G-Code to RAPID translator for Robot-Studio

Daniel Nilsson
A THESIS SUBMITTED TO THE DEPARTMENT OF ENGINEERING SCIENCE
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE WITH SPECIALIZATION IN ROBOTICS
AT UNIVERSITY WEST

2016

Date: June 09, 2016
Author: Daniel Nilsson
Examiner: Bo Svensson
Advisor: Svante Augustsson, University West
Programme: Master Programme in Robotics
Main field of study: Automation with a specialization in industrial robotics
Credits: 60 Higher Education credits (see the course syllabus)
Keywords: SME, RobotStudio, G-Code, CAM, add-in, C#, 3d-printing, industrial robot, printing robot, robot machining, robot milling
Publisher: University West, Department of Engineering Science
S-461 86 Trollhättan, SWEDEN
Phone: + 46 520 22 30 00 Fax: + 46 520 22 32 99 Web: www.hv.se
Summary

With the emerging development of new technology and the constantly falling prices, more companies will find interest in industrial robots. Until today, the typical robot users have been large scale car manufacturers. But there exists a big potential market within the small to medium businesses that already uses of CNC machines. Attracting smaller businesses to start using industrial robots could open up the doors to new possibilities and increase their production. Unfortunately, most people still lack the knowledge of operating and programming industrial robots. But many companies have knowledge in G-code which is normally used in CNC machines. That is why this work is focussing on the development of a software that opens up the possibility to make use of G-code to program advanced robot paths with minimal user input. By striving for easier handling of robots, the vision about a more efficient and automated society will become one step closer. The introduction straightens out the different basic principles of CNC milling machines and robots. This will introduce the reader and highlight the different similarities and differences that exist between robots and CNC mills. The goal of this work has been to create an add-in application for ABB’s off-line programming software RobotStudio that was able to import at least one type of file format commonly used for CNC milling. The program should be able to handle some basic functionality, the focus has been on the standard iso6983 type of G-code. The project started with a literature study that gave the author a better insight in both the previous research within the area but also deeper knowledge of the systems CNC mills and robots. The work continued with the development of a software able to import the specified file format. The software has been built in C# and is built as an add-in software for ABB’s offline programming software RobotStudio. The result presents a software that is able to read different types of G-code and translate them into generated paths in RobotStudio. The software also has an inbuilt function in order to parameterize the G02 and G03 commands that represent curves in G-code into straight line segments of the type MoveL in RobotStudio.
Affirmation

This thesis work, *G-Code to RAPID translator for RobotStudio*, was written as part of the master degree work needed to obtain a Master of Science with specialization in Robotics degree at University West. All material in this report, that is not my own, is clearly identified and used in an appropriate and correct way. The main part of the work included in this degree project has not previously been published or used for obtaining another degree.

Signature by the author

Daniel Nilsson

06/09/2016
Date
## Contents

**Preface**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY</td>
<td>III</td>
</tr>
<tr>
<td>AFFIRMATION</td>
<td>IV</td>
</tr>
<tr>
<td>CONTENTS</td>
<td>V</td>
</tr>
</tbody>
</table>

**Main Chapters**

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.1 PROJECT BACKGROUND</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2 AIM</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1.3 LIMITATIONS</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>BACKGROUND</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>2.1 G-CODE</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>2.2 RAPID CODE</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.3 COORDINATE SYSTEMS CNC MACHINES</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.4 COORDINATE SYSTEMS INDUSTRIAL ROBOTS</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>LITERATURE REVIEW</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3.1 SIMILAR EXISTING SOLUTIONS</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3.2 RESEARCH WITH THE USE OF AUTOMATED CODE</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>3.3 DISCUSSION</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>METHOD</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>4.1 LITERATURE STUDY</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>4.2 PLANNING</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>4.3 LEARNING C#</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>4.4 CODING</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>4.5 DEBUGGING</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>4.6 TEST AND VERIFICATION</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>SOFTWARE DEVELOPMENT</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>5.1 INITIALIZATION</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>5.2 WIREFRAME OF THE FRONTEND DESIGN</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>5.3 FLOWCHART AND BACKEND DESIGN</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>5.4 CODE</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>5.5 ADD-IN SKELETON AND UI</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>5.6 FILE READER FUNCTION</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>5.7 CREATING TARGETS FOR G00 AND G01</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>5.8 TRANSLATING THE G02, G03 INTO RAPID</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>5.9 CREATING A PATH</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>5.10 THE INTEGRATION PROCESS</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>5.11 TEST AND VERIFICATION</td>
<td>22</td>
</tr>
</tbody>
</table>
1 Introduction

The industrial robot has been a tool for the manufacturing industry for the last 5 decades. Where the automotive industry is the main user of robots and stood for 43% of the world total industrial robot sales in 2014 [1]. The total industrial robot sales are expected to grow by an average 15% a year at the same time as the ease of use is reported to be one of the main trends during these upcoming years [2]. This makes up for a good opportunity to introduce the robots as an automation tool for the small to medium enterprises (SME).

1.1 Project background

Pan et.al gives an example in their article were they state that manual programming of the spot arc welding for a vehicle hull takes more than eight months but the actually executed motion cycle performed by the robots only takes 16h which gives a ratio of programming to executed motion of 360:1 for advanced robot path programming [3]. This means that few SME’s will benefit from the use of industrial robots that require more advanced paths due to their often quite low batch sizes. According to Eurostat, European Union’s statistics division. SME’s stands for 99.8% of all the non-financial businesses in EU. Eurostat also states that the value added by the manufacturing and construction sector of the SME’s alone during 2011 exceeded 1200 thousand million euros [4]. And all the SME’s total value added represents 58.6% of all the value added by the non-financial businesses in EU [4]. This potential in getting the SME’s to adopt the industrial robot to a wider extent by reducing the complexity of the path planning is the very foundation in this litterateur research. The proposed method for easier preparation of the robot is to create G-Code support for the ABB robots. The reason for this is to let the SME’s with knowledge of G-Code and CAM systems use their existing software to prepare robot paths in a more simplified manner. This by creating an add-in for the default off-line programming software for ABB robots Robot Studio [5]. The proposed method is then meant to translate G-Code paths to robot paths in RAPID code which then could be used by the actual robot controller. Therefore, the need for a G-code to RAPID translator in RobotStudio is investigated in the initial literature review to this thesis work. This was done by looking for different existing solutions on how to create machining paths and how to be able to use paths generated from CAM tools.
1.2 Aim

- Develop an add-in for ABB RobotStudio in order to attract more businesses to make use of industrial robots.
  - The add-in shall make it possible to create robot paths based on input in the form of G-Code in an easy manner.
  - The add-in shall consist of a logical user interface that is easy to understand and also allows the inexperienced robot user to benefit from the software.
- In order to achieve the previous aims, the initial part of the project will focus on gaining enough knowledge in both C# and ABB’s API for RobotStudio to be able to complete the programming tasks needed.

1.3 Limitations

- The thesis work will be limited to ABB’s articulated robotic arms.
- The add-in will mainly focus on translating standard (iso6983-1) G-Code with basic functionality.
2 Background

Robots today still lacks the use of a standardized programming language. As it is today each vendor depends on their own type of control system. The code between ABB and KUKA differs quite a lot in the way it’s structured. In the KRL language for KUKA robots, the user can define an initial block that decides different parameters for example the tool. This is then not needed to define in the movement commands while the tool is defined for each movement command in the RAPID code. Another difference between these vendors are that KUKA robotics control system relies on x, y, z coordinates for translation and Roll, Pitch, Yaw to define the rotational angles in degrees [6]. The roll, pitch, yaw way of expressing rotation could be described as the same movements an airplane does according to Bolmsjö [7], where roll represents a rotation along the hands longitudinal axis. The pitch is a rotation along a horizontal axis that creates an up and down rotation of the end effector. The yaw represents a rotation along a vertical axis with respect to the hand longitudinal axis, this gives a side to side rotation of the end effector. ABB robotics is also using x, y, z coordinates to define the translation but then instead uses quaternions which give the rotational angles. The quaternion notation is an advanced mathematical expression of describing the rotational position in the three-dimensional space, the notation is commonly used in computer graphics [8].

Pan et.al [3] believes that more suitable programming methods for SME’s will be a growing market the next coming years. Something that Brogård confirms as he writes “What is needed in this environment is low-cost safe robot systems that are easy to install, configure, calibrate, program and maintain” [9] regarding robot systems for SME’s.

If the robots could get easier to program for machining and printing applications this would open up a lot of new areas where SME’s could benefit from the use of industrial robots. They could then also more easily apply the industrial robots which would result in the extended use of industrial robots for these companies.
2.1 G-code

G-code is an ISO standard (ISO6983-1) for programming of NC machines. Although it is an ISO standard each vendor usually has their own refined version of this standard, this means that some commands may differ a bit between different vendors of machines. G-code navigates the machine by sending one x, y, z coordinate after another to the machine in combination with one of the four existing G-commands. There are four types of movement in the G-code language. The first G00 defines high-speed linear movement and is usually used for transportation of the machine head from one point to another without performing any work. The three remaining commands are usually used when the machine is executing its specified task, for example, milling. G01 means linear motion with a reduced user defined speed in order to perform some type of machining. G02 means clockwise circular motion and the third one is G03 counter-clockwise circular motion. [10]

3d-printers is also using G-code but in a slightly different way. The G-code used for 3d-printing is a simplified version of the CNC machining code. 3d-printers only relies on the G00 command and the G01 command. They don’t use anything else than a linear motion to define the paths. Instead, they consume a lot of code since a curved object is built by a lot of short straight lines. [11]
2.2 RAPID code

RAPID is the programming language used to program ABB’s industrial robots. The robots rely on three different types of motion. MoveL, this is a linear motion which forces the robot to move in a straight line between two points that the programmer defines. MoveL could therefore not be used for all paths. This since there are several position in the robot space where the robot can’t pass in a certain configuration. An obvious example of an impossible movement with MoveL is if the programmer wants to move the robot to the opposite side of the base. Then the robot will try to move through itself causing an error. MoveJ is a point to point motion that lets the robot move to the second point in the easiest way. A benefit of letting the robot decide the easiest way is that if the programmer now wants to move the robot to the opposite side of the base as in the previous example, then the robot will avoid to collide with its own base and create a motion around the base instead. MoveJ is also often required in the beginning of the robot path in order to let the robot reconfigure itself without the constraint of following a specific path. A drawback with this motion is that it is almost impossible for the programmer to predict exactly how the robot will move when using MoveJ. This due to the articulated robot’s natural movement is a circular motion for each joint, the construction of serial linked rotational joints results in complicated total joint motion that is hard to predict for a human [12]. MoveJ is therefore not a suitable command if space is tight and the robot risking to collide on the way to the next path, since the robot won’t follow a straight line to the next point but instead take the easiest way.

Although when performing advanced paths like welding and other applications the robot programming is usually done by MoveL commands exclusively except for the start and end points. This is done in order to maintain total control of the TCP during the whole path.

The third command is MoveC. This command is used to create arc motion that doesn’t need to be circular. It is used by defining two points. The first point is a point along the circle the second point is the end point of the circle. The robot will then start from where it ended the last command and create a circular motion that goes through the first point and then ends at the second point. The command allows for a maximum angle of 180 degrees for each circular motion. A full circle can, therefore, be made by two MoveC commands using 4 points totally.

2.3 Coordinate systems CNC machines

A 3-axis vertical mill will be taken as an example in the following paragraph. In order to be able to navigate a CNC machine by the coordinates generated in the G-code the machine needs to have some kind of starting point. As an addition to the starting point, the machine also needs to know the direction. If the machine is told to go 100mm in the x-direction it has to know what direction x is. The point needs to refer to a predefined coordinate system. Further, a CNC machine has multiple coordinate systems in order to be able to perform a milling task. To begin with it has a machine reference point. This is the origin of the coordinate system that the machine works in and all the other reference points refer to. The machine reference point is also called machine zero or home position. The machine reference point is defined by the manufacturer for the specific machine. When the power has been turned off and the machine is turned on again the machine needs to do a machine zero home command. This is done in order for the machine axes to always start at the same position [10].
The tool also has a coordinate system called tool reference point that is defined at the tip of the tool along the vertical center line of the tool [10]. The reason for this is that the machine should be able to handle tools of different lengths. By defining a coordinate system at the tip of the tool, the operator tells the machine where the tip of the tool is located.

The part reference point of the CNC machine is a floating point which means that it is defined by the operator, this is done according to the specific task [10]. This point can be defined anywhere within the workspace of the machine. This coordinate system is used to establish a relationship between the drawing of the part, the machine reference point and tool reference point [10]. To simplify, if a square block is about to be milled, the machine reference point will be defined at one of the cubes top corners assumed that a vertical milling machine is used. By defining this third coordinate system the CNC-mill will have the information about where the object, that is about to be milled, is located with respect to the machine reference point. Since the machine also knows where the tool tip is located with respect to the machine reference point. It is now possible for the machine to calculate how it should move in order to translate the tool to start milling the square block. This, however, requires that if more than one square block should be milled after each other the blocks need to be placed at the exact same location each time. To accomplish this the operator could use a specialized fixture. If not the part reference point will have to be redefined for each square block that is about to be machined.

A five-axis CNC machine has some additional parameters on top of the ones described for the three axes vertical mill. The three axis mill, which has three degrees of freedom is able to translate in x, y, z-direction. The translation in the x,y,z-direction also applies to the five-axis mill. The additional two axes are a vertical rotation and also a horizontal rotation. The horizontal rotation gives the tool its’ tilting angle while the vertical axis controls the rotation in what direction the tool should be tilted at. This makes for a better surface finish and often also allows greater tolerances directly out of the mill compared to the three axis mill. The reason is the possibility to adjust the angle of the tool to follow the surface. The definition of the five-axis mill needs, therefore, extended definition compared to the three-axis version. It still has the x,y,z variables for the translation but also a,b,c to define the rotation. There are several different methods of how to define the angle of the tool. It could be described by using direct vector programming as a vector between the tool tip and an offset length in the three directions the first a3 then the length of b3 which is perpendicular to a3 then c3 which is vertical. The angle could also be described by rotary axis programming where the programmer defines the angles of the two rotational axis’s A and B for example A=35deg and B=40deg. The third method is by using RPY (roll, pitch, yaw) angles to describe the angle of the tool, this is described in more detail in the introduction of chapter 2. There is also a fourth method that is using Euler angles which are a similar principle to the RPY notation [13].
2.4 Coordinate systems Industrial robots

For an articulated industrial robot, the base coordinate system defines the origin of the robot’s workspace. This coordinate system is defined at the base of the robot along the center line of the first joint.

Next, the tool center point is often called TCP. This coordinate system is defined at the tip of the tool with the z-axis pointing out in the direction of the tools centreline. If no tool is defined the tool zero point will be used which is a predefined coordinate system at the center point on the robot’s hand, also here is the z direction pointing out from the robot’s hand along the robot hands centreline. When defining a custom made tool the measurements are done with respect to the tool0. A robot can handle several different TCP’s which allows tool changes and also the possibility to define multiple TCP’s for the same tool.

The coordinate system called work object is built up by two different coordinate systems. A user coordinates system and an object coordinate system. This allows the user to perform offsets of the object coordinate system with respect to the user coordinate system in order to make the programming easier. By using the offset function, the programmer is able to just define one single start point and program the robot with some logic to automatically place each point at a new location with a specified distance in between. The user frame is used to orient the robot so it can adapt to different workstations [12].

The robot also allows a global coordinate system that is located somewhere within the robot cell. The purpose of having yet another coordinate system is that it will let the robot know where it is located in space which allows the use of external axes. It also allows for multiple cooperating robots in the same cell. The global coordinate system is then acting as a reference point for where the robot is located in space with respect to another robot since both robots refer their position with respect to the global coordinate system.

One of the main challenges regarding the programming and simulation of robots is the handling of the robot’s configuration values. On top of the translation and rotational information that describes where the hand shall be positioned and what rotation it should have. A robot also needs to handle the issue of different possible configurations and since different robots will have different configurations the exportation of a robot program from one robot to another robot isn’t always easy.
3 Literature review

A literature review was done in order to create a foundation of knowledge for the upcoming project regarding the previous research in the subject and the already commercial alternatives.

3.1 Similar existing solutions

In the following sections will different existing solutions be presented shortly in order to give the reader a brief overview of the different available solutions today.

3.1.1 Grasshopper

Grasshopper is a software for Rhinoceros CAD system. Rhinoceros is a CAD software for surface modeling. Grasshopper is an add-in that enables the user to create parametric structures without the need of knowing how to write scripts. Instead, Grasshopper uses a visual scripting tool where the user can drag lines between different blocks with different functions where the coding is embedded in the block so that the user doesn’t need to write the code [14]. The principle of the usage is the same as for many other visual scripting tools such as the, to engineers more commonly known software Labview, and Simulink for Matlab.

3.1.2 HAL

HAL robot programming tool is an add-on framework for Grasshopper. HAL could be used to create robot paths with visualization directly in the CAD software. The software supports ABB IRC5 controllers which make it possible to simulate the robot motions. The premium version also has the possibility to directly create robot code in the HAL software that is executable on the robot controller [14]. The software is also capable of importing G-code and translate the code into paths [14]. The software is mainly focusing towards architectural applications.

3.1.3 ABB machining PowerPac

The ABB machining PowerPac is ABB’s own add-in software for ABB Robot Studio. The software makes it possible for the user to create machining paths by importing CAD geometries directly into Robot Studio. The software does support path generation from G-Code. It also features to act as a CAM software and calculate the toolpath in the off-line programming software. The benefits of using ABB’s own software is that the user is guaranteed that if the path is executable in the simulation it will also be executable on the robot. A feature that other software’s does not have. A drawback with the software is that it is closed source and it is therefore not possible for the user to customize the software for their specific needs. [5]

3.1.4 Boot the Bot

This is a Java-based software that enables the user to import points and additional data and export valid RAPID code from Boot the Bot. Since the software is built in Java it is designed to have cross-platform support. The software is able to automatically calculate the best-suited configuration for the robot. It also allows for FTP (file transfer protocol) connection to the robot, which allows for remote upload of robot programs and remote robot control. The
software also gives the user a full simulation of the workflow in the program. It is designed to enable a more creative freedom for the designer and is primarily targeting users working as architectures and artists [15]. The software is developed under the Creative Commons license. The software is created by Dank and Wegerer at the Graz University of Technology, Austria. [15]

3.1.5 RISE from Robotmaster
A full offline programming software that is also able to import CAD geometries and create robot paths for multiple vendors of robots [16]. The software has an in-built CAD engine that makes it possible to create and edit CAD models within the software. It has the capability to import CAM data from 2-5 axis machines and translate it into robot code. And it is able to handle configuration problems and singularities. [16]

3.2 Research with the use of automated code translation for industrial robots
The research articles summarized is the articles found that uses some type of tool to automate the translation into robot code.

3.2.1 Customized software
In Ribeiro’s article [17] a customized software has been used. Even though the technique could seem a bit primitive today, the principle of using a customized additional software for translation still appears in several articles. In Fernando’s article, the customized software was used in an experiment where an attempt on 3d printing in metal was done by using a robot and a welding torch. The customized software then took an already sliced file from a CAM system, the welding parameters and user inputs about desired parameters as the input. The program then took all the input data and outputted the robot program with the paths and two reports, one for the welding technician and one for the production manager.

A newer study that has been using a customized software is Dank’s Boot the Bot [15] that previously have been briefly presented in this literature review. In this study Dank developed a software that is a postprocessor which is using CAD data from different formats as input and from there simulates the path and generates RAPID code for ABB robots as output. Boot the Bot is written in Java which makes it cross platform compatible.

Yet another article that presents a setup where they use a customized software is Keating and Oxman’s [18]. The article presents a setup where the robot is used both for 3D-printing an object and for milling or grinding. The researchers were first 3D-printing an object and afterward finishing it up with grinding. For the relevance of this literature study, the focus will be on the software used. To accomplish this the researchers used the CAM software HSMWorks for the CAD software SolidWorks to create the tool paths. The tool paths were then translated into KRL code by a custom script written in python. Then the paths were tested in KUKA SimPro to check the paths against singularities.

Chen [19] presents a different way of representing their models compared to the other studies in this review. The research presents a system where a robot is used for cutting applications. In the study, the researchers using the robot to mill for rough machining applications. They also highlight the flexibility of a robot as the strength when used for machining applications but also outlines the drawbacks of the system. The drawbacks outlined are the lack of stiffness in the robot design itself, the non-orthogonal motion in a robot and the complexity of toolpath planning due to these weaknesses. The approach for creating paths in this study differs from other studies reviewed in this literature review. In this case, the
researchers used an algorithm to create a type of voxel representation of the object. The object was represented in square pillars of different heights with a uniform grid size. The size of the grid is controlling the resolution of the milled surface. This grid representation was then also done to both the tool and the tool holder. The method presented in this paper using the milling tool head aligned with the z-axis at all time making it correspond to a 3-axis milling system.

3.2.2 Articles using Grasshopper
In order create a smoother workflow for architects, the researchers Brell-Cokcan et.al wanted to create a shorter and more integrated way to work with robots. Brell-Cokcan et.al [20] is using the previously described software grasshopper in order to accomplish the researchers aim. They created their own software in grasshopper with the following requirements.

- The software was going to integrate like a component in grasshopper.
- The focus was flexible and dynamic robot control and not to compete with CAM programs in terms of the range of functions.
- In order to have near real-time feedback in the program, the calculations should be done in less than a second. This since they wanted to be able to have a preview of the robot path directly in the CAD software.
- The designer should have a transparent view of the programs functionality and also be able to customize them.
- The direct output from the program should be readable KRL code that shall be able to run directly on the robot controller.

By designing a program that just had the basic functionality of a CAM software, an integrated preview of the tool path directly in the CAD software and an output that is able to directly run the robot. The system has a tighter integration and lets the user have a near real-time overview of the project during the design phase. The main need for architects is to be able to play around with the robots with less complexity regarding path planning, configuration set up and other robot-related difficulties. The study shows a step in that direction.

Grasshopper have been used in several other studies within the architectural field [21], [22], [23], [24], [25]. The software is mainly highlighted for its ability to show visualizations of the robot paths in real time during the design process. The users could in this way instantly see if a path is possible for the robot or not. The software also highlights the path segment that is wrong which makes debug process easier. Some of the articles also highlight the software for its user friendliness.

3.3 Discussion
Chen [26] outlines another factor that complicates the use of the robot for machining applications that puts a lot of stress on the robot that is the difficulties in calculating and compensate for the flex of the robot arm, a statement that is being confirmed also by Gian [27]. But with the rapid advancements in both sensor technology and optimization algorithms, this issue will probably be reduced enough to be neglected in a near future.

After reviewing different solutions and the approach they have had in order to translate CAM data or G-code for use with robots it is clear that several solutions to accomplish this already exists on the market. It seems to be three main approaches in order to generate robot code from CAM data, see Figure 1.
1. The first one is to create a standalone post processor that translates the data from a CAM system into robot code that is then imported into the robots off-line programming software [17], [18]. This type of software is the smallest and most simple one. It doesn’t need to take robot configurations or singularities into account when performing the calculations. It simply just needs to translate from one language to another, the configuration problems and so on will have to be solved later on in the robots offline programming software. It is the easiest and most lightweight type of software in this literature study. But it is also the least integrated type of software. To be able to control the result of the output from this type of software, the user will have to import the code into an offline programming software and test if the robot is able to perform the path there.

2. The second approach is to use a program that not only translates the code but also takes care of the robot configuration and solves for singularities [15]. This type of solution could, in theory, have its code uploaded directly to the robot’s controller. The creator of Boot the Bot, however, recommends that the user tests the program in RobotStudio anyway as a safety precaution.

3. The third approach is to create a translator inside the CAD/CAM software like the solutions using grasshopper inside the CAD system Rhinoceros. The main advantage of these solutions is as mentioned before the ability to see real-time visualization of the robot paths.

The different researchers reviewed shows that almost all of them have been using custom software to a more or less large extent. There could be different reasons for this, the first is that the commercial software hasn’t met the researcher’s demands or that the researchers put a lot of value in having a complete understanding of the software that they use. It could also be that this type of software might not have been fully developed at the time of the research or it could be due to a strict budget.

Regarding the commercial software that is available on the market, it seems to be two quite sophisticated solutions. The first one is HAL which is an extension to the Grasshopper framework that is frequently used by architects. The software shall be able to simulate robot code to multiple robot vendors and models. Since the software is chained to the CAD software Rhinoceros it is mostly targeting architects and artists. This because the software itself is a made for surface modeling. Engineers have high demands on accuracy and the ability to create exact measurements in their CAD drawings, which Rhinoceros does not fulfill.

![Figure 1 Showing the different categories of existing solutions today.](image-url)
The second more sophisticated commercial software is Robotmaster. The software is made as a complete solution that can handle the whole process from modeling to uploading the code into the robot controller. It still allowing the user to import CAD files from different CAD systems or CAM files directly from the Cam system that the user may be more familiar with. This makes Robotmaster the most versatile system for translation of G-code into robot code.

Although multiple solutions have been found no software has been found for ABB’s RobotStudio except machining PowerPac. The ABB machining PowerPac does support importation of CAM data into the application. The application itself can also act as a CAM tool. The drawbacks with ABB’s software is that it is a closed source which does not allow the users to customize the software for their specific needs.

The idea is to develop an open add-in to let the users import CAM data directly into ABB’s OEM software RobotbStudio. This since RobotStudio supports extensions able to use the built-in virtual controller to verify the configuration for the paths and get collision detection. But still be able to have it integrated into the robot off-line programming software. This will also cut down the chain of software used by one link at least in terms of experience, hence, the add-in will be experienced more like an extension of the program than a separate software.

By looking at the 5-axis CNC G-code it shows close similarities to the RAPID code. The ABB robot is able to use Euler angles to generate the angular position by using the function OrientZYX. The 5-axis CNC is also able to use Euler angles as mentioned previously in chapter 2.3 to define the tool’s angular position. The movement functions are also very similar to each other when the CNC and robot movement is compared. The G00 and G01 correspond to the MoveL command in RAPID. Although the CNC lacks the corresponding function to the MoveJ that the robot has. The G02 and G03 functions aren’t corresponding to the robots MoveC function either. The G02 and G03 commands are creating circle arcs while the MoveC command creates arcs that not necessarily are circular. Therefore, the easiest way to handle the translation might be to only use linear movements. This is already the case in many robot applications, if we look at the Volvo car plant in Torslanda Sweden, they use MoveL exclusively in their factory in order to be able to have more precise control of the robot’s path. It is as previously also mentioned the same principle as the 3D-printers are using. Looking at the coordinate systems one could see that close similarity exist between the CNC machine and the Robot. The tool reference point in the CNC machine and the TCP in the robot works in the same way. Although there exist some differences in the way of movement and the way the different machines normally has their setup. For example, the robot has its base coordinate system in the center of the base of the robot and the CNC has its machine reference point in one of the corners of the workspace. However, there seems to be good potential in finding a working solution for importing G-code into RobotStudio thru a translating add-in software.

The 3-axis G-code is missing the angular definition of the tool. In order to use this type of code the rotational angles need to be added, this could probably be done easiest by locking the tool in a fixed position by user input and set those values to all the points in the paths.

In order to specify how the software should work in more detail, the application programming interface (API) will in further work have to be examined.
4 Method

The methods used has been chosen specifically for the project or for a part of the project in order to secure the project against its deadline. The method chapter is divided into the different parts of the work.

4.1 Literature study

The literature study gave a good foundation for the upcoming project which helped to define the specification of the project. The first part of the literature study presented in chapter 2 Background was performed in order to give the author a deeper understanding of both the theory behind CNC-machines and robots. The second part presented in chapter 3 Literature review was performed to give a holistic view of the research within the field of robots used for machining and the use of G-code connected to robots. The articles were mainly found using the keyword robot plus any of the following: machining, CNC, G-Code, milling, grasshopper, and architecture. The study also aimed to collect enough information regarding the subject to be able to draw conclusions on how such a software would have to work in order to simplify the path generating process. The study should in case that a need for the software was stated, also conclude for whom the software would benefit. The literature study resulted in the decision of creating an add-in for RobotStudio rather than a standalone program. This since the add-in could make use of the in-built collision detection inside RobotStudio and achieve a more integrated user experience.

4.2 Planning

The project started with planning and structuring the workload to get a better overview of the work that was needed to be done and to plan the time resource for the project more efficiently. A project time plan was made together with a SWOT analyze in order to define the internal and external strengths and weaknesses. The result pointed out that the most critical threats to the project were the lack of knowledge in C# and lack of experience in software development. The project plan was then adjusted to provide protection against these threats. It was redesigned to start aggressively with gaining as much knowledge as possible in the C# language. All of the mentioned actions above in planning was made in order to make the project more efficient and secure the project against the deadline. The software presented in this paper combines the standalone postprocessor with RobotStudio trough the creation of an add-in. In this way, the project will be able to benefit from the simplicity of the post processor and at the same time make use of the inbuilt functionality in RobotStudio.

4.3 Learning C#

The main work of this project is written in C#, a programming language based on Microsoft’s .NET framework.

In the initialisation of this project, the author lacked previous experience of C#. The first step was there for to gain as much knowledge as possible in a short amount of time, a good learning platform was needed. Reading programming books isn’t an effective way of learning unless the reader simultaneously has access to a computer. The primary choice became
therefore instead android application Learn C# by SoloLearn. A platform with short lessons followed by one to ten questions. With its inbuilt code editor and emulator, it lets the user test and run code on the android device.

4.4 Coding
The main project has been created using Microsoft visual studio. In addition, ABB has an API (application programming interface) for RobotStudio. The API allows the user to create Add-In software to RobotStudio that is written in C#. The API also includes specific libraries from RobotStudio that has a lot of prebuilt functions for communicating with RobotStudio. The full documentation of the API’s functionality is available at ABB’s developer website [5]. Since the author did not have any previous software development experience the software development method used in this project will follow the agile software development method. The method is suitable since the author lacked the experience to be able to accomplish a detailed project plan right from the start. The agile method will instead focus on tight part time deliveries in order to secure the project. The part-time deliveries will also create a better overall conception of how the project is proceeding.

4.5 Debugging
The tricky part of this process was to find good methods for debugging the software. Since the add-in runs through RobotStudio when testing a lot of problems can be hard to detect because RobotStudio doesn’t always know what’s wrong. By using breaking-points to identify values of variables in different parts of the code most number and variable related problems has been possible to solve. For more diffuse error messages, a lot of trial and error has led to the solving of most problems. Comments have been frequently used to identify a wide range of errors. They have both be used to print out values as well as labeling different parts of the code to identify where the software was before it crashed.

4.6 Test and verification
In order to test the add-in a robot was used as a simplified 3d-printed. The testing was done with an ABB IRB1600 1.45m 6kg robot. The robot was equipped with an external microcontroller with a fixed federate of the filament. The add-in was then used to import a G-code file for 3d-printing. The imported path was then used with the robot setup to accomplish the 3d-prints.
5 Software development

5.1 Initialization

In order to start the project an overview of the add-in was made, see Figure 2. The overview prints down a brief summary of core functionality, for example, the target group and the vision of the project. This basic overview has then been helpful throughout the project as a short summary of the project whenever needed. It was also helpful in the making of the wireframe for the front end design in the next step, see chapter 5.2. The wireframe was then, in turn, the foundation for creating the flowchart that represents the back-end design of the project.

Figure 2 Shows the application overview form. The form has been used throughout the project to give the author a quick and easy holistic view of aims, target group etc.
5.2 Wireframe of the frontend design

The start of the software development process was to make a wireframe of the frontend design, see Figure 3. The wireframe is a sketch made in the initial state, describing what the final user interface should look like. By starting with the user interface it was easier to identify what kind of functionality the program needed in the back end design.

The first four fields in the design of the wire frame are based on the essential input data needed in order to create a path in RobotStudio. The options auto-config and auto-simulate were chosen based on knowledge gained from the literature study. The options were later considered as redundant and never implemented due to two factors, the first one was that auto configuration and auto simulation runs at the same time when using auto configuration. The auto configuration which also runs a simulation is a rather slow process. Due to the fact that the process itself is time-consuming, it makes sense for the user to visually check that the path is looking correct before running autoconfiguration.

![G-Code Importer](image)

**Figure 3** Describes the initial wireframe design
5.3 Flowchart and Backend design

Based on the wireframe a flowchart was created that structured the core functionality and how the frontend design correlates with the different functions in the program, see Figure 4. The purpose of this was to have a visual document which could act as map later on in the project during the coding. The flowchart starts with opening the UI as the initial step in the add-in. The AddInMain() is called as soon as the user clicks on the icon for the add-in inside RobotStudio. From there it shows how the different functions are called with the wide arrows. The black arrows from the UI to the Read() and CreateTarget() functions represents the input data from the UI for these specific functions.

![Flowchart Image]

Figure 4 Flowchart describing the backend design of the application. The narrow arrows illustrate the information flow from the UI to the functions.
5.4 Code

The main parts of the project have been the coding and it was primarily done in three stages, creating the functions, creating the add-in and integrating the functions into the add-in. First, off the functions were built as a standalone console program. These programs were then tested by manually inputting data and then checking the output data against the expected results. When the functions were fully functional as standalone programs, they were implemented in the main software. By first testing the function itself the author has been able to save valuable time of debugging during the project. This by being able to exclude the function itself as the source of error when debugging the main software during the integration. The coding started with the most vital part in order to succeed with the project, the extraction of data. The file reading function created needed to extract certain parts of a text document while ignoring others. Next, the focus shifted towards integration of the function into ABB’s skeleton code for add-ins in RobotStudio. The reason for this work order was to allow for easier debugging and closer integration with RobotStudio from the start. By dividing the work into these steps the project has delivered tight part-time deliveries, following the agile software development method.

5.5 Add-in skeleton and UI

The project builds upon ABB’s Add-In template. The first thing created in the template was to build the UI based upon the wireframe earlier created. The idea behind it was to keep it as simple as possible to be able to target the non-experienced users. The UI was done using the regular windows forms from the toolbox inside Visual Studio. Once the add-ins UI was finished it was ready for integration with the File reader function and the other functions to build up the add-ins functionality one function at a time.

5.6 File reader function

This is the function called when the user has chosen a valid file to import. The main functionality in the file reader function is to read through the file and only store the necessary parts of the code that is needed in order to create a robot target in RobotStudio later on. The structure of the G-Code could be seen in Figure 5. Based on the structure of the file format the commands G, X, Y, Z, I, J, K was selected as the only letters to extract although the search pattern is prepared for some more letters. As seen in the example code in Figure 5, each of these letters is followed by at least one digit. After the first digit, the code starts to differ. Sometimes a dot separator will follow other times another digit will follow. If a dot appears this could either represent the end or it is followed by another digit sometimes more than one digit. From these observations, the method that seemed most suitable for the matching task was regular expressions. The regular expressions let the user specify a more advanced search pattern than the methods otherwise built into the regular string function.

The search pattern used looks like this: @"[gxyzijkabcf][+-]?(\d+)\(,\d+\)*". The pattern looks for one of the 10 letters inside the first set of brackets. After one of the letters has been matched the pattern will look for a plus or minus sign if there exists one after the previously found letter, that’s what the question mark outside the second set of brackets means. Then one or more digits needs to follow, the plus sign represents one or more digits in the first parenthesis. The last of the previously found set of digits could be followed by a dot separator, that dot separator has to be followed by one or more digits, this is stated in the last parenthesis. If the dot separator happens to not have any following digits it will not
5.7 Handling of both absolute and incremental positioning

G-Code can be defined in two ways. With absolute positioning, the coordinates are calculated with respect to the origin or point 0,0,0. The command used to express absolute positioning is done by G90, shown in the first row in the code above. Note that the G90 command is active until the positioning type is changed to incremental by using the command G91 seen at the fifth row in the code above. Another important detail is shown in the second row in the code above. The expression “G1 X-130” does not say -130,0,0 instead it says -130,-84.07,9.29. The expression X-130 is essentially keeping the previous values and changing only the new values that are defined, in this case; X-130.

The incremental positioning expressed by G91 takes the previous values and then adds the new values to the previous ones. Looking at the fifth row in the code above, the expression will be interpreted as follows. Add 130,0,0 to -150,-100,10 resulting in the new position -20,-100,10. If the same row would have been using absolute positioning instead the result would have been 130,-100,10.

The last important detail with the interpretation of positioning types is that I,J,K which represents the center point when creating circular movements only can be expressed by incremental change regardless of positioning type. I,J,K will be calculated with an incremental change with respect to the starting point.

The understanding of how the interpretation of these positioning types worked was essential to create a script that handled them both. The solution is expressed as a flowchart in Figure 6.

| 1: | N110 G90 G1 X-100 Y-84.07 Z9.29 |
| 2: | N115 G1 X-130 |
| 3: | N120 G2 Y-65.49 J9.29 |
| 4: | N125 G1 X-150 Y-100 Z10 |
| 5: | N130 G91 G1 X130 |
| 6: | N135 G3 Y-46.91 J9.29 |
| 7: | N140 G1 X-130 |

Figure 5 Example of G-code structure.
5.8 Creating targets for G00 and G01

If the G-Code command corresponds to one of the linear movements G00 or G01, this function will be called. It uses the array created in the file reader function together with tool data, speed data and work object data all specified by the user in the user interface. By using these inputs, the function is able to use an inbuilt function in ABB’s application programming interface (API) in order to create targets that will be visualized in RobotStudio.
5.9 Translating the G02, G03 into RAPID commands

The translation of G02 and G03 can be approached in two different ways. Either by converting the G02 and G03 data into MoveC data or by parameterizing the curve into small MoveL segments.

The difference between G02/03 and MoveC is the location of the center point, see Figure 7. The G-code commands have the middle point located in the middle of the “circle” where the start and end point is located along the circle. In order to complete a valid G02/03 command, the distance between the start and center point must be equal to the distance between the end and center point.

The MoveC command on the other side uses all three points along the curve. This allows the creation of elliptical curvatures and not only circles. This could be managed by changing the location of the middle point towards either the start or end point.

In order to extract and convert the data from G-code into RAPID code, a new point needs to be created at the center of the curve. This point was calculated using the starting point which is equal to the previous point, the end point which is defined in the G02/03 command as X, Y, and Z. Also, the G-codes center point is used which is expressed by I, J, and K. Where I represent the translational difference from the starting point in the X direction, J in Y direction and K in Z direction.

The parameterizing of the curve is done by creating targets along the curve which later becomes MoveL commands. The distance between the points depends on the user-specified resolution of the parameterization. The function then loops from the starting point to the end point creating a new point each iteration and moves one step of the size equal to the resolution until the end point is reached.

5.10 Creating a path

The last part of the process is to create a path from the targets already created. In order to do so, the program once more uses the API for RobotStudio. Before the path is generated a set point for marking the start of a undo step is created and the closing point is placed outside of the loop that generates the path. In this way, it is possible to undo the created path by a single undo command from RobotStudio.

Figure 7 Illustrates the location of the points required to create a curve in both G-code and RAPID to the left, MoveC and to the right G02/03
The first part of the path created will automatically be created as a MoveJ instruction in order to exclude the potential of error due to the wrong motion type for the first path element achieved by a counter. The following parts of the path will then be generated as MoveL instructions. The function does also use the chosen speed and zone from the user interface as the settings for the path segments created. Once the path is fully completed the user gets a message in the information window in RobotStudio and the path will pop up on the simulation screen in RobotStudio.

5.11 The integration process

The integration of functions into the add-in itself was done one function at a time during the project. The functions were tested separately, or for the smaller functions built step by step directly in the integrated software. This kept the number of potential errors reduced throughout the rest of the project.

5.12 Test and verification

The test and verification was done to verify that the different functions worked through the whole process chain and that the created path actually was able to run on a real robot. The process tested has been to take a G-code file created for 3d-printing, import it in to RobotStudio using the add-in and test run the path by 3d-printing with the robot.

The external hardware used for printing and cooling was not possible to control from the robot in terms of federate and cooling since the project has focussed on the path generation and code translation, these functions hasn’t been implemented in the software. However does G-code support the ability to control them both.

5.12.1 First test object

The testing was done based on setup presented in chapter 4.6. During the testing the printing parameters were adjusted based on the visual appearance of the prints, no mechanical properties has been tested.

The first object printed was a 20x20x20 mm cube, see part A in Figure 8. During the first iteration the print was done in manual mode and changes in speed was adjusted during the print. After approximately 7 mm of printing the object showed tendencies to have insufficient cooling. An external cooling system with compressed air was therefore added during the print, the cooling however was too intense and led to a temperature drop of the hot end which resulted in a failed print.

The second print was adjusted based on the observations during the first print. The speed was set to 35 % of the robots speed v100. The cooling was set further away from the print and the airflow was decreased. These changes resulted in the first successfully printed test cube, see part B in Figure 8.

During the third print the cooling was mounted on to the robot in order to give a more even airflow around the print. This time the air flow was moved much closer to the printed object with the airflow pointed to the side of the hot end in order to not because temperature drop on the hot end again. Also the speed was adjusted to 30 %. The results show that the airflow once again was too heavy and it was also pointed directly in to the side of the printed object causing deformation to that side, see part C in Figure 8. The adjusted speed also shows that 30 % was too slow given better layer characteristics from 35 % speed.
During all the first three prints the software proved to be stable and allowed the user to change tool, work object, speed, zone, resolution and rotation according to the preferred values.

5.12.2 Second test object
Next object to be tested was a more complex shape see part D in Figure 8, a low polygon shaped vase sliced with a single parameter outer shell and spiral printing pattern. The reason for the low polygon shape was to decrease the stress on the computer since low polygon shapes result in less targets. The spiral print pattern was expected to deliver the best result since the test equipment did not control the feed-rate of the filament. The spiral pattern does not use any retractions of the filament during the print, instead the object is printed with one continues path moving up along the Z-axis like a spiral. This print needed some further adjustments regarding the speed settings since the slicing was done with 0.75 mm layer height compared to the 0.7 mm layer height for the cube. The print was performed with 35 % speed during the solid bottom layers, then the speed was adjusted down to 15 % for the spiral pattern. After closer inspection the print seemed to have some small holes between the layers. A second print was there performed with the same solid layer settings but further decreased speed down to 13 %. The second object required a more complex path that included more targets. This results in slightly longer translation times and simulation times when RobotStudio should simulate the imported robot path. Besides the slightly longer time needed the software still proved to be stable during all the tests.

Figure 8 A – First print first object, B – second print first object, C – Third print first object, D – Final print second object.
6 Results and discussion

The add-in developed in this project allows the user to import G-code files with their basic functionality into RobotStudio. The tests show that the add-in is fully capable to perform 3d-prints with the use of an industrial robot with great results. The software is a good foundation to build upon for a variety of applications. The add-in lets the user import a file of G-code by simply pointing at the file that is chosen to be imported. Once this is done the program automatically reads through the file, creates targets and creates a path from the targets created. The user interface lets the user choose which tool and work object the program shall be using to create the targets and the path. Additional options in the UI is a selection of speed, zone, and selection of the resolution for parameterized curves, see Figure 9. The project aimed mainly to comprehend with .nc files and then extend the functionality to other file formats if there was time. After testing different file formats the author concludes that the file reading function is compatible with several file formats that are using the letters “X, Y, Z” to express translation. The file formats tested that seem to generate valid RAPID paths are the file extensions .nc, .cnc and .gcode.

6.1 Functionality

The software shows full functionality for all linear paths. It handles both absolute and incremental positioning with automatic detection, so multiple switches between the positioning types could be used in the G-code. The circular commands G02 and G03 have both been successfully integrated by parameterizing the curve into a number of straight line segments.

Figure 9 Illustrates the two versions of layout for the UI. The simple layout to the left and the advanced to the right.
Another further research would be to investigate exactly which file formats that the reading function supports. The only conclusion so far is that the file reading function works better than expected since it has been able to successfully import at least three different file formats so far. An interesting discovery was that the function also did support the file format “.gcode” generated from Repetier slicing software for 3d-printing [28]. This aim of the project was to focus on the ISO6983-1 flavor of G-code. The success of multiple file format support is seen as a proof of the strength in the regular expressions flexibility compared to other commonly used solutions like the string method split. The possibility to import 3d-printing G-code allows the software to be implemented into 3d-printing applications as well as milling and grinding applications. The imported G-code is illustrated in Figure 10

Summary:
- Handles several types of G-code.
- Automated generation of both targets and path.
- Automatically converts G02/03 commands into a parameterized curve consisting of straight line (MoveL) segments.
- The resolution is adjustable through the software’s simple UI.
- Works with both absolute positioning and incremental positioning. The recognition and switching between them are automated.
- Automatically loads the existing tools and work object from the open project and showing them in a drop down list in the UI.
- The UI allowing the user to switch between a basic and a more advanced layout. With the simple setting the user only need to specify work object and tool before importing the G-code file, making it suitable for the non-experienced user as well.

Figure 10 Shows imported G-code for 3D-printing where the layers could be seen in these side-views
7 Conclusion

Creating paths based on G-code makes it possible to use CAM software to create the paths and in an easy manner import them into RobotStudio. The most interesting finding might be the power of using regular expressions in order to interpret the G-code and only extract the useful information out of it. The use of this type of software is something that could be useful for companies that would like to use the robot’s large workspace in order to 3d-print, grind, laser engrave, polish or mill parts. With some further work on the software, it could be used for all of these applications. The possibility to use a CAM software which many machinists today already are familiar with to create paths for robots could save many hours of off-line programming for companies. Reducing the programming time would especially benefit SME’s that normally produces smaller batch sizes of their products which result in larger expense per unit compared to companies the produces larger batch sizes. It is therefore of extra interest for the low volume producers to lower the programming costs. Another important note here is the low price of a robot in relation to its workspace compared to a CNC mill with the same workspace. The price itself might attract SME’s to start using robots for machining applications if the off-line programming time could be reduced. With the success of the test and verification of the software, the project has proved that the use of a robot does not need to be more complex than using a regular CNC mill.

7.1 Future work and research

The project has shown a proposed method to easy create large and complex robot paths without any significant previous robot experience. Further development in order to extend the functionality further could simply follow the same method as the existing functionality has been using.

The main research that is left to do for the presented method is to find out if there is a suitable way to handle different configuration settings. The configuration handling is necessary to be able to make use of a large part of the robot’s workspace.

Suggested future work would be to develop this software further in order to adapt it for specific purposes such as 3d-printing, polishing, grinding or milling applications. A great advancement would be to implement the support for five axis milling. Something that also could open up exciting new possibilities for 3d-printing in order to overcome the limitations of the orthotropic mechanical properties in 3d-printed parts.

Also in order to reduce the size of large robot paths a finalized function for using MoveC would be a great improvement for the software. This would be especially useful when working with older robot that has less memory capacity. The implementation of MoveC has been started but not completed, the code for the creation of the steering point located on the curve at half of the curves length is completed and works as a standalone console application that prints out the coordinates of the target, but the implementation into the main software still remains due to complications with the API during the integration with the add-in.

The software is possible to extend in order to create the possibility to control an output for any application that could be appropriate and normally uses a regular text-based file format. The existing code could also easily be extent in order to control the feed rate for 3d-printing, controlling the power of a laser or adjustment in spindle speed for milling and
grinding. This could be especially interesting since the software already supports the commonly used file format for 3d-printing which is proved by the tests in chapter 5.12. The flexibility in the software could open up opportunities for the SME’s to use robots in new areas with already commonly known G-code based file formats.

The software could easily be extent to create RAPID code right out of the application in order to skip the rather slow auto-configuration procedure in RobotStudio although this is not recommended due to safety. The highly modular reading function could also easily be extent or modified to search for additional parameters in order to include more G-code commands or new file formats.

Finally the author encourage researchers to explore new ways of reducing the expertise needed for companies to be able to make use of robots in their businesses.

7.2 Critical discussion

The author had no previous knowledge at all from either C# or the API for RobotStudio at the start of the project. This resulted in a very frustrating learning process from time to time, but starting from scratch really paid off later on in the project since it resulted in a deep understanding of the software that was developed. Since the author lacked experience from both C# and software development projects many things could probably have been done differently. One thing to point out here is that the code has not been optimized for maximal performance. Many things in the software could probably have been written in a more efficient or tidier way from a programming point of view. By trying to improve the efficiency of the code it might be possible to save time on the path generation, especially on large objects with a high number of points. The lack of experience also resulted in spending most of the first three weeks in this project on learning both C# and getting familiar with RobotStudio’s API, these weeks could otherwise have been used to develop the software further.

The parameterizing method of handling curves works fine, but it requires a lot of unnecessary targets to be created. This rapidly results in paths with huge numbers of segments which could be hard for the computer to simulate. A possible solution to this would be if the function for using MoveC was successfully be implemented.

If the project was started today more time would have been spent on learning different strategies for debugging the program right from the start. This was by far the most time consuming part of the work and a part that didn’t got the methodical attention that it deserved. It was also first towards the end of this project that the author started to understand how to debug the program in a more efficient way by using both run to breakpoint and using comments at strategical places to identify bugs.

7.3 Generalization of the result

The result shows that it is possible to create new tools to simplify the robot handling. The author hopes that this will motivate researchers to help to push the development for easier handling of robots further. The success of creating such tools would result in many new applications for robots and by allowing the inexperienced users to gain profit of the benefits from using robots. The easy handling would also directly benefit the SME’s that right now wouldn’t profit from a robot due to the lack of knowledge and the high cost of programming them due to small batch sizes.
As presented in the literature study a lot of alternative software already exists that creates the opportunity to use the robot for machining. However, the software presented for translation of G-code into RAPID in this report could be used as a general solution for translating text based file formats into RAPID code. This work could easily be built upon in order to extend or adapt the software to the specific user need in order to use the robot for painting, milling, cutting, engraving etc. where G-code or some other text based file format would be a good option to have as input from where the robot code will be generated.
8 References


