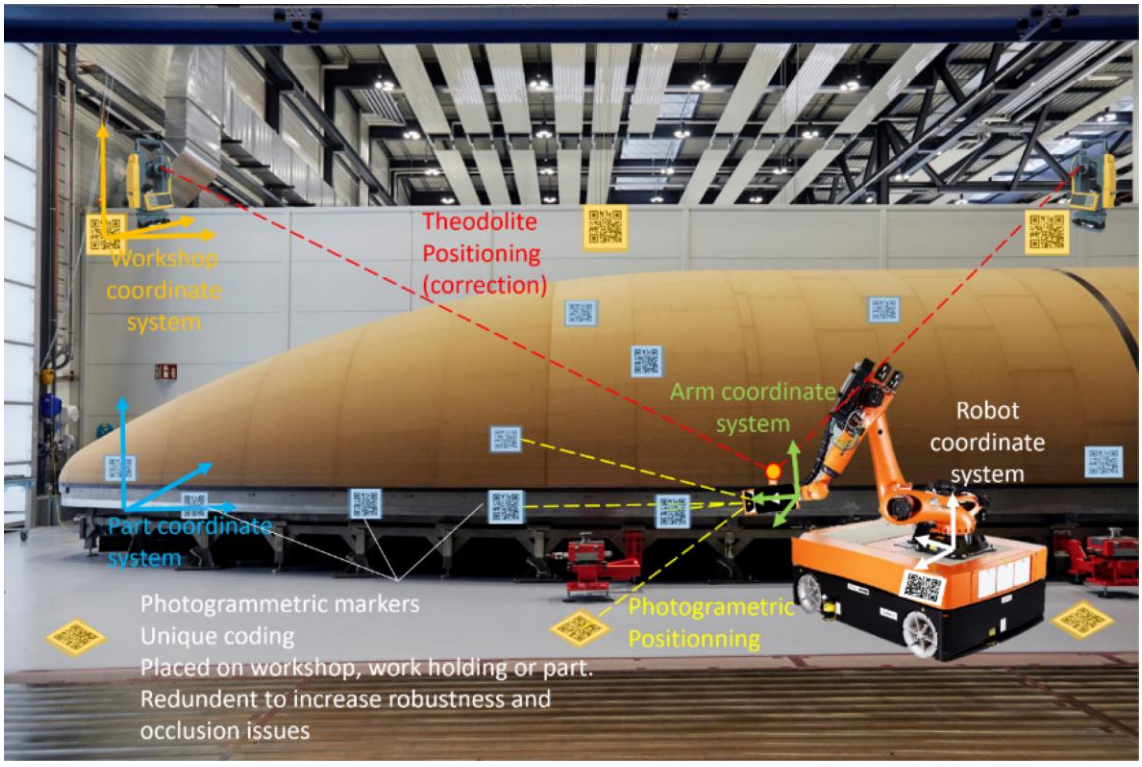


Figure 3. Pictorial presentation of estimated finished product. [7.]

Project is planned to be finished in September 2021, and for that reason everything cannot be accomplished in this thesis work.

Previously it was found, that ROS meets needs as a middleware. For a hardware is used Stäubli TX40 6-DOF robot hand with attached custom made AM printer. Idea is to recreate a 3D printer with previously mentioned method, and after that project could be moved forward by the next developers, so it can recognize different types of surface, be mobile and navigate by its own into desired location. Aim of this thesis work was to promote the project, by configuring the robot and thermoplastic extractor, with the target to make physical object printing possible.

1.5 Approach to the Problem

Basic 3D printers already exists, but they have own limitations, and does not meet the requirements of the task. Most common 3D printers are following cartesian coordinate path, whose three principal axes of control are linear, and they move in a straight line rather than rotate. Printers are built in a frame, and are successfully adding material layer by layer horizontally. With use of 6-DOF robot and mobile platform, it is possible to get rid of previously mentioned limits. It will not be so dependent of the placement and specifically, the body is free to change position and rotate.

Figure 4. Example of 3D printer: Ultimaker S5 Pro Bundle [8.]

This kind of topic seems not to be widely worked on, and none of the optimal solutions have not been found to use straightly with current method. For that reason, project is approached step by step, and researching what is the best solution to implement.

2 Hardware Introduction

2.1 Stäubli TX40

Stäubli TX40, which is a 6-DOF industrial robot, was used as a robotic arm. Robot controller was CS8C with a robotic language VAL3. Using this kind of robot provides an opportunity to rotate the tool around curved surfaces, and providing better work quality because of additional degrees of freedom. When 6-DOF robot is attached to the mobile platform, the working environment is no longer limited, and becomes almost infinite. Stäubli TX40 robotic arm is shown in figure 5.

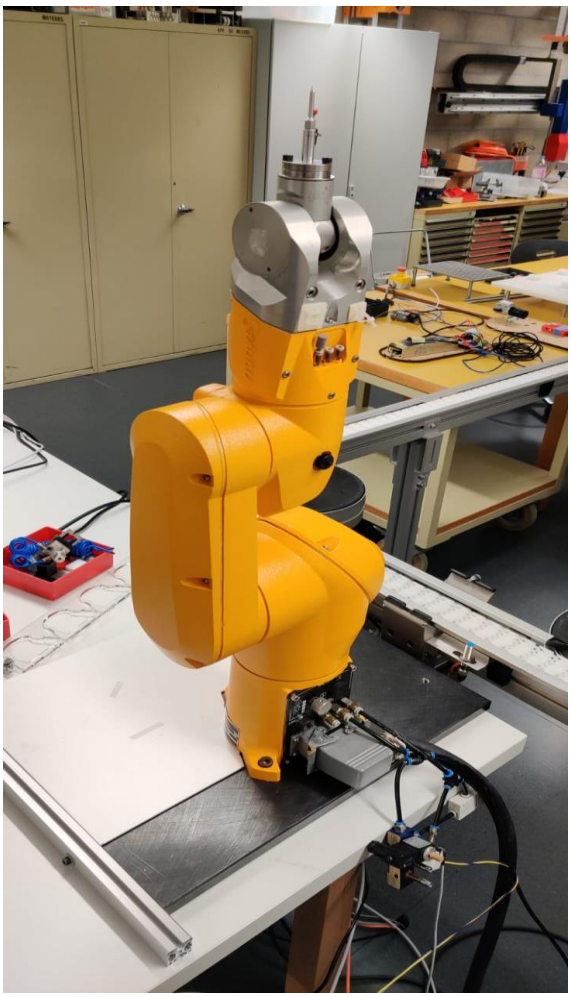


Figure 5. In the picture is shown Stäubli TX40 robot, without attachments related to printing.

2.2 Thermoplastic Extractor

Thermoplastic extractor has been created in same principle as in a typical 3D printers, based on fusion deposit modeling (FDM), which is an AM process. Principle of this fabrication process is that the filament is first loaded into the printer, and once the nozzle has reached desired temperature, extrusion head starts to feed filament to the nozzle, where it starts to melt. The molten material is applied to the predetermined surface layer by layer, where it cools and solidifies. [9.] FDM process is shown in figure 6.

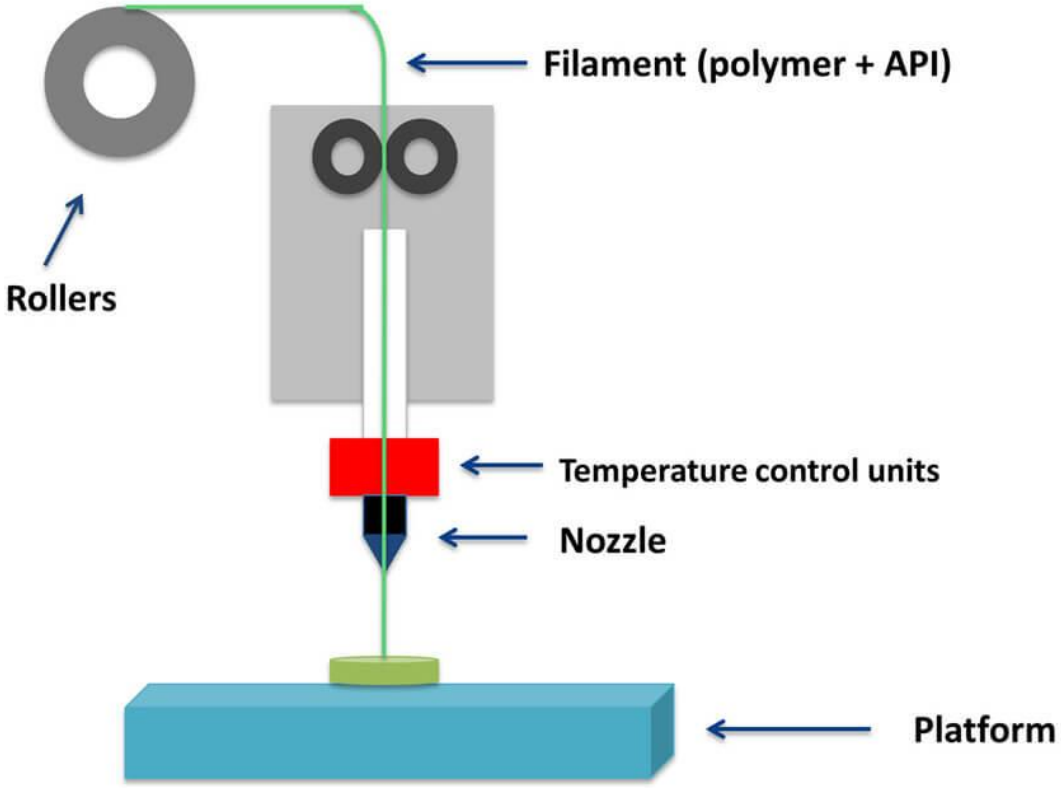


Figure 6. Pictorial presentation of FDM process. [10.]

Mounting to the robot has been accomplished to the original tool placement. Sometimes the cooling of the material is accelerated through the use of cooling fans. With the additional tool, there is an option to attach air pressure cooler around the nozzle as shown in figure 7.

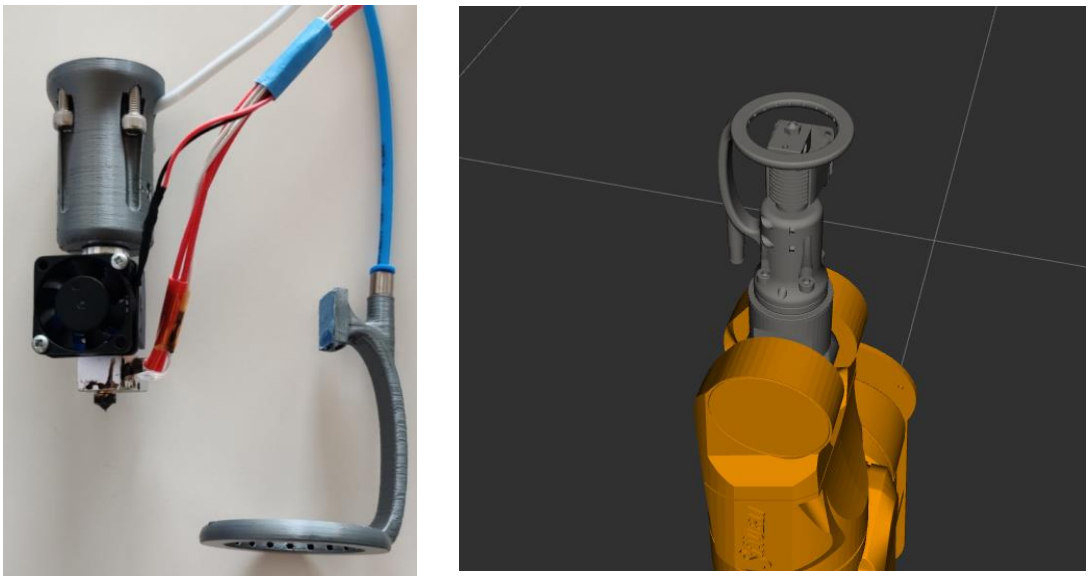


Figure 7. Left picture of thermoplastic extractor (left) and nozzle head cooler (right). Right picture is simulation of attached thermoplastic extractor to the Stäubli TX40.

Extruder and filament coil are mounted separately from the thermoplastic extractor in one plate. This is shown in fugure 8.

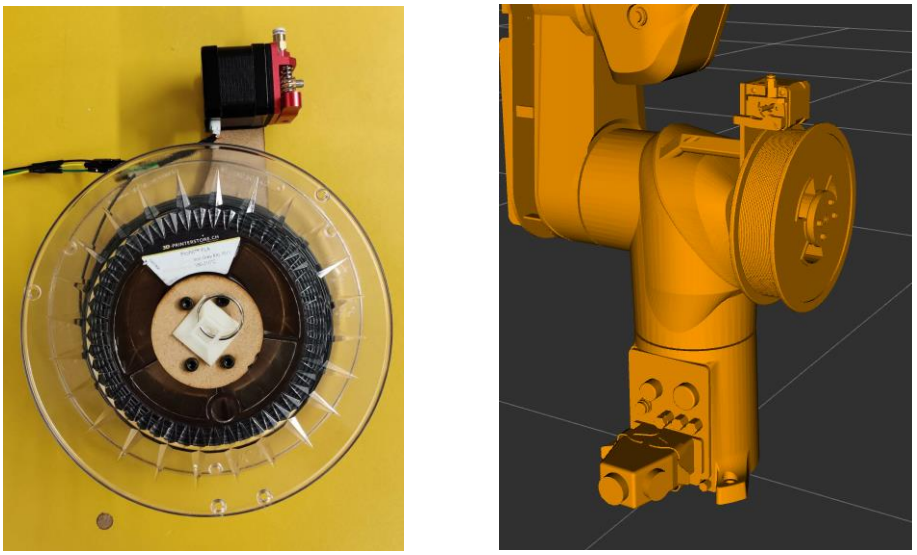


Figure 8. Mounting plate with extruder and filament coil at the left. Picture of attached parts from simulation at the right.

2.2.1 Controller

Printing is controlled by Arduino MEGA 2560 board, RepRap Arduino Mega Pololu Shield (RAMPS) 1.4 and DRV8825 stepper motor driver carrier. This assembly corresponds to the conventional custom made 3D printer. Inside is running Marlin 1.1.9 firmware, which is configurated for the attached parts. All this is powered by S-360-12 power supply. In the future development, RAMPS 1.4 offers different types of additions, for example adding displays and more extruders. Controller board and mounting to the robot is shown in figure 9.

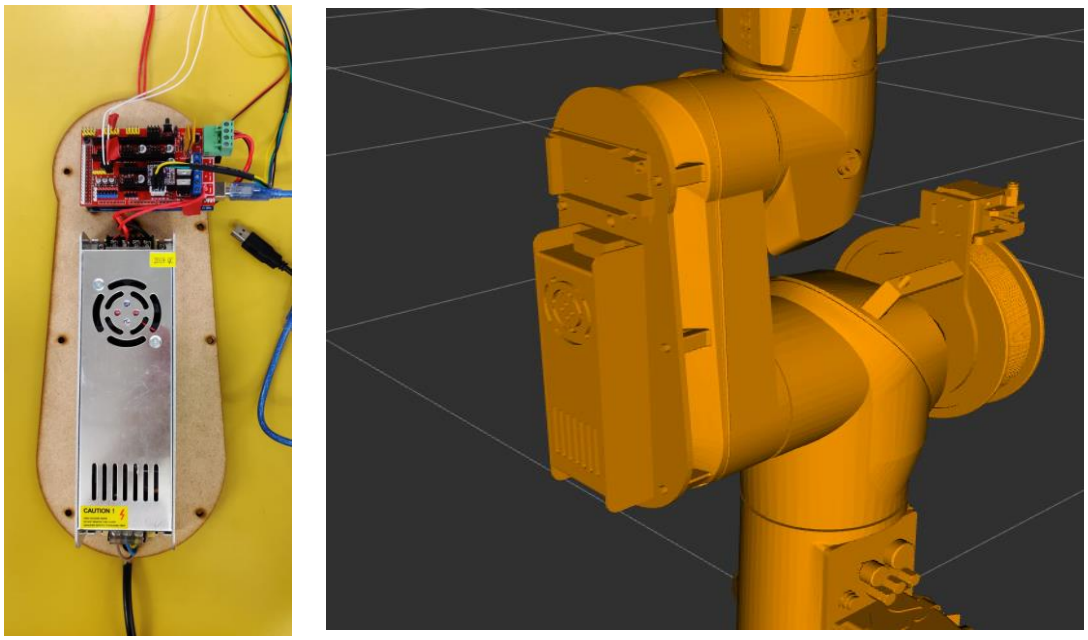


Figure 9. Controller board with Arduino MEGA 2560, RAMPS 1.4, stepper motor driver carrier and power supply at the left. Picture of attached parts from simulation at the right.

3 Software Introduction

3.1 Ubuntu 18.04 LTS Bionic Beaver

For a computer operating system Ubuntu 18.04 LTS Bionic Beaver was used. Ubuntu is a free and open-source Linux distribution, which is based on Debian. Ubuntu is a popular operating system for developing robots, and usually used in the ROS community. That is, because ROS chose Ubuntu as its primary supported platform. [11.]

3.2 Robot Operating System (ROS) Melodic

Creating truly robust, general-purpose robot software is hard, and for that reason ROS was built to encourage collaborative robotics software development. Although ROS is not an operating system, it is a robotic middleware, which is licensed under an open source, BSD license. It is a flexible framework for writing robot software, with a collection of tools, libraries, and conventions, whose purpose is to simplify task of creating complex robots across many different robotic environments. Processing takes place in nodes that may receive, post and multiplex sensor data, control, state, planning, actuator, and other messages. ROS is independent from programming language, and can work across such on C++, Python and Lisp. [12; 13.] Figure 10 shows ROS basis.

Figure 10. Pictorial presentation of ROS basis. [12.]

ROS basis are:

- Plumbing – provides publish-subscribe messaging infrastructure.
- Variety of tools, which allow to visualize and record data, navigate the ROS package structures, debugging, testing, create scripts and setup processes.
- Capabilities provided by packages and libraries.

- Ecosystem with tutorials, documentations and forum supported by developers around the world.

The software components are separated in nodes. Each node can be executed individually, and they have usually one single purpose. Nodes can communicate with each other, and are managed by the ROS master node - roscore. Simplified ROS communication environment is shown in figure 11.

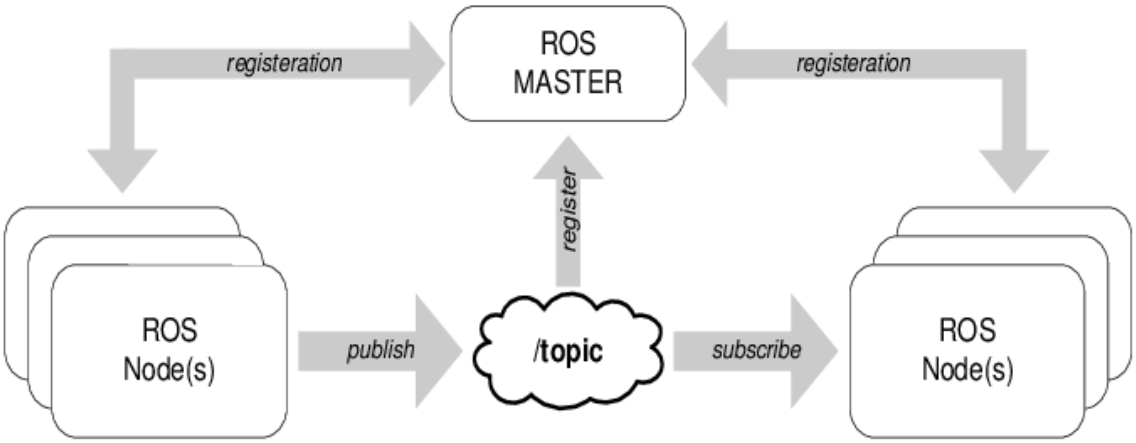


Figure 11. Simplified ROS communication environment. [14.]

3.2.1 ROS Computation Graph Level

The Computation Graph is peer to peer network, where ROS processes are processing data together. Basic concepts of ROS are Master, nodes, bags, topics, services, messages and Parameter Server. All of them provides data to the graph in different ways. [15.]

3.2.1.1 Master

The ROS Master is responsible for making individual nodes recognize each other. Once nodes have located each other, they can communicate with each other peer to peer. ROS Master keeps track on publishers and subscribers to topics as well as services.

Without Master running on background, communication will not work. Most commonly Master is run using roscore command in terminal. [16.] Running this command in terminal, it will load ROS Master along with other essential components:

```
$roscore
```

3.2.1.2 Node

A node is a process that performs computation. Robot control system usually involve many nodes. Nodes include different tasks, for example one node controls robot hand motors, one node controls attached tool to the robot, one node controls sensors, one node controls path planning and so on. Nodes are combined together into a graph and they can communicate with each other using Parameter Server, streaming topics and RPC services. [17.] Running nodes are executed with the following command:

```
$rosrun <package name> <executable name>
```

3.2.1.3 Parameter Server

The parameter server runs inside of the ROS Master. To store and retrieve parameters at runtime, nodes use this server. It is best used for static, non-binary data such as configuration parameters, as it is not designed for high performance actions. Parameter server is meant to be globally viewable, so that tools can easily inspect and modify configuration state of the system. Rosparam tool allows different settings from the command-line. [18; 19.] Currently supported commands are:

```
$ rosparam set    set parameter
$ rosparam get    get parameter
$ rosparam load   load parameters from file
$ rosparam dump   dump parameters to file
```

```
$ rosparam delete delete parameter
$ rosparam list   list parameter names
```

3.2.1.4 Messages

Nodes are communicating with each other by publishing messages to a topics, or subscribing messages from it. Message is a simple data structure, which types are integers, Booleans, floats etc. As part of ROS service call, nodes can also exchange request and response messages. Nodes can only exchange messages with same data type, which are using standard ROS naming conventions. [20.]

3.2.1.5 Topics

Nodes exchange messages over topics, which are named buses. Nodes are not aware of who they are communicating with, because they are interested in the data, which is published or subscribed in the topic. For clarification, nodes that are generating data to a topic are publishers, and nodes which are listening data from a topic are subscribers. Multiple nodes can publish and subscribe to a topic, and topics are intended for one way streaming communication. [21.]

rostopic is a command-line tool for interacting with ROS topics. For example:

```
$ rostopic list
```

will list the current topics and

```
$ rostopic echo /topic_name
```

will display Messages published to /topic_name.

3.2.1.6 Services

Nodes are using services to perform remote procedure calls, in other words receiving response to a request. It is another way to pass data between nodes, like with topics.

Requesting and replying is done via Service, which is defined by a pair of messages for each action. Providing ROS node is offering a service under a string name, and client is sending request message and awaiting for the reply. [22.]

The currently supported commands are: [23.]

```
$ rosservice call    call the service with the provided args
$ rosservice find    find services by service type
$ rosservice info    print information about service
$ rosservice list    list active services
$ rosservice type    print service type
$ rosservice uri     print service ROSRPC uri
```

3.2.1.7 Bags

Bags are used in ROS for storing message data from topics. Bags are often created by subscribing to a topic, and storing received message data. Variety of tools have been created to process, store, visualize and analyze them. [24.]

Command-line tool rosbag provides functionality: [25.]

```
$ rosbag record <topic-names>   Record a bag file with the contents of the
                                specified topics.

$ rosbag info <bag-files>       Display a summary of the contents of the bag
                                files.

$ rosbag play <bag-files>       Publish the contents of the given bags.
$ rosbag check <bag-file>       Determine whether or not a bag is playable in
                                the current system.


$ rosbag fix <in-bag> <out-bag> [rules.bmr]
Repairs a bag using registered rules (and optionally locally defined rules).

$ rosbag filter <in-bag> <out-bag> <expression>
Convert a bag file using the given Python expression.

$ rosbag compress <bag-files>         Compress one or more bag files.
$ rosbag decompress <bag-files>       Decompress the given bag files.
$ rosbag reindex <bag-files>          Reindex the given bag files.
```

3.2.2 ROS Filesystem Level

Typical ROS project is built in a catkin workspace. Workspace is a folder, including catkin packages. Workspace can be named as desired, and the typical structure is shown in the figure 12.

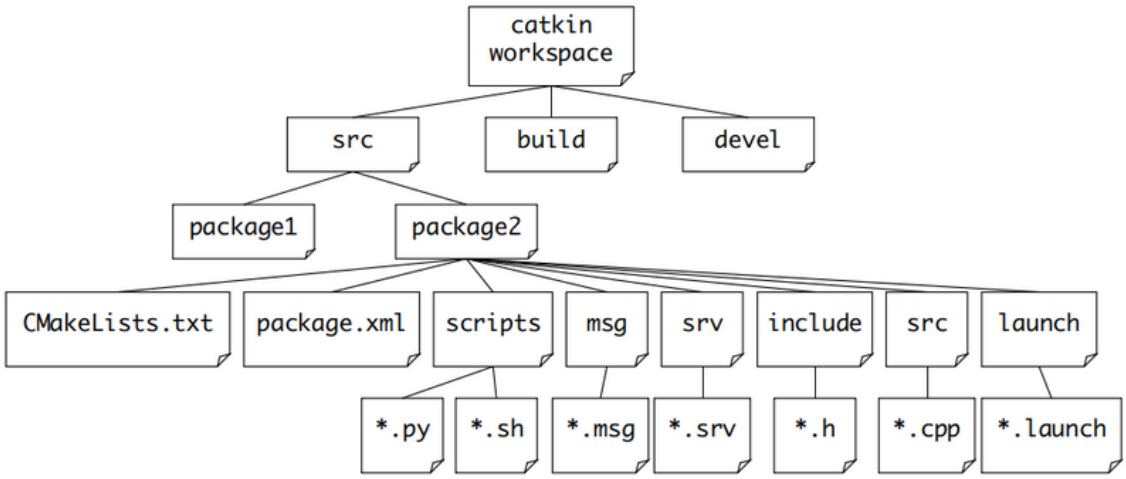


Figure 12. Following chart is representing typical structure of a ROS project [26.]

3.2.2.1 Workspace

Workspace usually contains three different subdirectories - src, build and devel, which serve different role in the software development process. Source (src) space contains the source code, build space corresponds for building catkin packages in the source space, development (devel) space is where built targets are placed before being installed. [27.]

3.2.2.2 Package

Software in ROS is organized in packages, and it is a main unit. Package is providing enough functionality to be useful, and avoiding to be heavyweight and difficult to use from the software. The files that the package may contain are described below. [28.]

CMakeLists.txt

Describes how to build code and where to install it. [29.]

Package.xml

Defines properties about the package, such as the package name, version numbers, authors, maintainers and dependencies on other catkin packages. [30.]

Scripts

This folder includes scripts. Script is a list of commands, that are executed.

Msg

Nonstandard message definitions are stored in this folder.

Srv

Defines the request and response data structure for services.

Include

Includes the headers of the libraries.

Src

In this folder source codes are stored. Nodes can be found in this directory.

launch

Launch files are stored here. Those files are used to launch one or more ROS nodes.

3.2.2.3 Metapackage

Metapackages are specialized packages in ROS. Metapackage references one or more related packages, which are loosely grouped together, and is not containing any files usually found in packages. With metapackage, it is convenient to group multiple packages as a single logical package. [31.]

3.2.3 ROS Community Level

The ROS community level is a level with ROS resources for communities, to exchange knowledge and software. These resources include distributions, repositories, ROS wiki, mailing lists, ROS Answers and blog. [32.]

3.2.4 Graph and Package Resource Names

Graph resource names provide a hierarchical naming structure. It is used for all resources in a computation graph, such as topics, nodes, parameters and services. It is critical to understand how these names work, as they are powerful and central to how larger and more complicated systems are composed in ROS. Each resource is defined within a namespace, which it may share with many other resources. Valid name has first character in an alpha character, tilde or forward slash, and subsequent characters can be alphanumeric, underscores or forward slashes.

Package resource names are used in ROS filesystem-level to simplify the process of referring to data types on disk and files. Package resource names consists of the name of the package that resource is in, and the name of the resource. Names are very similar to file paths, except they are shorter, due to the ability of ROS to locate packages on the disk. [33.]

3.3 Pronterface

Pronterface is a 3D printing program, that allows to directly control 3D printers through a USB cable. This program was found to be useful with troubleshooting and figuring out thermoplastic extractor working principle and parameters, as it is also including a console view.

4 Implementation

Previously foundation had been laid for the robot to run with ROS. Communication had been created and few steps had been made for sending commands to the robot for execution.

Node Descartes_manufacturing had been created to be able to send trajectory to the robot, but this topic has not been studied widely. Descartes_manufacturing is a path-planner. This node was sending trajectories for the robot, including XYZ IJK parameters in randomly defined test points, without availability to define a movable frame in what relation it would be executed. It has not been studied what type of information robot is expecting, to be able to generate a G-code file from CAD model. Also velocity parameters have not been defined.

Tool to be used with the robot for printing was not studied from the software point of view, but mountings to the robot had been created. This tool was previously mounted to another, older robot, and which was using different type of working principle. This tool had to be adjusted for the current task, and had to be found a way to communicate with it using ROS middleware, synchronously with the robot. Previously it was including also Raspberry pi single board computer, but after researching was found, that Raspberry pi only mission was to work as a message bridge between old robot and thermoplastic extractor. This project is using newer robot and different method, and for that reason Raspberry pi was not included.

Because of current world situation with COVID-19, access to the lab was restricted. For that reason, work was done mostly using simulation tools with the robot, but additions related to the printing was available physically.

4.1 Converting CAD Model to a G-code

Cura software was used for slicing a CAD model into layers and generating G-code. Cura is an open source slicing application for 3D printers, made by Ultimaker, a 3D printer manufacturing company. G-code file includes coordinate positions for the robot and commands for the attached tool, that are responsible for manufacturing of the physical object. In the machine settings was set starting part of G-code to activate the printer, and at the ending part to turn printer off and to drive robot to the corner of defined frame. While setting those parameters, it is required to be careful with comments, because path planner Descartes_manufacturing is reading whole line, searching XYZ IJK and is not recognizing the comments. Figure 13 shows machine settings in the Cura software.

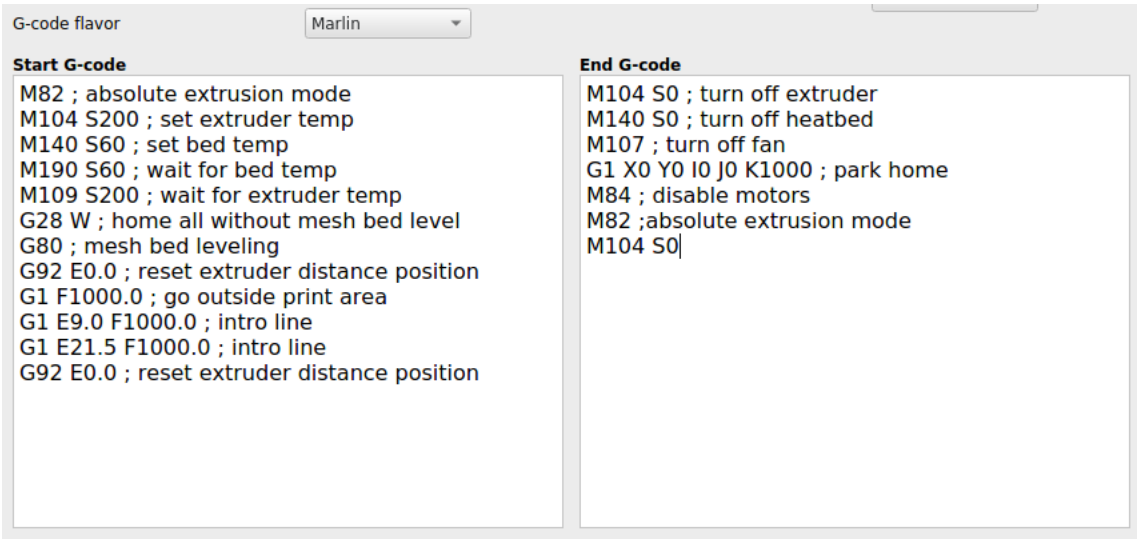
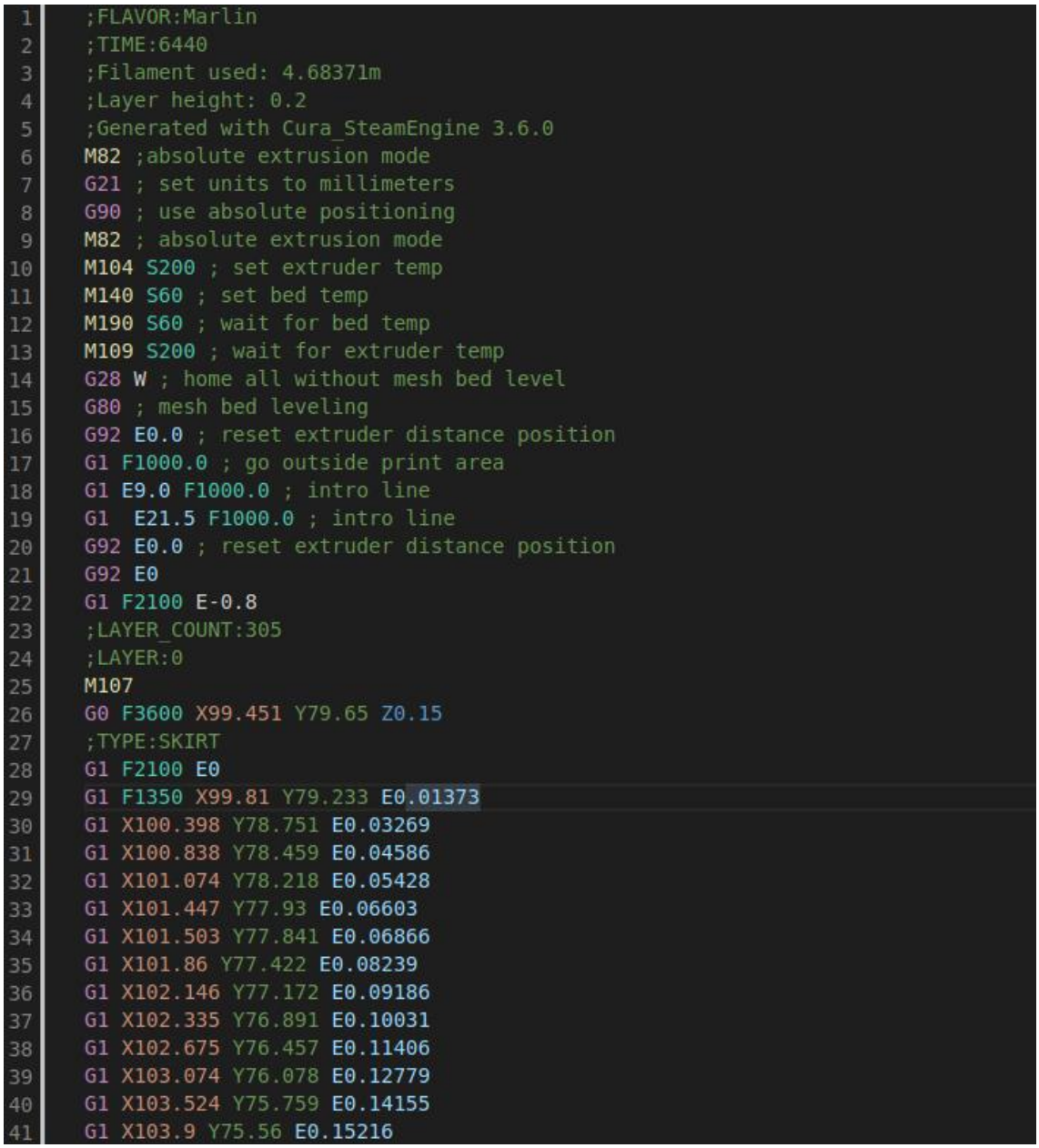


Figure 13. Starting and ending part of G-code, which could be customized and found in Preferences/Configure Cura/Printers/Machine settings

Example of sliced CAD model:



```
;FLAVOR:Marlin
;TIME:6440
;Filament used: 4.68371m
;Layer height: 0.2
;Generated with Cura_SteamEngine 3.6.0
M82 ;absolute extrusion mode
G21 ; set units to millimeters
G90 ; use absolute positioning
M82 ; absolute extrusion mode
M104 S200 ; set extruder temp
M140 S60 ; set bed temp
M190 S60 ; wait for bed temp
M109 S200 ; wait for extruder temp
G28 W ; home all without mesh bed level
G80 ; mesh bed leveling
G92 E0.0 ; reset extruder distance position
G1 F1000.0 ; go outside print area
G1 E9.0 F1000.0 ; intro line
G1  E21.5 F1000.0 ; intro line
G92 E0.0 ; reset extruder distance position
G92 E0
G1 F2100 E-0.8
;LAYER_COUNT:305
;LAYER:0
M107
G0 F3600 X99.451 Y79.65 Z0.15
;TYPE:SKIRT
G1 F2100 E0
G1 F1350 X99.81 Y79.233 E0.01373
G1 X100.398 Y78.751 E0.03269
G1 X100.838 Y78.459 E0.04586
G1 X101.074 Y78.218 E0.05428
G1 X101.447 Y77.93 E0.06603
G1 X101.503 Y77.841 E0.06866
G1 X101.86 Y77.422 E0.08239
G1 X102.146 Y77.172 E0.09186
G1 X102.335 Y76.891 E0.10031
G1 X102.675 Y76.457 E0.11406
G1 X103.074 Y76.078 E0.12779
G1 X103.524 Y75.759 E0.14155
G1 X103.9 Y75.56 E0.15216
```

Figure 14. Picture of the starting part from the G-code. The model shown in example includes 156 855 lines of code.

G-code includes XYZ parameters which is needed for the robot to calculate the trajectory, and different commands for the thermoplastic extractor. Mostly used are E and F. E is affecting to the length of filament to feed into the extruder between the start and end point, and F is for a feed rate. Feed rate is the maximum movement rate of the move between the start and end point. Usually G-code also includes at the start and end of the

file commands relating for turning thermoplastic extractor on and off. Documentation about Marlin commands can be found here [URL] https://marlinfw.org/meta/gcode/. Deeper explanation about the G-code and how it is used will be in the later topics.

4.2 Staubli TX40

This part describes developments related to the 6-DOF robot.

4.2.1 Node descartes_manufacturing

Descartes_manufacturing is a node, which is responsible for performing path-planning on under-defined Cartesian trajectories, and specifies where to put the tool. This node is using trajectory points to define what the path looks like, robot model which will solve forward and inverse kinematics, and planner that will do the work of finding a valid route along the trajectory using the provided model of the robot. [34.] Command for running this node from the console is:

```
$ rosrun manufacturing_6dof descartes_manufacturing
```

After this command has been executed in the console, it asks user to enter G-code file name. When file is entered, it calculates joint trajectory, calculate overall time and looks that the robot will not collide in any point. If all parameters are not met, for example point is not reachable or the robot will collide, it will not send trajectory to the robot for execution and will print an error message to the console.

4.2.2 Adjusting Generated G-code for the Robot

After the G-code was generated, descartes_manufacturing node had to be adjusted, as it was previously reading coordinate points in meters:

```
X0.1 Y0.2 Z0.4 I0.0 J0.0 K1.0
```

And generated G-code is in millimeters, example of the line from G-code:

```
G0 F3600 X99.451 Y79.65 Z0.15 I0 J0 K1000
```

This adjustment was done by dividing extracted values by 1000. At the same time was added offsetting possibility, if offset would be required:

```
  // Position given by the G-Code (X,Y,Z)
  // This is also starting position (X,Y,Z)
  Eigen::Vector3d position(0.0, 0.0, 0.0);
  // Orientation given by the G-Code (I,J,K)
  // This is also starting orientation (I,J,K)
  Eigen::Vector3d toolAxisIJK(0.0, 0.0, 1.0);
…
…
…
// set XY offset
int offsetXY = 0;

// set Z offset
int offsetZ = 0;
…
gcodeFile.open(completeFilename);
if(gcodeFile){
while(std::getline(gcodeFile,line)){
boost::split(splitLine, line, boost::is_any_of(delimiters));
// Loop to extract datas from each character.
for(std::string element : splitLine){
// Simply add a new condition if you want to extract and store datas from another
character
if(element[0] == 'X')
position(0) = (std::stod(&element[1])+offsetXY)/1000;
else if(element[0] == 'Y')
position(1) = (std::stod(&element[1])+offsetXY)/1000;
else if(element[0] == 'Z')
position(2) = (std::stod(&element[1])+offsetZ)/1000;
else if(element[0] == 'I')
toolAxisIJK(0) = std::stod(&element[1])/1000;
else if(element[0] == 'J')
toolAxisIJK(1) = std::stod(&element[1])/1000;
else if(element[0] == 'K')
toolAxisIJK(2) = std::stod(&element[1])/1000;
else ROS_WARN("Element: %s, line: %i not handled.",
element.c_str(), i);
}
…
```

Listing 1. Adjustment of G-code readability. [Appendix 1.]

With previously mentioned modifications readability of G-code was achieved, and most of the slicing programs are using same type of values with the coordinate points.

In generated G-code file is used XYZ parameters, as in this task the tool is pointing all the time downwards to the printing surface. Orientation of the tool could be changed from the Descartes_manufacturing node if needed, or by adding new IJK parameters in the G-code file, and the changes would affect to the robot. In this way, rotation would follow lastly set value, before it finds new rotation parameters from the G-code, if they are provided. In this type of task, rotation parameters have not been changed, because printing on the flat surface is not requiring such action. More about rotation with respect to the printing surface will be explained in paragraph 4.2.4.2. Tool orientation is shown in figure 15.

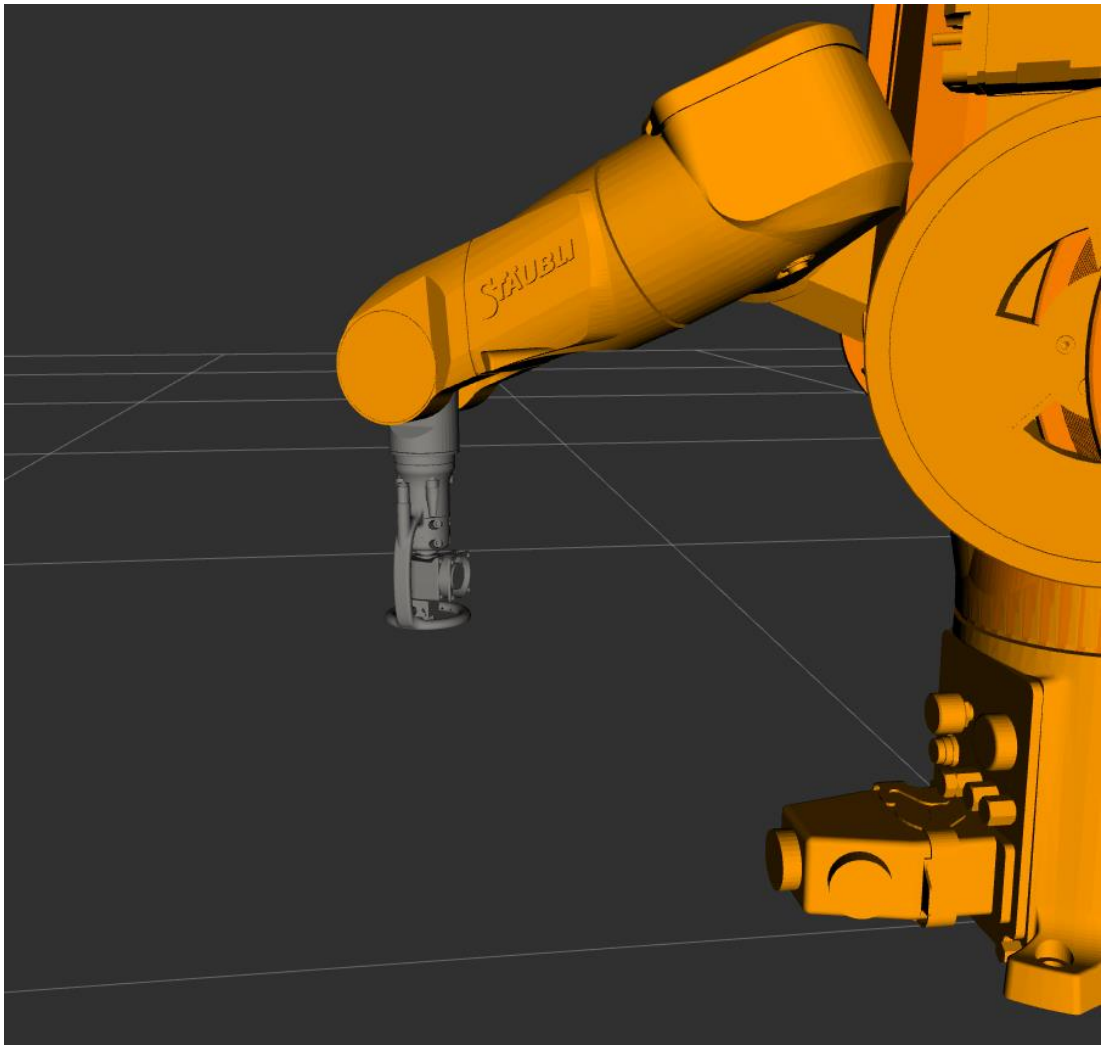


Figure 15. Tool orientation, pointing down to the printing surface.

4.2.3 Setting Velocity Parameters

It was found that this node is using delta time to get to the next pose. Delta time was defining robot to move from point A to point B in 1 second. In this way, robot velocity was dependent on the distance between the points. Using distance formula, new velocity parameters was calculated and corrected:

$$dt = \frac{\sqrt{(x_2-X_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}}{velocity*60} \quad (1)$$

Implementation:

```
// Calculate delta time to match 40mm/min velocity.
double calculateDT(Eigen::Vector3d position, Eigen::Vector3d previousPosition)
{
      // Set velocity. 0.001 = 1mm/min
      double velocity = 0.04;
      // The distance formula
      return sqrt(pow(position(0) - previousPosition(0), 2) + pow(position(1) - previ-
      ousPosition(1), 2) + pow(position(2) - previousPosition(2), 2)) / velocity*60;
}
…
// Point of the path trajectory
descartes_core::TrajectoryPtPtr pt;
…
// Position given by the G-Code (X,Y,Z)
Eigen::Vector3d position(0.0, 0.0, 0.0);
  // Orientation given by the G-Code (I,J,K)
  // This is also starting orientation (I,J,K)
  Eigen::Vector3d toolAxisIJK(0.0, 0.0, 1.0);
…
// If not defined, the delta time used to get to next pose
static double dt;
…
// Define "home" position, for setting right velocity between starting position and
first gcode coordinate.
Eigen::Vector3d previousPosition(0.0, 0.0, 0.0);
…
// Apply velocity changes
dt = calculateDT(position, previousPosition);
previousPosition = position;

// Allow degree of freedom for the tool axis
pt = descartes_core::TrajectoryPtPtr(new descartes_trajectory::AxialSymmetricPt(po-
seN, M_PI / 12.0, descartes_trajectory::AxialSymmetricPt::Z_AXIS, des-
cartes_core::TimingConstraint(dt)));
pathTrajectory.push_back(pt);
```

Listing 2. Calculating new velocity parameters using delta time. [Appendix 1.]

Currently velocity has been set to be constant 40mm/min between the points, but this value could be set to any other. It was found that this velocity parameter is safe to use with the printer, and the same velocity parameter is set in the printertool node for thermoplastic extractor. Both are set to the same velocity parameters, and in this way they are working simultaneously. More about thermoplastic extractor will be explained in paragraph 4.3.

4.2.4 Understanding Unified Robot Description Format

Unified Robot Description Format (URDF) is an XML file used in ROS to describe all elements of a robot. It is a code-independent, human-readable way to describe the geometry of robots and their cells. URDF file builds a robot model by determining how model links and joints are connected together. Link element describes a rigid body with an inertia, visual features, and collision properties. The joint element describes the kinematics and dynamic of the joint and also specifies the safety limits of the joint. [35.] Example of URDF file structure is shown below.

At the beginning of the XML file, robot is being named and links are determined as follow:

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://wiki.ros.org/xacro">
  <xacro:include filename="$(find staubli_resources)/urdf/common_materi-
als.xacro"/>

  <xacro:macro name="staubli_tx40" params="prefix">
    <!-- links: main serial chain -->
    <link name="${prefix}base_link">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/vis-
ual/base_link.stl"/>
        </geometry>
        <xacro:material_staubli_ral_melon_yellow />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/colli-
sion/base_link.stl"/>
        </geometry>
      </collision>
      <inertial>
        <mass value="5.82375"/>
        <origin xyz="-0.00892 -0.00017 0.07857" rpy="0.0 0.0 0.0"/>
        <inertia ixx="0.020725706" ixy="0.00008955" ixz="0.00193426"
                                    iyy="0.026779837"  iyz="0.000024618"
                                                         izz="0.021448879"/>
      </inertial>
…
…
…
```

Listing 3. Determining the links. [Appendix 6.]

After all links are added, it is needed to determine how they are connected together, because otherwise the parser won't know where to put them:

```
…
…
…
<!-- joints: main serial chain -->
    <joint name="${prefix}joint_1" type="revolute">
      <origin xyz="0 0 0.32" rpy="0 0 0"/>
      <parent link="${prefix}base_link"/>
      <child link="${prefix}link_1"/>
      <axis xyz="0 0 1"/>
      <limit lower="${radians(-180.0)}" upper="${radians(180.0)}" ef-
fort="40.0" velocity="${radians(287.0)}" />
      <dynamics damping="0.0" friction="0.0"/>
    </joint>
    <joint name="${prefix}joint_2" type="revolute">
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <parent link="${prefix}link_1"/>
      <child link="${prefix}link_2"/>
      <axis xyz="0 1 0"/>
      <limit lower="${radians(-125.0)}" upper="${radians(125.0)}" ef-
fort="11.0" velocity="${radians(287.0)}" />
      <dynamics damping="0.0" friction="0.0"/>
    </joint>
…
…
…
```

Listing 4. Creating joints and determining links positions. [Appendix 6.]

When all links are added and joints are created, it is also important to add several standardised frames to match common industrial robot frame. They are base_link, flange and tool0. Base_link is positioned in the first frame of the robot tied to the first link. Flange is attachment point for an end effector. Tool0 lies on the physical robot's mounting flange. [36.]

```
…
…
 <!-- ROS-Industrial 'base' frame: base_link to Staubli World Coordinates
transform -->
    <link name="${prefix}base" />
    <joint name="${prefix}base_link-base" type="fixed">
      <origin xyz="0 0 0.32" rpy="0 0 0"/>
      <parent link="${prefix}base_link"/>
      <child link="${prefix}base"/>
    </joint>

    <!-- ROS-Industrial 'flange' frame: attachment point for EEF models -->
    <link name="${prefix}flange" />
    <joint name="${prefix}joint_6-flange" type="fixed">
      <origin xyz="0 0 0" rpy="0 ${radians(-90.0)} 0" />
      <parent link="${prefix}link_6" />
      <child link="${prefix}flange" />
    </joint>

    <!-- ROS-Industrial 'tool0' frame: all-zeros tool frame -->
    <link name="${prefix}tool0" />
    <joint name="${prefix}flange-tool0" type="fixed">
      <origin xyz="0 0 0" rpy="0 ${radians(90.0)} 0" />
      <parent link="${prefix}flange" />
      <child link="${prefix}tool0" />
    </joint>
```

Listing 5. Standardized frames. [Appendix 6.]

Those frames should not be changed, but it is possible to add additional frames as a children frames.

4.2.4.1 Change Robot Model

Robot model was configured to be without printing related parts attached, and was not aware if it would have any collisions caused because of attachments. To change robot model, meshes needs to be replaced. They are found in src/staubli_experimental/staubli_tx40_support/meshes. This folder includes two subfolders, collision and visual, with .stl files. Overwriting .stl files from “normal” to “With3Dprinting” inside both subfilders will apply changes. Changes are shown in figure 16.

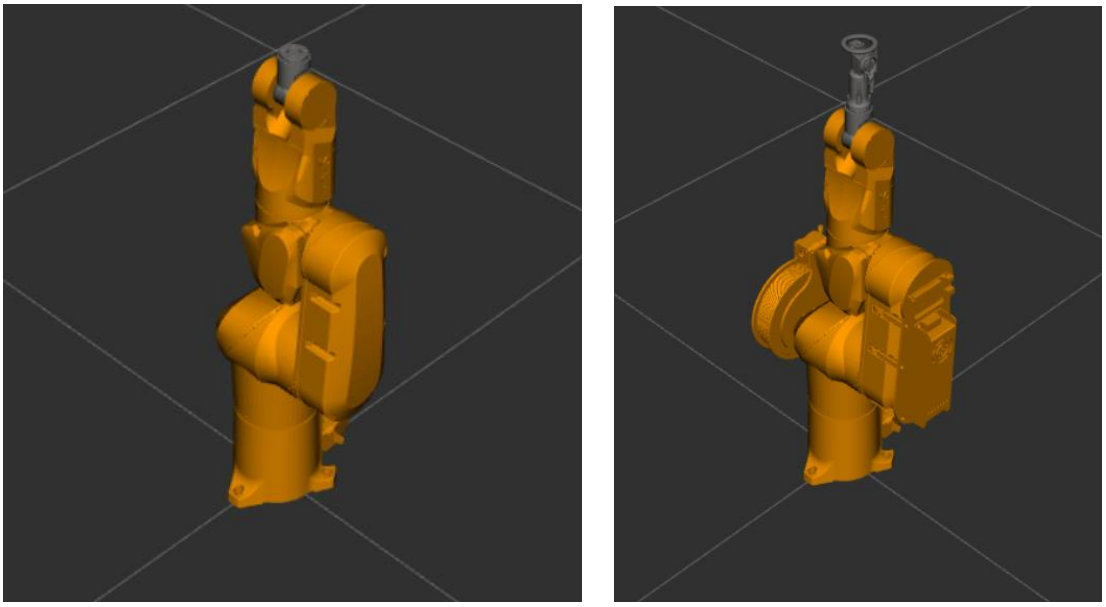


Figure 16. Robot model without printing parts attached at the left, and with attached parts at the right.

Replacing files from “normal” to “With3Dprinting” will change robot model visually, and will have correct collision parameters to be used with printing related parts, attached to the physical robot.

4.2.4.2 Creating Reconfigurable Frame Where Trajectory Will Be Executed

Generated trajectory from g-code file needs to be executed in specific reconfigurable place. Node Descartes_manufacturing is expressing poses in relation from standardized frames base_link to tool0, which is shown in figure 17.

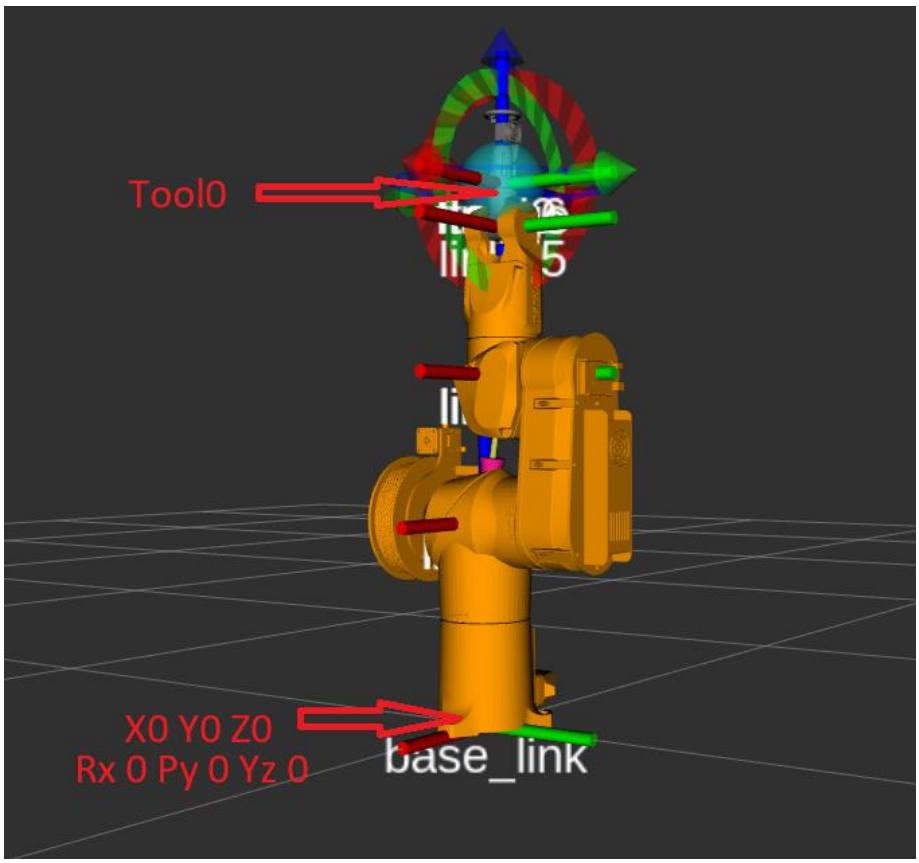


Figure 17. Trajectory is executed in relation from base_link to tool0. Base_link coordinates are not reconfigurable, because it is standardized frame.

To make reconfigurable frame, additional frame was added to the URDF file, which is children frame from base_link. File is located in src/staubli_experimental/staubli_tx40_support/urdf/tx40_macro.xacro. This frame was named “custom”.

```
<!-- Custom frame -->
<link name="${prefix}custom" />
<joint name="${prefix}custom-frame" type="fixed">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <parent link="${prefix}base_link"/>
  <child link="${prefix}custom"/>
</joint>
```

Listing 6. Creating new reconfigurable frame to the tx40_macro.xacro. [Appendix 6.]

Origin of this frame is fully reconfigurable. This point could be set to the corner of the printing surface, so the model would be printed in correct place related to the origin. If new origin was set when simulation window is open, simulation should be restarted to apply the changes. Running simulation is done by commanding in the console:

```
$ roslaunch staubli_tx40_moveit_config moveit_planning_execution.launch sim:=
```

If the physical robot is connected to the computer, connection is made with command:

```
$ roslaunch staubli_tx40_moveit_config moveit_planning_execution.launch
sim:=false robot_ip:=192.168.125.40
```

More about how to get robot position data, which should be implemented in origin will be explained in 4.2.5.

Node Descartes_manufacturing needs to be edited, to send the trajectory related to the recently created custom frame. It is done by changing world_frame from “base_link” to “custom” frame:

```
  // name of the kinematic group you defined when running MoveitSetupAssis-
tant. For many industrial robots this will be
  // "manipulator"
  const std::string group_name = "manipulator";

  // Name of frame in which you are expressing poses. Typi-
cally "world_frame" or "base_link".
  /* Previously we were looking at base_link, so at the middle bot-
tom of the robot. Now there is added new frame named "custom",
     which we can move from the tx_40_macro.xacro. This is done, to be able mo
ve needed frame anywhere we want, for example
     plan is that this whill be point in the corner of the printing ta-
ble. Similar changes are done to the printertool.cpp.
     It is "looking" transformation between custom frame and tool0, so it rec-
ognizes coordinate points correctly.
  */
  const std::string world_frame = "custom";

  // tool center point frame (name of link associated with tool). The ro-
bot's flange is typically "tool0" but yours
  // could be anything. We typically have our tool's positive Z-
axis point outward from the grinder, welder, etc.
  const std::string tcp_frame = "tool0";
```

Listing 7. Changing frame where poses are being expressed, from descartes_manufacturing source code. [Appendix 1.]

When changes were applied, trajectory is sent related to the reconfigurable frame named “custom”.

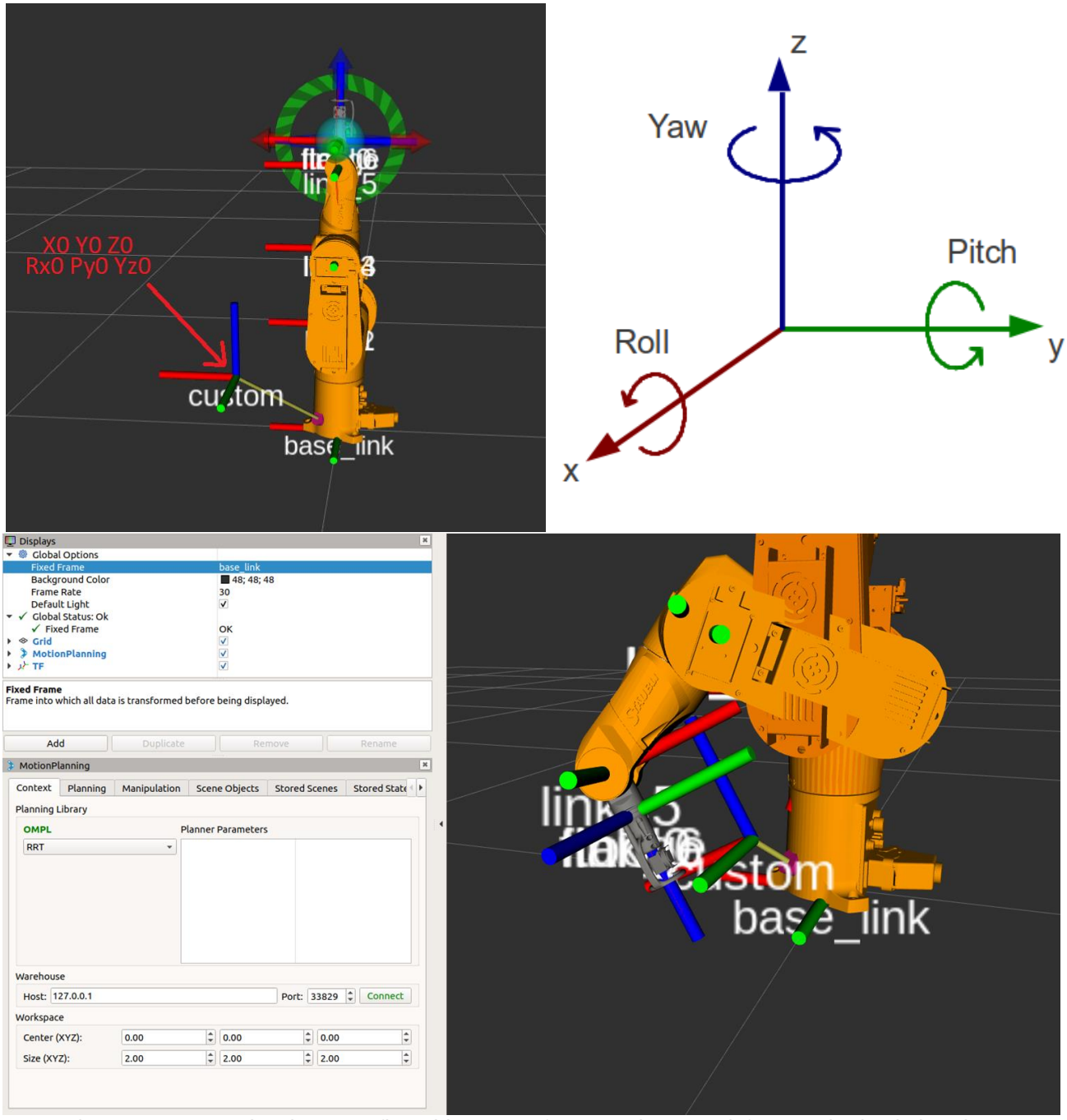


Figure 18. Image showing reconfigurable custom frame. In the upper left corner is simulation view of a frame set horizontally. In the picture down is seen, that when the frame is rotated, tool orientation is following the rotation.

To make custom frame visible in RViz, fixed frame should be set to the base_link, and TF display should be added from the menu - Add -> TF.

4.2.5 Getting Robot Position Data for Defining a New Frame

For receiving robot position data, node staubli_tf_liseter was created. This node base was taken from writing a tf listener (C++) tutorial. [37.] Tf is a package, that is letting to keep a track of coordinate frames. It maintains relationship between coordinate frames, and let's to transform points and vectors. [38.] Staubli_tf_listener is refreshing robot position data on 50Hz frequency between base_link and tool0, and prints received data to the console. Coordinate data was multiplicated by 1000 to get more clean prints to the console, and for that reason, coordinates entered to redefining a new custom frame position should be divided by 1000. Command for running staubli_tf_listener node:

```
$ rosrun staubli_tf staubli_tf_listener
```

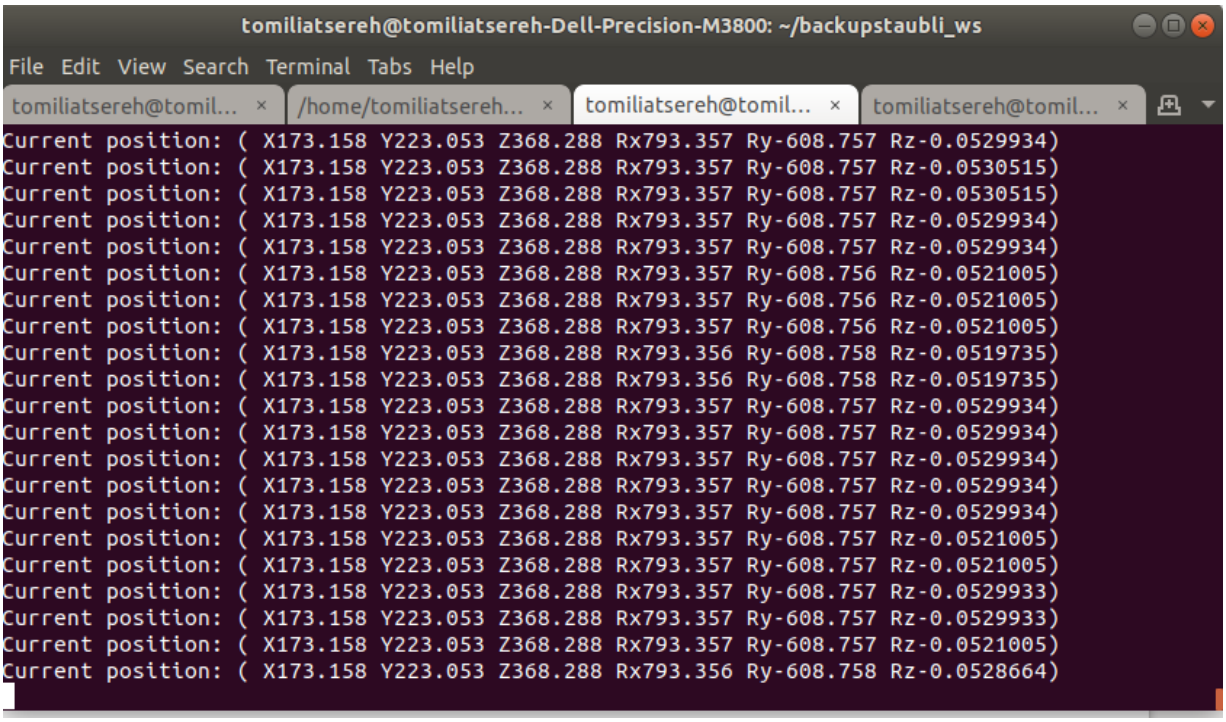


Figure 19. Staubli_tf_listener node is printing robot position coordinates to the console. [Appendix 2.]

XYZ comes from robot flange center coordinates, and RPY are attitude components:

- XYZ are coordinate values, which defines the position of the robot tool center
- Rx is a Yaw angle: Rotation angle around X axis
- Ry is a Pitch angle: Rotation angle around Y axis
- Rz is a Roll angle: Rotation angle around Z axis

When desired location for a new frame is found, it should be used in tx40_macro.xacro file as follows:

```
    <!-- Custom frame -->
    <link name="${prefix}custom" />
    <joint name="${prefix}custom-frame" type="fixed">
      <origin xyz="0.173158 0.223053 0.368228" rpy="0.793356 -0.608758 -
0.0521005"/>
      <parent link="${prefix}base_link"/>
      <child link="${prefix}custom"/>
    </joint>
```

Listing 8. Reconfiguring new frame.

This is just an example values, and the same information is used that could be seen in image 20.

4.3 Developing Thermoplastic Extractor

Hardware for thermoplastic extractor was pre-made, but from software point of view it has not been examined. Parts related to printing was taken from other, non ROS related project. After the investigation, it was found, that Arduino MEGA2560 was running on an Marlin 1.1.9 firmware, and parameters related to attached parts was configured.

None of compatible “ready to use” solutions was found to run Marlin with ROS middle-ware, and for that reason was decided to develop a ROS node from scratch. This node would represent ideologically something similar, that could be experienced in graphical user interface type of programs for 3D printers, but in a C++ format and adapted for current task. It would send the G-code line by line to the Marlin, with converted values and at a certain time.

4.3.1 Marlin

Marlin is an open source firmware for replicating rapid prototypers, also known as 3D printers. It is also used in several popular 3D printers, few of them are for example Ulti-maker, Prusa and Printrbot. Marlin is also capable of driving CNC’s and laser engravers. To reproduce a model with Marlin, it must be converted to G-code, which is including several action commands.

Marlin aims to be adaptable to as many boards and configurations as possible. Marlin aims to be configurable, customizable, extensible and economical for users. It offers many features, and one of them is full featured G-code with over 150 commands. Marlin Firmware runs on the 3D printer's main board and manages all the real-time activities of the machine. It can coordinate heaters, steppers, sensors and everything else involved in the 3D printing process.

Marlin implements an AM process. The control-language for marlin is coming from G-code, which is telling to a machine how it should act. For example, G-code includes commands like "set heater to desired temperature" and "feed this length of filament (E) into the extruder between the points, using this feed rate (F)". [39.]

4.3.2 Creating Printertool Node

Operating principle of this node is to make communication with the Arduino MEGA 2560, open the G-code file, convert the parameters from G-code and send them line by line to the USB port at a certain time, synchronized with the robot trajectory. After the line is sent to the USB port, Marlin receives the commands, and with that information controls extruder and extractor. G-code file entered for the robot should be entered separately to the printertool, as they are two different nodes, with own working principles. But the G-code file should be the same, as it is including the information meant for the current action.

4.3.2.1 Communication with the USB Port

Before any data could be sent to the Marlin via USB port, serial communication should be created. It was done by creating a third party library named mySerial, which source code was found from Raspberrypi forum. [40.] Including this third party library, mySerial.cpp and mySerial.h, sending data over serial link has been made possible. This library is located in [WORKSPACE]/src/printer/src and is used in printertool.cpp header file as follows:

```
#include "mySerial.h"
```

This library has to be also specified in CMakeLists.txt, so it will be compiled correctly and be in use. To add the library in CMakelists.txt, it should be mentioned as follows:

```
## target_link_libraries(mySerial ${catkin_LIBRARIES})

add_executable(${PROJECT_NAME}_printertool src/printertool.cpp src/mySe-
rial.cpp src/mySerial.h)

add_dependencies(${PROJECT_NAME}_printertool ${${PROJECT_NAME}_EXPORTED_TAR-
GETS} ${catkin_EXPORTED_TARGETS})

target_link_libraries(${PROJECT_NAME}_printertool ${catkin_LIBRARIES})
```

Listing 9. Adding library to the CMakeLists.txt

This library is responsible for making communication, defining port and baudrate. To find information about what port and baudrate should be implemented, Pronterface was used. With this application is used ttyUSB0 port, and baudrate is 9600. It is defined in the printertool.cpp code as follows:

```
    // Define port and baudrate, and make communication.
    mySerial serial = mySerial("/dev/ttyUSB0", 9600);
```

Now when communication is created and program knows where to send the commands, it is possible to use USB port.

4.3.2.2 Converting G-code Values

Because robot moves at the same speed between the points, printing should also be executed with the same parameters. This was done by searching F<pos> from the G-code, calculating new federate, and overwriting existing value from G-code with a new value, before sending it to the Marlin. Feed rate was calculated using formula:

$$FeedRate = \frac{L_1}{L_2/speed} = \frac{4*\text{ØNozzle}*h_{layer}*Speed}{\text{Ø}Filament^2*\pi} * 60\ [mm/min] \quad (2)$$

With this formula, federate was calculated to be 40mm/min between every point and implemented in the printertool.cpp as follows:

```
float printspeed = 40;          // Set speed
float diambuse = 0.4;           // NozzleDiam
float hcouche = 0.2;            // hLayer
float diamfil = 1.75;           // FilamentDiam
…
// Calculate new parameters to F
float feedrate = (60 * 4 * diambuse * hcouche * printspeed)/(std::pow(diam-
fil, 2) * M_PI);

    std::ostringstream ssF;
    ssF << feedrate;
    std::string feedrateString = ssF.str();

    std::string filename, line, path;
    std::ifstream read;

…
    if (read.is_open())
    {
…
        // Search F and change feedrate value. Overwrite with new parameters.
        while (getline(read, line))
        {
            if (isdigit(line[line.find("F") + 1]))
            {
                int fStartPos = line.find("F") + 1;
                int fSpacePos = line.find(" ", fStartPos);
                int feedrateLength = fSpacePos - fStartPos;
                line.replace(fStartPos, feedrateLength, feedrateString);
```

Listing 10. Calculating new F. [Appendix 3.]

After this manipulation was done, new federate is corresponding to F79.824 in every line, which is representing 40mm/min velocity.

As mentioned before, E<pos> is the length of filament to feed into the extruder between the start and end point. In the G-code, this value is growing in every line, and at the last line could be seen whole length of the printed model. (The increase of this value can be seen in image 15.) Probably basic 3D printer software is taking this into account and calculating it, but because this program is made from scratch, E should be calculated separately for every line, so the value will correspond how much filament extruder should feed at the current time. It was done in a similar way that was used with converting F, but by searching E value and using different formula:

$$NewE = E_{Current} - E_{previous} \quad (3)$$

In the formula Ecurrent is the E value found on the line from G-code, which has been recently entered, and Eprevious is the E value from the previous line. When calculation is done, E value from the G-code is overwritten with the new E value, which is including command for the filament length meant exactly for current action, before sending the line to the Marlin.

4.3.2.3 Synchronization with the Robot Trajectory and Sending G-code to the Printer

For reproducing physical object, it is not enough to just send G-code to the robot to execute trajectory, and for the printer to preheat and extrude the filament. Robot movement and printer extrusion must take place simultaneously in a printing area. To make synchronization possible, node printer_printertool is using tf package to keep track on robot position. Tf package was previously explained in paragraph 4.2.4 “Getting robot position for defining a new frame”.

Node printer_printertool is comparing coordinate values from the entered G-code, with the robot’s current position, between frames “custom” and “tool0”, where robot trajectory is executed through node descartes_manufacturing. When it finds matching coordinate values, it is sending next line from the G-code to the Marlin for execution and commanding the printer. In this way, when robot is moving from point A to point B, Marlin is also commanding to print from point A to point B. But additional measures had to be included, because the task is more complicated.

Because G-code is handled line by line, and it is including at the beginning of the file startup parameters, in the middle extra commands and at the end commands for the printer, sending lines for the printer would freeze. That is, because it would search to compare some lines in the G-code, related to the printer, with the robot’s coordinates. For that reason, printertool node is sending lines without coordinate information (XYZ) immediately for the Marlin, because this data includes important commands only related to the printer. With this addition, when G-code is entered to the printer_printertool node, it sends startup commands for the printer, so it will preheat and will stop for the first point with coordinates, because it will wait them to match. In the working process it would not freeze in the lines, which are including only printer related commands. And at the end, when G-code is gone through, node will send command lines for turning printer off.

Also tolerance setting ability was necessarily to add, because it was found, that when double values were compared together, they looked to be as the same value, but digging deeper it was found, that the last decimal values were differing. Robot position is also

refreshing at 50hz frequency, and misfires could happen. It was accomplished by calculating difference between robot's coordinates and coordinates from the G-code, and allowing the deviation.

Moreover coordinate parameters listened from the robot position are in metric units, and in G-code units are in millimeters, and for that reason they had to be calculated to be equal. This was done by multiplying coordinates, listened from the robot by *1000.

After additions were added, printer is able to see robot's current position and act simultaneously. For running the node related to the printer, this command should be used in the console:

```
$ rosrun printer printer_printertool
```

5 Application

When robot and the printer are connected to the computer, several steps should be performed. Communication with the robot has to be created and commands for running the nodes should be performed.

1. Creating TCP-IP Connection:

    - Connection between computer and robot controller is performed using TCP-IP protocol. Network connection needs to be set in the computer with the relation of the controller IP-address: 192.168.125.40
    - For checking that connection between computer and controller has been made succesfully, it could be ensured by pinging the controller:

        ```
        $ ping 192.168.125.42
        ```

2. Launching ros_server from the Robot's Physical Controller:

    - Controller needs to be connected with ROS server. This is performed in the controller settings by running:

```
Application manager / val3 applications / disc / ros_server
```

3. Starting Master

- In the computer console, start ROS master with the following command:

```
$ roscore
```

4. Load URDF Parameters

- Open new terminal and load URDF parameters with the following command:

```
$roslaunch staubli_tx40_support load_tx40.launch
```

5. Start Communication Between ROS and Controller:

- Command for starting the communication between ROS and controller:

```
$roslaunch staubli_val3_driver robot_interface_streaming.launch ro-
bot_ip:=192.168.125.40
```

6. Launch Stäubli:

- After the communication is created between computer and the controller, Stäubli is ready to use. Command for running physical robot with the simulation view in the computer:

```
$ roslaunch staubli_tx40_moveit_config moveit_planning_execution.launch
sim:=false robot_ip:=192.168.125.40
```

- Or only in simulation without physical robot, using command below:

```
roslaunch staubli_tx40_moveit_config moveit_planning_execution.launch sim:=
```

After communication has been created, robot and thermoplastic extractor are ready to use. To perform printing, frame should be defined in URDF first, for executing the trajectory in the correct place. Setting frame is explained in paragraph “4.2.4 Getting robot position for defining a new frame”. When frame is defined, thermoplastic extractor is ready to be activated.

7. Using Thermoplastic Extractor

- Before sending the trajectory to the robot for execution, thermoplastic extractor should be activated, so it will preheat and be ready to follow robot's movement. Launching thermoplastic extractor in a new terminal with the following command:

```
$ rosrun printer printer_printertool
```

- When printertool node is activated, it will ask for a G-code file. When the G-code file is entered, it will send commands for starting the printer, and will wait robot to arrive to the first coordinate point. The starting lines include preheating commands, and preheating operation is not time consuming. Nozzle will be preheated before robot arrives to the first coordinate point.
- After printer is activated, trajectory could be sent to the robot.

8. Sending Trajectory for the Robot

- To send trajectory for the robot, node descartes_manufacturing should be used for path-planning, with the following command in a new terminal:

```
$ rosrun manufacturing_6dof descartes_manufacturing
```

- When descartes_manufacturing node is activated, it will ask for a G-code file. The file entered here should be exactly the same file which was entered to the printertool node, as they are performing similar actions based on a G-code, to reproduce physical object.
- After the file is entered, trajectory will be calculated and robot will start to move between the coordinate points. Printer is already activated and is awaiting for the robot to reach the destination points, to perform extrusion.

6 Results, Future Development and Bugs

Purpose of the work was to successfully develop forward Stäubli and the parts, which are responsible for the 3D printing. COVID-19 caused restrictions to the physical robot, and work has been done mostly using simulation tools. Because of that, tests with the real use were very minor.

6.1 Results

It was proven, that CAD model, which was generated to the G-code with the Cura is working. Adjustments were made to the path planner Descartes_manufacturing, and robot is now able to read the G-code in the right value. This topic was also tested with the real robot. Only note is, that the physical robot controller is waiting to receive coordinate points at least in the first two lines. For example zero position coordinates could be added in the beginning of the file.

Creating a possibility to set velocity parameters sent to the robot was found, and it was affecting in simulation. When the physical robot was connected to the computer, issue came up. Robot's physical controller was overwriting the values, which were sent along with the trajectory. Solution was not found in the short time given for testing, when physical robot was connected.

Robot model was successfully changed to the model with the attachments, responsible for the printing. Visual model changed in simulation, and collision was taken into account.

It was found a method how to define where trajectory needs to be executed. This was done by creating a new frame, possibility to read robot coordinates and adjusting path planner Descartes_manufacturing. Results were tested with the physical robot, and it has been found to work.

Creating a node, which is responsible for the communication with the parts related to the printer has been made. Communication with the Marlin was created via USB port. It was proven, that this node is able to send commands from the G-code file to the Marlin, and Marlin was able to command the printer. Extruder was moving, nozzle was able to reach

a specified temperature, filament was melted and supplied from the nozzle. Synchronization with the robot had been created. Testing printing with the physical robot was not done, because of the physical robot velocity issue. It was not studied how the thermoplastic extractor and robot are working together in the real life use.

6.2 Future Development

More tests should be performed in the real life use. Issue with the velocity overwriting should be figured out at first. It could be related with the robot initialization. I would recommend to study staubli_tx40_moveit_config files, ros_control, looking that nothing is missing regarding initialization and explore physical controller of the robot. This is just an idea, and solution could be figured out anywhere else.

When the issue regarding robot's velocity would be fixed, thermoplastic extractor would be attached to the robot and printing physical object from the G-code file could be tested. During the test, should be looked how thermoplastic extractor and robot are working together. If any problems occur, development could be continued from this part. No future issues are known at this time, but for example it is unknown, if any delays could appear in the process.

6.3 Bugs

At the work progress was found two bugs, which were minor and did not cause any problems to the development. Both of them were associated with the simulation view.

First bug was related to the robot figure, when trajectory was sent using Descartes path planner. When in simulation view robot was commanded using built-in functions, everything looked as it should be. When external application Descartes_manufacturing had been used, it caused to spawn second model in the simulation. The second model was only visual bug, and was inactive. It was not causing any problems to the usability, and was disappearing when built-in functions had been used again. This problem could be caused because external application is used, and simulation tool is not adapted to recognize this application.

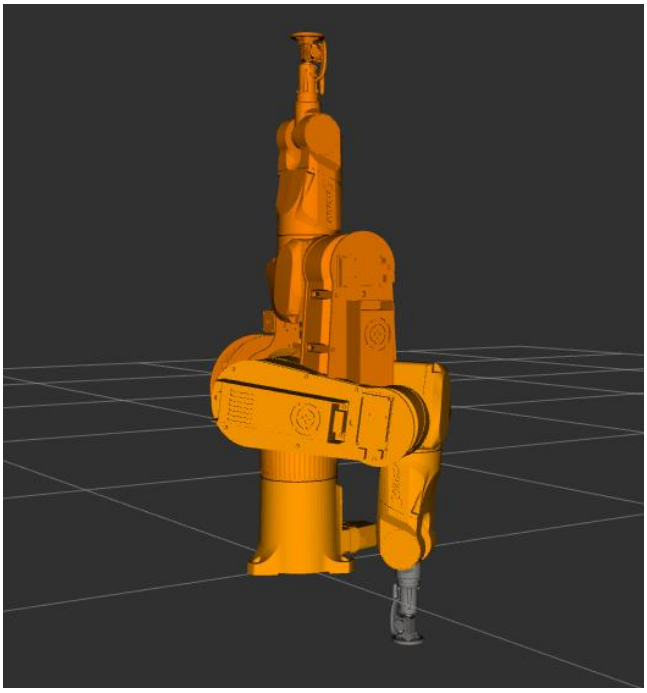


Figure 20. Bug related to simulation tool. Robot which is pointing down is active robot, and figure of the second robot pointing up is a visual bug.

Second open point was related to the orientation, picked up from a simulation view. When the same orientation from the simulation view was sent using node Descartes_manufacturing, it was quite opposite. This did not cause any problems in the development, as application was not related to data read from simulation view.

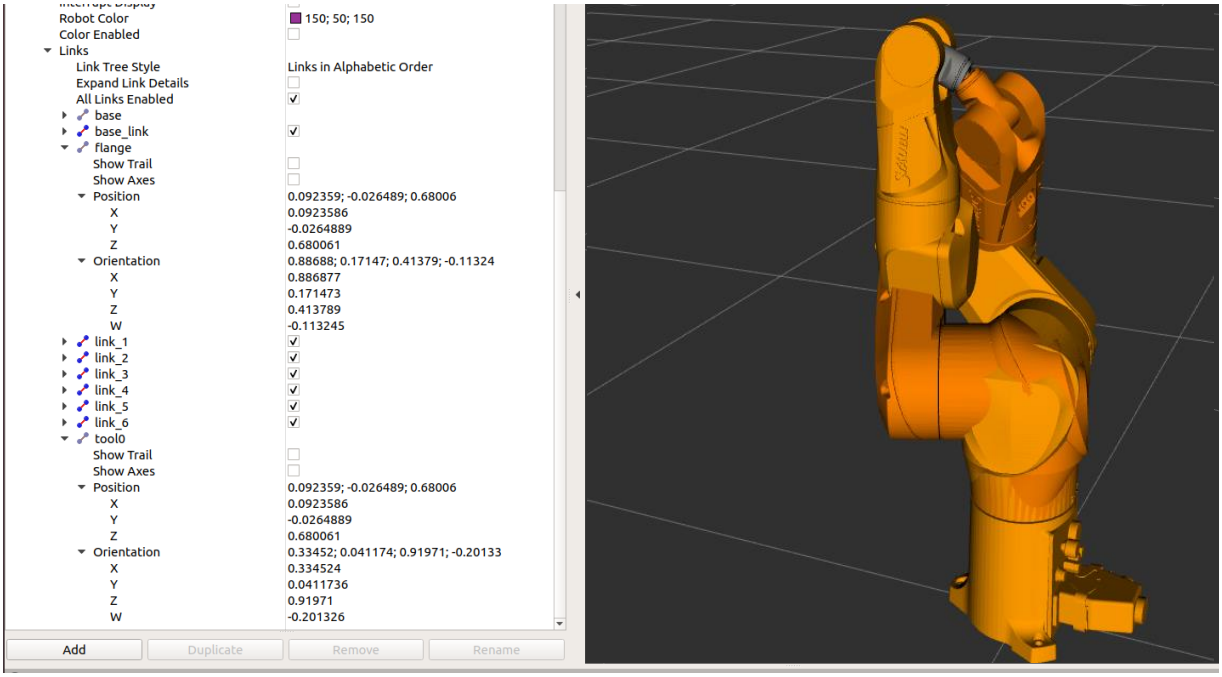


Figure 21. In this picture could be seen, that the orientation sent using Descartes_manufacturing is mirrored.

7 Summary

The purpose of this thesis work was to continue the project from the previous stage, and push the development forward using ROS middle-ware, with the aim of creating 6-DOF 3D printer. Current COVID-19 situation around the world caused restrictions, and made already complicated task much more difficult to carry on. A lot of adaptation and improvisation had to be done for moving forward. Work had been done mostly simulating the robot, and working with physical parts related to printing. Topic of the work turned out also to be more complicated as it was planned, because none of the similar works were found in the internet, related to printing using ROS middle-ware and Marlin firmware. For that reason, many developments had to be figured out from scratch. Used method was also new for me, but it got carried out good with really hard work. Previously made work related to robot initialization and the report was written in French, which caused difficulties due to language incompetence.

The work achieved the set aim, and pushed development process forward. In the simulation view with the robot, it was possible to use generated G-code of the CAD model, setting velocity parameters and defining where the trajectory should be executed. Connection with the printing related parts had been created, and commands sent to the thermoplastic extractor worked. Nozzle was able to get to the desired temperature, and was able to control the substance supply. It was also found how to get robot position data and pair it with the printer. Everything looked to work in simulation, and because of that, extra permission to the laboratory was requested and accepted for the last working days, to test the results with the real robot.

While testing results with the real robot, it was found, that mostly done work in simulation was successful, but unexpected issue came up. When physical robot controller had been connected to the computer, it was possible to send the trajectory generated from CAD model to the desired location, and robot seemed to execute it correctly. But the problem came up with the robot physical controller, as it was overwriting velocity parameters with its own, or not respecting the sent parameters. The problem was totally unexpected, because in simulation was proven, that defined velocity was affecting the movement between the coordinate points. The time was limited and solution has not been found regarding velocity issue. This problem could be related with the robot's initialization. For

that reason, it was impossible to move forward in testing and try to reproduce physical object with the use of CAD model, which was converted to G-code. Because of that issue, remained an unsolved topic with the printing physical object, and if there could be any delays regarding synchronization.

Also sending G-code file to the real robot had a slight difference, compared to the testing results in the simulation. While working with the simulation tools, robot was able to follow the trajectory generated with the Cura, without any modifications. But when the physical robot was connected, controller was expecting to receive coordinate points at least in the first two lines from the G-code file, to calculate the trajectory. This issue was fixed by adding two coordinate points in the very beginning of the G-code, and the testing could be continued. It has been found, that this method is working, and single lines in between of the G-code, which belong only to the printer, are not triggering the robot.

Even though the issue with robot velocity overwriting appeared, it is solvable. The work opened up an opportunity for further development, as this is a long term project, and it is a good to move on from this point.

After the issue with velocity would be figured out, and the topic with the printing using above methods would be completed, it could be upgraded to be more productive. As at the present time, printer and robot are set to work on a specified speed to match, this part could be developed further. Robot could take E value into a count, which is determining velocity, and define movement speed based on that. Similar idea of implementation was found on the RoboDK website, with modifying robot post processor, to calculate movement speed before executing. URL to the document:

https://robodk.com/doc/en/Robot-Machining.html #Print3Dpost

References

1 School of Business and Engineering Vaud. 9 June 2017. [online]. Wikipedia. URL: <https://en.wikipedia.org/wiki/School_of_Business_and_Engineering_Vaud>. Accessed 17.5.2020

2 GE Additive. 2020. What is additive manufacturing? [online]. GENERAL ELECTRIC. URL: <https://www.ge.com/additive/additive-manufacturing>. Accessed 1.5.2020

3 3D printing. 18 May 2020. [online]. Wikipedia. URL: <https://en.wikipedia.org/wiki/3D_printing>. Accessed 3.5.2020

4 Alkaios Bournias Varotsis. 2020. 3D printing vs. CNC machining. [online]. 3D HUBS. URL: <https://s3-eu-west-1.amazonaws.com/3dhubs-knowledge-base/cnc-vs-3d-printing/2-additive-manufacturing.png>. Accessed 1.5.2020

5 Creative Mechanisms. January 4, 2016. Additive Manufacturing vs Subtractive Manufacturing. [online]. URL: <https://www.creativemechanisms.com/blog/additive-manufacturing-vs-subtractive-manufacturing>. Accessed 3.5.2020

6 Alkaios Bournias Varotsis. 2020. 3D printing vs. CNC machining. [online]. 3D HUBS B.V. URL: <https://s3-eu-west-1.amazonaws.com/3dhubs-knowledge-base/cnc-vs-3d-printing/1-substractive-manufacturing.png>. Accessed 3.5.2020

7 HEIG-VD internal material.

8 Ultimaker S5 Pro Bundle. [online]. URL: <https://images.ctfassets.net/7cnpidfipnrw/3wpqCaf8ZgPTEbNgZ9izV/ea7d2d5630afefb8b9ebfc916c4635fa/Ultimaker-S5-Pro-Bundle-product-hero.jpg?f=center&fit=fill&fm=webp&w=2048>. Accessed 3.5.2020

9 Alkaios Bournias Varotsis. 2020. What is FDM? [online]. 3D HUBS B.V. URL: <https://www.3dhubs.com/knowledge-base/introduction-fdm-3d-printing/#what>. Accessed 4.5.2020

10 Fused deposition Modeling (FDM). 2019. [online]. Pick3DPrinter. URL: <https://pick3dprinter.com/3d-printer-types/>. Accessed 4.5.2020

11 What is ROS. 2020. [online]. Canonical Ltd. URL: <https://ubuntu.com/robotics/what-is-ros> . Accessed 17.5.2020

12 About ROS. 2020. [online]. ROS wiki. URL: <https://www.ros.org/about-ros/>. Accessed 5.5.2020

13 Robot Operating System. 1 May 2020. [online]. Wikipedia. URL: <https://en.wikipedia.org/wiki/Robot_Operating_System>. Accessed 5.5.2020

14 M.S. Hendriyawan Achmad. 9.2017. Fig 4. [online]. ResearchGate GmbH. URL: <https://www.researchgate.net/figure/Nodes-communication-model-in-the-ROS-environment-Fig-5-describes-the-proposed-ROS_fig2_319566597>. Accessed 5.5.2020

15 ROS computation graph level. 2014-06-21. [online]. ROS wiki. URL: <http://wiki.ros.org/ROS/Concepts>. Accessed 7.5.2020

16 Understanding ROS master. 2018-01-15. [online]. ROS wiki. URL: <http://wiki.ros.org/Master>. Accessed 7.5.2020

17 Understanding ROS nodes. 2018-12-04. [online]. ROS wiki. URL: <http://wiki.ros.org/Nodes>. Accessed 7.5.2020

18 Parameter server. 2018-11-08. [online]. ROS wiki. URL: <http://wiki.ros.org/Parameter%20Server>. Accessed 7.5.2020

19 Rosparam command-line tool. 2014-06-29. [online]. ROS wiki. URL: <http://wiki.ros.org/rosparam>. Accessed 7.5.2020

20 Messages. 2016-08-26. [online]. ROS wiki. URL: <http://wiki.ros.org/Messages>. Accessed 7.5.2020

21 Topics. 2019-02-20. [online]. ROS wiki. URL: <http://wiki.ros.org/Topics>. Accessed 7.5.2020

22 Services. 2019-07-18. [online]. ROS wiki. URL: <http://wiki.ros.org/Services>. Accessed 7.5.2020

23 Rosservice command-line tool. 2011-07-15. [online]. ROS wiki. URL: <http://wiki.ros.org/rosservice>. Accessed 7.5.2020

24 Bags. 2015-05-02. [online]. ROS wiki. <URL: http://wiki.ros.org/Bags>. Accessed 7.5.2020

25 Rosbag/ Commandline. 2019-03-27. [online]. ROS wiki. URL: <http://wiki.ros.org/rosbag/Commandline>. Accessed 7.5.2020

26 Study notes of autolabor 2.5 (3) — working space and compiling system of ROS. 16.1.2020. [online]. Develop Paper. URL: <https://developpaper.com/study-notes-of-autolabor-2-5-3-working-space-and-compiling-system-of-ros/>. Accessed 7.5.2020

27 Catkin/ Workspaces. 2017-07-07. [online]. ROS wiki. URL: <https://wiki.ros.org/catkin/workspaces>. Accessed 7.5.2020

28 Packages. 2019-04-14. [online]. ROS wiki. URL: <http://wiki.ros.org/Packages>. Accessed 7.5.2020

29 Catkin/ CMakeLists.txt. 2019-07-25. [online]. ROS wiki. URL: <http://wiki.ros.org/catkin/CMakeLists.txt>. Accessed 7.5.2020

30 Catkin/ package.xml. 2019-07-24. [online]. ROS wiki. URL: <http://wiki.ros.org/catkin/package.xml>. Accessed 14.5.2020

31 Metapackages. 2014-03-28. [online]. ROS wiki. URL: <http://wiki.ros.org/Meta-packages>. Accessed 7.5.2020

32 ROS community level. 2014-06-21. [online]. ROS wiki. URL: <http://wiki.ros.org/ROS/Concepts>. Accessed 7.5.2020

33 Names. 2014-06-21. [online]. ROS wiki. URL: <http://wiki.ros.org/ROS/Concepts>. Accessed 7.5.2020

34 Descartes. 2018-09-24. [online]. ROS wiki. URL: <http://wiki.ros.org/descartes>. Accessed 10.5.2020

35 Urdf/ Tutorials. 2016-04-11. [online]. ROS wiki. URL: <http://wiki.ros.org/urdf/Tutorials>. Accessed 11.5.2020

36 Additional/Standard Frames. 2018-07-11. [online]. ROS wiki. URL: <http://wiki.ros.org/Industrial/Tutorials/Create%20a%20URDF%20for%20an%20Industrial%20Robot#Additional.2FStandard_Frames>. Accessed 11.5.2020

37 Writing a tf listener (C++). 2017-07-12. [online]. ROS wiki. URL: <http://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20listener%20%28C%2B%2B%29>. Accessed 11.5.2020

38 Tf Package Summary. 2017-10-02. [online]. ROS wiki. URL: <http://wiki.ros.org/tf>. Accessed 11.5.2020

39 What is Marlin? 2020. [online]. Marlin. URL: <https://marlinfw.org/docs/basics/introduction.html>. Accessed 13.5.2020

40 danjperron. Jan 04, 2016. Sending serial data out (via USB) with C++. [online]. RASPBERRY PI FOUNDATION. URL: <https://www.raspberrypi.org/forums/viewtopic.php?t=131208>. Accessed 10.3.2020

Appendix 1. Node descartes_manufacturing

Command: rosrun manufacturing_6dof descartes_manufacturing

Function: Reading G-code and planning the path.

Code:

```
// Core ros functionality like ros::init and spin
#include <ros/ros.h>
#include <ros/package.h>

// ROS Trajectory Action server definition
#include <control_msgs/FollowJointTrajectoryAction.h>
// Means by which we communicate with above action-server
#include <actionlib/client/simple_action_client.h>

// Includes the descartes robot model we will be using
#include <descartes_moveit/ikfast_moveit_state_adapter.h>

// Includes the descartes trajectory type we will be using
#include <descartes_trajectory/axial_symmetric_pt.h>
#include <descartes_trajectory/cart_trajectory_pt.h>

// Includes the planner we will be using
#include <descartes_planner/dense_planner.h>

// Includes the utility function for converting to trajec-
tory_msgs::JointTrajectory's
#include <descartes_utilities/ros_conversions.h>

// Include cpp
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <math.h>
#include <cmath>

// Boost
#include <boost/algorithm/string.hpp>

/**
 * Makes the trajectory for the robot to follow, from G-Code file
 */
std::vector<descartes_core::TrajectoryPtPtr> makePath(std::string filename);

/**
 * Sends a ROS trajectory to the robot controller
 */
bool executeTrajectory(const trajectory_msgs::JointTrajectory& trajectory);
```

```
int main(int argc, char** argv)
{
  // Initialize ROS
  ros::init(argc, argv, "descartes_manufacturing");
  ros::NodeHandle nh;

  // Since we're not calling ros::spin() and doing the planning in a callback,
but rather just handling this
  // inline, we need to create an async spinner if our publishers are to work.
Note that many MoveIt components
  // will also not work without an active spinner and Descartes uses moveit
for its "groups" and "scene" descriptions
  ros::AsyncSpinner spinner (1);
  spinner.start();

  // 1. First thing first, let's create a kinematic model of the robot. In
Descartes, this is used to do things
  // like forward kinematics (joints -> pose), inverse kinematics (pose ->
many joints), and collision checking.

  // All of the existing planners (as of Nov 2017) have been designed with the
idea that you have "closed form"
  // kinematics. This means that the default solvers in MoveIt (KDL) will NOT
WORK WELL. I encourage you to produce
  // an ikfast model for your robot (see MoveIt tutorial) or use the OPW kine-
matics package if you have a spherical
  // wrist industrial robot. See http://docs.ros.org/kinetic/api/moveit_tuto-
rials/html/doc/ikfast/ikfast_tutorial.html

  // This package assumes that the move group you are using is pointing to an
IKFast kinematics plugin in its
  // kinematics.yaml file. By default, it assumes that the underlying kinemat-
ics are from 'base_link' to 'tool0'.
  // If you have renamed these, please set the 'ikfast_base_frame' and
'ikfast_tool_frame' parameter (not in the
  // private namespace) to the base and tool frame used to generate the IKFast
model.
  descartes_core::RobotModelPtr model (new descartes_moveit::IkFastMoveitSta-
teAdapter());

  // Name of description on parameter server. Typically just "robot_descrip-
tion". Used to initialize
  // moveit model.
  const std::string robot_description = "robot_description";

  // name of the kinematic group you defined when running MoveitSetupAssis-
tant. For many industrial robots this will be
  // "manipulator"
  const std::string group_name = "manipulator";

  // Name of frame in which you are expressing poses. Typically "world_frame"
or "base_link".
  /* Previously we were looking at base_link, so at the middle bottom of the
robot. Now there is added new frame named "custom",
     which we can move from the tx_40_macro.xacro. This is done, to be able
move needed frame anywhere we want, for example
     plan is that this whill be point in the corner of the printing table.
Similar changes are done to the printertool.cpp.
     It is "looking" transformation between custom frame and tool0, so it rec-
ognizes coordinate points correctly.
  */
```

```
  const std::string world_frame = "custom";
  // const std::string world_frame = "base_link";

  // tool center point frame (name of link associated with tool). The robot's
flange is typically "tool0" but yours
  // could be anything. We typically have our tool's positive Z-axis point
outward from the grinder, welder, etc.
  const std::string tcp_frame = "tool0";

  // Before you can use a model, you must call initialize. This will load ro-
bot models and sanity check the model.
  if (!model->initialize(robot_description, group_name, world_frame,
tcp_frame))
  {
    ROS_INFO("Could not initialize robot model");
    return -1;
  }

  model->setCheckCollisions(true); // Let's turn on collision checking.

  // 2. The next thing to do is to generate a path for the robot to follow.
The description of this path is one of the
  // cool things about Descartes. The source of this path is where this li-
brary ties into your application: it could
  // come from CAD or from surfaces that were "scanned".

  // Make the path by calling a helper function. See makePath()'s definition
for more discussion about paths.
  std::string filename;
  std::cout << "Enter G-Code filename: ";
  std::getline(std::cin, filename);
  std::vector<descartes_core::TrajectoryPtPtr> points = makePath(filename);

  // 3. Now we create a planner that can fuse your kinematic world with the
points you want to move the robot
  // along. There are a couple of planners now. DensePlanner is the naive,
brute force approach to solving the
  // trajectory. SparsePlanner may be faster for some problems (especially
very dense ones), but has recieved
  // less overall testing and evaluation.
  descartes_planner::DensePlanner planner;

  // Like the model, you also need to call initialize on the planner
  if (!planner.initialize(model))
  {
    ROS_ERROR("Failed to initialize planner");
    return -2;
  }

  // 4. Now, for the planning itself. This typically happens in two steps.
First, call planPath(). This function takes
  // your input trajectory and expands it into a large kinematic "graph".
Failures at this point indicate that the
  // input path may not have solutions at a given point (because of reach/col-
lision) or has two points with no way
  // to connect them.
  if (!planner.planPath(points))
  {
    ROS_ERROR("Could not solve for a valid path");
    return -3;
  }
```

```
  // After expanding the graph, we now call 'getPath()' which searches the
graph for a minimum cost path and returns
  // the result. Failures here (assuming planPath was good) indicate that your
path has solutions at every waypoint
  // but constraints prevent a solution through the whole path. Usually this
means a singularity is hanging out in the
  // middle of your path: the robot can solve all the points but not in the
same arm configuration.
  std::vector<descartes_core::TrajectoryPtPtr> result;
  if (!planner.getPath(result))
  {
    ROS_ERROR("Could not retrieve path");
    return -4;
  }

  // 5. Translate the result into something that you can execute. In ROS land,
this means that we turn the result into
  // a trajectory_msgs::JointTrajectory that's executed through a con-
trol_msgs::FollowJointTrajectoryAction. If you
  // have your own execution interface, you can get joint values out of the
results in the same way.

  // get joint names - this could be from the robot model, or from the parame-
ter server.
  std::vector<std::string> names;
  nh.getParam("controller_joint_names", names);

  // Create a JointTrajectory
  trajectory_msgs::JointTrajectory joint_solution;
  joint_solution.joint_names = names;

  // Define a default velocity. Descartes points without specified timing will
use this value to limit the
  // fastest moving joint. This usually effects the first point in your path
the most.
  const static double default_joint_vel = 0.5; // rad/s
  if (!descartes_utilities::toRosJointPoints(*model, result, de-
fault_joint_vel, joint_solution.points))
  {
    ROS_ERROR("Unable to convert Descartes trajectory to joint points");
    return -5;
  }

  // 6. Send the ROS trajectory to the robot for execution
  if (!executeTrajectory(joint_solution))
  {
    ROS_ERROR("Could not execute trajectory!");
    return -6;
  }
  std::cout << joint_solution << "\n";

  // Wait till user kills the process (Control-C)
  ROS_INFO("Done!");
  return 0;
}

descartes_core::TrajectoryPtPtr makeCartesianPoint(const Eigen::Isometry3d&
pose, double dt)
{
  using namespace descartes_core;
  using namespace descartes_trajectory;
```

```
  return TrajectoryPtPtr( new CartTrajectoryPt( TolerancedFrame(pose), Tim-
ingConstraint(dt)) );
}

descartes_core::TrajectoryPtPtr makeToleracedCartesianPoint(const
Eigen::Isometry3d& pose, double dt)
{
  using namespace descartes_core;
  using namespace descartes_trajectory;
  return TrajectoryPtPtr( new AxialSymmetricPt(pose, M_PI / 12.0, AxialSymmet-
ricPt::Z_AXIS, TimingConstraint(dt)) );
}


// Calculate delta time to match 40mm/min velocity. 40mm/min could change,
this is test value at the moment, and printertool.cpp have the same value.
double calculateDT(Eigen::Vector3d position, Eigen::Vector3d previousPosition)
{
  // Set velocity. 0.001 = 1mm/min
  double velocity = 0.04;

  // The distance formula
  return sqrt(pow(position(0) - previousPosition(0), 2) + pow(position(1) -
previousPosition(1), 2) + pow(position(2) - previousPosition(2), 2)) / veloc-
ity*60;
}

std::vector<descartes_core::TrajectoryPtPtr> makePath(std::string filename)
{
  // In Descartes, trajectories are composed of "points". Each point describes
what joint positions of the robot can
  // satisfy it. You can have a "joint point" for which only a single solution
is acceptable. You might have a
  // fully defined cartesian point for which many (8 or 16) different robot
configurations might work. You could
  // allow extra tolerances in any of these and even more points satisfy the
constraints.


  // Variables

  // First we define the folder and G-Code file from which we get the trajec-
tory
  std::ifstream gcodeFile;
  std::string completeFilename = ros::package::getPath("manufacturing_6dof") +
"/gcode/" + filename;
  // The file will be read line by line
  std::string line;
  // Each element of the line is delimited by the character 'space'
  std::string delimiters = "' '";
  // Each element is stored as a string
  std::vector<std::string> splitLine;

  // Path trajectory reurned by the function
  std::vector<descartes_core::TrajectoryPtPtr> pathTrajectory;
  // Point of the path trajectory
  descartes_core::TrajectoryPtPtr pt;
  // Initialize the pose as an identity matrix
  Eigen::Isometry3d poseN = Eigen::Isometry3d::Identity();
  // Position given by the G-Code (X,Y,Z)
  // This is also starting position (X,Y,Z)
  Eigen::Vector3d position(0.0, 0.0, 0.0);
```

```
  // Orientation given by the G-Code (I,J,K)
  // This is also starting orientation (I,J,K)
  Eigen::Vector3d toolAxisIJK(0.0, 0.0, 1.0);
  // The free axis of the tool, corresponding to the URDF tool0 frame
  Eigen::Vector3d toolAxis = Eigen::Vector3d::UnitZ();
  // Rotation axis used to align the tools axis
  Eigen::Vector3d rotationAxis;
  // Rotation angle used to align the tools axis
  double rotationAngle;
  // The orientation of the pose
  Eigen::AngleAxisd orientationIJK;
  // If not defined, the delta time used to get to next pose
  static double dt;

  // Loop variable
  int i = 1;

  // set XY offset
  int offsetXY = 0;

  // set Z offset
  int offsetZ = 0;

  // Define "home" position, for setting right velocity between starting posi-
tion and first gcode coordinate.
  // If you change home position, remember to change parameters in lines 225 &
228 too, and set .gcode end part.
  Eigen::Vector3d previousPosition(0.0, 0.0, 0.0);

  gcodeFile.open(completeFilename);
  if(gcodeFile){

    while(std::getline(gcodeFile,line)){
        boost::split(splitLine, line, boost::is_any_of(delimiters));
        // Loop to extract datas from each character.
        for(std::string element : splitLine){
            // Simply add a new condition if you want to extract and store
datas from another character
            if(element[0] == 'X')
                position(0) = (std::stod(&element[1])+offsetXY)/1000;
            else if(element[0] == 'Y')
                position(1) = (std::stod(&element[1])+offsetXY)/1000;
            else if(element[0] == 'Z')
                position(2) = (std::stod(&element[1])+offsetZ)/1000;
            else if(element[0] == 'I')
                toolAxisIJK(0) = std::stod(&element[1])/1000;
            else if(element[0] == 'J')
                toolAxisIJK(1) = std::stod(&element[1])/1000;
            else if(element[0] == 'K')
                toolAxisIJK(2) = std::stod(&element[1])/1000;
            else ROS_WARN("Element: %s, line: %i not handled.",
                element.c_str(), i);
        }


        std::cout << position << "\n";

        // Calculs to align tools axis vector
        toolAxisIJK.normalize();
        rotationAxis = toolAxis.cross(toolAxisIJK);
        rotationAngle = asin(rotationAxis.norm())+M_PI;
        orientationIJK = Eigen::AngleAxisd(rotationAngle,rotationAxis);
```

```
        // Homogeneous transformations
        poseN.translate(position);
        poseN.rotate(orientationIJK);

        // Apply velocity changes
        dt = calculateDT(position, previousPosition);
        previousPosition = position;

        // Allow degree of freedom for the tool axis
        pt = descartes_core::TrajectoryPtPtr(new descartes_trajectory::Axial-
SymmetricPt(poseN, M_PI / 12.0, descartes_trajectory::AxialSymmet-
ricPt::Z_AXIS, descartes_core::TimingConstraint(dt)));
        pathTrajectory.push_back(pt);

        // Format for next iteration
        i++;
        splitLine.clear();
        poseN = Eigen::Isometry3d::Identity();
    }
    gcodeFile.close();
  }else{
    ROS_ERROR("G-Code file not read");
  }


  return pathTrajectory;
}

bool executeTrajectory(const trajectory_msgs::JointTrajectory& trajectory)
{
  // Create a Follow Joint Trajectory Action Client
  actionlib::SimpleActionClient<control_msgs::FollowJointTrajectoryAction> ac
("joint_trajectory_action", true);
  if (!ac.waitForServer(ros::Duration(2.0)))
  {
    ROS_ERROR("Could not connect to action server");
    return false;
  }

  control_msgs::FollowJointTrajectoryGoal goal;
  goal.trajectory = trajectory;
  goal.goal_time_tolerance = ros::Duration(1.0);

  return ac.sendGoalAndWait(goal) == actionlib::SimpleClientGoalState::SUC-
CEEDED;
}
```

Appendix 2. Node staubli_tf_listener

Command: rosrun staubli_tf staubli_tf_listener

Function: Listen for robot position and print TF data to the console.

Code:

```
#include <ros/ros.h>
//The tf package provides an implementation of a TransformListener to help
make the task of receiving transforms easier.
//To use the TransformListener, we need to include the tf/transform_listener.h
header file.
#include <tf/transform_listener.h>


int main(int argc, char** argv){
  ros::init(argc, argv, "my_tf_listener");

  ros::NodeHandle node;

  /*
  Create a TransformListener object. Once the listener is created, it starts
receiving tf transformations over the wire, and buffers them for up to 10 sec-
onds.
  The TransformListener object should be scoped to persist otherwise it's
cache will be unable to fill and almost every query will fail. A common method
is to make the
  TransformListener object a member variable of a class.
  */
  tf::TransformListener listener;

  ros::Rate rate(50.0);
  while (node.ok()){
    tf::StampedTransform transform;
    try{

      /*
      The waitForTransform() takes four arguments:
      1. Wait for the transform from this frame...
      2. ... to this frame,
      3. at this time, and
      4. timeout: don't wait for longer than this maximum duration.
      So waitForTransform() will actually block until the transform between
the frames becomes available (this will usually take a few milliseconds),
      OR --if the transform does not become available-- until the timeout has
been reached.
      */
      listener.waitForTransform("base_link","tool0", ros::Time(0), ros::Dura-
tion(10.0));
```

```
      // Using this also helps to figure out what printer is reading. Useful
for bug fixes.
      // listener.waitForTransform("custom","tool0", ros::Time(0), ros::Dura-
tion(10.0));

      /*
      Here, the real work is done, we query the listener for a specific trans-
formation. Let's take a look at the four arguments:
      1. We want the transform from frame 1 to frame 2.
      2. The time at which we want to transform. Providing ros::Time(0) will
just get us the latest available transform.
      3. The object in which we store the resulting transform.
      We can also use ros::Time::now(), but for real tf use cases, it is often
perfectly fine to use Time(0).
      */
      listener.lookupTransform("base_link", "tool0", ros::Time(0), transform);

      // As previously mentioned, for printertool bug fixes.
      // listener.lookupTransform("custom", "tool0", ros::Time(0), transform);


      // Print location to the console
       double x = transform.getOrigin().x();
       double y = transform.getOrigin().y();
       double z = transform.getOrigin().z();
       double rx = transform.getRotation().x();
       double ry = transform.getRotation().y();
       double rz = transform.getRotation().z();

       std::cout << "Current position: (" " X" << (x*1000) << " Y" << (y*1000)
<< " Z" << (z*1000) << " Rx" << (rx*1000) << " Ry" << (ry*1000) << " Rz" <<
(rz*1000) << ")" << std::endl;

    }
    catch (tf::TransformException &ex) {
      ROS_ERROR("%s",ex.what());
      ros::Duration(1.0).sleep();

      continue;
    }

    rate.sleep();
  }
  return 0;
};
```

Appendix 3. Node printer_printertool

Command: rosrun printer printer_printertool

Function: Creates communication with printer related components. Sending the G-code, in synchronization with the robot movement.

Code:

```
#include <iostream>
#include <fstream>
#include "mySerial.h"
#include <math.h>
#include <cmath>
#include <sstream>
#include <vector>
// Core ros functionality like ros::init
#include <ros/ros.h>
//The tf package provides an implementation of a TransformListener to help
make the task of receiving transforms easier.
//To use the TransformListener, we need to include the tf/transform_listener.h
header file.
#include <tf/transform_listener.h>

// printer variables & parameters (F)
float printspeed = 40;      // Set velocity
float diambuse = 0.4;       // nozzle
float hcouche = 0.2;        //layer
float diamfil = 1.75;       //filament diameter
// F = feedrate = (60 * 4 * diambuse * hcouche * printspeed)/(diamfil^2 * pi)
// E = ((pi * diamfil^2) / (4)) * L1.

// Function for searching coordinates.
std::string getGcodeValue(std::string gcodeOperand, std::string gcodeLine) {
    if (isdigit(gcodeLine[gcodeLine.find(gcodeOperand) + 1]) ||
gcodeLine[gcodeLine.find(gcodeOperand) + 1]=='-')
    {
        int valueStartPos = gcodeLine.find(gcodeOperand) + 1;
        int spacePos = gcodeLine.find(" ", valueStartPos);
        int valueLength;
        std::string gcodeValue;
        if (spacePos == std::string::npos)
        {
            // Space not found after operand, value ends at the line end
            valueLength = gcodeLine.length() - valueStartPos;
            gcodeValue = gcodeLine.substr(valueStartPos);
        }
        else
        {
            // After operand found space, value is between operand and " "
            valueLength = spacePos - valueStartPos;
            gcodeValue = gcodeLine.substr(valueStartPos, valueLength);
```

```
        }
        return gcodeValue;
    }
    return "";
}


int main(int argc, char** argv)
{
    // Initialize ROS
    ros::init(argc, argv, "printertool");
    ros::NodeHandle nh;

    // Calculate new parameters to F
    float feedrate = (60 * 4 * diambuse * hcouche * printspeed)/(std::pow(di-
amfil, 2) * M_PI);

    std::ostringstream ssF;
    ssF << feedrate;
    std::string feedrateString = ssF.str();

    std::string filename, line, path;
    std::ifstream read;

    // Define port and baudrate, and make communication.
    mySerial serial = mySerial("/dev/ttyUSB0", 9600);

    std::vector<std::string> gcodeLines = {};

    // UNCOMMENT THIS, IF you want your file to be read from same directory
with the script. ////////////////////////
    /*
//
    std::cout << "Enter G-Code filename: ";
//
    std::getline(std::cin, filename);
//
    read.open(filename);
//
    */
//

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////


    // UNCOMMENT THIS, IF you want your file to be read from absolute path.
//////////////////////////////////////////////////////////////////////////
    //*
//
    std::cout << "Enter G-Code filename: ";
//
    std::getline(std::cin, filename);
//
    read.open("/home/tomiliatsereh/backupstaubli_ws/src/manufactur-
ing_6dof/gcode/" + filename); // "home/[USERNAME]/[PATH_TO_YOUR_FILE]" <- edit
//

//

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
```

```
    //* UNCOMMENT THIS, IF you want to ask user for path and filename.
////////////////////////////////////////////
    /*
//
    std::cout << "Enter full path to G-code file folder: ";
//
    std::getline(std::cin, path);
//

//
    std::cout << "Enter G-Code filename: ";
//
    std::getline(std::cin, filename);
//

//
    read.open(path + filename);
//
    */
//

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////


    if (read.is_open())
    {
        std::string oldE = "0";
        std::string actE;
        double newE;

        // Search F and change feedrate value. Overwrite with new parameters.
        while (getline(read, line))
        {
            if (isdigit(line[line.find("F") + 1]))
            {
                int fStartPos = line.find("F") + 1;
                int fSpacePos = line.find(" ", fStartPos);
                int feedrateLength = fSpacePos - fStartPos;
                line.replace(fStartPos, feedrateLength, feedrateString);
            }
            /*
            // There is possibility, that F is not stored and should be sended
in every line.
            else
            {
                If in future there occurs problems with printing, start with
adding F in every line. This could be the main point to start
                figuring out what is wrong. Second point is to look at G0 and
G1. If this will not work, other way could be to send to the printer
                lines in shape like only "G0 E0.123 F12.345". Remember, that
in Gcode E parameter is growing all the time, so it should be calculated
                for only one point.
            }
            */


            // Search E and change parameters
            if (isdigit(line[line.find("E") + 1]))
            {
                int eStartPos = line.find("E") + 1;
```

```
                int eSpacePos = line.find(" ", eStartPos);
                int eLength;
                if (eSpacePos == std::string::npos)
                {
                    // Space not found after "E", extruder value ends at the
line end
                    eLength = line.length() - eStartPos;
                    actE = line.substr(eStartPos);
                }
                else
                {
                    // After "E" found space, extruder value is between "E"
and " "
                    eLength = eSpacePos - eStartPos;
                    actE = line.substr(eStartPos, eLength);
                }
                newE = std::stod(actE) - std::stod(oldE);
                // Convert newE to string
                std::ostringstream ssE;
                ssE << newE;
                std::string extruderString = ssE.str();
                line.replace(eStartPos, eLength, extruderString);
                oldE = actE;
            }

            // cout just to see what is happening
            // std::cout << line << "\n";

            gcodeLines.push_back(line);
        }
        read.close();

        /*
        Create a TransformListener object. Once the listener is created, it
starts receiving tf transformations over the wire, and buffers them for up to
50 seconds.
        The TransformListener object should be scoped to persist otherwise
it's cache will be unable to fill and almost every query will fail. A common
method is to make the
        TransformListener object a member variable of a class.
        */

        tf::TransformListener listener;
        ros::Rate rate(50.0);   //  frequency (hz)
        int lineIndex = 0;

        // Robot start position coordinates
        double destinationX = 0;
        double destinationY = 0;
        double destinationZ = 0;

        // Sending all lines at the beginning, before first line with XY AND Z
is found. After line with XYZ is found, waiting coordinates to match with the
robot and sending them line by line.
        // When robot reaches point A, send point B to the USB etc.
        // If line in gcode does not include XY or Z, send it straightly to
the USB.
        while (destinationX==0 && destinationY==0 && destinationZ==0)
        {
            // Sending lines to USB port. When XYZ found, stop and wait for
matching coordinates from robot.
```

```
            if (getGcodeValue("X", gcodeLines[lineIndex])!="" && getGcode-
Value("Y", gcodeLines[lineIndex])!="" && getGcodeValue("Z", gcodeLines[lineIn-
dex])!="")
            {
                destinationX = std::stod(getGcodeValue("X", gcodeLines[lineIn-
dex]));
                destinationY = std::stod(getGcodeValue("Y", gcodeLines[lineIn-
dex]));
                destinationZ = std::stod(getGcodeValue("Z", gcodeLines[lineIn-
dex]));
            }
            std::cout << gcodeLines[lineIndex] << "\n";
            serial.Send(gcodeLines[lineIndex]);
            lineIndex++;
        }
        while (nh.ok()){
            tf::StampedTransform transform;
            try{
                /*
                The waitForTransform() takes four arguments:
                1. Wait for the transform from this frame...
                2. ... to this frame,
                3. at this time, and
                4. timeout: don't wait for longer than this maximum duration.
                So waitForTransform() will actually block until the transform
between the frames becomes available (this will usually take a few millisec-
onds),
                OR --if the transform does not become available-- until the
timeout has been reached.
                */
                listener.waitForTransform("custom","tool0", ros::Time(0),
ros::Duration(10.0));

                /*
                Here, the real work is done, we query the listener for a spe-
cific transformation. Let's take a look at the four arguments:
                1. We want the transform from frame 1 to frame 2.
                2. The time at which we want to transform. Providing
ros::Time(0) will just get us the latest available transform.
                3. The object in which we store the resulting transform.
                We can also use ros::Time::now(), but for real tf use cases,
it is often perfectly fine to use Time(0).
                */
                listener.lookupTransform("custom", "tool0", ros::Time(0),
transform);

                // Latest XYZ transform
                double x = transform.getOrigin().x();
                double y = transform.getOrigin().y();
                double z = transform.getOrigin().z();

                // Print robot location to the console
                // std::cout << "Current position: (" << x << "," << y << ","
<< z << ")" << std::endl;

                // Checking and waiting matching values, from gcode file and
robot position.
                // std::cout << (x*1000) << " == " << destinationX << "\n";
                // std::cout << (y*1000) << " == " << destinationY << "\n";
                // std::cout << (z*1000) << " == " << destinationZ << "\n";
```

```
                // Set tolerance between points. Decrease value for better ac-
curacy.
                double tolerance = 0.05;
                if (std::abs(x*1000-destinationX)<tolerance &&
std::abs(y*1000-destinationY)<tolerance && std::abs(z*1000-destinationZ)<tol-
erance)
                {
                    // Send line, if it is not including XYZ straight away
                    while (getGcodeValue("X", gcodeLines[lineIndex])=="" &&
getGcodeValue("Y", gcodeLines[lineIndex])=="" && getGcodeValue("Z",
gcodeLines[lineIndex])=="")
                    {
                        std::cout << gcodeLines[lineIndex] << "\n";
                        serial.Send(gcodeLines[lineIndex]);
                        lineIndex++;
                    }
                     // XYZ found, send line and save coordinate values.
                    std::cout << gcodeLines[lineIndex] << "\n";
                    serial.Send(gcodeLines[lineIndex]);

                    if (getGcodeValue("X", gcodeLines[lineIndex])!="")
                    {
                        destinationX = std::stod(getGcodeValue("X",
gcodeLines[lineIndex]));
                    }
                    if (getGcodeValue("Y", gcodeLines[lineIndex])!="")
                    {
                        destinationY = std::stod(getGcodeValue("Y",
gcodeLines[lineIndex]));
                    }
                    if (getGcodeValue("Z", gcodeLines[lineIndex])!="")
                    {
                        destinationZ = std::stod(getGcodeValue("Z",
gcodeLines[lineIndex]));
                    }

                    lineIndex++;
                    std::cout << gcodeLines[lineIndex] << "\n";
                }
            }
            catch (tf::TransformException &ex) {
                ROS_ERROR("%s",ex.what());
                ros::Duration(1.0).sleep();

                continue;
            }

            rate.sleep();
        }
    }

    else
        std::cout << "Unable to open file \n";

    return 0;
}
```

Appendix 4. Source code mySerial.cpp

Source code, which is responsible for serial communication in printer_printertool node.

```
/* FYI!!! THIS CODE IS ORIGINALLY FROM HERE! -> https://www.raspber-
rypi.org/forums/viewtopic.php?t=131208
 I uncommented lines 90-103 from this code. Didn't have any problems with it,
but just to be sure, because
 i was thinking that we could maybe have problems in the future.
 - T-I.Tsereh
*/

#include <asm/termbits.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <fcntl.h>
#include <iostream>
using namespace std;

#include "mySerial.h"

mySerial::mySerial(string deviceName, int baud)
{
   handle=-1;
   Open(deviceName,baud);
}

mySerial::~mySerial()
{
  if(handle >=0)
      Close();
}

void mySerial::Close(void)
{
   if(handle >=0)
      close(handle);
   handle = -1;
}


bool mySerial::Open(string deviceName , int baud)
{
    struct termios tio;
    struct termios2 tio2;
    this->deviceName=deviceName;
    this->baud=baud;
    handle  = open(this->deviceName.c_str(),O_RDWR | O_NOCTTY /* | O_NONBLOCK
*/);

    if(handle <0)
       return false;
    tio.c_cflag =  CS8 | CLOCAL | CREAD;
    tio.c_oflag = 0;
    tio.c_lflag = 0;       //ICANON;
    tio.c_cc[VMIN]=0;
    tio.c_cc[VTIME]=1;     // time out every .1 sec
```

```
    ioctl(handle,TCSETS,&tio);

    ioctl(handle,TCGETS2,&tio2);
    tio2.c_cflag &= ~CBAUD;
    tio2.c_cflag |= BOTHER;
    tio2.c_ispeed = baud;
    tio2.c_ospeed = baud;
    ioctl(handle,TCSETS2,&tio2);

//   flush buffer
    ioctl(handle,TCFLSH,TCIOFLUSH);

    return true;
}

bool mySerial::IsOpen(void)
{
   return( handle >=0);
}

bool mySerial::Send( unsigned char  * data,int len)
{
   if(!IsOpen()) return false;
   int rlen= write(handle,data,len);
   return(rlen == len);
}

bool mySerial::Send( unsigned char value)
{
   if(!IsOpen()) return false;
   int rlen= write(handle,&value,1);
   return(rlen == 1);
}

bool mySerial::Send(std::string value)
{
   if(!IsOpen()) return false;
   int rlen= write(handle,value.c_str(),value.size());
   return(rlen == value.size());
}


/* int  mySerial::Receive( unsigned char  * data, int len)
{
   if(!IsOpen()) return -1;

   // this is a blocking receives
   int lenRCV=0;
   while(lenRCV < len)
     {
       int rlen = read(handle,&data[lenRCV],len - lenRCV);
       lenRCV+=rlen;
     }
   return  lenRCV;
}
*/

bool mySerial::NumberByteRcv(int &bytelen)
{
   if(!IsOpen()) return false;
   ioctl(handle, FIONREAD, &bytelen);
   return true;
```

```
}
```

Appendix 5. Header file mySerial.h

Header file, which is responsible for serial communication in printer_printertool node.

```
#ifndef SERIAL
#define SERIAL
#include <string>


class  mySerial
{

public:

  int handle;
  std::string  deviceName;
  int baud;

  mySerial(std::string deviceName, int baud);
  ~mySerial();

  bool Send( unsigned char  * data,int len);
  bool Send(unsigned char value);
  bool Send( std::string value);
  int Receive( unsigned char  * data, int len);
  bool IsOpen(void);
  void Close(void);
  bool Open(std::string deviceName, int baud);
  bool NumberByteRcv(int &bytelen);
};

#endif
```

Appendix 6. File tx40_macro.xacro

Function: Macro file, which is affecting to the Unified Robot Description Format (URDF).

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://wiki.ros.org/xacro">
  <xacro:include filename="$(find staubli_resources)/urdf/common_materi-
als.xacro"/>

  <xacro:macro name="staubli_tx40" params="prefix">
    <!-- links: main serial chain -->
    <link name="${prefix}base_link">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/vis-
ual/base_link.stl"/>
        </geometry>
        <xacro:material_staubli_ral_melon_yellow />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/colli-
sion/base_link.stl"/>
        </geometry>
      </collision>
      <!--
      <inertial>
        <mass value="5.82375"/>
        <origin xyz="-0.00892 -0.00017 0.07857" rpy="0.0 0.0 0.0"/>
        <inertia ixx="0.020725706" ixy="0.00008955" ixz="0.00193426"
                                    iyy="0.026779837"  iyz="0.000024618"
                                                         izz="0.021448879"/>
      </inertial>
      -->
    </link>
    <link name="${prefix}link_1">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/vis-
ual/link_1.stl"/>
        </geometry>
        <xacro:material_staubli_ral_melon_yellow />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/colli-
sion/link_1.stl"/>
        </geometry>
      </collision>
      <inertial>
        <mass value="8.81740"/>
        <origin xyz="-0.00029 0.00931 -0.03386" rpy="0.0 0.0 0.0"/>
        <inertia ixx="0.050149637" ixy="0.000391605" ixz="400337.06"
```

```
                                        iyy="0.043332881"  iyz="0.003817008"
                                                             izz="0.032839343"/>
      </inertial>
    </link>
    <link name="${prefix}link_2">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/vis-
ual/link_2.stl"/>
        </geometry>
        <xacro:material_staubli_ral_melon_yellow />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/colli-
sion/link_2.stl"/>
        </geometry>
      </collision>
      <inertial>
        <mass value="6.12815"/>
        <origin xyz="0.00008 0.14711 0.10312" rpy="0.0 0.0 0.0"/>
        <inertia ixx="0.056410712" ixy="-0.000058219" ixz="0.000080107"
                                        iyy="0.060088807" iyz="-0.000308218"
                                                             izz="0.010289325"/>
      </inertial>
    </link>
    <link name="${prefix}link_3">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/vis-
ual/link_3.stl"/>
        </geometry>
        <xacro:material_staubli_ral_melon_yellow />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/colli-
sion/link_3.stl"/>
        </geometry>
      </collision>
      <inertial>
        <mass value="3.36342"/>
        <origin xyz="0.00765 0.01281 -0.01551" rpy="0.0 0.0 0.0"/>
        <inertia ixx="0.008592818" ixy="0.000098473" ixz="-0.000358456"
                                        iyy="0.008894321"  iyz="-0.000370698"
                                                             izz="0.006740357"/>
      </inertial>
    </link>
    <link name="${prefix}link_4">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/vis-
ual/link_4.stl"/>
        </geometry>
        <xacro:material_staubli_ral_melon_yellow />
      </visual>
      <collision>
```

```
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/colli-
sion/link_4.stl"/>
        </geometry>
      </collision>
      <inertial>
        <mass value="2.63788"/>
        <origin xyz="-0.00283 0.00 0.14236" rpy="0.0 0.0 0.0"/>
        <inertia ixx="0.009920969" ixy="0.000000028" ixz="-0.000208997"
                                    iyy="0.008812832"  iyz="0.000000954"
                                                         izz="0.00406691"/>
      </inertial>
    </link>
    <link name="${prefix}link_5">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/vis-
ual/link_5.stl"/>
        </geometry>
        <xacro:material_staubli_ral_grey_aluminium />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/colli-
sion/link_5.stl"/>
        </geometry>
      </collision>
      <inertial>
        <mass value="0.21237"/>
        <origin xyz="0.00 0.00 0.01975" rpy="0.0 0.0 0.0"/>
        <inertia ixx="0.000117818" ixy="0.00"          ixz="-0.000000003"
                                    iyy="0.000132032" iyz="-0.00000000345"
                                                        izz="0.000054314"/>
      </inertial>
    </link>
    <link name="${prefix}link_6">
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/vis-
ual/link_6.stl"/>
        </geometry>
        <xacro:material_staubli_ral_grey_aluminium />
      </visual>
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <mesh filename="package://staubli_tx40_support/meshes/colli-
sion/link_6.stl"/>
        </geometry>
      </collision>
      <inertial>
        <mass value="0.01746"/>
        <origin xyz="-0.00023 0.00 -0.00463" rpy="0.0 0.0 0.0"/>
        <inertia ixx="0.000002221" ixy="0.00"          ixz="-0.000000005"
                                    iyy="0.000002156" iyz="0.0"
                                                        izz="0.000004187"/>
      </inertial>
    </link>
```

```
    <!-- joints: main serial chain -->
    <joint name="${prefix}joint_1" type="revolute">
      <origin xyz="0 0 0.32" rpy="0 0 0"/>
      <parent link="${prefix}base_link"/>
      <child link="${prefix}link_1"/>
      <axis xyz="0 0 1"/>
      <limit lower="${radians(-180.0)}" upper="${radians(180.0)}" ef-
fort="40.0" velocity="${radians(287.0)}" />
      <dynamics damping="0.0" friction="0.0"/>
    </joint>
    <joint name="${prefix}joint_2" type="revolute">
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <parent link="${prefix}link_1"/>
      <child link="${prefix}link_2"/>
      <axis xyz="0 1 0"/>
      <limit lower="${radians(-125.0)}" upper="${radians(125.0)}" ef-
fort="11.0" velocity="${radians(287.0)}" />
      <dynamics damping="0.0" friction="0.0"/>
    </joint>
    <joint name="${prefix}joint_3" type="revolute">
      <origin xyz="0 0.035 0.225" rpy="0 0 0"/>
      <parent link="${prefix}link_2"/>
      <child link="${prefix}link_3"/>
      <axis xyz="0 1 0"/>
      <limit lower="${radians(-138.0)}" upper="${radians(138.0)}" effort="8.0"
velocity="${radians(430.0)}" />
      <dynamics damping="0.0" friction="0.0"/>
    </joint>
    <joint name="${prefix}joint_4" type="revolute">
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <parent link="${prefix}link_3"/>
      <child link="${prefix}link_4"/>
      <axis xyz="0 0 1"/>
      <limit lower="${radians(-270.0)}" upper="${radians(270.0)}" effort="4.0"
velocity="${radians(410.0)}" />
      <dynamics damping="0.0" friction="0.0"/>
    </joint>
    <joint name="${prefix}joint_5" type="revolute">
      <origin xyz="0 0 0.225" rpy="0 0 0"/>
      <parent link="${prefix}link_4"/>
      <child link="${prefix}link_5"/>
      <axis xyz="0 1 0"/>
      <limit lower="${radians(-120.0)}" upper="${radians(133.5)}" effort="2.6"
velocity="${radians(320.0)}" />
      <dynamics damping="0.0" friction="0.0"/>
    </joint>
    <joint name="${prefix}joint_6" type="revolute">
      <origin xyz="0 0 0.065" rpy="0 0 0"/>
      <parent link="${prefix}link_5"/>
      <child link="${prefix}link_6"/>
      <axis xyz="0 0 1"/>
      <limit lower="${radians(-270.0)}" upper="${radians(270.0)}" effort="1.4"
velocity="${radians(700.0)}" />
      <dynamics damping="0.0" friction="0.0"/>
    </joint>


    <!-- ROS-Industrial 'base' frame: base_link to Staubli World Coordinates
transform -->
    <link name="${prefix}base" />
    <joint name="${prefix}base_link-base" type="fixed">
```

```
      <origin xyz="0 0 0.32" rpy="0 0 0"/>
      <parent link="${prefix}base_link"/>
      <child link="${prefix}base"/>
    </joint>

    <!-- ROS-Industrial 'flange' frame: attachment point for EEF models -->
    <link name="${prefix}flange" />
    <joint name="${prefix}joint_6-flange" type="fixed">
      <origin xyz="0 0 0" rpy="0 ${radians(-90.0)} 0" />
      <parent link="${prefix}link_6" />
      <child link="${prefix}flange" />
    </joint>

    <!-- ROS-Industrial 'tool0' frame: all-zeros tool frame -->
    <link name="${prefix}tool0" />
    <joint name="${prefix}flange-tool0" type="fixed">
      <origin xyz="0 0 0" rpy="0 ${radians(90.0)} 0" />
      <parent link="${prefix}flange" />
      <child link="${prefix}tool0" />
    </joint>


    <!-- Custom frame -->

    <link name="${prefix}custom" />
    <joint name="${prefix}custom-frame" type="fixed">
      <origin xyz="-0.108687 0.178755 0.324798" rpy="0.0 0.0 0.0"/>
      <parent link="${prefix}base_link"/>
      <child link="${prefix}custom"/>
    </joint>


  </xacro:macro>
</robot>
```