CODEN:LUTEDX/(TEIE-5480)/1-76(2022)

Industrial Electrical Engineering and Automation

Enhancement and Verification of a Motion Profile Design Tool

David Karlsson
Viktor Lunderquist

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Enhancement and Verification of a Motion Profile Design Tool

Master Thesis

Authors:
David Karlsson, dawe.97@live.se
Viktor Lunderquist, viktorlundq@gmail.com

Examinator: Ulf Jeppsson, ulf.jeppsson@iea.lth.se
Academic Supervisor: Gunnar Lindstedt, gunnar.lindstedt@iea.lth.se
Industrial Supervisor: Tobbe Bengtsson, tobbe.bengtsson@tetrapak.com

Winter and Spring of 2022

Abstract

Robots and other machines utilised in industries are using servo motors that need precise, well defined and controlled motions. Often are these motions to move to a precise point with an exact velocity. Motion profiles provide the physical motion information for such a precise motion as well as the graphical interpretation on how it would move in terms of position, velocity and acceleration. The motion profile is used by the servo controller to determine which power level should be used to drive the motor.

This Master Thesis project aims to improve an existing tool for calculating motion profiles for servo motors used in the manufacturing industry by implementing features to calculate more advanced motion profiles. These new features include motion profiles with non-zero jerk in the beginning and end points, motion profiles focusing on velocity changes and motion profiles where the focus is on position changes. The final step in this project is to show the strength of designing motion profiles using this tool by comparing it to a motion profile created using Beckhoff motion profile designer run on a setup resembling a real application.

All the goals of the project were achieved. The comparison with a motion profile from Beckhoff showed that with our tool a smaller servo motor could drive a bigger load faster and still be within its limits. This is beneficial since the motion profile can be adapt to the motor, and not the other way around. This means one can get a bulk price of the servo motors and the machines will be cheaper, since several different servo motors are not in need.

Preface

During the summer of 2021, a summer worker developed a tool for calculating basic motion profiles for servo applications. This Master Thesis project aims to improve this tool by making it able to calculate more advanced motion profiles, where the acceleration behaviour is defined by a motion profile designer. This project was carried out during the winter and spring of 2022 at Tetra Pak in Lund, during an ongoing pandemic.

We would like to express our special thanks to our supervisor at Tetra Pak, Tobbe Bengtsson, for all his help, openness and assistance provided during the project as well as all the colleagues at Tetra Pak that made us feel welcomed and included. We would also express our thanks to our academic supervisors Gunnar Lindstedt and Ulf Jeppsson for their assistance and guidance throughout the project.

The authors have taken equal part in the project and thus the project can not say have been divided in any way. We have done everything together, which is an advantage since we can ask each other about everything within the project. We have had good discussions between each other about how to implement different parts, which would not be possible if only one of us were familiar with each part.

Terms and Abbreviations

- CAM - Mechanical linkage to produce a motion out of a rotary motion.
- PLC - Programmable Logic Controller
- GUI - Graphical User Interface
- SCCA - Sine Constant Cosine Acceleration
- TwinCAT - The Windows Control and Automation Technology, which is Beckhoff's PLC-environment
- RPM - Revolutions Per Minute

- Beckhoff CAM Design Tool - Tool provided by Beckhoff for designing motion profiles
- CAMTool - The name of the applications (both Python and PLC) that is in focus of this master thesis, which builds the motion profiles
- Jerk - the derivative of acceleration
- Knots - the points that bind two splines together
- Ducks - same as Knots
- Splines - the functions between the different knots
- Cubic splines - splines which are two times differentiable
- Antiderivative - is the function F(x) to its derivative f(x)
- Integral - the numerical value of the area under the graph of a function

- Acceleration Increase - the part of a motion profile where the acceleration gets bigger
- Acceleration Constant - the part of a motion profile where the acceleration is unchanged
- Acceleration Decrease - the part of a motion profile where the acceleration gets smaller
- Deceleration Increase - the part of a motion profile where the deceleration gets bigger
- Deceleration Constant - the part of a motion profile where the deceleration is unchanged
- Deceleration Decrease - the part of a motion profile where the deceleration gets smaller

Contents

1 Introduction

This chapter will give a first insight of what to expect and what should be achieved during this Master Thesis project. This is done through a section of context, which will give some background to the project as well as some explanations to its purpose. The following section is about goals and problem formulation, where these are stated. The last section is the disposition of the report, which will tell more about what will be included and in which order.

1.1 Context

During the summer of 2021, another student developed a Python program including a graphical interface for calculating motion profiles, which describes the motion in terms of position, velocity, acceleration and jerk. The motion profiles consisted of Point to Point motion as well as Dwell / Cruise, but more about that later. The calculations of motion profiles were according to the principle of the Sine Constant Cosine Acceleration method, SCCA, which will be described further down in this section. The Python program was then used as a base to implement the corresponding features in Beckhoff's PLC-environment TwinCAT, which the same student also did, to be able to calculate and run the motion profiles in a real-time environment. All the work the student did has been evaluated and a need of further development is necessary to achieve the full potential of the idea behind the software. The requested functionalities are non-zero jerk at different locations of the motion profile as well as motion profiles for describing a velocity change or a position change. The task of this Master Thesis is to implement the new functions both in Python and in TwinCAT.

SCCA is originally a method for calculating motion profiles for mechanical CAMs, which is the linkage to produce a motion out of a rotary motion. It uses a sinus function to calculate the acceleration increase part, a constant function to derive the constant acceleration part and a cosine function to derive the acceleration decrease part. Consequently, for the deceleration part of the motion, a cosine function is used to calculate the deceleration increase part, a constant function for the constant part of the deceleration motion and to calculate the deceleration decrease a sin function is used. After that the functions for velocity and position can be derived by integration of the acceleration function and velocity function, respectively. Jerk is the derivative of acceleration, i.e. the rate of acceleration change over time and thus can the function for jerk be derived by taking the derivative of the acceleration function. Non-zero jerk refers to a value of the jerk that is non-zero at a given point in time. [1]

There are also other methods for calculating motion profiles that builds on the same principle as SCCA. One of these uses polynomial functions to describe the motion. Continuous polynomial functions for velocity, acceleration and jerk can then be derived from the position by derivation if the degree of the polynomial function is high enough.[1] [2] [3] [4]

1.2 Overall Goals and Problem Formulation

The purpose of this Master Thesis is to introduce and implement new features to the CAMTool software that already exists at Tetra Pak. All of the potential improvements listed below have been requested from those who have used the software, mainly motion design engineers. The new features should not affect the graphical user interface, GUI, in a negative way, but rather the opposite. A simple GUI that is easy to use will make the application more attractive among the engineers. This will have to be kept in mind while developing the application so that it would be more usable among the motion profile designers.

The features that are to be implemented are the following:

1. A motion profile where the jerk is non-zero at the start of the acceleration increase, the end of acceleration decrease, the beginning of deceleration increase and the end of deceleration decrease.
2. A motion profile where the start and end velocity are defined by the user, including the acceleration behaviour.

If times permits the following features should also be implemented:

3. Rewrite the PLC program to optimise the execution with a real time system in mind.
4. A motion profile where the start and end positions are defined by the user, including the acceleration behaviour.
5. A motion profile where both the start and end velocities as well as the start and end positions are defined by the user.

Feature number 3 is only meant to be applied to the TwinCAT program since it is a real time environment. Likewise, feature 5 will only be implemented in Python since it is an advanced way to design motion profiles. However, the rest of the features should be implemented both in the Python program as well as in the TwinCAT. The goals will be further introduced in chapter 3 and the order of implementation will be discussed in chapter 4.

The project will be evaluated in two perspectives, both theoretical and physical. The theoretical evaluation will be used to determine if the new features are the ones required to improve the tool according to the specifications. The reason why it has to be evaluated is because a feature might sound good and necessary, but turns out to be useless when implemented. This means every new feature will be evaluated one by one and if it is needed it will be kept, otherwise not. The physical evaluation will include a servo motor with auxiliary equipment making it a scaled down replica of a conveyor belt application with a gearbox provided by Tetra Pak. More exactly a comparison will take place between the application CAMTool and the way motion profiles are designed using Beckhoff motion design tool, to evaluate the benefits of the application and show the strength in designing motion profiles this way.

The overall idea and goals of CAMTool are threefold. Firstly, it is supposed to aid in the design of motion profiles and reducing the amount of different servo motors by adapting the movement to the servo motor, instead of the other way around. Secondly, to become some type of reference point between the automation engineers and the mechanical designer to aid in the design and specification of large systems where there is both mechanical and electrical components. Lastly, to streamline the process of designing motion profiles and hopefully move all the motion profile design to one tool in the feature.

1.3 Disposition

This report is divided mainly into six different chapters. The introduction, which this section is part of, introduces the topic of this project, the context of the existing programs, the overall goal and problem formulation. The next chapter is the theory, which aims to describe the underlying theory on which this work stands. It includes the basics of how a servo motor works, gearboxes and backlash. It will also give an introduction into motion profiles and present the relevant maths behind it as well as give an introduction into the software library used to construct the different software as well as present the necessary theory about real time systems. The chapter following on the theory is the background which will describe what has been done in the existing software as well as further explanation of the overall goals and problem formulation. The methodology chapter will describe the approach taken. More exactly, a more detailed description of what to do and in which order. The last two chapters, the results as well as discussion and conclusions, aim to describe the result of the project as well as a discussion about these results. At the end of the report a reference list is included, which provides a detailed list of the sources referenced in this project.

1.4 Limitations

The setup for the verification of CAMTool is quite small because of the order delay due to the pandemic situation. This means the application verification will be limited. The only motion profile that will be run and presented in this report is Point to Point. Another limitation for the project is to continue where this tool left of. Since the work aims to enhance an existing tool, the project already has a specific structure.

2 Theory

In this chapter the theory behind the Master Thesis will be presented. Firstly, a section about servo motors and a section on gearboxes and backlash to illustrate the physical side of the project, followed by a section describing motion profiles. Next section will present the mathematics behind the motion profiles, including equations and descriptions. Lastly, a section presenting the software techniques used and a small introduction to the programming languages, Python and TwinCAT, as well as the libraries used in the project.

2.1 Servo Motor

A servo motor is able to rotate in both directions where one is defined as positive and the other as negative. This results in either a positive or a negative velocity. The torque of the servo motor is defined as if it tries to work against the rotation its negative otherwise its positive. The power of the motor is the product of torque and velocity as:

$$P = T\omega \tag{1}$$

where T is the torque (Nm) and ω is the angular velocity (rad/s). The servo motor that will be used in this project is from Beckhoff, more specific a AM8041-0DG0 rotary servo motor. It has a standstill torque of 2.37 Nm, a standstill current of 1.65 A, a rated velocity of 3000 rpm, a rated torque of 2.29 Nm as well as a rated power of 0.72 kW.[5] Standstill torque is the torque the motor produces at a velocity of less than 100 rpm. Corresponding standstill current is the current the motor draws at a velocity of less than 100 rpm. [6] Rated torque and rated velocity are the torque and velocity values at which the motor produces the rated power which is a value used to compare different motors [7].

2.1.1 Characteristics

Any general, electrical motor has two operating modes, motoring and generating. Depending on if the motor is rotating forward or backward, as well as if the torque is positive or negative, the motor will end up in four different quadrants, shown in figure 1. In the first quadrant, where the velocity and the torque of the motor are both positive, the motor is in a mode called forward motoring, where it converts electrical energy to mechanical energy. The same operating mode can be found in quadrant three, but this case is called reverse motoring since the motor is rotating backwards. The common result in these two cases is the positive result of the power in equation 1 above. The two other quadrants are called reverse generating and forward generating, which happens when the torque is positive while the velocity is negative and when the torque is negative while the velocity is positive, respectively. The common result in these two cases is the negative result in equation 1 of the power, since one of the factors is positive while the other one is negative. The motor operates as a generator thus it converts mechanical energy to electrical energy. This explains the letters M and G in figure 1, which stands for motoring and generation, respectively. [8]

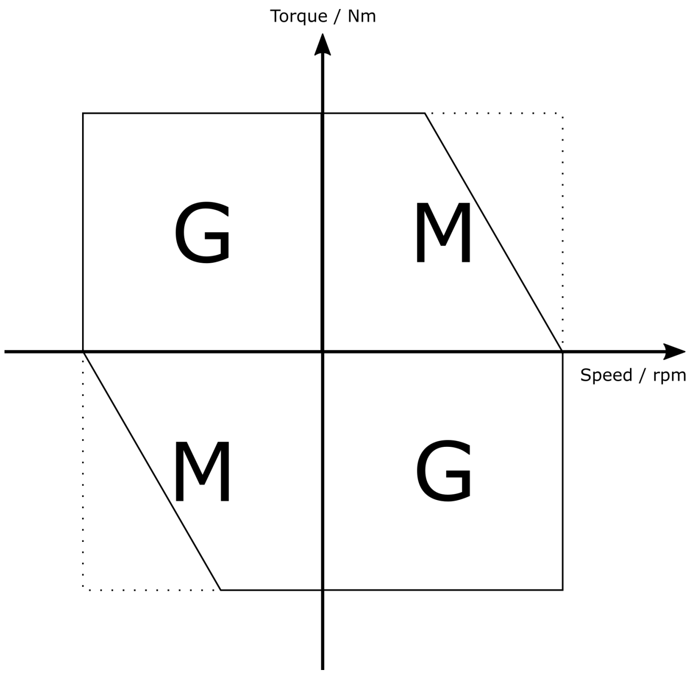


Figure 1: The servo motor characteristic in the four quadrants. M = motoring, G = generation

The velocity torque diagram in figure 1 shows the torque and velocity limits for the motor. To avoid damaging the motor, it should not be used outside of these limits by for example running the motor too fast or with a load that is too heavy. When no load is added the servo motor can reach its maximum velocity. The same results holds for the torque, when it has no velocity the servo motor can reach its maximum torque. See figure 2 of how velocity and torque are linked, with the limitations in red. More advanced requirements comes into play when the servo motor operates for a long time. In industry, the servo motors can be running for months, where an overheated servo motor can stop the whole process or even cause damage on other components and machines. The generated heat from the motor is in direct correlation with the velocity and torque the motor is running at.

A motor's torque-velocity diagram has two zones where one is called continuous duty zone and the other one is called intermittent duty zone. They refer to the lower and the upper area of $T_{rms-limit}$, respectively, in figure 2. The RMS of the torque, here denoted as T_{rms}, is a way of keeping track of the root mean square of the torque required during a full duty cycle of the servo motor. If the servo motor only operates in the continuous duty zone, which means it never crosses the $T_{rms-limit}$, the T_{rms} will be in the same zone and will never be working outside the limitations. But the more interesting thing happens when the servo motor enters the intermittent duty zone, as shown by the blue closed path in figure 2. Depending on how long time the servo motor keeps each level of torque in the duty cycle the T_{rms} is varying. If the motor keeps too high torque levels for too long the T_{rms} will cross the $T_{rms-limit}$ line and the servo motor might overheat. Using a motor with the correct dimension for the application will keep the T_{rms} point below the $T_{rms-limit}$ in the velocity-torque diagram, thus ensuring that the motor will stay operational.

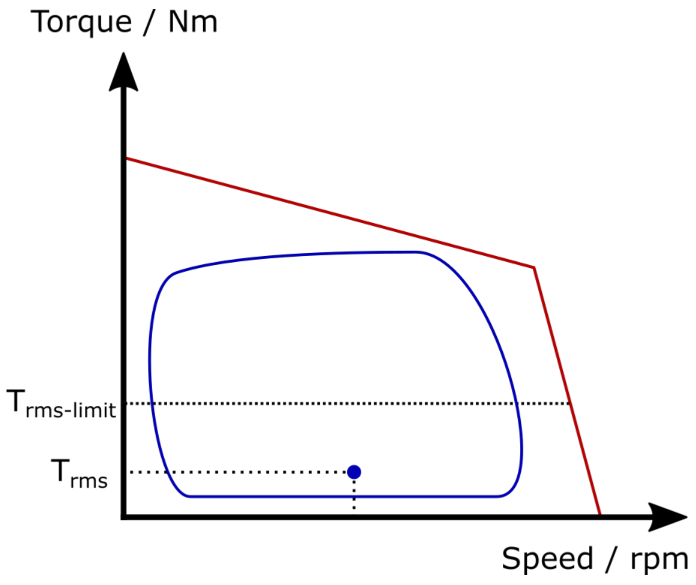


Figure 2: Motor Velocity-Torque diagram including RMS of the torque. The red boundary constitutes the velocity - torque limit of the motor, and the blue line is how a servo motor might operate with its RMS point in blue

2.2 Gearbox and Backlash

The purposes of the gearbox are several. Firstly to match the servo motor output with the application, i.e for example low velocity, with a high torque or only high torque with a smaller servo motor. Secondly and the most common one is to match the inertia from the application with the one at the motor. This is to stabilise the application and minimising the effect the load has on the behaviour of the motor. Using a gearbox allows the designer to more finely control the end velocity since the gear ratio controls the output velocity. The gear ratio can either increase the velocity on the secondary side compared to the primary or decrease it. [9] But introducing additional gears does not always mean more accuracy when the servo motor is changing direction all the time. This is due to the play between the teeth in the gears, which is called backlash. More exactly backlash is the play between two mated gear teeth at the pitch circle, as illustrated in figure 3. It is necessary to prevent heat generation, wear, noise, failure or overload between two gears, and is achieved by cutting the teeth to one half of what the backlash should be stated in the requirements for the application. Backlash is wanted when the operation is only forward drives or with a load in one direction, but when the operation drives changes direction frequently and the precision as well as timing are critical to the application, backlash is not tolerated. [10]

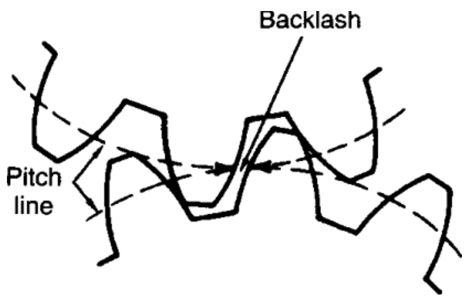


Figure 3: This figure shows two gears and where backlash exists. The picture is taken from [10]

2.3 Motion Profiles

If a servo motor gets connected to a constant power supply it starts to accelerate to a specific velocity and keeps that velocity until the power supply changes level. If the power supply enters a higher constant level the servo motor starts to accelerate again and reaches a new higher velocity. The opposite holds if the power supply enters a lower constant level, the servo motor will decelerate to a new lower constant velocity. One can not anticipate the behaviour of the acceleration in that way. Instead motion profiles are used to define the behaviour of the motion in terms of position, velocity, acceleration and jerk. This information can then be used to control the voltage to the servo motor to achieve the correct velocity and position. To achieve good performance when utilising the servo motor at high velocity, continuous derivatives are necessary when calculating the motion profiles. Therefore, the position profile is defined with a polynomial of the fifth degree. This means that the velocity is defined by a fourth degree polynomial, acceleration a third degree polynomial and the jerk a second degree polynomial and thus ensuring continuous function for all derivatives. [1]

Figure 4 shows how the position and velocity could look like in a motion profile and figure 5 shows the jerk and acceleration profiles for the same motion. Those figures show an example of a motion profile where the behaviour of position, velocity, acceleration and jerk are defined in the interval of 0 to 100 of the Master Position, which also is the x-axle. This section will go deeper into details of the figures and all parameters further down, but for now this is just an example.

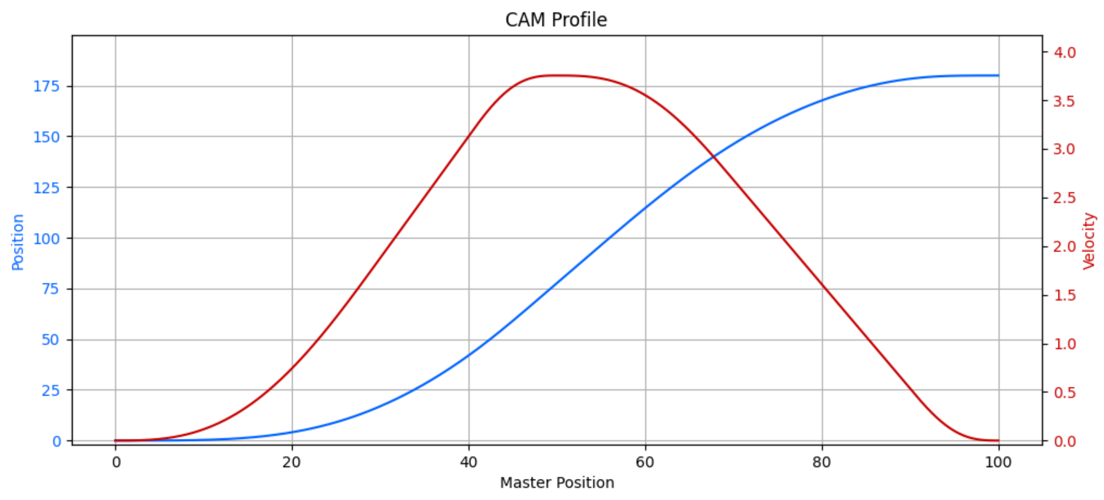


Figure 4: This figure shows an example of the behaviour for the motion profiles of position (blue line) and velocity (red line), created by the application CAMTool

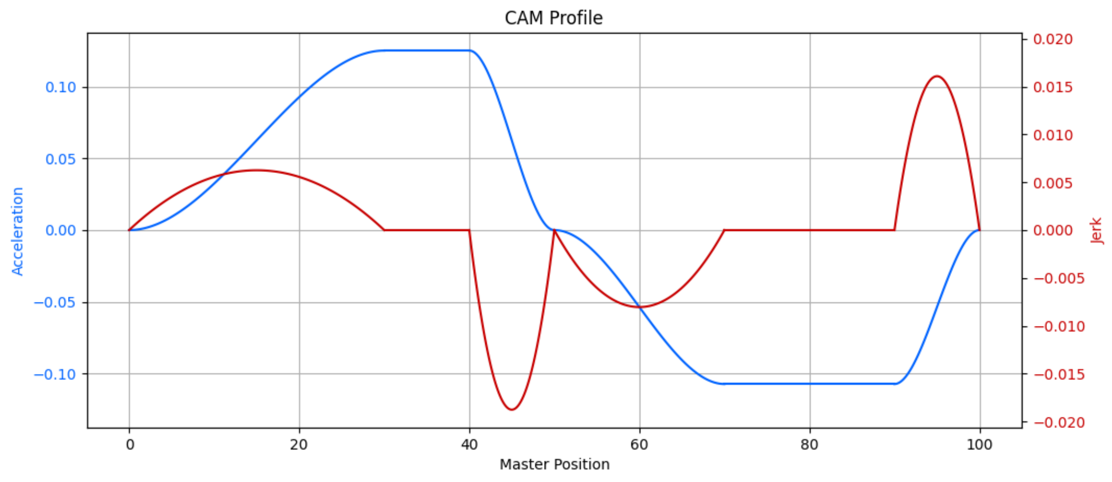


Figure 5: This figure shows the same example as figure 4 but the behaviour for a motion profile of acceleration (blue line) and jerk (red line), created by the application CAMTool

The conditions of the motion profile depend on the requirements of the application or decisions of the designer. One such condition could for example be when the servo motor is standing still and goes to another position ending up standing still, which is a case called Point to Point motion. This correspond to path A in figure 6 (a). Another case is when the servo motor has an initial velocity (can be zero) and wants to keep that velocity for a time, which is a case called Dwell / Cruise. This corresponds to path B in figure 6 (a) and path A in figure 6 (b). A further condition pair is when the engineer wants to increase the velocity, either from stand still or from another initial velocity. This case is called Velocity Increase, and corresponds to path C in figure 6 (a) and path B in figure 6 (b). The opposite, when the servo motor should decrease the velocity, is called Velocity Decrease, which corresponds to path D in figure 6 (a) and path C in figure 6 (b). In total there exists seven cases, see figure 6. Beyond these seven cases there are two additional cases called Position Increase and Position Decrease. These cases makes it possible to start with an initial velocity in a given start position and end in a given position but with an undefined end velocity. Position Increase is when the start position is smaller than the end position and consequently Position Decrease is when the start position is bigger than the end position. Figure 6 is not relevant for these additional cases as the start and end velocities are not the focus to the software when doing these calculations.

Figure 4 can now be determined as a Point to Point motion, since the motion profile starts with no velocity and a predetermined start position, and goes to another position ending up standing still.

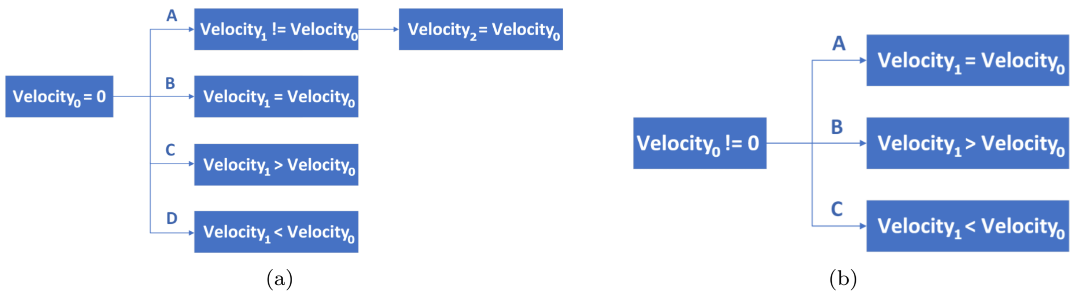
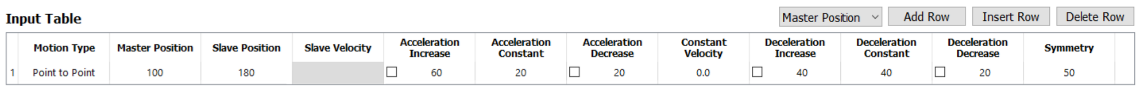

A
B
C
D
Velocity$_0$ = 0
Velocity$_1$!= Velocity$_0$
Velocity$_2$ = Velocity$_0$
Velocity$_1$ = Velocity$_0$
Velocity$_1$ > Velocity$_0$
Velocity$_1$ < Velocity$_0$
(a)
Velocity$_0$!= 0
Velocity$_1$ = Velocity$_0$
Velocity$_1$ > Velocity$_0$
Velocity$_1$ < Velocity$_0$
(b)

Figure 6: This figure shows different conditions for the program, where the left hand side in (a) and (b) are initial conditions and the right hand side are the end conditions

To create a motion profile the engineer enters a set of parameters. These parameters are what kind of motion profile to create, how the acceleration behaviour should be, how the deceleration behaviour should be, for how long time the motion profile should last etc. The input table from the application can be seen in figure 7. A deeper explanation about the different parameters and what they are doing are explained in the upcoming sections. From this table the application calculates a motion profile for position, velocity, acceleration and jerk based on the given parameters, with some built in constrains that the designer takes for granted. The constraints could be that the motion should not overshoot, or behave in other strange ways.


Input Table

	Motion Type	Master Position	Slave Position	Slave Velocity	Acceleration Increase	Acceleration Constant	Acceleration Decrease	Constant Velocity	Deceleration Increase	Deceleration Constant	Deceleration Decrease	Symmetry
1	Point to Point	100	180		☐ 60	20	☐ 20	0.0	☐ 40	40	☐ 20	50

Figure 7: This figure shows the graphical designer interface for writing the different parameters to the program

2.3.1 Acceleration

The motion profile is defined by a set of acceleration points. The acceleration points are used to define the acceleration behaviour in terms of acceleration increase, acceleration constant and acceleration decrease as well as deceleration increase, deceleration constant and deceleration decrease. See figure 8 for how and where all three acceleration parameters as well as all three deceleration parameters are defined in the motion profile. The acceleration and the deceleration parameters are given as a percentage of the total acceleration or deceleration time. Figure 8 also gives a parameter for the length of the motion called Master Position and can be a reference to an internal machine position, machine degrees. The Master Position can be recalculated to time if that is preferred instead. In this way, when the motion profile has one section of acceleration and one section for deceleration, the velocity motion gets divided into two parts around the symmetry. The first one is Velocity Increase and the second one is Velocity Decrease as can be seen in figure 9, which gives the engineer freedom in terms of acceleration to design the overall motion behaviour.

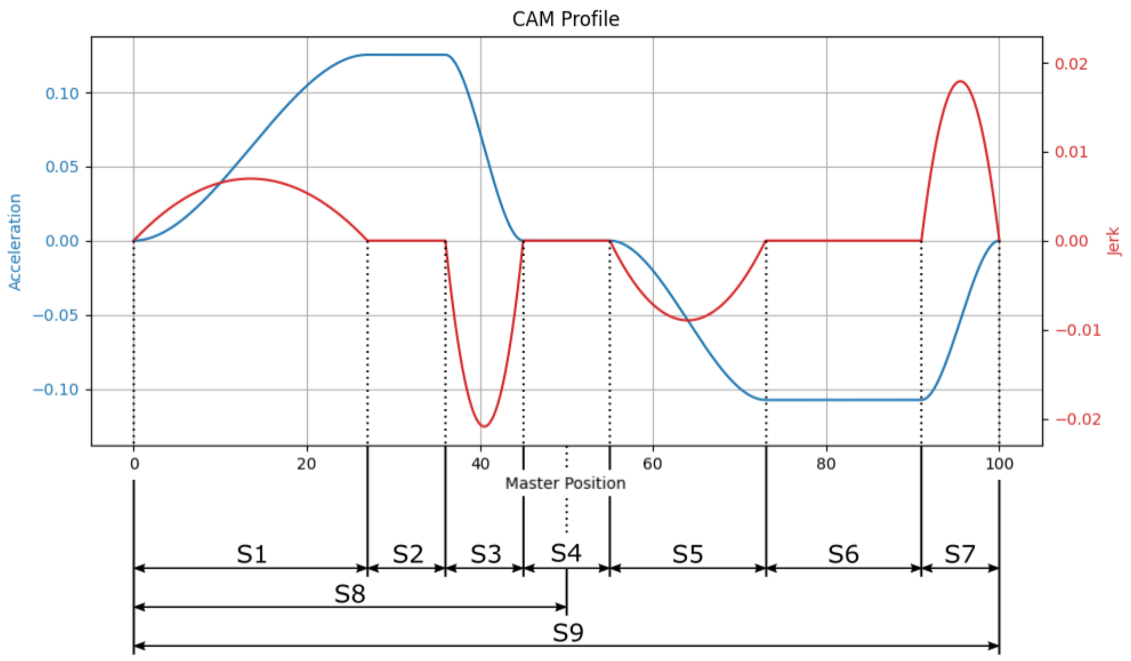


Figure 8: This figure explains the use of the different parameters. S1 is Acceleration Increase, S2 is Acceleration Constant, S3 is Acceleration Decrease, S4 is Constant Velocity, S5 is Deceleration Increase, S6 is Deceleration Constant, S7 is Deceleration Decrease, S8 is Symmetry, S9 is Master Position

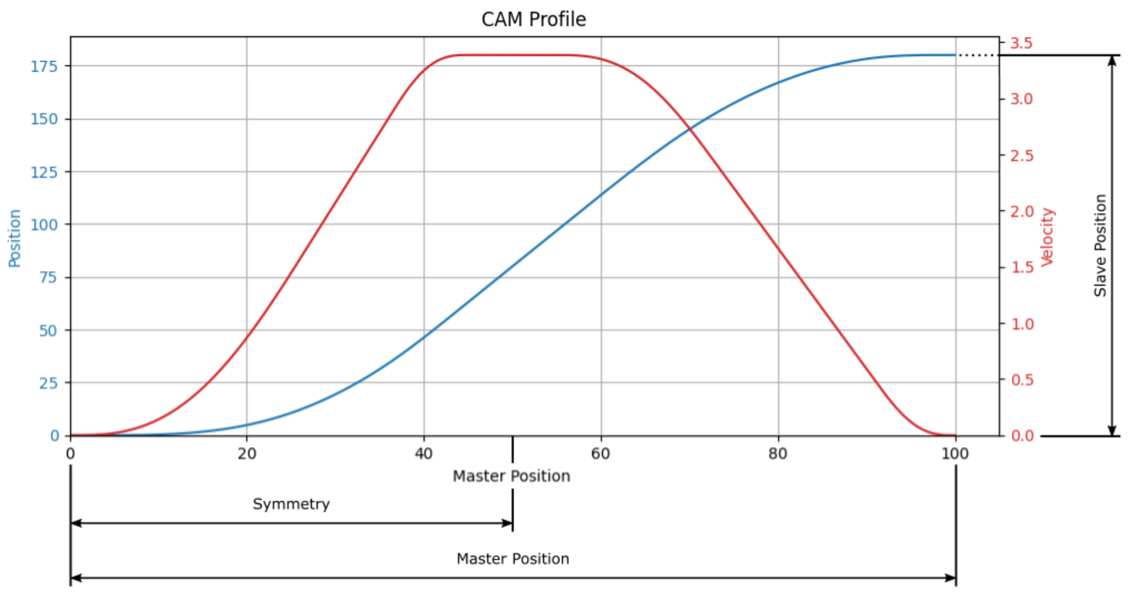


Figure 9: This figure explains the use of the parameters symmetry, Master Position, slave position

The different parameters for acceleration increase and decrease as well as deceleration increase and decrease can be defined in the range of 1 - 99, depending on how big part of the motion should be either increase or decrease of either acceleration or deceleration. For example 99 acceleration increase means that 99% of the acceleration part of the profile should be dedicated to increasing the acceleration. The constant acceleration and constant deceleration parts can be defined in the range of 0-98. The total of the acceleration part must be 100. This also applies to the deceleration part.

An additional parameter used to define the acceleration behaviour is the symmetry of the profile, defined in the range of 1 - 99. A symmetry of 50 means that the motion is divided in equal parts acceleration and deceleration. This means that in motions where the forces acting upon the object in motion is uneven, this can be compensated for. Examples of such motions are when moving vertically, i.e. when moving downwards the gravitational force will help the motion and when moving upwards the gravitational force will counteract the motion.

2.3.2 SCCA Method

SCCA stands for Sine-Constant-Cosine-Acceleration and is a set of functions used for defining different types of acceleration behaviour. There are a few standard functions defined, e.g. modified sine, modified trapezoid and cycloidal profiles, simple harmonic acceleration and constant acceleration. All theses different profiles can be obtained by changing the parameters presented in equations 2 and 3, which changes the acceleration behaviour.[1]

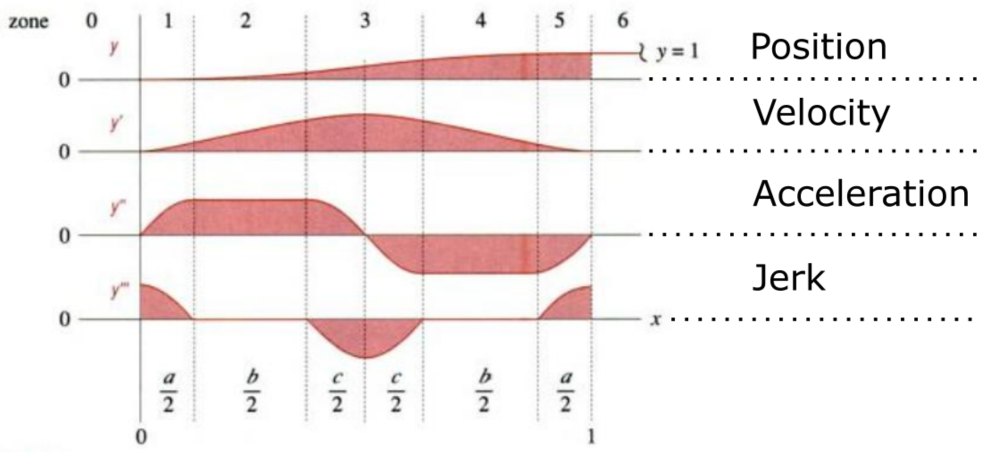


Figure 10: This figure shows a modified sine motion profile constructed using SCCA. Part of the figure is taken from Norton, 2009 [1]

Given a motion profile as shown in figure 10, the acceleration increase, acceleration constant and acceleration decrease can be defined by the following functions, which corresponds to the first three sectors in figure 10.

$$\begin{aligned} y_{accelerationincrease} &= C_1 sin(\frac{\pi}{a}x) \\ y_{accelerationconstant} &= C_1 \\ y_{accelerationdecrease} &= C_1 cos(\frac{\pi}{c}(x - \frac{1-c}{2})) \end{aligned} \tag{2}$$

where x is the independent variable, a and c are the periods of the motion. The use of π is to normalise the period of the motion. C_1 is a constant to get the correct amplitude. And similarly, the equations for deceleration increase, deceleration constant and deceleration decrease are the following. It corresponds to the final three sectors in figure 10.

$$\begin{aligned} y_{decelerationincrease} &= C_1 cos(\frac{\pi}{c}(x - \frac{1-c}{2})) \\ y_{decelerationconstant} &= -C_1 \\ y_{decelerationdecrease} &= C_1 sin(\frac{\pi}{c}(x - 1)) \end{aligned} \tag{3}$$

where, again, x is the independent variable, a and c are the periods of the motion. The use of π is to normalise the period of the motion. C_1 is a constant to get the correct amplitude.

The expression for the velocity in the different sectors is as follows. First is the velocity for the increasing part of the motion and then the decreasing part of the motion. For the first three velocity functions, shown in equation 4, the first is derived by taking the antiderivative of the function for acceleration increase, the second velocity function is the antiderivative of the constant acceleration function and finally the third function is the antiderivative of the acceleration decrease function. All these acceleration functions can be seen in equation 2.

$$\begin{aligned}
y_{velocity} &= C_2[\frac{a}{\pi} - \frac{a}{\pi}cos(\frac{\pi}{a}x)] \\
y_{velocity} &= C_2[x + a(\frac{1}{\pi} - \frac{1}{2})] \\
y_{velocity} &= C_2(\frac{a}{\pi} + \frac{b}{2} + \frac{\pi}{c}sin[\frac{\pi}{c}(x - \frac{1-c}{2})])
\end{aligned} \tag{4}$$

where, x is the independent variable. a, b and c are the periods of the motion. The use of π is to normalise the period of the motion. C_2 is a constant to adjust to the correct amplitude. For the following three velocity functions, shown in equation 5, the first function is derived by taking the antiderivative of the function for deceleration increase, the second velocity function is the antiderivative of the constant deceleration function and finally the third function is the antiderivative of the deceleration decrease function. All these deceleration functions can be seen in equation 3.

$$\begin{aligned}
y_{velocity} &= C_2(\frac{a}{\pi} - \frac{a}{p}sin[\frac{\pi}{c}(x - \frac{1-c}{2})]) \\
y_{velocity} &= C_2(-x + \frac{a}{\pi} + 1 - \frac{a}{2}) \\
y_{velocity} &= C_2[\frac{a}{\pi} - \frac{a}{\pi}cos(\frac{\pi}{a}(x-1))]
\end{aligned} \tag{5}$$

where, x is the independent variable. a, b and c are the periods of the motion. The use of π is to normalise the period of the motion. C_2 is a constant to adjust to the correct amplitude. And then the following three functions, shown in equation 6, is for the position in the different sectors in the acceleration part of the motion. The position equations are also split into two parts, one for the acceleration part of the motion, which is shown first, and one for the deceleration part of the motion. The first function is for the acceleration increase part of the motion, the second function is for the constant acceleration part of the motion and the third and final function is for the acceleration decrease part of the motion.

$$\begin{aligned}
y_{position} &= C_3[\frac{a}{\pi}x - (\frac{a}{\pi})^2 sin(\frac{\pi}{a}x)] \\
y_{position} &= C_3[\frac{x^2}{2} + a(\frac{1}{\pi} - \frac{1}{2})x + a^2(\frac{1}{8} - \frac{1}{\pi^2})] \\
y_{position} &= C_3([\frac{a}{\pi} + \frac{b}{2}]x + (\frac{c}{\pi})^2 + a^2(\frac{1}{8} - \frac{1}{\pi^2}) - (\frac{c}{\pi})^2 cos[\frac{\pi}{c}(x - \frac{1-c}{2})])
\end{aligned} \tag{6}$$

where, x is the independent variable. a, c and b are the periods of the motion. The use of π is to normalise the period of the motion. C_3 is a constant to adjust to the correct amplitude. And then the expression shown below is for the different sectors of position for the deceleration part of the motion. One for the acceleration part of the motion, which is shown first, and one for the deceleration part of the motion.

$$\begin{aligned}
y_{position} &= C_3([\frac{a}{\pi} + \frac{b}{2}]x + (\frac{c}{\pi})^2 + a^2(\frac{1}{8} - \frac{1}{\pi^2}) - (\frac{c}{\pi})^2 cos[\frac{\pi}{c}(x - \frac{1-c}{2})]) \\
y_{position} &= C_3[-\frac{x^2}{2} + (\frac{a}{\pi} + 1 - \frac{a}{2})x + (2c^2 - a^2)(\frac{1}{\pi^2} - \frac{1}{8}) - \frac{1}{4}] \\
y_{position} &= C_3[\frac{a}{pi}x + \frac{2(c^2 - a^2)}{\pi^2} + \frac{(1-a)^2 - c^2}{4} - (\frac{a}{pi})^2 sin(\frac{\pi}{a}x)]
\end{aligned} \tag{7}$$

where, x is the independent variable. a, c and b are the periods of the motion. The use of π is to normalise the period of the motion. C_3 is a constant to adjust to the correct amplitude. The expressions given in equations 2, 3, 4, 5, 6 and 7 are retrieved from Norton [1].

2.3.3 Splines

Splines are continuous mathematical functions that is a piece-wise polynomial with certain continuous derivatives in the considered interval [11]. The polynomial functions will start at one point and end in the other point. Spline fitting is the corresponding mathematical technique for fitting functions between discrete points to match some data [12]. The points that bind two splines together are knots [1]. This means the knots corresponds to the x-marks numbered one to eight in figures 11 and 12, while the splines corresponds to the line in between. The concept for splines evolved from using weights and bamboo sticks to find a curve between several points with the least tension. The weights corresponding to the points and would be called knots. The bamboo stick, now matching the way between the knots with the least tension, would be the splines.[1]

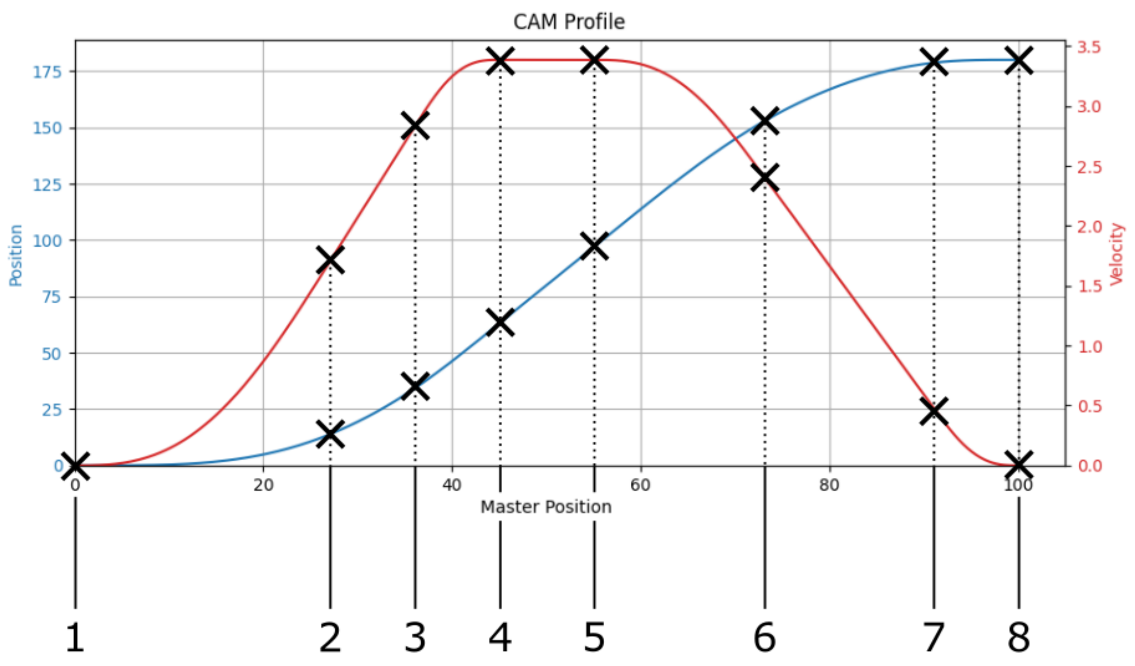


Figure 11: This figure shows the knots of a motion profile of position (blue) and velocity (red) numbered one to eight

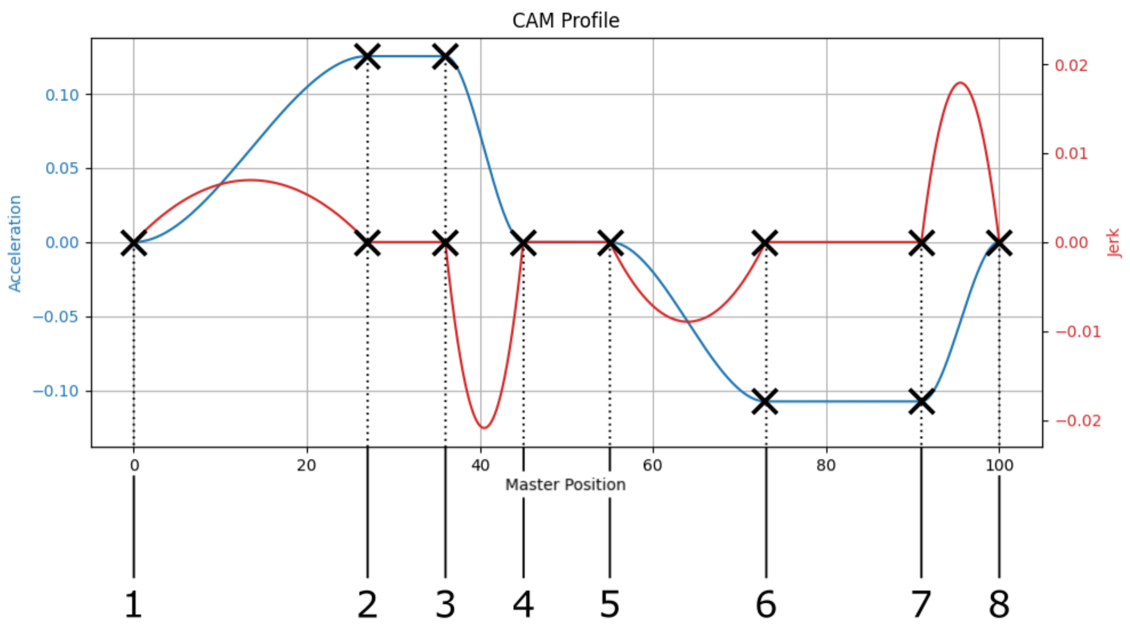


Figure 12: This figure shows the knots of a motion profile of acceleration (blue) and jerk (red) numbered one to eight

The important thing when using splines is to make the function continuously differentiable across the entire domain. This can be achieved by matching the boundary conditions for the different splines at the knots thus matching the different end and beginning derivatives at the same knot with each other [12]. Some common boundary conditions are for example to make the first derivative zero, which is called to make the spline "clamped" [1]. Another common boundary condition is to make the second derivative zero. This called to make the spline "natural" [12].

Some different polynomial functions can be used and some are better fitted for different purposes [12]. For use in CAM profiles a polynomial of the fifth degree for the position function is preferred since velocity, acceleration and jerk will be continuous functions and thus the motion will be smooth [1]. A spline of the fifth degree is called a quintic spline [12]. A spline of the third degree is called a cubic spline [1].

In this project cubic splines will be used to define the acceleration behaviour. Cubic splines are splines which are two times differentiable [13]. Since velocity is the antiderivative of acceleration, it will be defined by a quartic spline. The position is the antiderivative of velocity, which means it will be defined by a quintic spline. Jerk will be a second degree polynomial since it is the derivation of acceleration. Equation 8 shows the polynomial functions for acceleration, velocity, position and jerk.

$$\begin{aligned}
&f_{acceleration}(x) = ax^3 + bx^2 + cx + d \\
&f_{velocity}(x) = \frac{a}{4}x^4 + \frac{b}{3}x^3 + \frac{c}{2}x^2 + dx + e \\
&f_{position}(x) = \frac{a}{20}x^5 + \frac{b}{12}x^4 + \frac{c}{6}cx^3 + \frac{d}{2}x^2 + ex + f \\
&f_{jerk}(x) = 3ax^2 + 2bx + c
\end{aligned} \tag{8}$$

where x is the independent variable and a, b, c, d, e and f are the individual coefficients to the polynomials.

2.4 Mechanics

This section aims to give a brief overview on the laws of motions. It will also give an attempt to describe how these are relevant to our Master Thesis. The reason for the order of acceleration, jerk, velocity and position is to first introduce the acceleration since that is the main focus of this way of designing motion profiles. Then introduce the different topics as they are mentioned in the requirements. The first requirement was to introduce a non-zero jerk at the beginning of acceleration increase, at the end of acceleration decrease, at the beginning of deceleration increase and at the end of deceleration of decrease. The main reason for this was to have a faster acceleration and deceleration behaviour. The second requirement was to introduce a motion profile designed with a specific end velocity in mind. The third requirement was to introduce a motion profile with a specific end position. The reasons for the two last requirements were to make the tool more flexible with more motion types.

2.4.1 Acceleration

Acceleration is defined by a third degree polynomial interpolated between the different knots [1]. The splines, which are the functions between the different knots, are calculated by using the SciPy library, which will be described more in depth in the section about the software. The acceleration functions are by definition two times differentiable [13]. This property gives a smooth profile to work with and as will be discussed later, a second degree polynomial function for the jerk of the motion, which can be manipulated for different cases.

$$f_{acceleration}(x) = ax^3 + bx^2 + cx + d \tag{9}$$

This results in four unknowns, a, b, c and d, as can be seen in equation 9. Since we have four unknown variables we need four equations to solve for the values of a, b, c and d. This can be derived from the boundary conditions of the acceleration motion and thus gives us two additional conditions and the remaining two are derived from either the jerk end points or the end points of the derivative of the jerk, as will be presented in the next subsection.

2.4.2 Jerk

Jerk is the derivative of acceleration and defines how the acceleration changes over time [1] [14]. In this project, motion profiles where the jerk is non-zero at the beginning of acceleration increase, at the end of deceleration decrease, at the beginning of deceleration increase and at the end of deceleration decrease have been investigated. The jerk is defined by a polynomial of the second order and is derived by differentiating the acceleration profile once, as can be seen in equation 10.

$$f'_{acceleration}(x) = f_{jerk}(x) = 3ax^2 + 2bx + c \tag{10}$$

where x is the independent variable and a, b and c are polynomial coefficients. Notice that this is the same coefficients as in the acceleration equation mentioned in equation 9. If the motion profile should not have non-zero jerk at the beginning or at the end of a spline, equations 11 or 12 hold.

$$f_{jerk}(x_0) = 0 \tag{11}$$

$$f_{jerk}(x_n) = 0 \tag{12}$$

If the motion profile should have non-zero jerk at the beginning or at the end of a spline, equations 13 or 14 hold instead.

$$f'_{jerk}(x_0) = 0 \tag{13}$$

$$f'_{jerk}(x_n) = 0 \tag{14}$$

Using the equations in 11, 12, 13 and 14, shown above, in different combinations results in a jerk profile similar to those shown in figures 23, 24, 25, 26 and where the jerk value is either non-zero or zero in the beginning of acceleration increase, the end of acceleration decrease, the beginning of deceleration increase and at the end of deceleration decrease.

2.4.3 Velocity

As mentioned earlier, velocity is defined by a fourth degree polynomial and is derived by getting the antiderivative of the acceleration profile. The constant term is derived either by the start condition of the motion or the end position of the previous spline to achieve a continuous velocity profile over the entire motion.

$$\begin{aligned} &f_{velocity}(x) = \int f_{acceleration}(x)dx = \frac{a}{4}x^4 + \frac{b}{3}x^3 + \frac{c}{2}x^2 + dx + e \\ &e = f_{velocity}(x_0) \end{aligned} \tag{15}$$

2.4.4 Position

Position is the two time antiderivative of the acceleration profile. It is defined as mentioned earlier as a fifth degree polynomial. The constant term of the integral is either defined by the starting position or the end position of the previous spline to maintain a continues position profile. The constant term is derived by the initial condition of the motion or the end position of the previous spline to achieve a continuous velocity profile over the entire motion.

$$\begin{aligned} &f_{position}(x) = \iint f_{acceleration}(x)dx^2 = \frac{a}{20}x^5 + \frac{b}{12}x^4 + \frac{c}{6}x^3 + \frac{d}{2}x^2 + ex + f \\ &f = f_{position}(x_0) \end{aligned} \tag{16}$$

2.5 Python

Python is designed to be used for building software in a wide variety of application domains, which makes it a general-purpose programming language. It is also a high level, object oriented interpreted programming language. [15]

It is developed under open source license, which means that everyone can contribute to it and develop it. [16]

2.5.1 Graphical User Interface

Qt Designer is a tool to design graphical user interface (GUI), which then can be used in different applications [17]. It is based on "drag and drop" and a "what you see is what you get" type of approach [18]. The implementation of the GUI is built around the QObject to provide runtime efficiency and the flexibility for creating a user interface. For adding elements to the user interface Qt designer uses the Qt widget object. [19] The widgets works well with the underlying platform and provide the native look and feel of any other program running on Windows, Linux and macOS. However, they are quite static in their implementation and thus only suitable for desktop centred user interfaces and for applications only running on desktops. [18]

Qt Designer is built around some main concepts to ease the development process and integration of GUI. The main concepts are: the application main window, desktop integration, dialog windows, layout management, model/view programming, rich text processing, drag and drop, internationalisation. [18] The main window of the interface is the feature which holds the entire GUI together. It also provides implementations for menus and toolbars as well as a status bar and more. [20] Desktop integration adds features to integrate the application with the system it is running on, such as the ability to open external resources, adding system tray icons and support for running the application on more than one screen [21].

Layout management orders the applications widget and their child widgets in a simple and powerful way within the application to use the available space to the max. Child widgets are widgets within another widget, for example a drop down menu inside a popup window. The layout management comes with a set of classes used to describe the specific layout. These classes also describes the size of the window, handles resizing and automatic updates of the layout and its child widgets. [22]

Dialog windows offer the user the ability to give input to the software [23]. Rich text processing provides a framework for handling text within the user interface [24]. Similarly, drag and drop offers the ability for the user to drag and drop files within an application [25]. Internationalisation makes the process of changing languages easier [26].

Model view controller is a design pattern used for implementing user interfaces and as such it is used by Qt designer. The design pattern aims to separate the data, from the view and as such simplifies the implementation of the user interface and gives the designer more freedom in how to create the user interface. [27]

2.5.2 Numerical Calculations

Two special libraries of functions were used to do all advanced numerical calculations, NumPy and SciPy. NumPy is a Python library designed to simplify scientific calculations that require to store and handle large amounts of data [28]. It is the foundation on which a lot of other scientific libraries are built that are used within many research fields, such as astrology, materials science, engineering and finance [29]. One of these libraries that utilises NumPy is SciPy, which adds functions for solving and modelling different scientific problems.

NumPy provides the NumPy array as a layer of abstraction to be able to handle the computation of scientific data more easily. The NumPy array is a data structure, similar to an array, which simplifies the handeling of large amount of data.[28] It also introduces features to support calculations on these arrays as if they were vectors as well as support for calculations on each vector element individually [30]. The new data structure significantly speeds up the calculations and computer programs by utilising the underlying hardware more efficiently [29].

SciPy supplies more advanced mathematical algorithms for optimisation, integration, interpolation and more [31]. In this project, interpolation and integration functions were used, especially in regards to spline interpolation and subsequent spline integration, spline derivation and spline antiderivation to retrieve the values and functions for acceleration, velocity, position and jerk. Other more specialised software, such as skictlearn which provides the engineer with more advanced machine learning algorithms for example uses SciPy [29].

NumPy and SciPy was originally built on a Python library called Numeric developed during the 90's for use in scientific calculations. In the middle of 2000's standardisation work and upgrading work was done and the result is NumPy and SciPy. [30]

SciPy includes a package for interpolating a third degree polynomial, a cubic spline, between two points. As additional constraints the value of the first derivative or the second derivative in those two points has to be provided. [32] SciPy also includes a package, called make_interp_spline, for deriving polynomial for higher degree polynomials between two points, thus a five degree polynomial could be derived. To be able to construct this function the method needs the end and start position, as well as the start and end velocity and start and end acceleration. [33]

The cubic spline package, as well as make_interp_spline package, provides methods for deriving the derivative and the antiderivative of the interpolated function [32] [33]. The packages also provide a method for numerical integration for calculating the finite integral of a function between two points [32] [33]. Numerical integration is based on an old FORTRAN routine which sums the product of a function value and weight at multiple instances [34].

2.6 TwinCAT

TwinCAT is a control software solution offered by Beckhoff that is used in PC-based automation system. It features a rich software library specially suited for automation and controlling industrial processes. The development of PC-based machine control started in 1986 and the first version of TwinCAT was released 10 years later. Now TwinCAT 3 is the third generation automation support software, which is meant to work as a comprehensive automation solution with support for both C/C++ and MATLAB/Simulink as well as an open support for third party solutions. It also supports version control, which is integrated in to the development environment to facilitate easy development in a team. [35]

TwinCAT comes in many different forms and modules. For this Master Thesis the TwinCAT XAE and TwinCAT XAR were used. XAE stands for Extended Automation Engineering and offers support for programming according to the IEC-61131-3 standard, as well as C/C++ and MATLAB/Simulink. It also features debugging and diagnostics functionalities for both code and hardware. [35] TwinCAT XAR stands for extended automation runtime. The module offers runtime capabilities to run the developed applications on real machines. The runtime capabilities enables control over the machine on field level. But since the PC-based operating system is always running in the background customer specific software and software for visualising the current effects of the running software can run at the same time as the control platform. This makes it a very versatile environment to develop automation platforms in. TwinCAT XAE software is available as a free to use software where the basic components are included. [35]

An additional feature that was used was the module CAM Design Tool from Beckhoff. This tool offers support for designing motion profiles. It also offers many different features such as different motion functions and set boundary conditions for derivatives. [36]

For this project the real time environment and PLC component of TwinCAT were utilised. Since TwinCAT is an integral part of the mechanical systems at Tetra Pak, development in TwinCAT was of uttermost importance.

2.7 Real Time Software

Real time software is a domain of software where there are usually harder constraints on the system in focus since it interacts with the real world [37]. According to the DIN 44300 standard real time is defined as: Real-time operation is an operating mode of a computer system in which programs for the processing of data are continuously operational in such a way that the processing results are available within a specified period of time. In other words this means that the output value of an application program, which is derived based on the inner states and input values, are available within a defined guaranteed time. This defined time is called cycle time. [38]

Cycle time is the time the program should execute its tasks in. If a task exceeds the cycle time in execution the processor keeps executing that task into to the next cycle and an exceed counter is raised. If then the following cycle is exceeded, the current task is completed and the program is halted until the next possible cycle start. The exceed counter is increased accordingly. This means that many cycles can be lost if a task or process runs too long. [38]

Timing is really important in a real time system. Timing describes the need for a result of a computation to be delivered at a specified time. This is a consequence of the cycle time, harder constraints on real time system compared to a non-real time system as well as the inputs and outputs of the system. For instance if a result is delivered too late the response to an activity outside the system might be too late and not effective. Even a result delivered too early might be harmful since the system might react too soon to an activity outside the system and thus damage the application. [37]

Another consequence of the DIN 44300 standard is the use of run time scheduling [38]. Run time scheduling means that the operations of the microprocessor are not predetermined and thus the schedule is determined as the program runs [37]. In TwinCAT this is achieved with the double tick method where the real time operations are executing first after the first tick. After the second tick the underlying operating system is running its tasks until the cycle time finishes. [38] The order of the execution of the different tasks are decided by a priority system where the task with the highest priority is done first. [38]

2.7.1 State Machines

An important concept in real time programming is the use of state machines. They are used to model and describe the real world seen from the computers perspective which can then be used to analyse the system [37]. For example the state machine of a door would be the states open and close. The transitions would then be open and closed. The open transition would go from the closed state to the open state and the close transition would then go from the open state to the closed state.

The transitions between the states are instantaneous but it is also here that the interaction with the real world happens. The transitions should be triggered by some event, for example a sensor trigger or a switch turning on or off. Since the state transitions are event triggered, the time spent in one state is unknown and therefore not a good transition trigger [37].

One way to implement the state machine is to use the switch case program statement. Then the different states would be the different cases and the transitions would be when going from one state to another. [37]

3 Background

In this chapter the background for this thesis will be presented. As mentioned in the introduction some work regarding the CAMTool had already been done, which this section aims to describe.

Firstly, the Python variant of the CAMTool will be introduced. This section is divided into two smaller parts, one for the GUI and one for the application. Secondly, the TwinCAT application will be described, followed by the goals of this project including illustrations to show what should be achieved.

3.1 Python Implementation

As mentioned above the Python application is divided into two parts. First the GUI will be described and how it relates to the theory previously presented. Next the application itself will be presented including how it works.

3.1.1 Graphical User Interface

If a new application should be appreciated and used in the future, the GUI has to be user friendly and nice looking. It should be easy to understand what every button does, as well as finding the functionalities fast. The GUI in the starting point of this thesis work can be found in figure 13 and figure 14, where the first one is for the input table and the second one is for the initial settings. Both figures are divided into smaller sections or boxes, numbered one to seven. Notice, box number three and four is the same for both.

Box number one is the input table, where the engineers can enter the parameters of the desired motion profile. The second box is where the user can add a new motion type which is added to the input table, or delete an existing row in the input table. In this box the motion profile can change between Master Position or Time at the x-axis. The next box is where the result is plotted in the first two tabs, named Position/Velocity and Acceleration/Jerk, while the last tab is the data of the plots, named CAM Table. The fourth and last box in figure 13 contains different functionalities as the explanation of the GUI and the ability to save the current CAM as well as load an old saved one. In figure 14 box five is where the user enters the starting values for the motion profile. The next box is where the variable name of the generated Beckhoff file is decided, while the last box includes some short instructions.

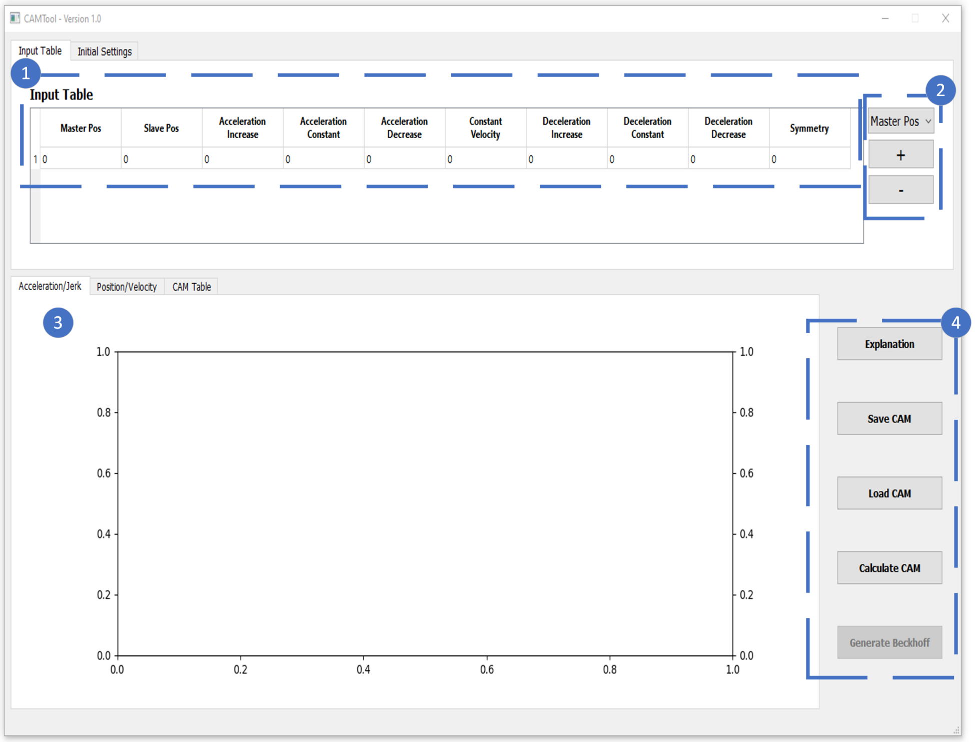
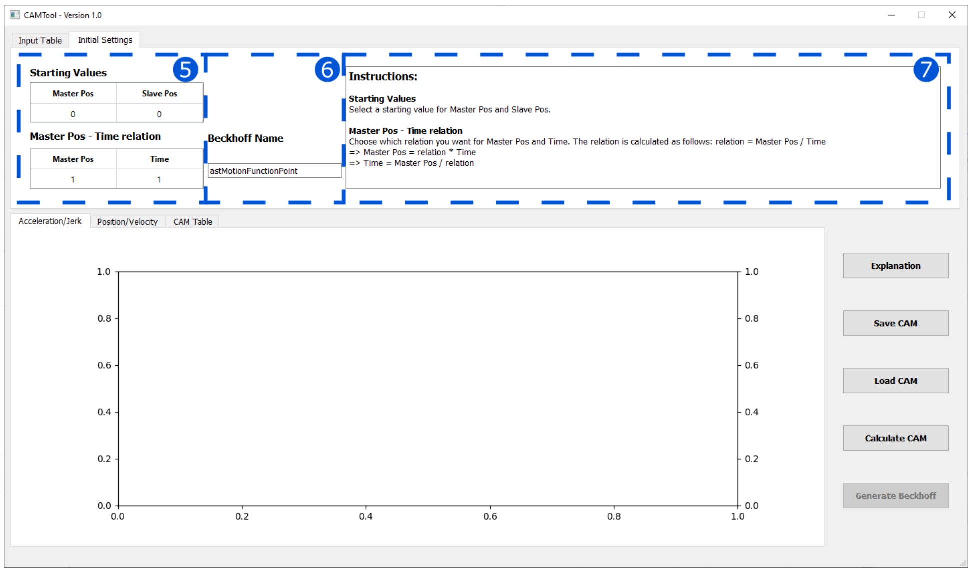


Figure 13: This figure shows the starting point for the GUI interface, under the tab Input Table



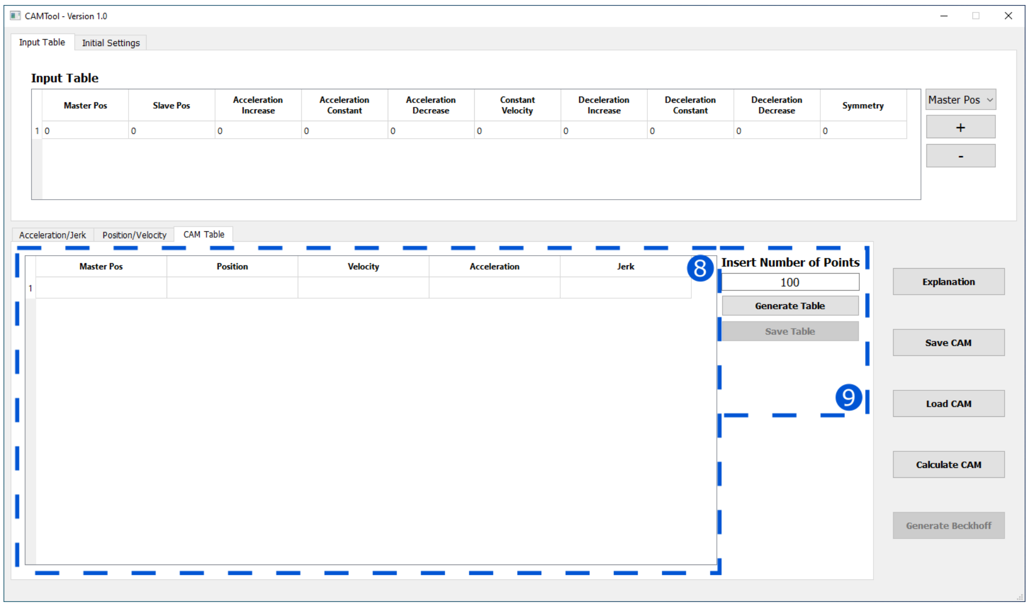
Instructions:

Starting Values
Select a starting value for Master Pos and Slave Pos.

Master Pos - Time relation
Choose which relation you want for Master Pos and Time. The relation is calculated as follows: relation = Master Pos / Time
=> Master Pos = relation * Time
=> Time = Master Pos / relation

Figure 14: Figure showing the starting point for the GUI interface, under the tab Initial Settings

As mentioned above there is a tab called CAM Table, which can be seen in figure 15. To generate the data, which will be shown in box number eight, the user must first create a motion profile. Thereafter the button called Generate Table in box nine can be pressed. This generated table, which can be seen in box ten in figure 16, can be saved for later use by the button Save Table in box number eleven.

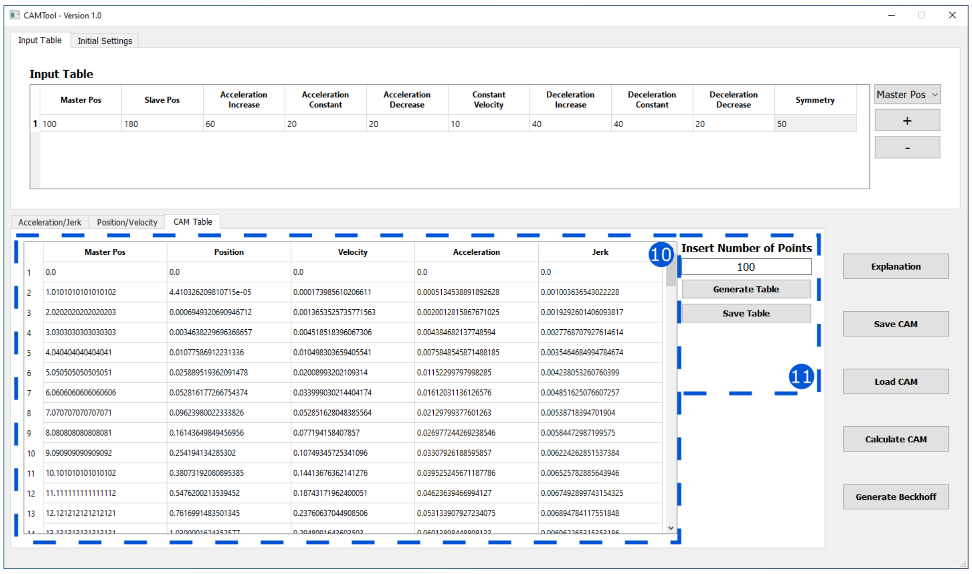


Input Table

	Master Pos	Slave Pos	Acceleration Increase	Acceleration Constant	Acceleration Decrease	Constant Velocity	Deceleration Increase	Deceleration Constant	Deceleration Decrease	Symmetry
1	0	0	0	0	0	0	0	0	0	0

	Master Pos	Position	Velocity	Acceleration	Jerk
1					

Figure 15: This figure shows the starting point for the GUI interface, under the tab CAM Table



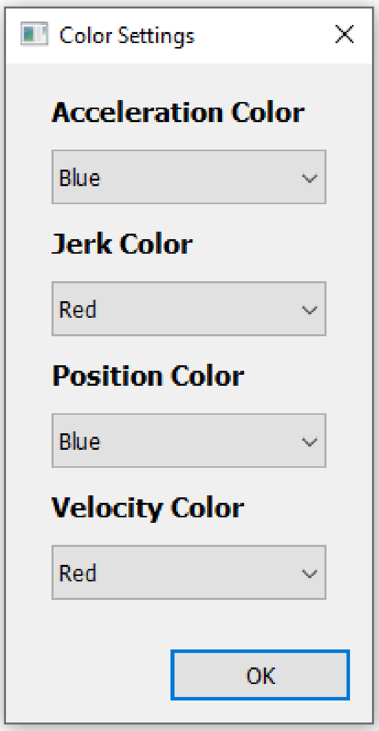
Input Table

	Master Pos	Slave Pos	Acceleration Increase	Acceleration Constant	Acceleration Decrease	Constant Velocity	Deceleration Increase	Deceleration Constant	Deceleration Decrease	Symmetry
1	100	180	60	20	20	10	40	40	20	50

	Master Pos	Position	Velocity	Acceleration	Jerk
1	0.0	0.0	0.0	0.0	0.0
2	1.0101010101010102	4.410326209810715e-05	0.000173985610206611	0.0005134538891892628	0.001003636543022228
3	2.0202020202020203	0.0006949320690946712	0.0013653525735771563	0.0020012815867671025	0.0019292601406093817
4	3.0303030303030303	0.0034638229696368657	0.004518518396067306	0.004384682137748594	0.0027768707927614614
5	4.040404040404041	0.01077586912231336	0.010498303659405541	0.0075848545871488185	0.0035464684994784674
6	5.050505050505051	0.025889519362091478	0.02008993202109314	0.01152299797998285	0.004238053260760399
7	6.0606060606060606	0.052816177266754374	0.033999030214404174	0.01612031136126576	0.004851625076607257
8	7.070707070707071	0.09623980022333826	0.052851628048385564	0.02129799377601263	0.00538718394701904
9	8.080808080808081	0.16143649849456956	0.077194158407857	0.026977244269238546	0.00584472987199575
10	9.090909090909092	0.254194134285302	0.10749345725341096	0.03307926188595857	0.006224262851537384
11	10.101010101010102	0.38073192080895385	0.14413676362141276	0.039525245671187786	0.006525782885643946
12	11.111111111111112	0.5476200213539452	0.18743171962400051	0.04623639466994127	0.0067492899743154325
13	12.121212121212121	0.7616991483501345	0.23760637044908506	0.053133907927234075	0.006894784117551848

Figure 16: Figure showing the starting point for the GUI interface, under the tab CAM Table with the generated data

One function that is not shown in figures 13 and 14 but still exists, is to change the colour of the motion profiles. This can be done by right clicking at the plot, corresponds to box three in figure 13, which will make a pop-up window with colour settings appear as shown in figure 17 (a). The user och operator can choose between ten different, predefined colours individually for each motion profile as shown in figure 17 (b).



Color Settings

Acceleration Color

Blue

Jerk Color

Red

Position Color

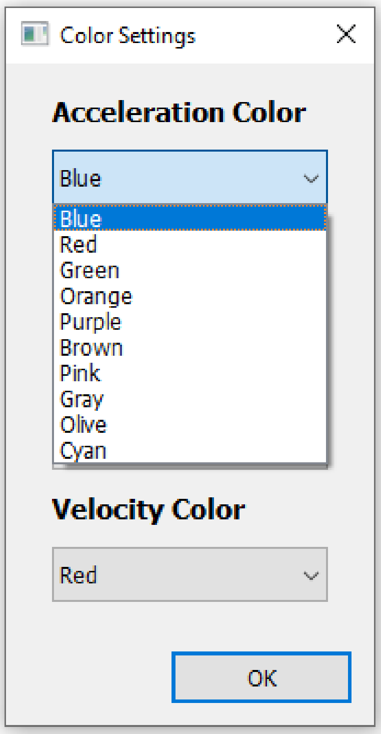
Blue

Velocity Color

Red

OK

(a)



Color Settings

Acceleration Color

Blue

Blue
Red
Green
Orange
Purple
Brown
Pink
Gray
Olive
Cyan

Velocity Color

Red

OK

(b)

Figure 17: This figure shows the starting point for how to change colour in the GUI with the default in (a) and an extended drop down in (b). Colour is spelt the American way in this picture. This is changed in all other pictures

3.1.2 Application

Figure 18 is a flow graph which shows the overall control flow of the Python application. The application starts executing in the Mainwindow module. Following the arrows yields the following execution order: Controller, InputHandler, SplineCalculator and lastly the GUI which shows the finished motion profiles.

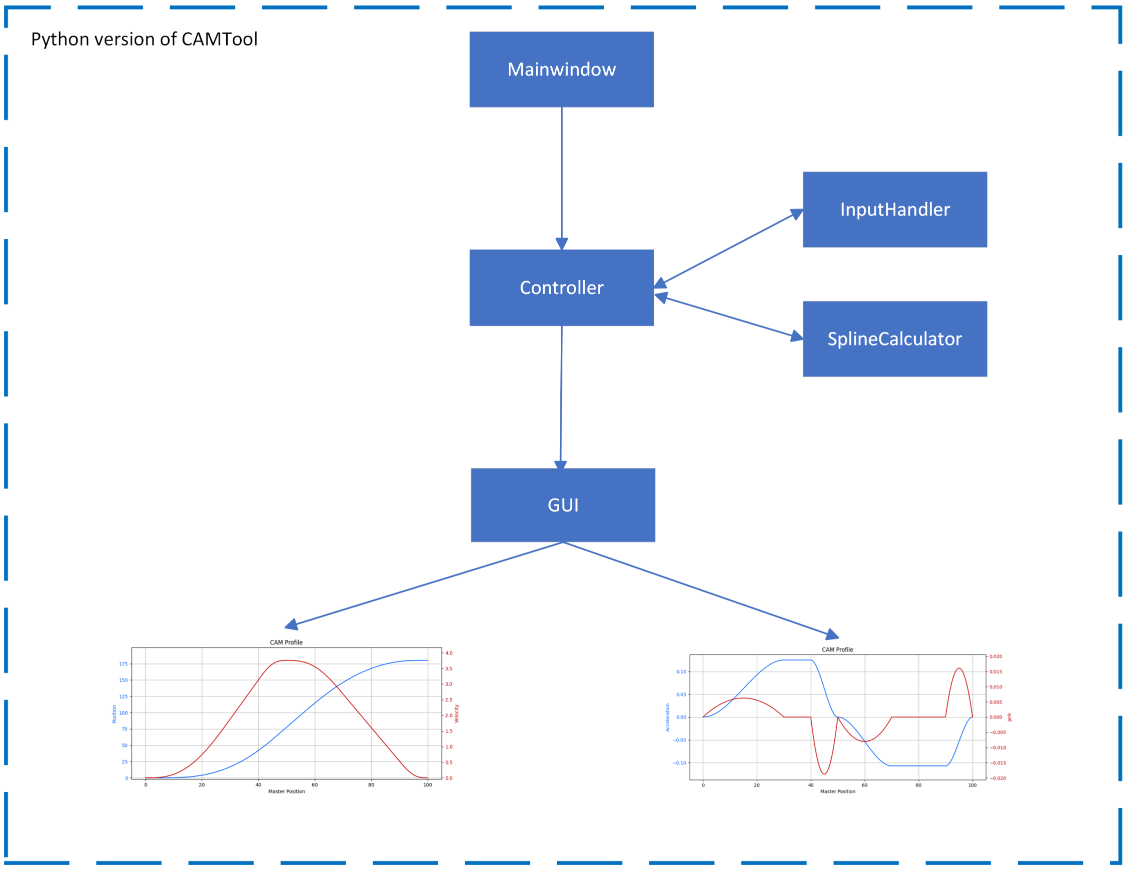


Figure 18: This figure shows the structure of the Python program

From the Mainwindow module, the application is run. It is also here all the code that connects the buttons with the functionality is located.

The Controller acts as the link, within the CAMTool application, between the InputHandler and SplineCalculator. It imports the data from the Mainwindow and sends it to the InputHandler. When the data have been processed by the InputHandler it gets passed to the SplineCalculator. This connection can be seen in figure 18 and it is also here that the motion functions get evaluated and the values are passed to the GUI where they are plotted in the graph. The Controller can also generate code that can be used to run a newly generated motion profile in TwinCAT. The code is generated when pressing the button Generate Beckhoff, which can be seen in box four in figure 13. The generated code consists of the knots that tie all the individual splines together.

The InputHandler handles the input and coverts it to the right format so that the SplineCalculator can calculate the splines. The acceleration percentages and deceleration percentages get converted to corresponding y- and x-values. Verification is implemented to ensure that the user inputs the correct form of inputs and that the numbers add up. For example, the sum of acceleration increase, acceleration constant and acceleration decrease has to be 100, while acceleration increase and acceleration decrease has to be bigger than or equal to one when calculating a motion profile for a Point to Point motion.

The SplineCalculator calculates the coefficients of the polynomial function, which will pass through the required points using the equations presented in subsection 2.3.1 and the Python packages presented in subsection 2.5.2. Since acceleration behaviour is user defined the polynomial coefficients for the acceleration profile are calculated first. The coefficients for the velocity polynomial and position polynomial are then derived by antiderivation. The jerk polynomial coefficients are derived by derivation of the acceleration splines.

The polynomial coefficients for jerk, acceleration, velocity and position are returned to the Controller along with x-values representing the Master Position. The Controller calculates the corresponding y-value for jerk, acceleration, velocity and position given a certain x-value.

The last arrow in figure 18 between the Controller and the GUI, represents sending the jerk-, acceleration-, velocity- and position values and the matching x-values to the GUI, which draws up the graphs of the jerk, acceleration, velocity and position of the motion.

As mentioned in section 1.1, Point to Point motion and Dwell / Cruise were the only types of motion able to design and calculate with the old version of CAMTool. An example of a Point to Point motion can be seen in figures 19 and 20, where the parameters are as followed: acceleration decrease 60 %, acceleration constant 20 %, acceleration decrease 20 %, constant velocity 0 %, deceleration increase 40 %, deceleration constant 40 %, deceleration decrease 20 % and the symmetry 50%. The motion of Dwell / Cruise can be found in figure 21, which have the following parameters: constant velocity 100 %, the rest are zero. Since the position and velocity are both zero all the time, the acceleration and jerk behaviour will be the same and are not included in the report.

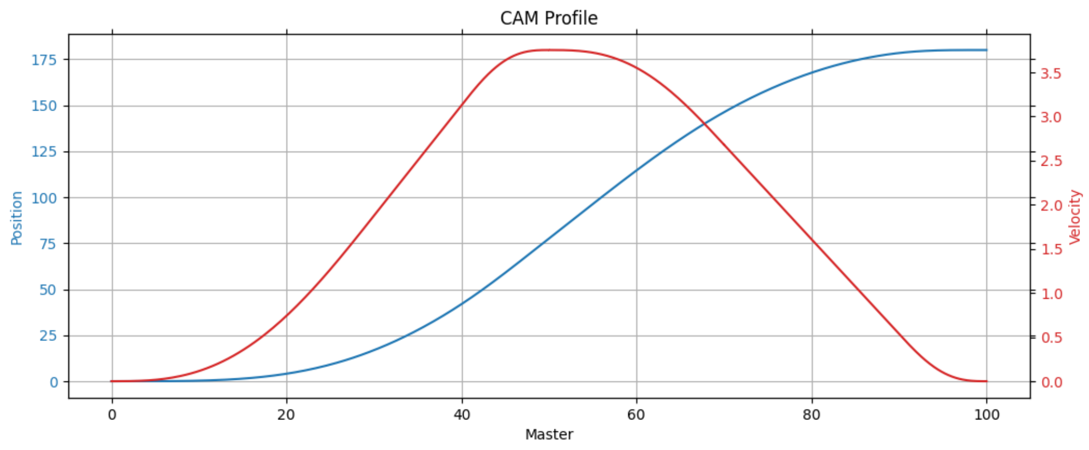


Figure 19: Shows an example of a Point to Point motion, where position is in blue and velocity in red

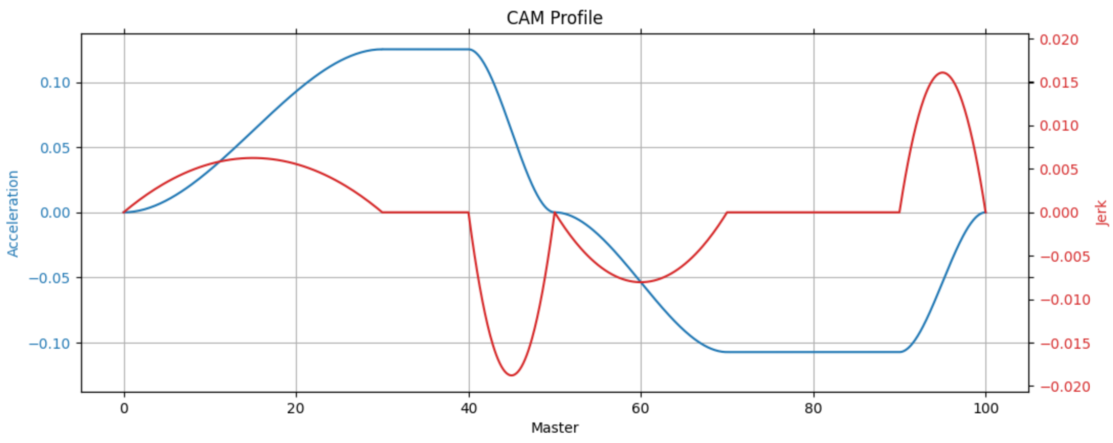


Figure 20: Shows the acceleration (blue) and jerk (red) behaviour of figure 19

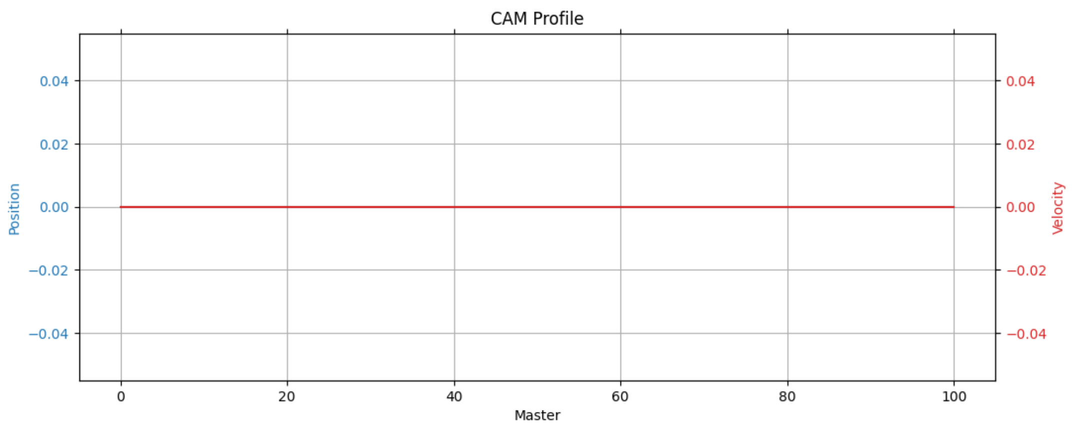


Figure 21: Shows an example of a Dwell / Cruise where position is in blue and velocity is in red

3.2 TwinCAT Implementation

The TwinCAT application tries to follow the same architecture as the Python implementation, where the FB_Inputhandler calculates the acceleration amplitude given the input parameters. The spline calculator function block, FB_SplineCalculator then calculates the acceleration function given the acceleration amplitude and the jerk, velocity and position functions given by the acceleration function. The functions are then passed to FB_BuildCamPoints, which calculates discrete points which are used to plot the motion profiles. A control flow graph can be seen in figure 22, which shows the overall control flow of the TwinCAT application.

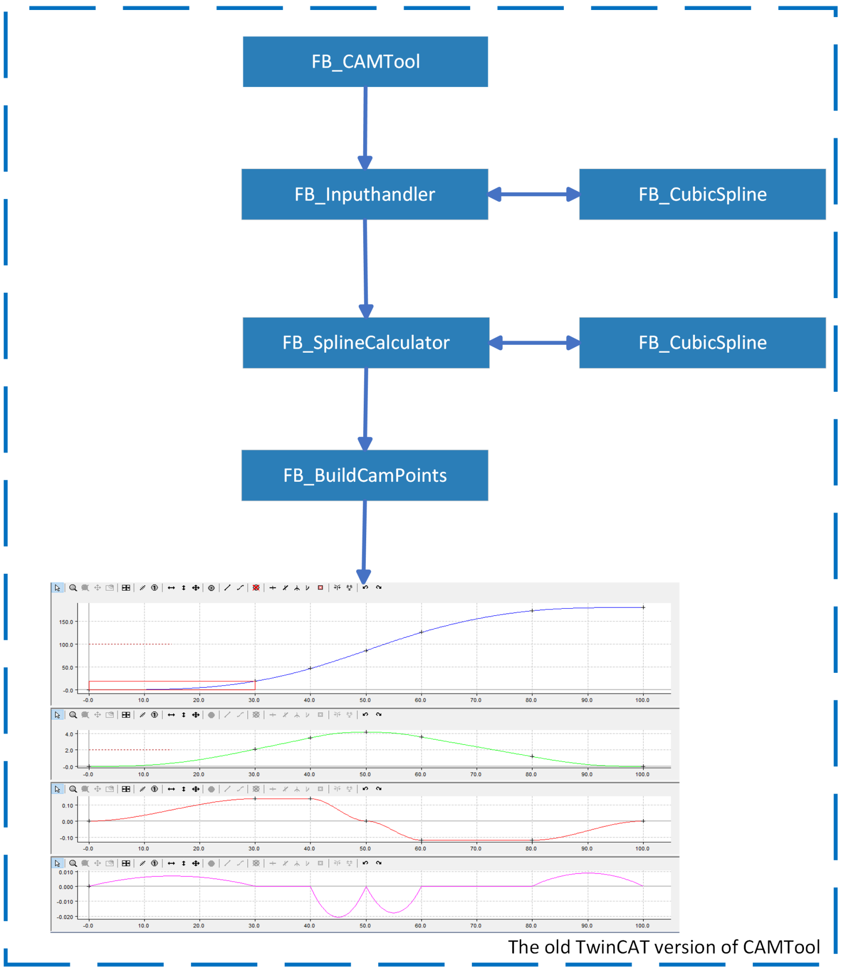


Figure 22: This figure shows the structure of the PLC program

3.3 Goals

This section aims to describe the different requirements that were stated in chapter 1 and is divided in the same way. The non-zero jerk motion profile will be described first followed by a description of Velocity Increase and Velocity Decrease. After that Position Increase and Position Decrease as well as Rendezvous will be described and finally the real time performance of the application will be discussed.

3.3.1 Non-Zero Jerk

This subsection will describe how goal number one that was setup in section 1.2 will be fulfilled. The implementation for that goal will from here on be called non-zero jerk and will build on the equations presented in the subsection 2.4.2.

The non-zero jerk function is only applicable at four points in the motion. At the beginning of the acceleration part, at the end of acceleration part, at the beginning of the deceleration part and at the end of deceleration part. At the beginning and end of the constant acceleration and constant deceleration motions the jerk value is always zero as can be seen in the graphs below. In figure 23 the first case where the difference between non-zero and zero jerk is in the beginning of the acceleration motion can be seen. The second case, where the difference between non-zero and zero jerk is in the end of the acceleration motion, can be seen in figure 24. The non-zero jerk at the beginning and end of the deceleration motion can be seen in figure 25 and figure 26, respectively.

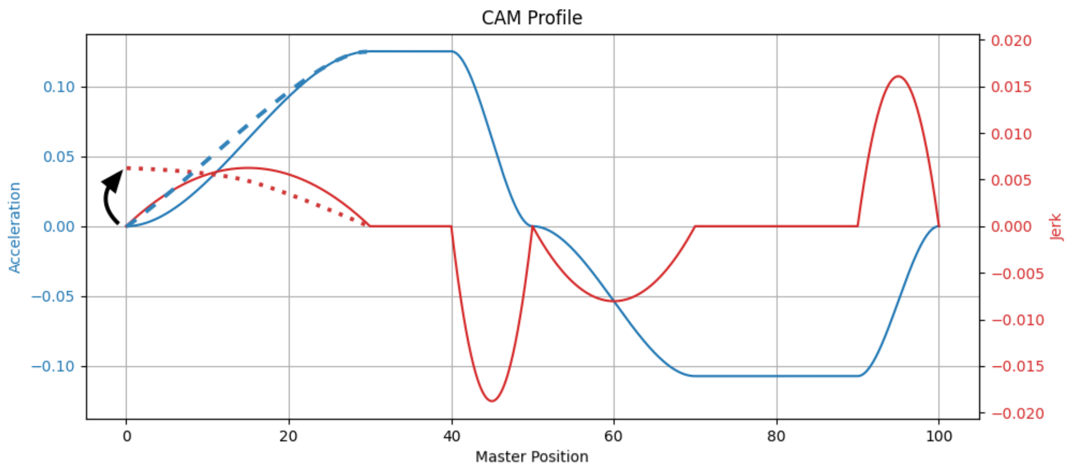


Figure 23: This figure illustrates the difference between non-zero jerk and zero jerk at the beginning of the increasing acceleration motion. Notice also the difference in acceleration behaviour. Acceleration is in blue with the non-zero jerk behaviour in dashed blue and zero jerk behaviour in straight blue. Jerk is in red, with non-zero jerk in dotted red

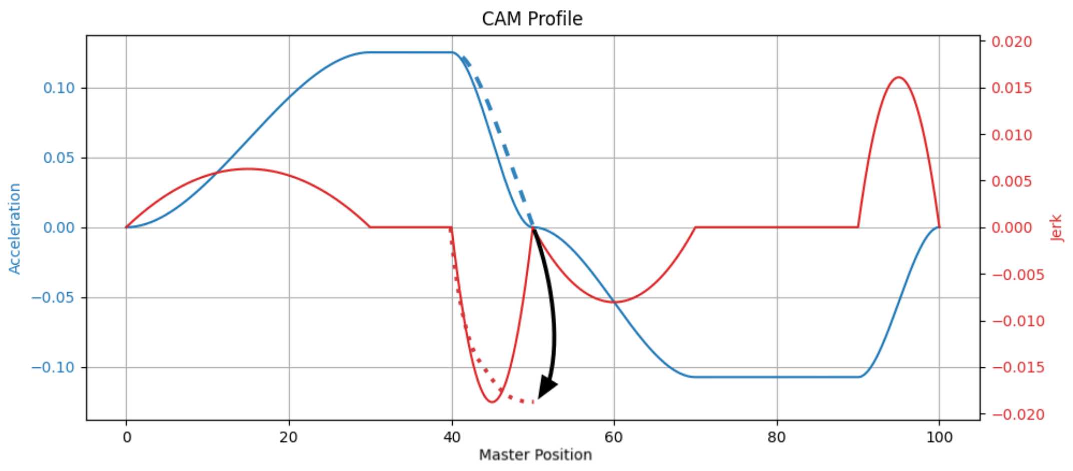


Figure 24: This figure illustrates the difference between non-zero jerk and zero jerk at the end of the decreasing acceleration motion. Notice also the difference in acceleration behaviour. Acceleration is in blue with the non-zero jerk behaviour in dashed blue and zero jerk behaviour in straight blue. Jerk is in red, with non-zero jerk in dotted red

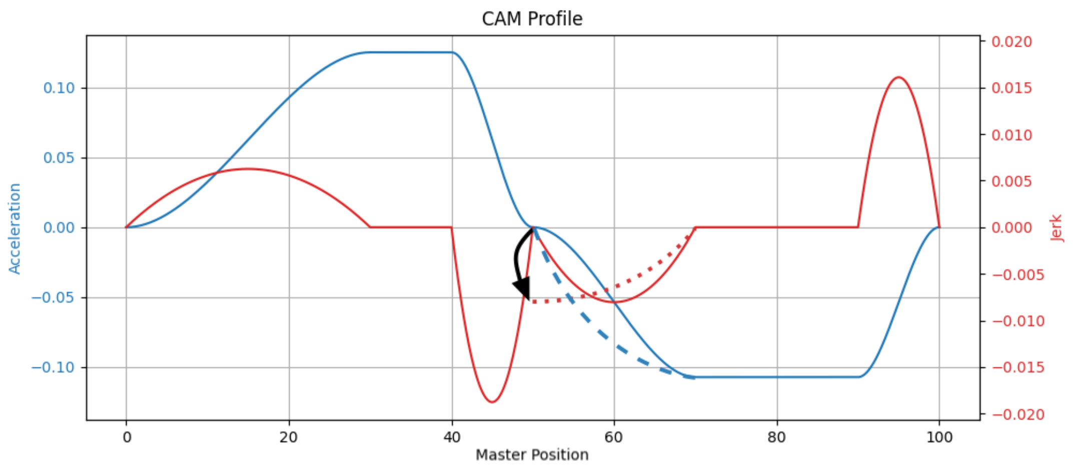


Figure 25: This figure illustrates the difference between non-zero jerk and zero jerk at the beginning of the increasing deceleration motion. Notice also the difference in acceleration behaviour. Acceleration is in blue with the non-zero jerk behaviour in dashed blue and zero jerk behaviour in straight blue. Jerk is in red, with non-zero jerk in dotted red

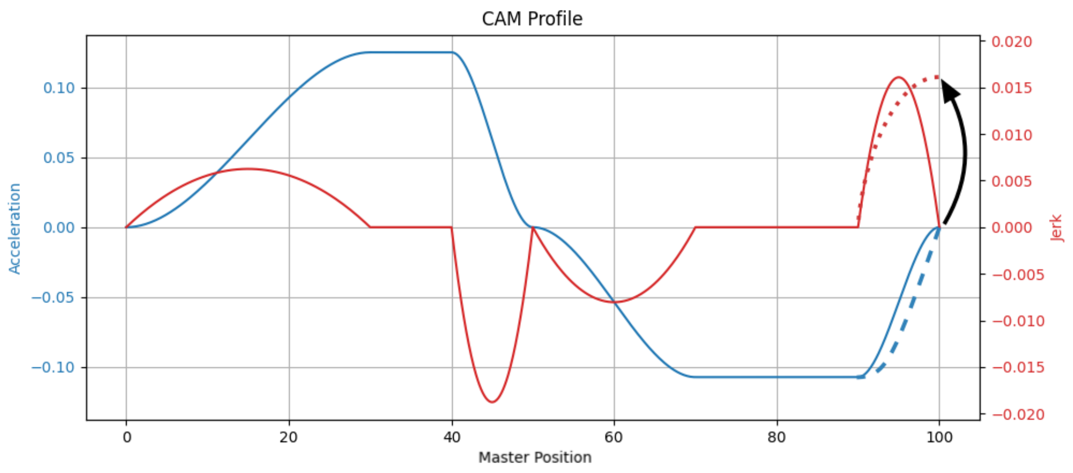


Figure 26: This figure illustrates the difference between non-zero jerk and zero jerk at the end of the decreasing deceleration motion. Also, notice the difference in acceleration behaviour. Acceleration is in blue with the non-zero jerk behaviour in dashed blue and zero jerk behaviour in straight blue. Jerk is in red, with non-zero jerk in dotted red

3.3.2 Velocity Increase and Velocity Decrease

Velocity is the first antiderivative of acceleration. In this project, a motion profile with a predefined initial velocity and position as well as a predefined end velocity will be investigated. This means that the initial velocity and end velocity can be any value. Depending on if the end velocity is bigger or smaller than the initial velocity it is defined as a Velocity Increase or Velocity Decrease, respectively. The aim is to achieve something like in figure 27 for Velocity Increase and figure 29 for Velocity Decrease. Along with these new velocity profiles the aim is to achieve an acceleration behaviour and thus a jerk behaviour as shown in figure 28 for Velocity Increase and figure 30 for Velocity Decrease, respectively. Please note that the figures shown below are not the actual motion profiles but more a representation of the behaviour hoped to be achieved.

The figures 27, 28, 29, 30 mentioned above aim to describe the fulfilment of goal number two setup in section 1.2. The implementation for that goal will from here on be called Velocity Increase or Velocity Decrease depending on if the initial velocity is smaller or bigger than the end velocity and will build on the theory presented in subsection 2.4.3, 2.4.1 as well as 2.4.2.

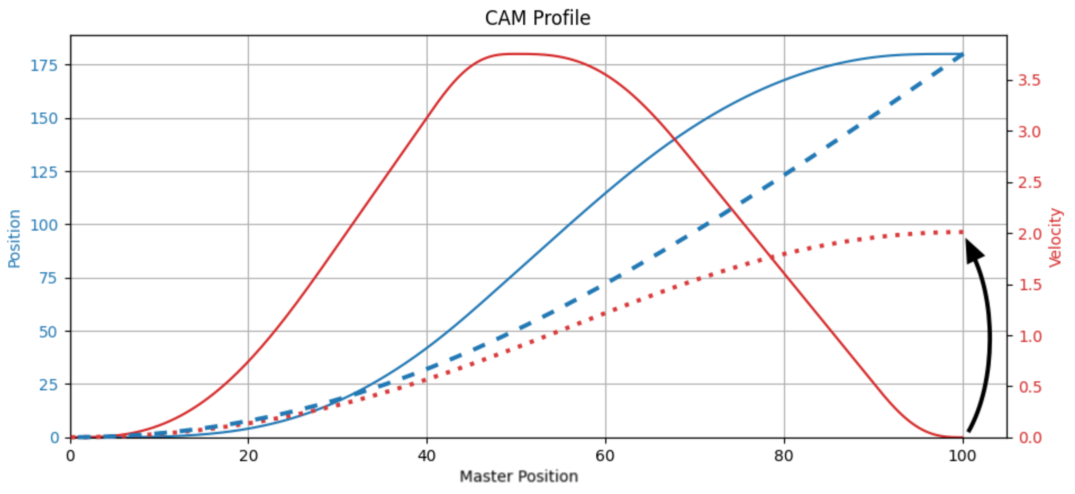


Figure 27: This figure tries to illustrate the difference between a Point to Point motion and a Velocity Increase in terms of position and velocity. Velocity is in red, with the targeted behaviour dotted. Position is in blue with the targeted behaviour dashed

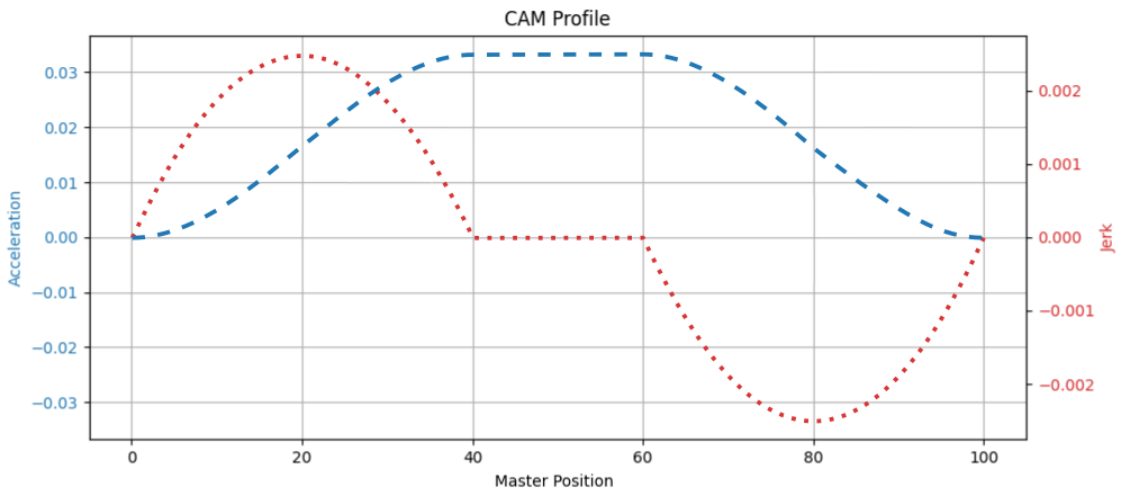


Figure 28: This figure shows the expected acceleration and jerk behaviour corresponding to figure 27. Acceleration is in blue, and jerk is in red

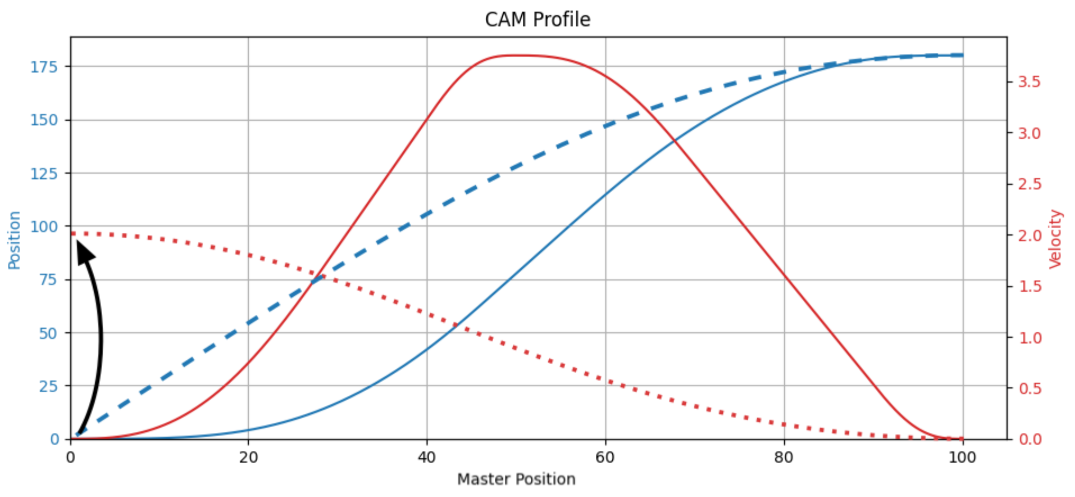


Figure 29: This figure tries to illustrate the difference between a Point to Point motion and a Velocity Decrease in terms of position and velocity. Velocity is in red, with the targeted behaviour dotted. Position in blue with the targeted behaviour dashed

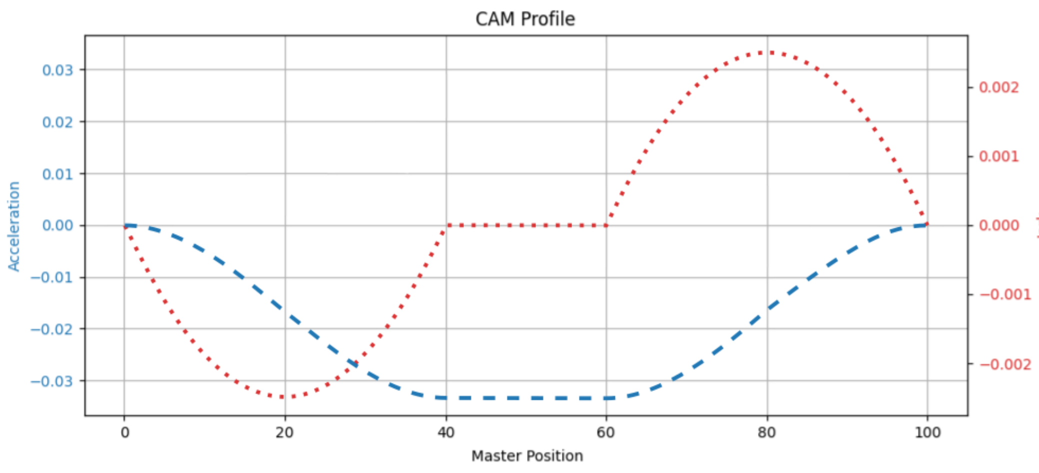


Figure 30: Shows the expected acceleration and jerk behaviour corresponding to figure 29. Acceleration is in blue, and jerk is in red

3.3.3 Position Increase and Position Decrease

This subsection will describe how goal number four in section 1.2 is achieved. That goal will from here on be called Position Increase or Position Decrease depending on if the initial position is smaller or bigger than the end position. The implementation will build on the theory presented in subsection 2.4.1 and 2.4.4 as well as 2.4.2.

Position is derived by taking the second antiderivative of the acceleration. During this project, a motion profile with a predefined initial start velocity and start position and a predefined end position will be investigated. Figures 31 and 32 below aims to show how the motion profile is supposed to look like when doing a Position Increase. With the new profile represented by the dotted line.

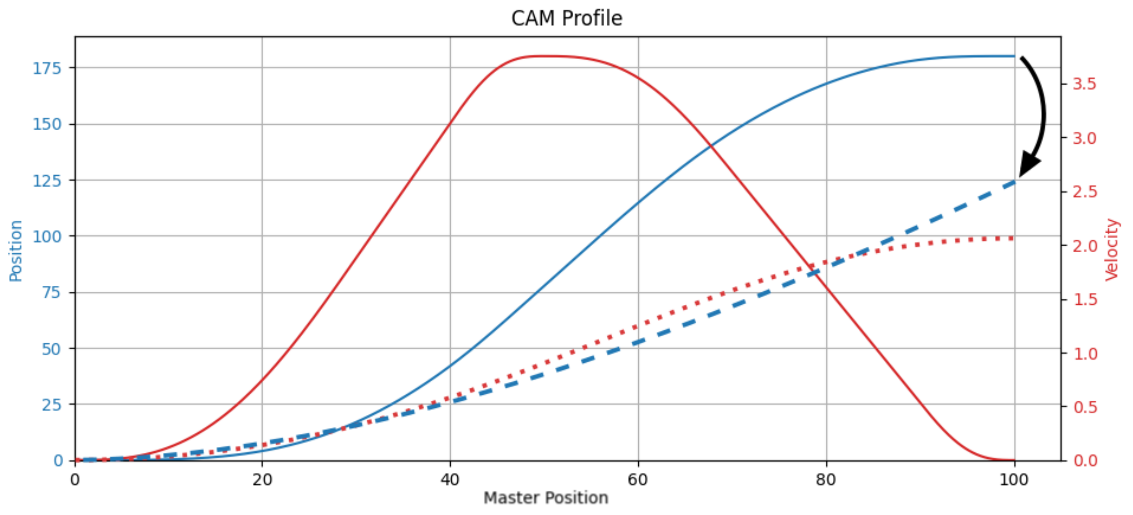


Figure 31: This figure shows the aim of the Position Increase motion compered to a Point to Point motion. The dashed blue line is supposed to represent the new position and the dotted red line is supposed to represent the new velocity

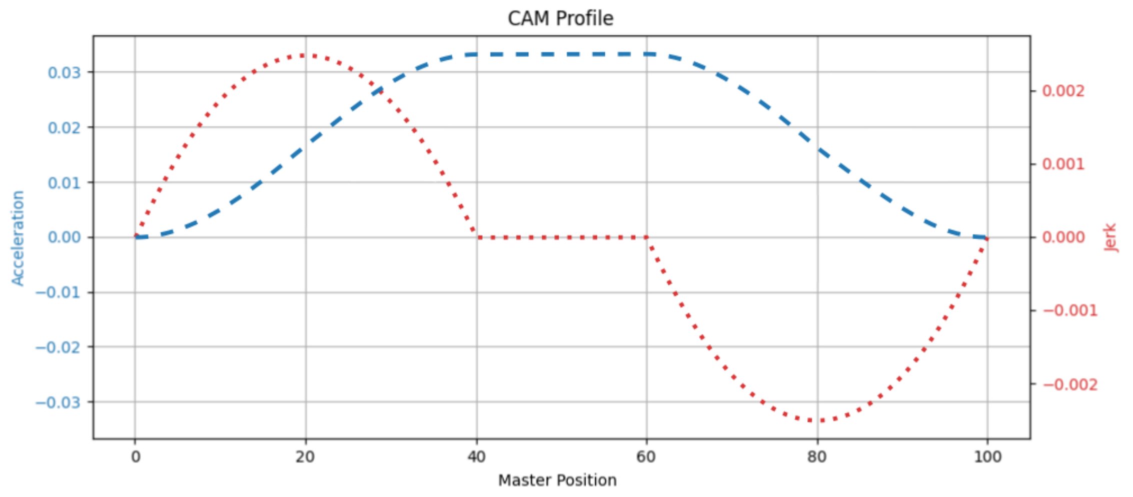


Figure 32: This figure shows how the acceleration and jerk behaviour is supposed to look when doing a Position Increase motion. The dotted line and the dashed line are supposed to represent the new motion. Blue is for acceleration and red is for jerk

Figures 33 and 34 below aim to show how a motion profile is supposed to look like when doing a Position Decrease. The new profile is represented by the dotted and dashed lines.

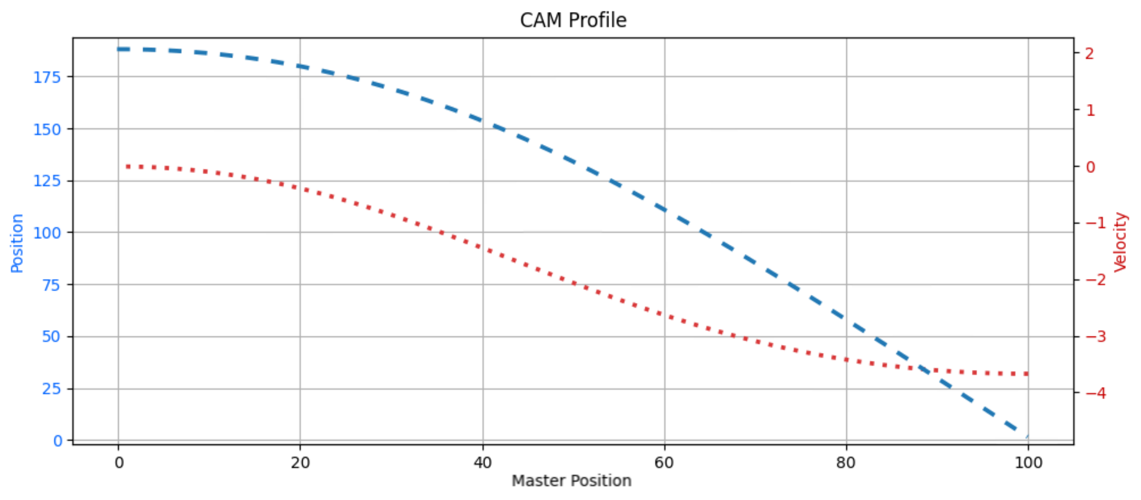


Figure 33: This figure shows the aim of the Position Decrease motion. The dashed blue line are supposed to represent the new position behaviour and the dotted red line is supposed to represent the new velocity behaviour

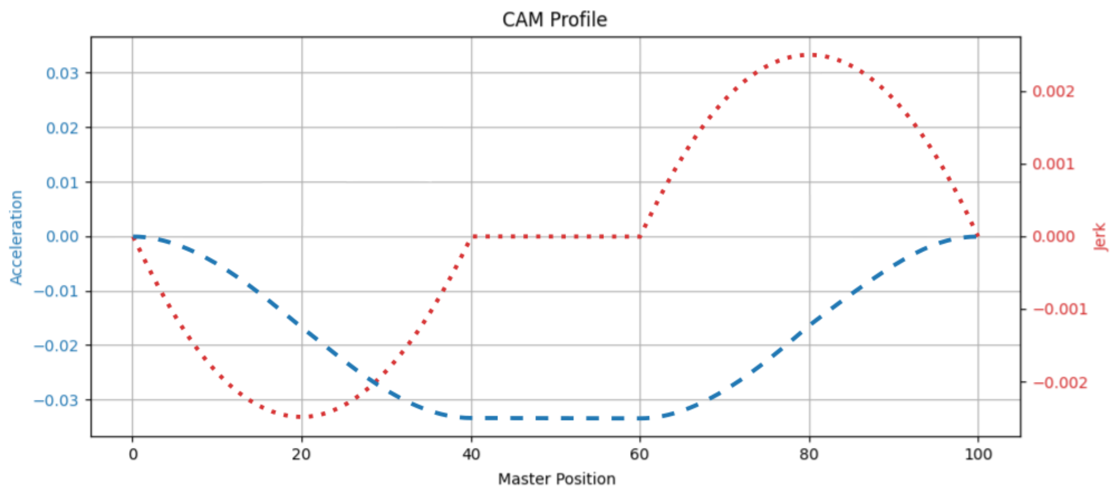


Figure 34: This figure shows how the acceleration and jerk behaviour is supposed to look when doing a Position Decrease motion. Blue is for acceleration and red is for jerk

3.3.4 Rendezvous

Rendezvous is a term that we use to describe a motion profile, which has a predefined start position and start velocity and reaches a predefined end position with a predefined end velocity. In this case the acceleration behaviour is left undefined and will thus be defined by the method used to calculate the motion profile. This is also referenced as goal number six in section 1.2 of this thesis. The implementation of this goal will from here on be called Rendezvous.

Revisiting equation 16 we see that there are six unknowns, a, b, c, d, e and f thus we need six equations to be able to solve for the unknowns. Two equations can be derived from the end and start positions. Two equations can be derived from the start and end velocities of the motion using equation 15. The final two equations can be derived from the acceleration equation of the motion, using equation 9. In our project, we have assumed that the acceleration will always be zero at the end points since the points defined are at the end and beginning of a motion. The following system of equations is thus obtained.

$$f_{position}(x_{start}) = ax_{start}^5 + bx_{start}^4 + cx_{start}^3 + dx_{start}^2 + ex_{start} + f$$
$$f_{position}(x_{end}) = ax_{end}^5 + bx_{end}^4 + cx_{end}^3 + dx_{end}^2 + ex_{end} + f \tag{17}$$

$$f_{velocity}(x_{start}) = 5ax_{start}^4 + 4bx_{start}^3 + 3cx_{start}^2 + 2dx_{start} + e$$
$$f_{velocity}(x_{end}) = 5ax_{end}^4 + 4bx_{end}^3 + 3cx_{end}^2 + 2dx_{end} + e \tag{18}$$

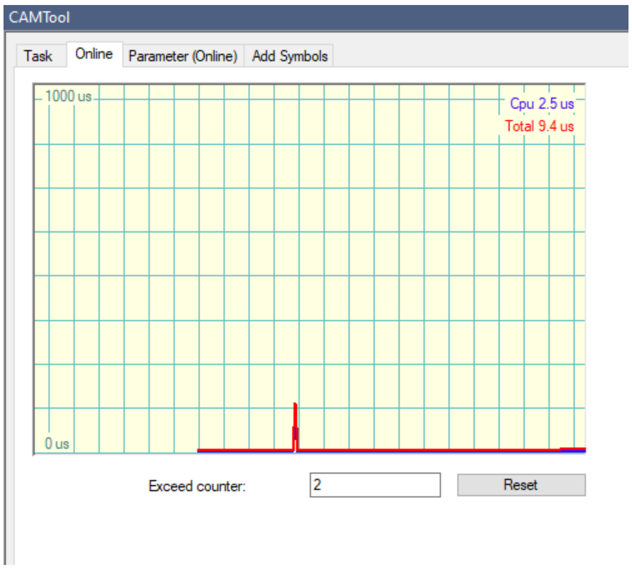
$$f_{acceleration}(x_{end}) = 20ax_{end}^3 + 12bx_{end}^2 + 6cx_{end} + 2d = 0$$
$$f_{acceleration}(x_{start}) = 20ax_{start}^3 + 12bx_{start}^2 + 6cx_{start} + 2d = 0 \tag{19}$$

And thus the coefficients a, b, c, d, e and f can be solved and the motion profile can be calculated.

3.3.5 Real Time Implementation

The PLC program was poorly implemented with respect to real time properties. The main issue was that the program occupied too much of the CPU and thus other routines and programs could not run within a cycle time and therefore there was not enough CPU time for other functions such as output and input actions or similar. Hence, as stated in section 1.2, this is goal number three of this project. The implementation of this goal will from here on be called Real Time Implementation.

In TwinCAT one can decide how long the cycle time should be. A short cycle time, where the minimum in TwinCAT is one millisecond, means the program can only occupy the CPU for a short time. The aim regarding the real time implementation for this project was to eliminate the number of times the program exceeded the cycle time when having a cycle time of one millisecond. As can be seen in figure 35 below, the application exceeds the cycle time twice. Although one millisecond is a rather short time it ensures that the program will run on all kinds of applications with different amount of cycle times without any problem.



Exceed counter: 2

Figure 35: This figure shows the exceed counter when running the old version of PLC version of CAMTool

4 Method

This chapter is meant to introduce the methodology of this project, which changes were implemented and in what order they were done. First the Python changes will be introduced, where the first changes were done to the GUI and then the changes to the CAM tool application. Then the changes to TwinCAT were implemented.

4.1 Python Implementation

The changes made to the Python implementation were done both in the GUI and in the application itself. The GUI changes were done first since they were estimated to be the easiest and offer a better introduction to how the application worked.

After the GUI changes were done the work on the application itself started. The functionalities that were to be implemented are shown in subsection 4.1.2 and correspond to the requirements determined in the introduction.

4.1.1 Graphical User Interface

Regarding the GUI, an application called QT Designer was used to design the layout of the interface, which is described in subsection 2.5.1. By ticking off each improvement in the list, the GUI started look as desired by our supervisor at Tetra Pak. When some of the changes had been implemented, the GUI was shown to our supervisor to get feedback directly. After another set of changes had been implemented, it was shown again. In this way the GUI took the best shape possible.

The first set of changes for improving the GUI is shown down below, numbered 1 to 8. It was decided to start with some simple changes to get to know the code and application better. These changes did not require too much code to be changed but some changes to the interface design in Qt Designer.

1. Change "Explanation" button text to "Help"
2. Change "Generate Beckhoff" button text to "Generate TwinCAT PLC Code"
3. Change "Load CAM" button text to "Open CAM"
4. Change the text "Insert number of points" to "Minimum number of points" under CAM Table tab
5. Move the Position/Velocity tab to be the most left tab
6. Make it consistent so that all values are floats (decimal numbers) as default
7. Centre all values in the Input Table
8. Change the order of the button to the right: Calculate CAM, Generate Beckhoff, Save CAM, Load CAM, Explanation

The second set of changes to improve the GUI is shown below, numbered 9 to 16. After the minor changes, as seen in the list above, it was time to do the changes that required more insight into the application. The reason why the changes below required more from the developer is because one had to deal with both the Qt Designer as well as to connect the new design with new functionality using code.

9. Add a button to change colour and a palette
10. Add a button to show or hide the knots in the graphs
11. Make it possible to delete a certain row in the input table
12. Make it possible to insert a row in the input table
13. Time-limits in the graph shall follow the times of the CAM, not exceed
14. Drop down menu to decide type of motion (Point to Point, Dwell and then future possibilities) instead of doing if through the parameters
15. Add text box to define the name of MC_CAM_REF variable under Initial-settings tab
16. Possible to introduce some kind of tool tip to display the values in a specific point in the CAM-diagram. It shall be the value of the CAM, not the value of the mouse pointer position in the graph

4.1.2 Application

After the changes to the GUI, as stated in subsection 4.1.1, were implemented, work on the actual goal of the Master Thesis was started. Those are listed below and were done in the following order.

1. A motion profile where the jerk is non-zero at the beginning of acceleration increase, end of acceleration decrease, beginning of deceleration increase and end of deceleration decrease
2. A motion profile where the start and end velocities are defined by the user, including the acceleration behaviour
3. A motion profile where the start and end position are defined by the user, including the acceleration behaviour
4. A motion profile where both start and end velocities as well as start and end positions are defined by the user

4.2 TwinCAT Implementation

The similar changes done to the Python code was also implemented in TwinCAT. In addition to that there was also a requirement to adapt the software to work in a real time context. The following things were done.

1. A motion profile where the jerk is non-zero at the beginning of acceleration increase, end of acceleration decrease, beginning of deceleration increase and end of deceleration decrease
2. A motion profile where the start and end velocities are defined by the user, including the acceleration behaviour
3. Rewrite the software so that its execution time does not occupy the processor for longer than a fraction of a cycle period
4. A motion profile where the start and end position are defined by the user, including the acceleration behaviour
5. A motion profile where both start and end velocities as well as start and end positions are defined by the user

The decision to take the TwinCAT application after the Python application was that the latter offered a better chance to learn how the code worked and understand its purpose. Since the idea of the Python and TwinCAT software are the same this offered the best way to understand both.

4.3 Verification

It is not possible to say anything about the new CAMTool without comparing it with Beckhoffs CAM Design Tool for creating motion profiles, which will be the way to preform the verification of the new version of CAMTool. The speed-torque diagram was introduced in subsection 2.1.1 and will be used to make sure the new version of CAMTool will make a better operate-path for the servo motor compared to Beckhoffs CAM Design Tool. Since the older version of CAMTool only is able to do the motion type Point to Point except Dwell, this is what is going to be compared. The rest of the motion types the new version of CAMTool is able to calculate are they are all based on the same principle, which means they do not have to be verified since the same behaviour and functionality are expected.

To produce a speed-torque diagram, both motion profiles are going to be run at a setup emulating a real life application, more exactly the setup in figure 36 and figure 37. The dark grey part is the servo motor, the blue part is the gearbox and the grey box with the belt is the conveyor belt rig. On the belt a rectangular piece of aluminium is mounted to have a better view of how the belt is moving.

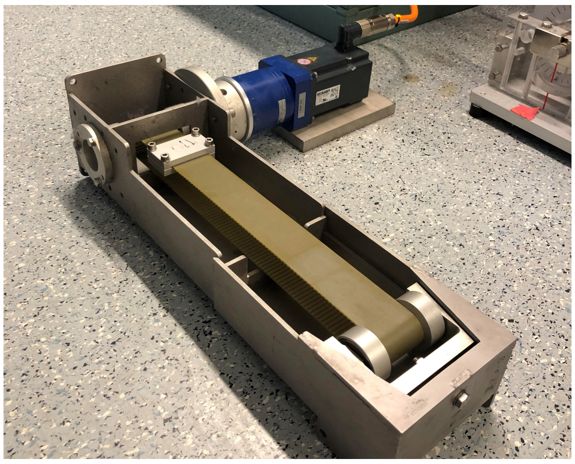


Figure 36: This figure shows the setup for the verification from the belt side

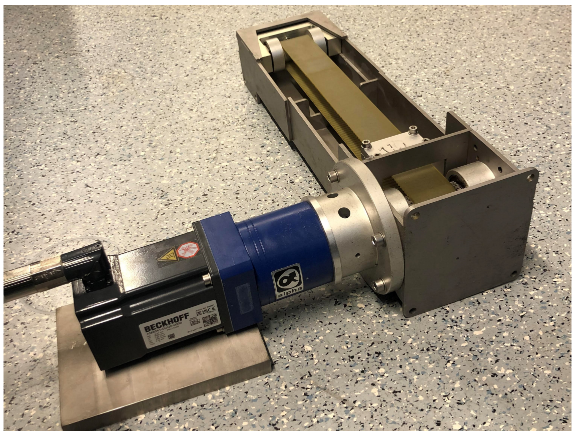


Figure 37: This figure shows the setup for the verification from the servo motor side

5 Results

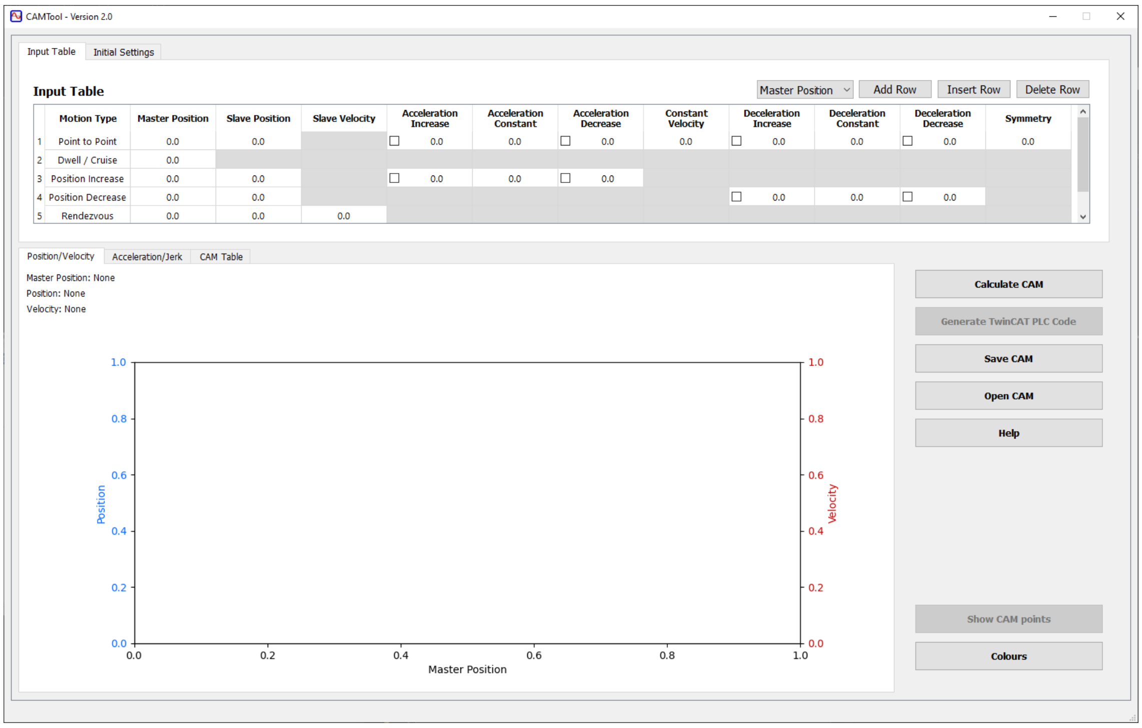
In this chapter the result of the work done during this project will be presented. First out is how the Python software turned out and its different parts, followed by the result of the TwinCAT software. In addition to that the result of adapting the software to real time is described. This chapter ends with a section about the results when a motion profile from the new version CAMTool was compared with a motion profile from Beckhoff's own CAM design tool.

5.1 Python Implementation

This section is divided in a similar way as the requirement chapters, 1 and 3. First out is the GUI, more exactly how the user interface and the colour palette turned out. This section is followed by the result of the non-zero jerk, Velocity Increase and Decrease as well as Position Increase and Decrease. The last section shows how the implementation of Rendezvous turned out.

5.1.1 Graphical User Interface

The new GUI of the software can be seen in figures 38, 39 and 40, where the first one shows the tab Input Table, the second one shows the tab Initial Settings and the third one shows the tab CAM Table. As can be seen in the input table there are already some motion types added. This is not the default, but to be able to show the result of how the input interface of some different motion types look when added. It is here that the engineer will input the acceleration and deceleration parameters when designing a new motion profile.

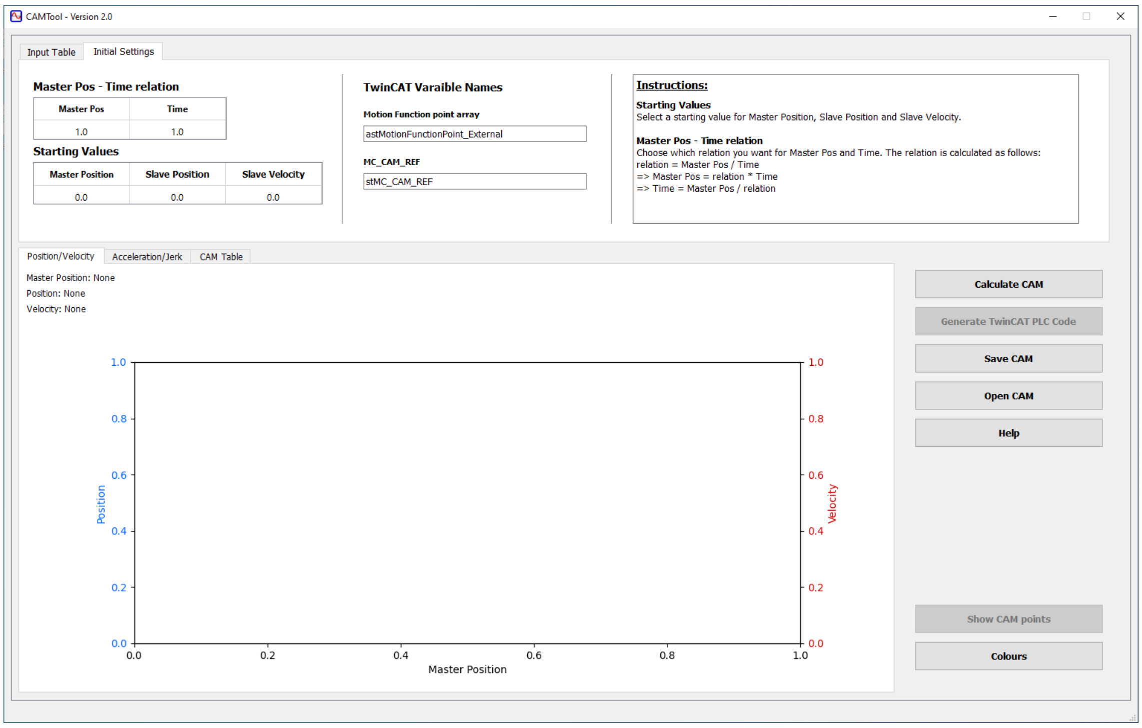


Input Table

	Motion Type	Master Position	Slave Position	Slave Velocity	Acceleration Increase	Acceleration Constant	Acceleration Decrease	Constant Velocity	Deceleration Increase	Deceleration Constant	Deceleration Decrease	Symmetry
1	Point to Point	0.0	0.0		☐ 0.0	0.0	☐ 0.0	0.0	☐ 0.0	0.0	☐ 0.0	0.0
2	Dwell / Cruise	0.0										
3	Position Increase	0.0	0.0		☐ 0.0	0.0	☐ 0.0					
4	Position Decrease	0.0	0.0						☐ 0.0	0.0	☐ 0.0	
5	Rendezvous	0.0	0.0	0.0								

Master Position: None
Position: None
Velocity: None

Figure 38: This figure shows the new GUI of Input table in the software



Master Pos - Time relation

Master Pos	Time
1.0	1.0

Starting Values

Master Position	Slave Position	Slave Velocity
0.0	0.0	0.0

TwinCAT Varaible Names

Motion Function point array: astMotionFunctionPoint_External

MC_CAM_REF: stMC_CAM_REF

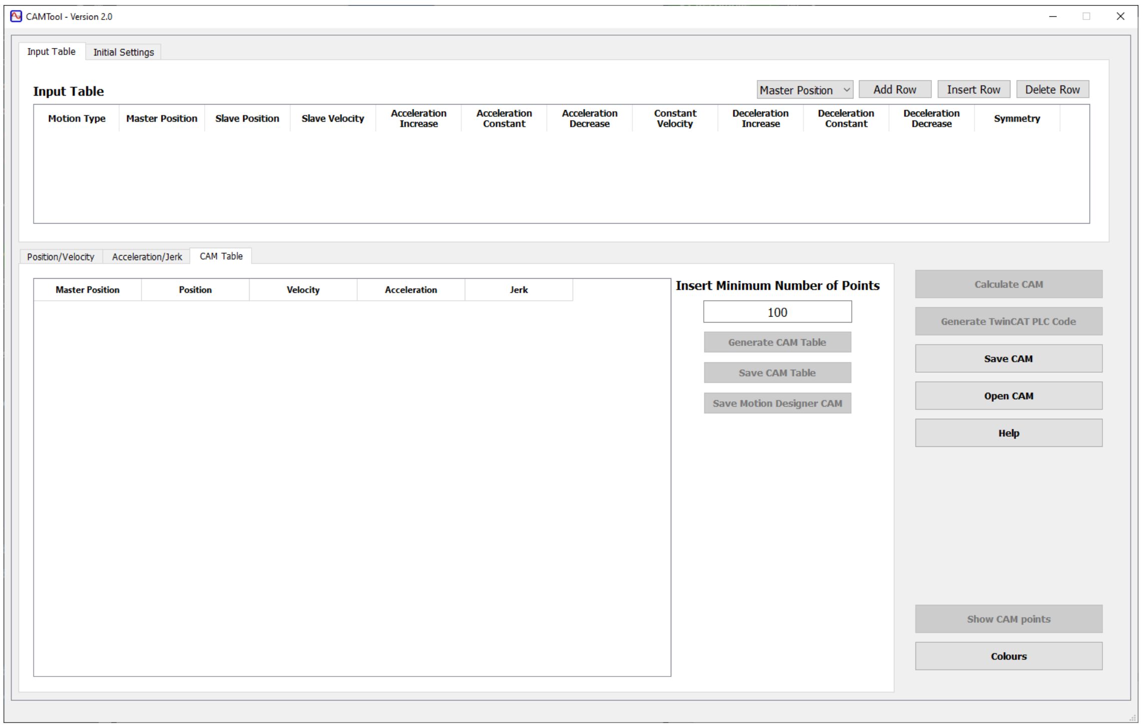
Instructions:

Starting Values
Select a starting value for Master Position, Slave Position and Slave Velocity.

Master Pos - Time relation
Choose which relation you want for Master Pos and Time. The relation is calculated as follows:
relation = Master Pos / Time
=> Master Pos = relation * Time
=> Time = Master Pos / relation

Master Position: None
Position: None
Velocity: None

Figure 39: This figure shows the new GUI Initial Settings tab



Input Table

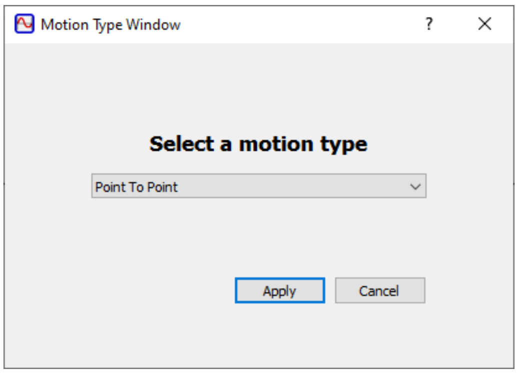
Motion Type	Master Position	Slave Position	Slave Velocity	Acceleration Increase	Acceleration Constant	Acceleration Decrease	Constant Velocity	Deceleration Increase	Deceleration Constant	Deceleration Decrease	Symmetry

Master Position	Position	Velocity	Acceleration	Jerk

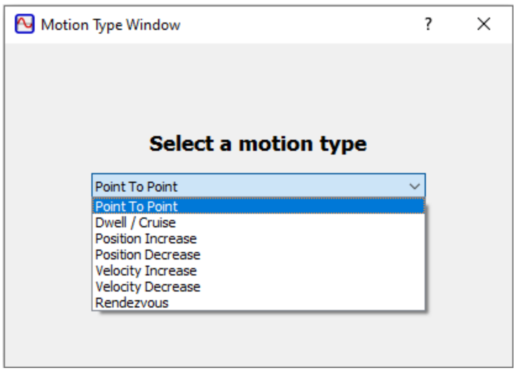
Insert Minimum Number of Points: 100

Figure 40: This figure shows the new GUI CAM Table tab

Both the Add Row and Insert Row buttons will make a pop-up window appear where the motion profile designer can choose which motion type to use. This pop-up window is shown in figure 41. Figure 41 (a) shows the pop-up window as it is when it first shows up, with Point to Point motion as the default motion type, while figure 41 (b) shows then the extended drop down list with the other motion types.



(a)



(b)

Figure 41: This figure is showing the Motion Type Window as default in (a) and when the drop down list is extended (b)

The button Colours in figures 38 and 39 will make a pop-up window appear where the designer can choose which colour position, velocity, acceleration and jerk will have. This pop up window can be seen in figure 42.

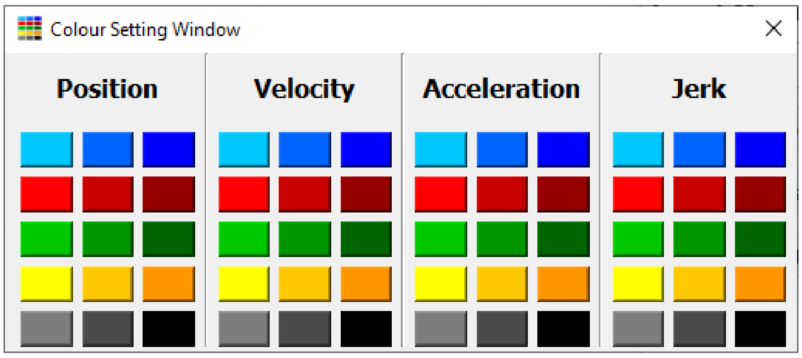


Figure 42: This figure shows the colour palette of the new GUI

5.1.2 Application

The overall architecture of the Python application was not in need of any big changes to be able to produce the new motion profiles. This means that the picture of the architecture in figure 18 still stands. However, some features in the application have been added to be able to calculate the new motion profiles.

One of these features is to be able to calculate a motion profile with non-zero jerk at the beginning of acceleration increase, end of acceleration increase, start of deceleration increase and end of deceleration increase. This was done by adding additional constrains to the function that calculated the acceleration splines, as described in the subsection 2.4.2 about jerk The constraints consisted of limitations on the derivative of jerk, it had to be zero which meant that the jerk could be a positive or negative value instead of just zero. If the constraint is that the derivative of the jerk is to be zero, then the jerk will have its maximum or minimum value at that point. However, if instead the jerk has to be zero, then the acceleration will have it is minimum or maximum value at that point. More about this in subsection 5.1.3.

Two other features that were added are Velocity Increase and Velocity Decrease, which were derived by firstly deriving an acceleration profile with the correct specifications according to the input parameters regarding acceleration or deceleration but with acceleration amplitude at one. Secondly, the correct amplitude were found by finding the quotient between the integral of the generated acceleration profile and the actual velocity change, see equation 20 and equation 21, respectively. The summation in the denominator is for summing the integral of the splines between the acceleration points, as can be seen in figure 44 for the acceleration and jerk points and in figure 43 for the corresponding points for position and velocity.

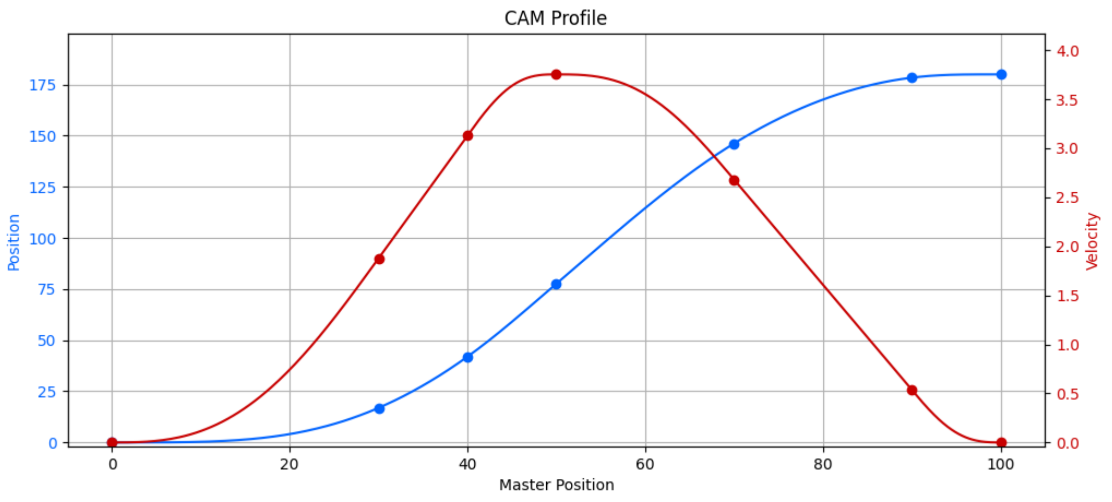


Figure 43: This figure shows the velocity (red) and position (blue) of a motion profile with acceleration points and deceleration points

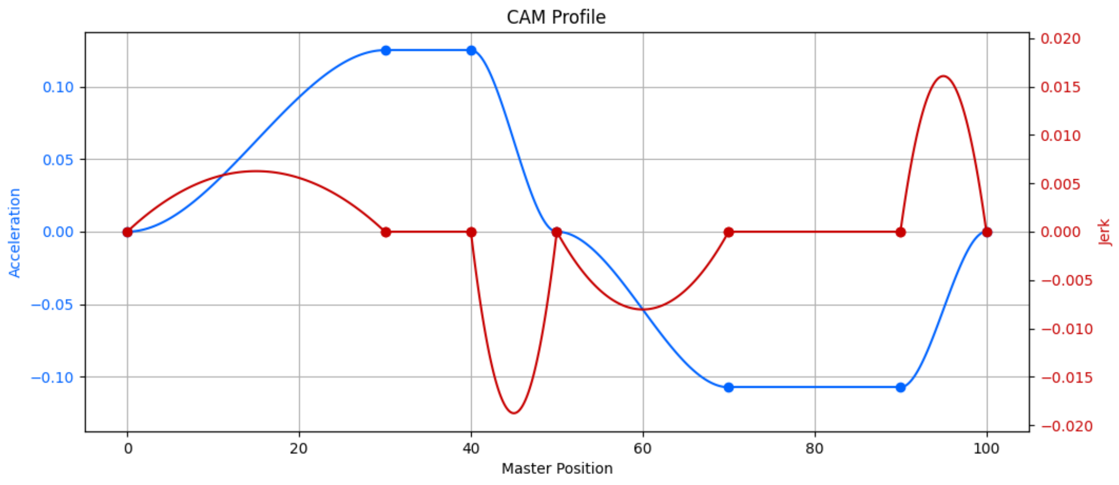


Figure 44: This figure shows the acceleration (blue) and jerk (red) of a motion profile with acceleration points and deceleration points

$$\text{acceleration amplitude} = \frac{f_{velocity}(x_{start}) - f_{velocity}(x_{end})}{\sum \int f_{acceleration}(x)dx} \tag{20}$$

$$\text{deceleration amplitude} = \frac{f_{velocity}(x_{start}) - f_{velocity}(x_{end})}{\sum \int f_{acceleration}(x)dx} \tag{21}$$

Two other features that were added are Position Increase and Position Decrease. The motion profile for these two motion types were calculated almost the same way as for Velocity Increase and Velocity Decrease. The difference is how to calculate the correct integral of the position that should be covered, since the motion profile can begin with an initial velocity not equal zero and thus the acceleration part has to be bigger or smaller than the actual distance specified in the input table. The acceleration and deceleration amplitudes for Position Increase and Position Decrease can be seen in equation 22 and equation 23, respectively.

$$\text{acceleration amplitude} = \frac{f_{position}(x_{start}) - f_{position}(x_{end}) - f_{velocity}(x_{start}) * (x_{end} - x_{start})}{\sum \iint f_{acceleration}(x)dx^2} \tag{22}$$

$$\text{deceleration amplitude} = \frac{f_{position}(x_{start}) - f_{position}(x_{end}) + f_{velocity}(x_{start}) * (x_{end} - x_{start})}{\sum \iint f_{acceleration}(x)dx^2} \tag{23}$$

5.1.3 Non-Zero Jerk

One of the criteria for this project was to have a non-zero jerk value in four different positions of the motion profile. The first position is at the beginning of the acceleration increase, the second position is at the end of acceleration decrease, the third position is at the beginning of the deceleration increase and the fourth and final position is at the end of deceleration decrease. This was introduced in subsection 3.3.1, in figures 23, 24, 25 and 26. It turned out to be solvable where the designer can choose in the GUI where the jerk should be non-zero by checking the correct check boxes in the input table and using the equations stated in 2.4.2. The four positions where it is possible to define different jerk values corresponds to the same position in the input table, i.e. the position at the start of the increasing part of the acceleration motion corresponds to the checkbox in acceleration increase box and the position at the end of the decreasing part of the acceleration motion corresponds to the checkbox in the acceleration decrease box. Consequently, the same for the deceleration part of the motion.

The result can be seen in figures 45, 46, 47 and 48 which are exactly the same motion profile as in figures 23, 24, 25 and 26 but with the sought after jerk and acceleration behaviour. More than at one position for the jerk can be non-zero at a time and actually all four can be non-zero at once if the designer wants to, see figure 49. It is also possible to mix the jerk options in any possible combination between the four different positions, for example only the two first, or the last two even though those options are not shown in the pictures.

In figure 45 the derivative of jerk is zero at the beginning of the acceleration increase motion and the jerk is zero at the end of acceleration decrease, at the beginning of deceleration increase and at the end of deceleration decrease.

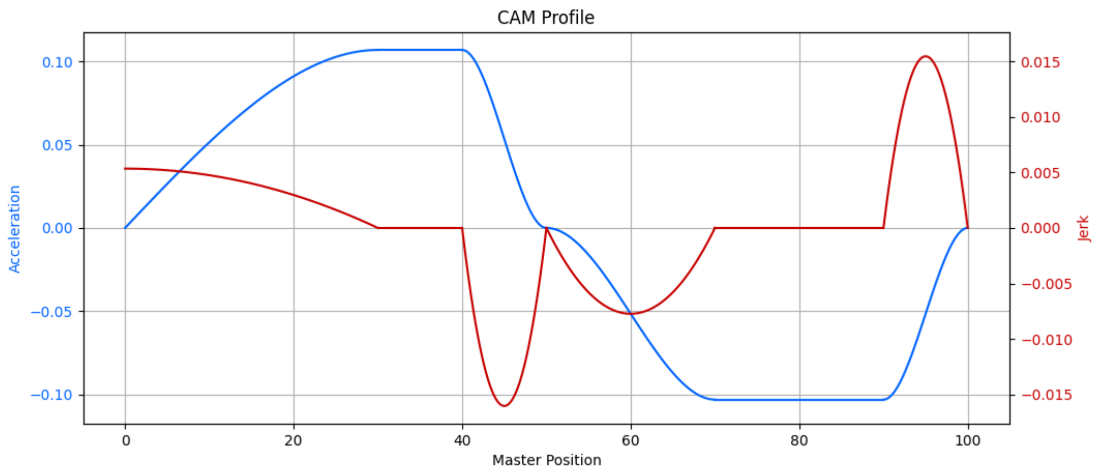


Figure 45: This figure shows the motion profile for acceleration (in blue) and jerk (in red) with a non-zero jerk value at the beginning of acceleration increase

In figure 46 the derivative of the jerk is zero at the end of the acceleration decrease motion and the jerk is zero at the beginning of acceleration increase, at the beginning of deceleration increase and at the end of deceleration decrease.

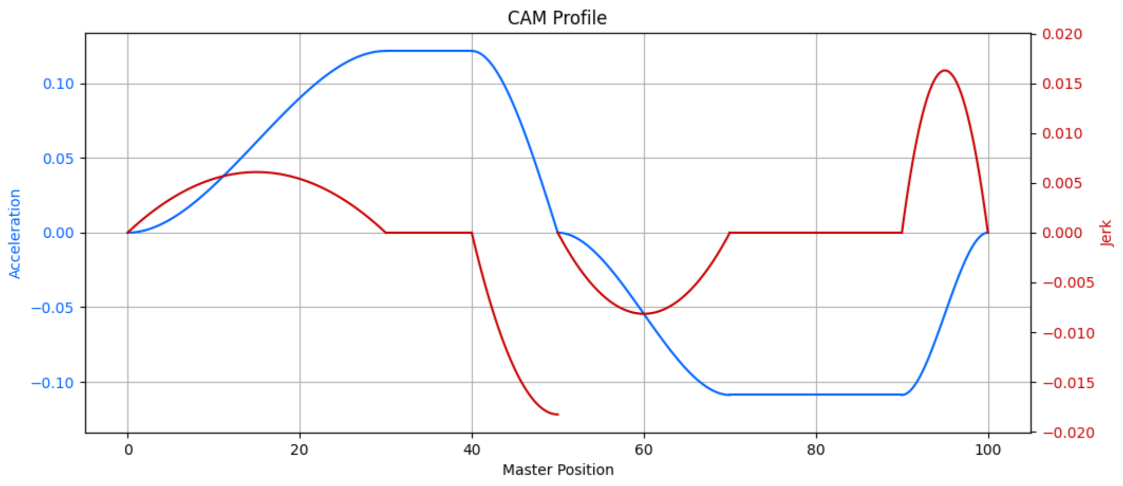


Figure 46: This figure shows the motion profile for acceleration (in blue) and jerk (in red) with a non-zero jerk value at the end of acceleration decrease

In figure 47 the derivative of the jerk is zero at the beginning of deceleration increase and the jerk is zero at the beginning of acceleration increase, at the end of the acceleration decrease motion and at the end of deceleration decrease.

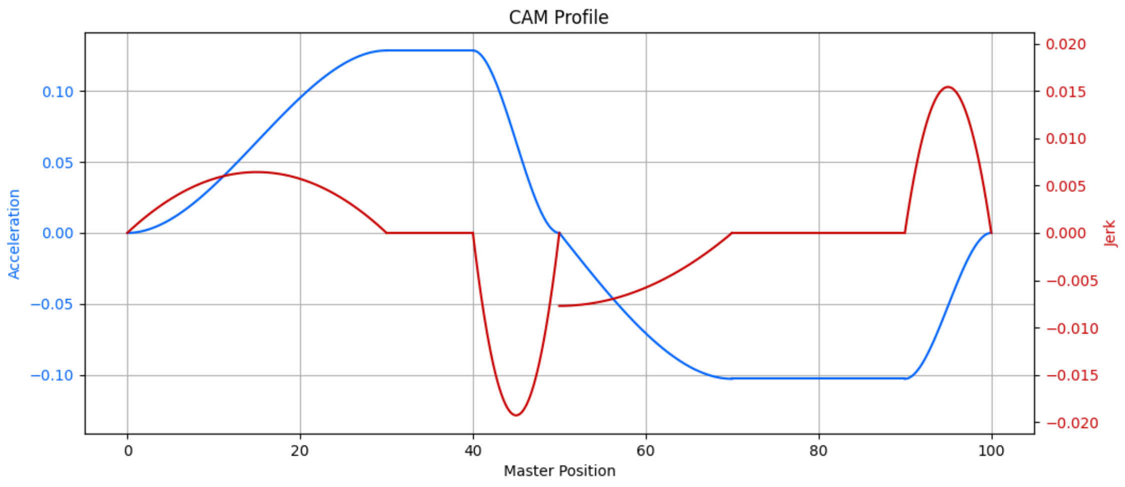


Figure 47: This figure shows the motion profile for acceleration (in blue) and jerk (in red) with a non-zero jerk value at the start of deceleration increase

In figure 48 the derivative of the jerk is zero at the end of the deceleration decrease motion and the jerk is zero at the beginning of acceleration increase, at the end of acceleration increase and at the beginning of deceleration increase.

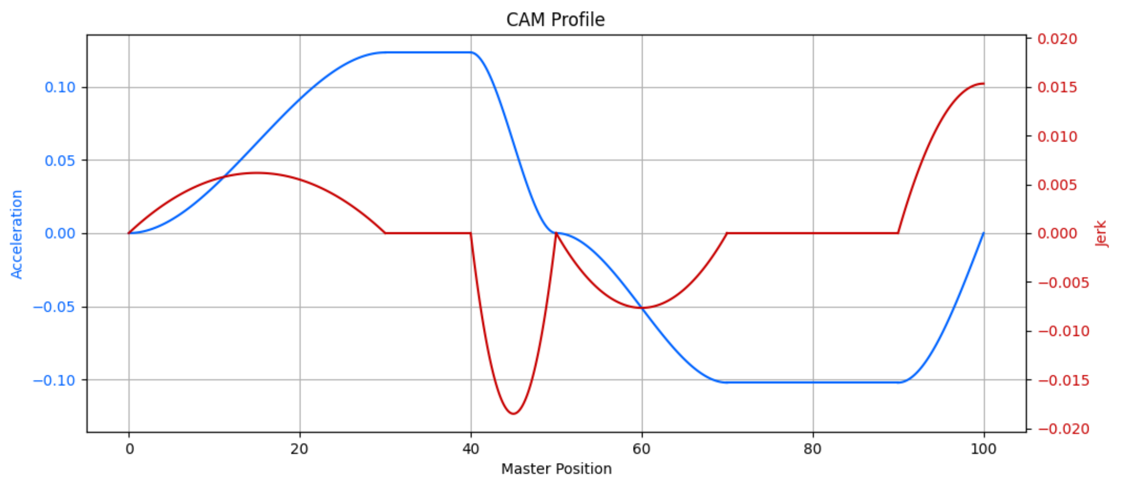


Figure 48: This figure shows the motion profile for acceleration (in blue) and jerk (in red) with a non-zero jerk value at the end of deceleration decrease

In figure 49 the derivative of the jerk is zero at the beginning of acceleration increase, at the end of the acceleration decrease motion, at the beginning of deceleration increase and at the end of deceleration decrease.

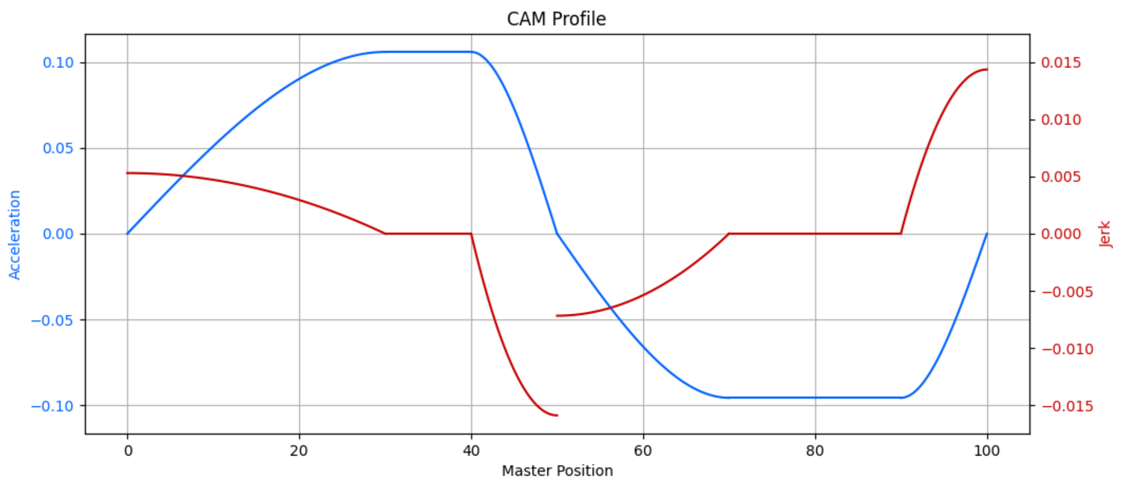


Figure 49: This figure shows the motion profile for acceleration (in blue) and jerk (in red) with a non-zero jerk value at all four possible places

5.1.4 Velocity Increase and Velocity Decrease

Another criteria for this thesis was to be able to change the velocity independent of the slave position. This means to increase the velocity from one value to another value or decrease the velocity from one value to another value. Those velocities can all be non-zero, both the initial velocity of the motion profile as well as the end velocity. This type of motion function, which we have decided to call Velocity Increase and Velocity Decrease respectively, was introduced in subsection 3.3.1 in figures 27 and 29 with its corresponding acceleration and jerk behaviour in figures 28 and 30. In the final version of the software the designer can choose via the GUI which velocity to start with and what the end velocity should be. The result can be seen in figures 50, 51, 52 and 53, which are exactly the same motion profiles as in figures 27, 28, 29 and 30 but with the sought after behaviour presented in subsection 3.3.1.

In figure 50 the Velocity Increases from 0 to 2, while the position goes from 0 to 100 and in figure 52 the Velocity Decreases from 0 to 2 and the position goes from 0 to 100.

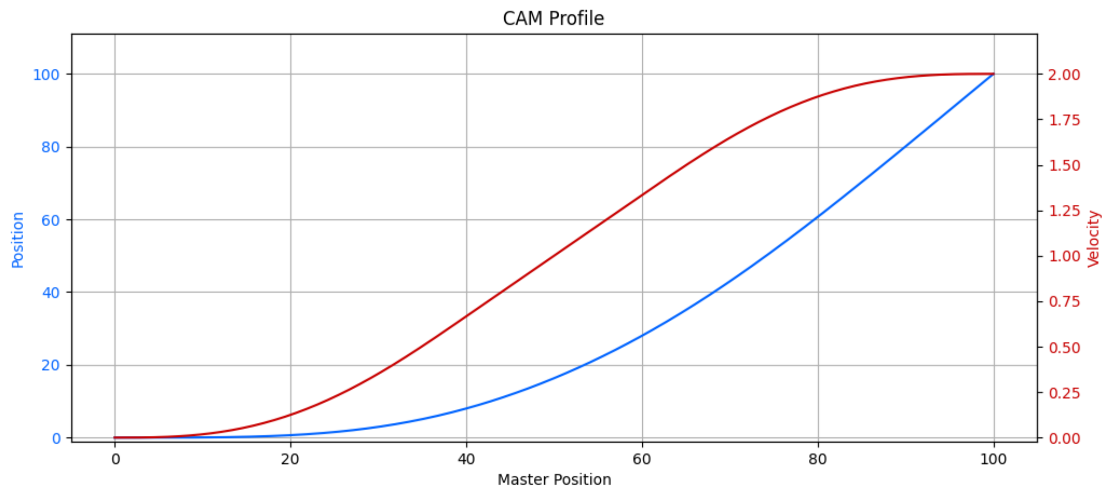


Figure 50: This figure shows a motion profile for position (in blue) and velocity (in red) of a Velocity Increase motion type

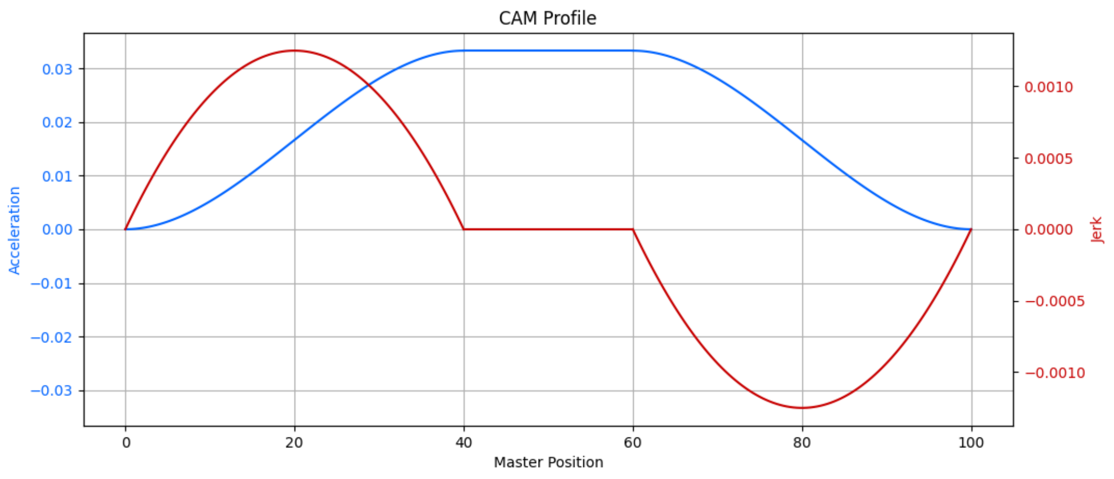


Figure 51: This figure shows a motion profile for acceleration (in blue) and jerk (in red) of figure 50

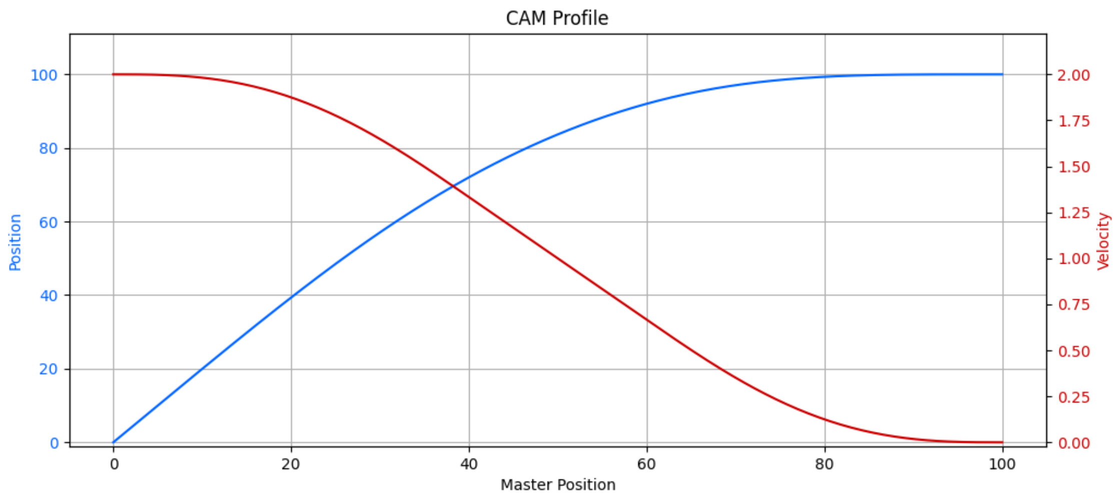


Figure 52: This figure shows a motion profile for position (in blue) and velocity (in red) of a Velocity Decrease motion type

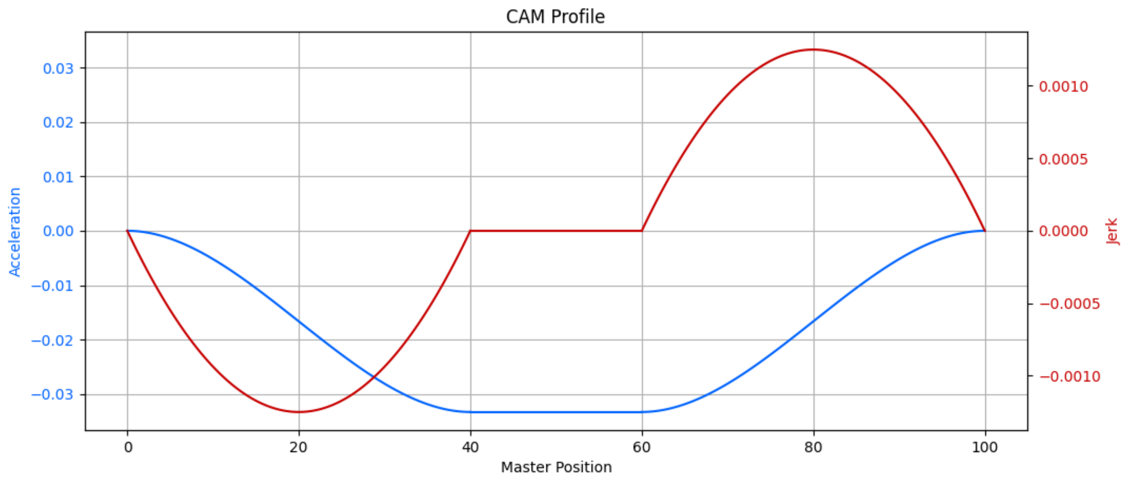


Figure 53: This figure shows a motion profile for acceleration (in blue) and jerk (in red) of figure 52

5.1.5 Position Increase and Position Decrease

An additional requirement added later during this project was to let the designer change the position by defining the start velocity, the start position, the end position and the acceleration behaviour. The start position has to be smaller than the end position when a Position Increase is created, and the opposite has to be true for Position Decrease. The software would then calculate the acceleration amplitude so the end position would be achieved, without any constraints on the end velocity, as can be seen in figure 54 where the motion starts in position zero and ends in position 180 with a start velocity of zero and an end velocity of over three. The acceleration behaviour as well as the jerk can be seen in figure 55.

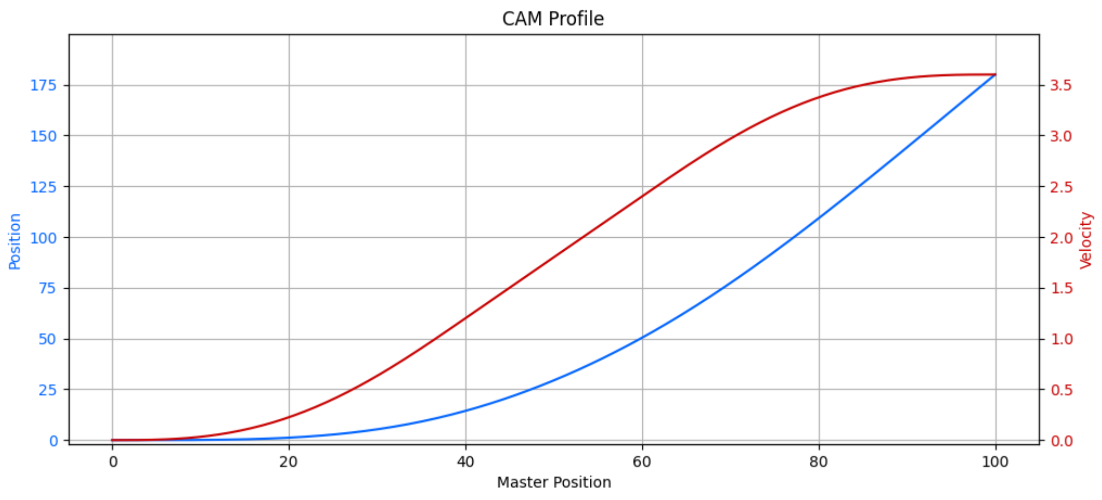


Figure 54: This figure shows a motion profile for position (in blue) and velocity (in red) of a Position Increase motion type

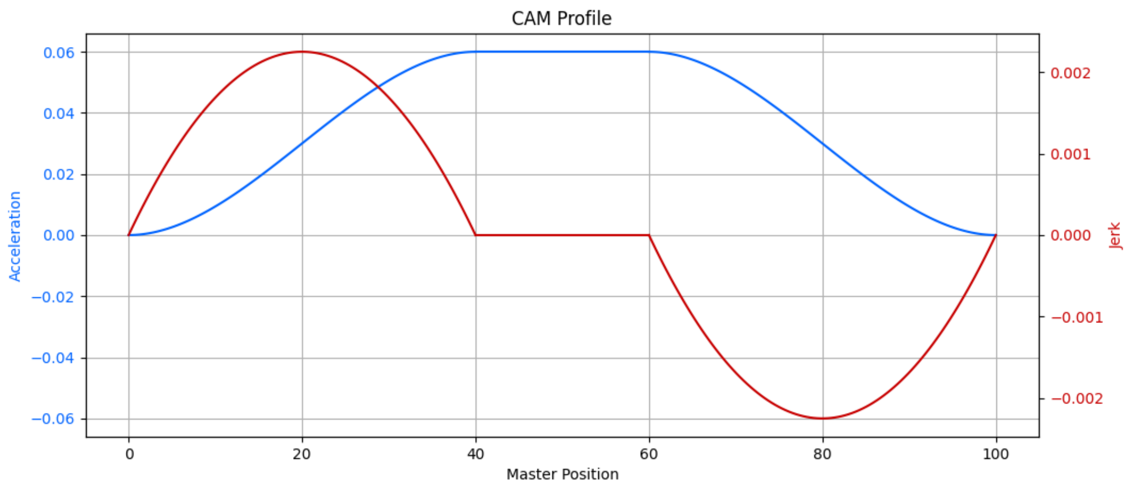


Figure 55: This figure shows a motion profile for acceleration (in blue) and jerk (in red) of figure 54

The opposite behaviour from Position Increase holds for Position Decrease regarding the start and end positions. The end position has to be smaller than the start position for a Position Decrease motion type. Like Position Increase the designer should define the start velocity, the start position, the end position and the acceleration behaviour. The software would then calculate the acceleration amplitude so the end position would be achieved without any constraints on the end velocity. As can be seen in figure 56 where the motion starts in position 180 and ends in position zero with a start velocity of zero and an end velocity of smaller than negative three. The acceleration behaviour as well as the jerk behaviour can be seen in figure 57.

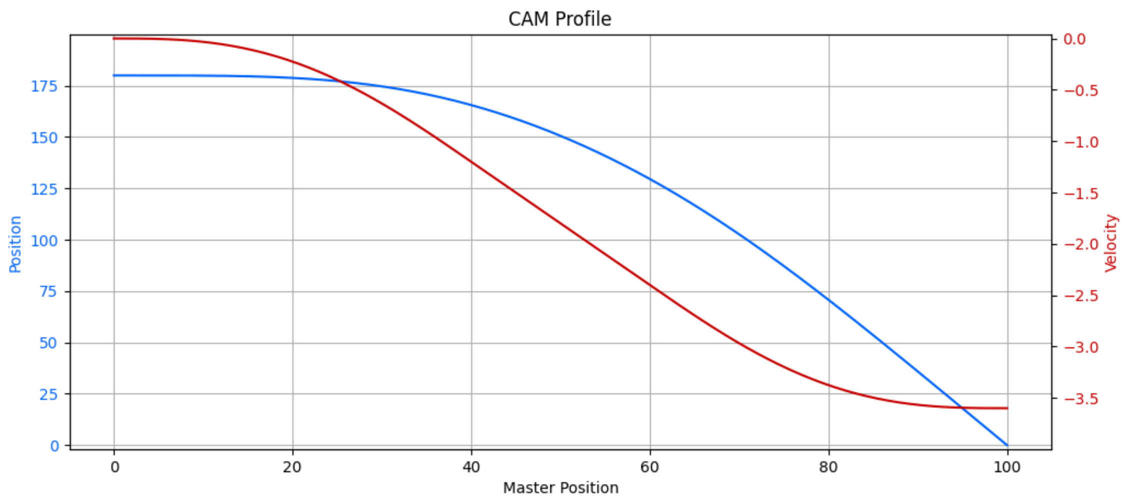


Figure 56: This figure shows a motion profile for position (in blue) and velocity (in red) for a Position Decrease motion type

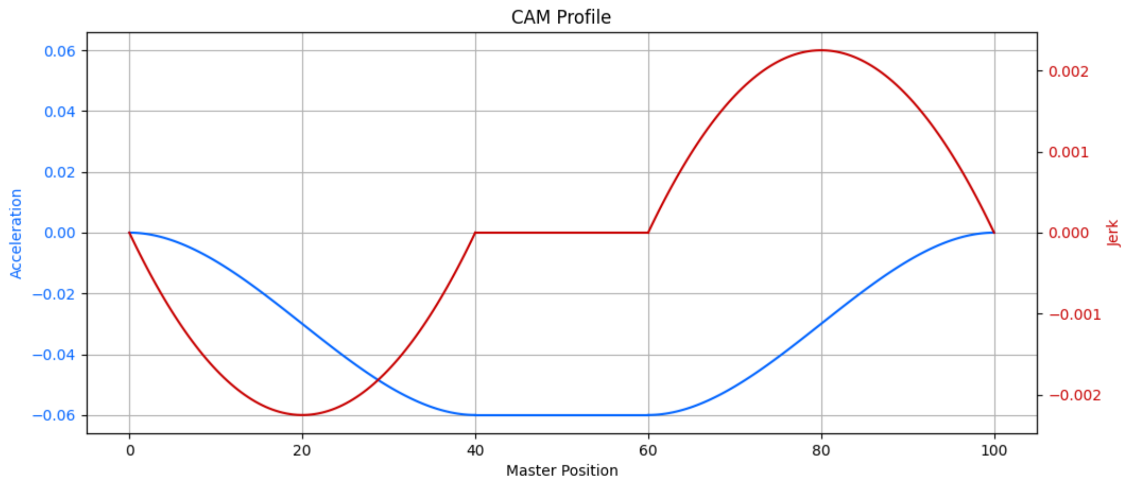


Figure 57: This figure shows a motion profile for acceleration (in blue) and jerk (in red) for figure 56

5.1.6 Rendezvous

Figure 58 shows the motion profile when moving from 0 to 180 with an initial velocity of zero and an end velocity of two. Master Position is from 0 to 100. Figure 59 shows the acceleration and jerk motion required to achieve the motion in figure 58 and are not derived from the parameters in the CAMTool.

The result, presented in the figures 58 and 59 is derived by using equations 17, 18 and 19 presented in subsection 3.3.4 and the Python function make_interp_spline presented in subsection 2.5.2.

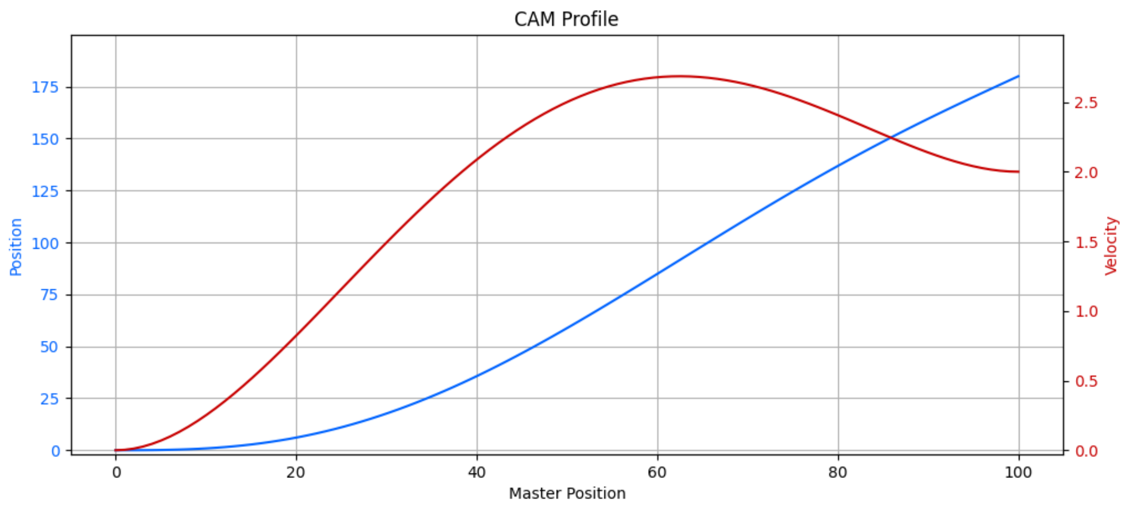


Figure 58: This figure shows a motion profile for position (in blue) and velocity (in red) for a Rendezvous motion type

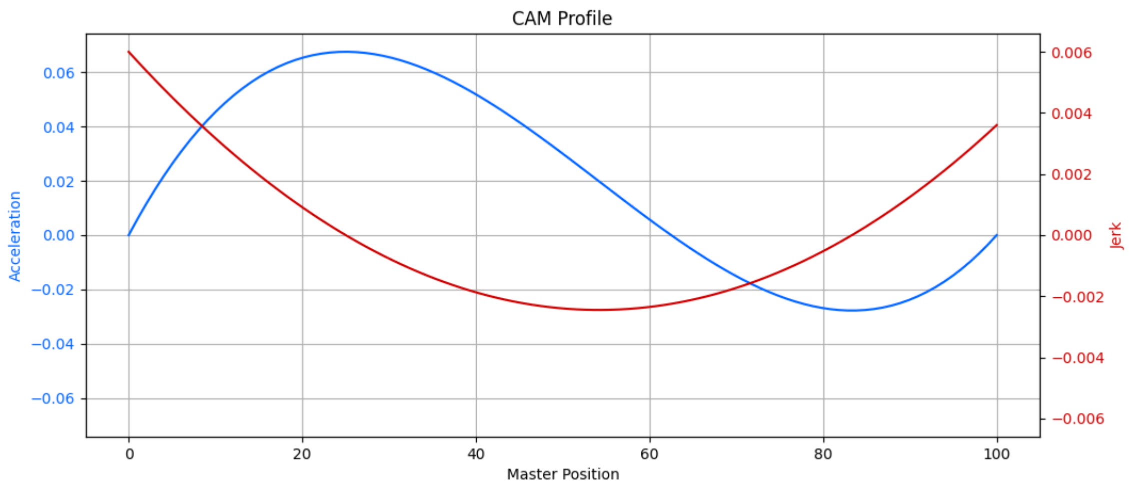


Figure 59: This figure shows a motion profile for acceleration (in blue) and jerk (in red) for figure 58

5.1.7 Dwell and Cruise

Figure 60 shows a motion profile for Dwell where the initial velocity and end velocity is zero. Since the velocity is constant the acceleration and jerk values are also zero. Figure 61 shows an example of a motion profile for Cruise where the initial velocity and the end velocity is the same value, in this case 2. Since the velocity is constant here as well the acceleration and jerk values are also zero. Figure 62 shows the acceleration and jerk behaviour for both Dwell and Cruise since the velocity is constant in both cases.

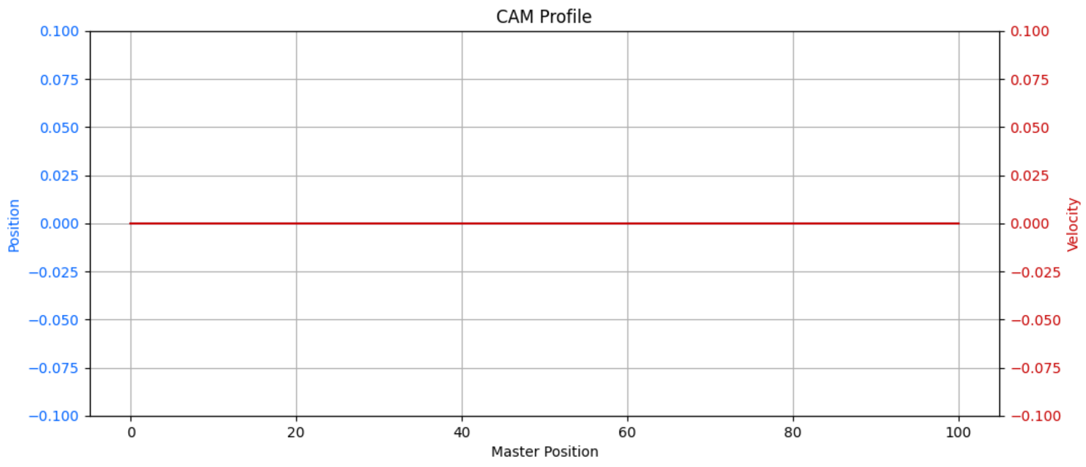


Figure 60: This figure shows a motion profile for Dwell. The velocity and position has the same value, 0, and are in red and blue

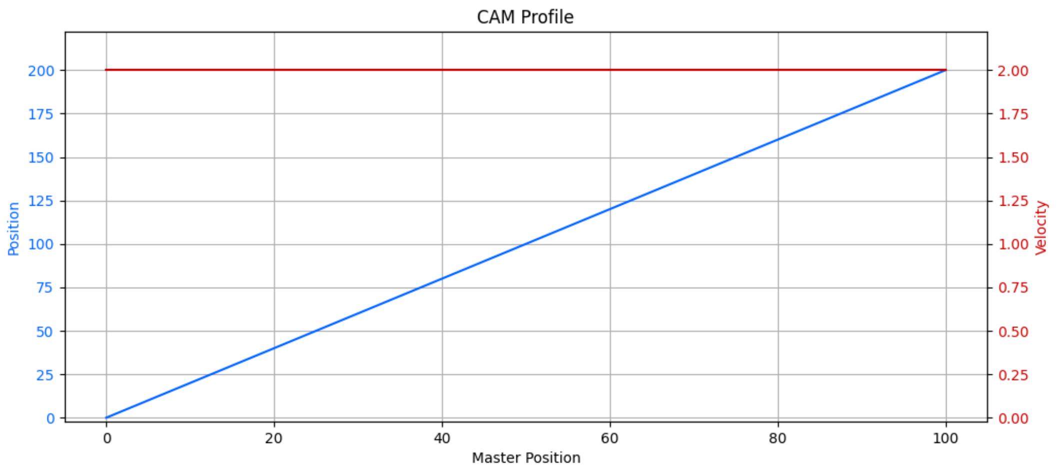


Figure 61: This figure shows a motion profile for Cruise. The velocity is in red and the position is in blue

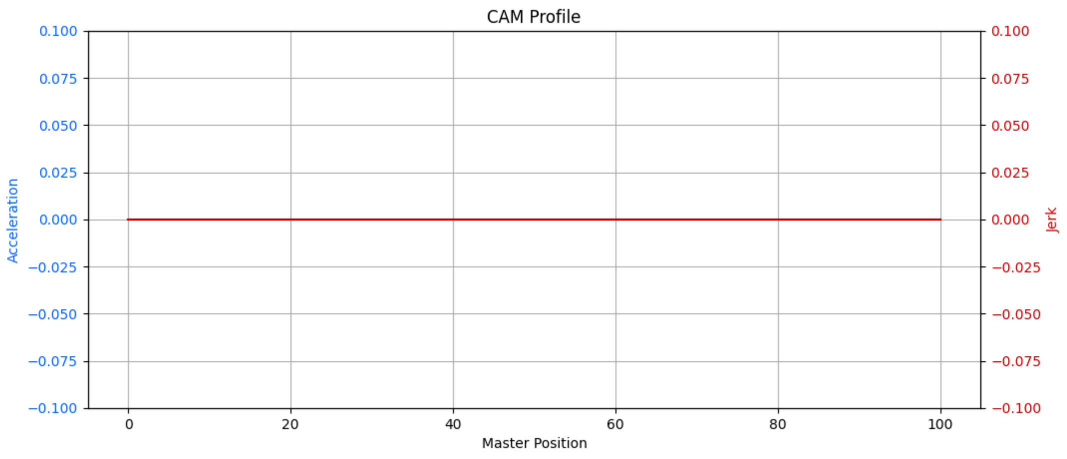


Figure 62: This figure shows the acceleration and jerk behaviour for Dwell and Cruise. Both are zero. Acceleration is in blue and jerk is in red

5.2 TwinCAT Implementation

In this section the result of the TwinCAT version of CAMTool will be presented. Firstly the application, where the new architecture is shown. The next subsection will go through the non-zero jerk results, followed by a subsection with the result of the velocity changes. This is followed by the position change result. The last subsection will show the result of the real time implementation.

5.2.1 Application

To introduce the new features, the architecture of the TwinCAT implementation had to be updated due to inaccuracy both in the numerical calculations of the integral of the acceleration and in the integral of the velocity. New function blocks were introduced, which either increased or decreased the acceleration points whether the end result was too large or too small in relation to the expected result. Figure 63, shows graphically what is happening to the acceleration points when moving the acceleration points. The function blocks adjust the acceleration points up and down. This will either decrease or increase the velocity of the motion and in turn increase or decrease the end position depending on the expected end result.

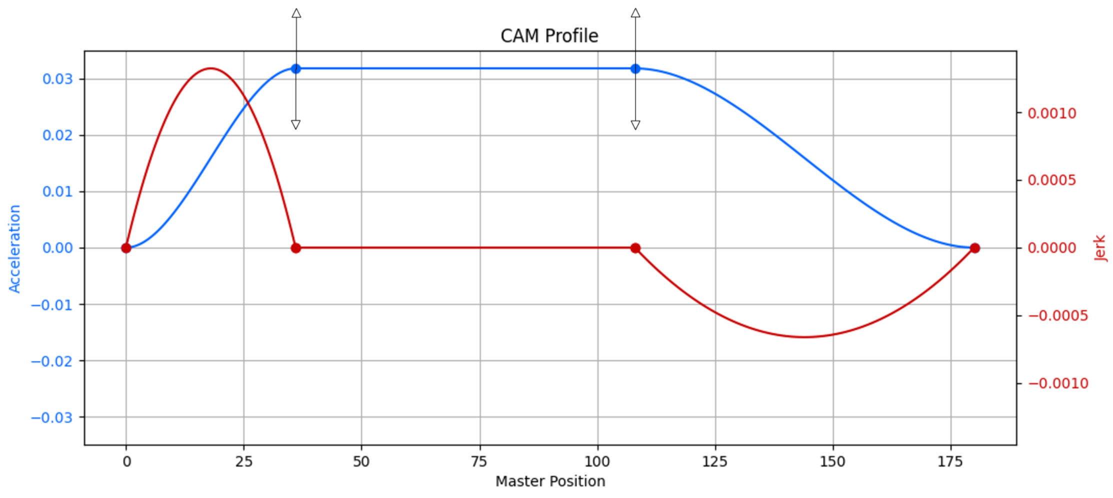


Figure 63: This figure illustrates what the algorithms does to the acceleration points when moving them up or down

The new function blocks can be seen in figure 64 and are FB_Recalculate, FB_Recalculate_VelIncDec and FB_Recalculate_PosIncDec. FB_Recalculate is used for adjusting the acceleration points for a Point to Point motion. FB_Recalculate_VelIncDec is used for adjusting the acceleration points for a Velocity Increase or a Velocity Decrease motion profile and Recalculate_PosIncDec is used for adjusting the acceleration points for a Position Increase and Position Decrease motion type. Depending on what motion type the motion is one of the parallel function blocks will be executed, which is shown in figure 64.

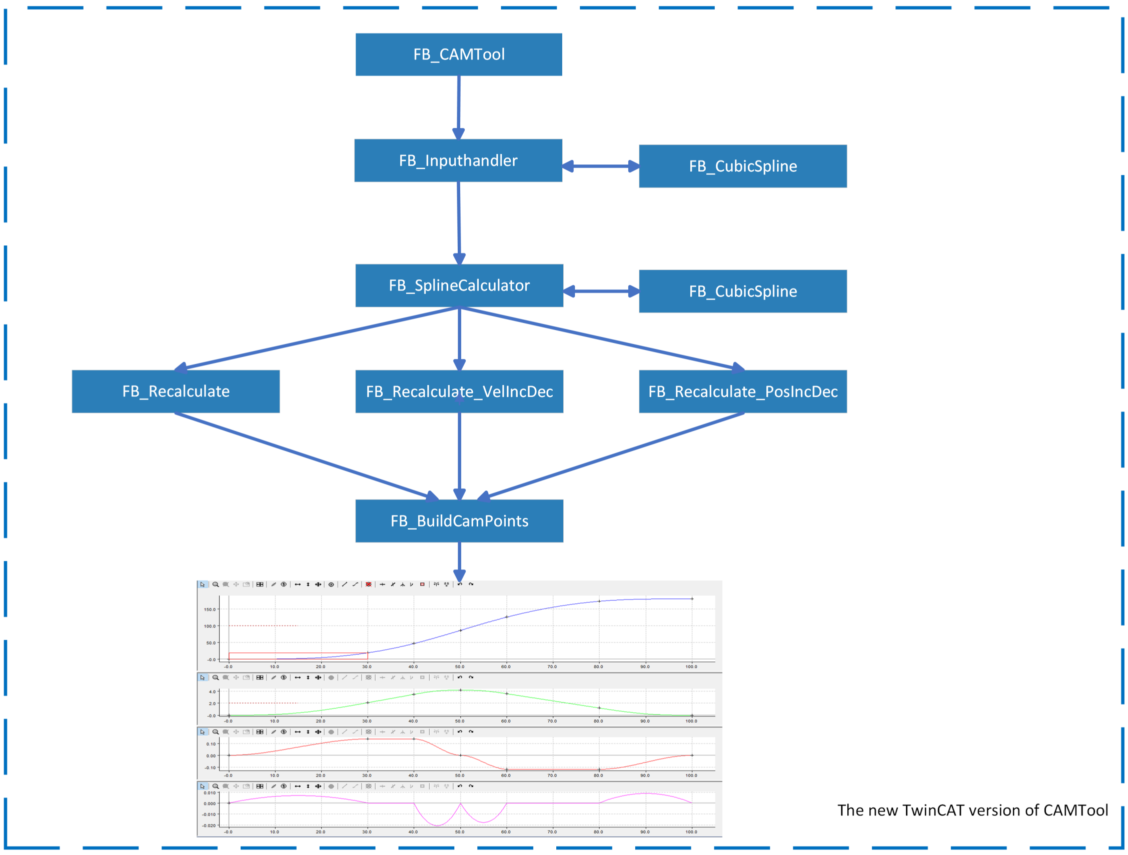


Figure 64: This figure shows the new architecture for the TwinCAT version of CAMTool

The algorithm for FB_Recalculate _VelIncDec and FB_Recalculate_PosIncDec is shown below. First the end value is calculated and compared with the predefined end value, here defined as slave position.

Algorithm 1 Algorithm for recalculate with focus on position

```
$endValue \leftarrow f_{position}(x_{end})$
$iterStep \leftarrow \frac{abs(endValue - slavePosition)}{(x_{end} - x_{start})^2}$
$k = 0$

while $abs(endValue - slavePosition) > 10^{-8} \wedge k \leq 1000$ do
    if $endValue \leq slaveposition$ then
        $accelerationpoints = accelerationpoints + iterstep$
    else
        $accelerationpoints = accelerationpoints - iterstep$
    end if
    recalculate the new motion profile
    $endValue \leftarrow f_{position}(x_{end})$
    $iterStep \leftarrow \frac{abs(endValue - slavePosition)}{(x_{end} - x_{start})}$
    $k++$
end while
```

Algorithm two focuses on minimising the difference between the required velocity and the current velocity.

Algorithm 2 Algorithm for recalculate with focus on velocity

```
$endValue \leftarrow f_{velocity}(x_{end})$
$iterStep \leftarrow \frac{abs(endValue - slaveVelocity)}{(x_{end} - x_{start})}$
$k = 0$

while $abs(endValue - slaveVelocity) > 10^{-8} \wedge k \leq 1000$ do
    if $endValue \leq slaveposition$ then
        $accelerationpoints = acccelerationpoints + iterstep$
    else
        $accelerationpoints = acccelerationpoints - iterstep$
    end if
    recalculate the new motion profile
    $endValue \leftarrow f_{velocity}(x_{end})$
    $iterStep \leftarrow \frac{abs(endValue - slaveVelocity)}{(x_{end} - x_{start})}$
    $k++$
end while
```

For FB_Recalculate the algorithm is slightly different since the end value of the position is always calculated by the CAMTool. Instead, the focus is on minimising the integral value. The following algorithm was used for that.

Algorithm 3 Algorithm for recalculate with focus of the integral of acceleration

```
$endValue \leftarrow \int f_{acceleration}(x)dx$
$iterStep \leftarrow \frac{abs(endValue)}{(x_{end} - x_{start})}$
$k = 0$

while $abs(endValue) > 10^{-8} \wedge k \leq 1000$ do
    if $endValue < 0$ then
        $accelerationpoints = acccelerationpoints + iterstep$
    else
        $accelerationpoints = acccelerationpoints - iterstep$
    end if
    recalculate the new motion profile
    $endValue \leftarrow \int f_{acceleration}(x)dx$
    $iterStep \leftarrow \frac{abs(E \int f_{acceleration}(x)dx)}{(x_{end} - x_{start})}$
    $k++$
end while
```

However, due to the accuracy of the input handler to calculate the correct acceleration amplitude (steps 1, 2 and 3 in figure 64) these function blocks do not need to iterate more than 10 times for Position Increase and Position Decrease and for the other motion types it is not necessary at all, which improves the performance of the program.

5.2.2 Non-Zero Jerk

One goal of this work was to implement motion profiles with non-zero jerk at four different positions in TwinCAT, which was presented in subsection 3.3.1 figures 23, 24, 25 and 26. The positions are at the beginning of acceleration increase, the end of acceleration decrease, the beginning of deceleration increase and the end at deceleration decrease. This is shown in figures 65, 66, 67, 68 and 69.

In figure 65 the derivative of the jerk is zero at the beginning of the acceleration increase motion and the jerk is zero at the end of acceleration decrease, at the beginning of deceleration increase and at the end of deceleration decrease.

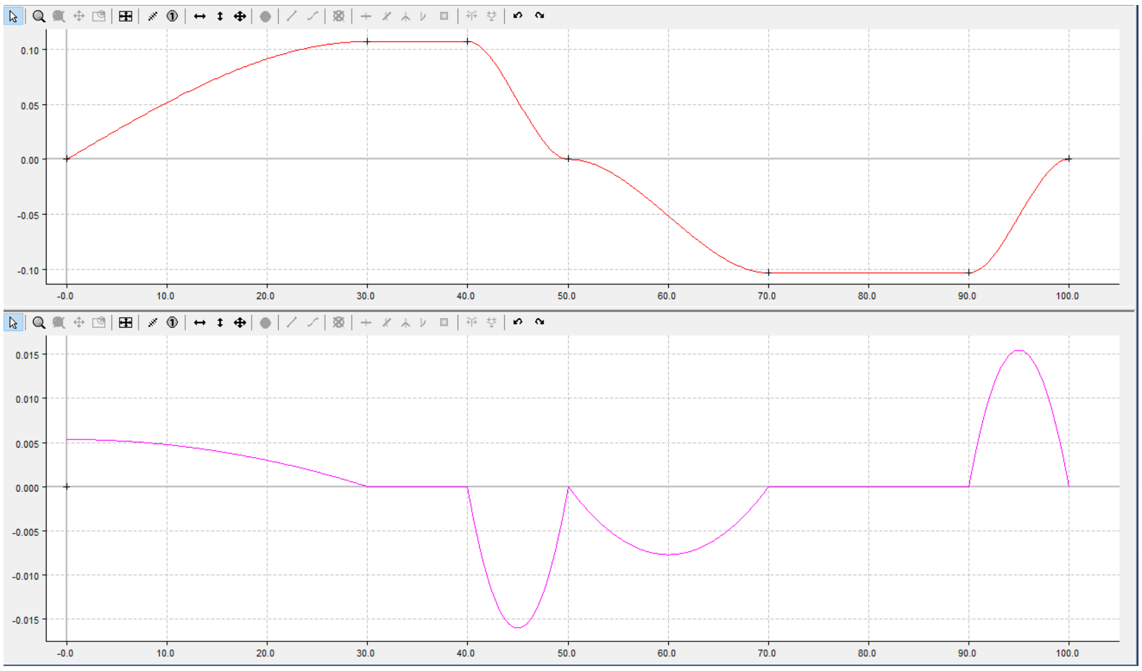


Figure 65: This figure shows a motion profile where the jerk is non-zero at the start of the acceleration increase motion in TwinCAT. The acceleration is in red, and jerk is in pink

In figure 66 the derivative of the jerk is zero at the end of the acceleration decrease motion and the jerk is zero at the beginning of acceleration increase, at the beginning of deceleration increase and at the end of deceleration decrease.

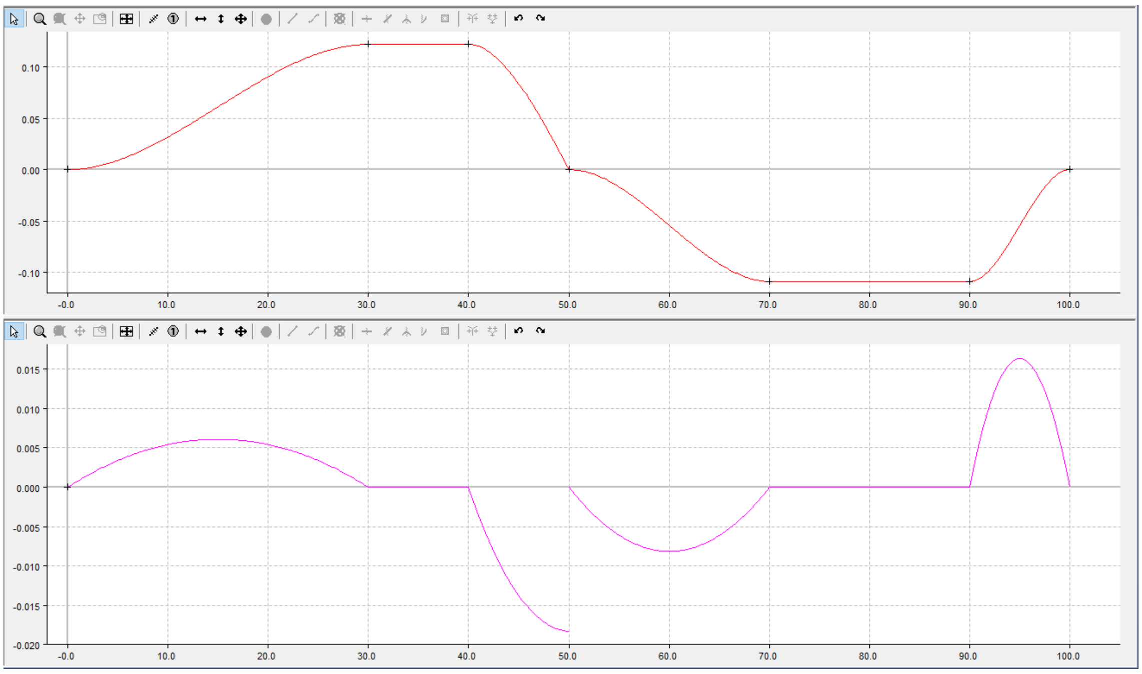


Figure 66: This figure shows a motion profile where the jerk is non-zero at the end of acceleration decrease motion in TwinCAT. The acceleration is in red, and jerk is in pink

In figure 67 the derivative of the jerk is zero at the beginning of deceleration increase and the jerk is zero at the beginning of acceleration increase, at the end of the acceleration decrease motion and at the end of deceleration decrease.

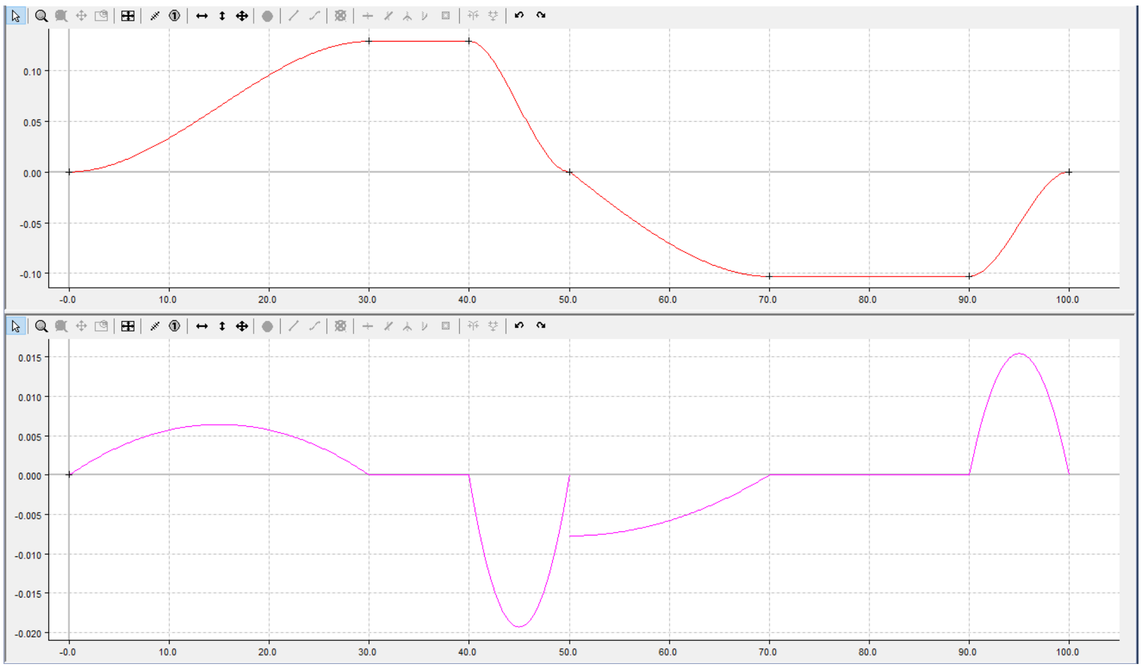


Figure 67: This figure shows a motion profile where the jerk is non-zero at the start of the deceleration increase motion in TwinCAT. The acceleration is in red, and the jerk is in pink

In figure 68 the derivative of jerk is zero at the end of the deceleration decrease motion and the jerk is zero at the beginning of acceleration increase, at the end of acceleration increase and at the beginning of deceleration increase.

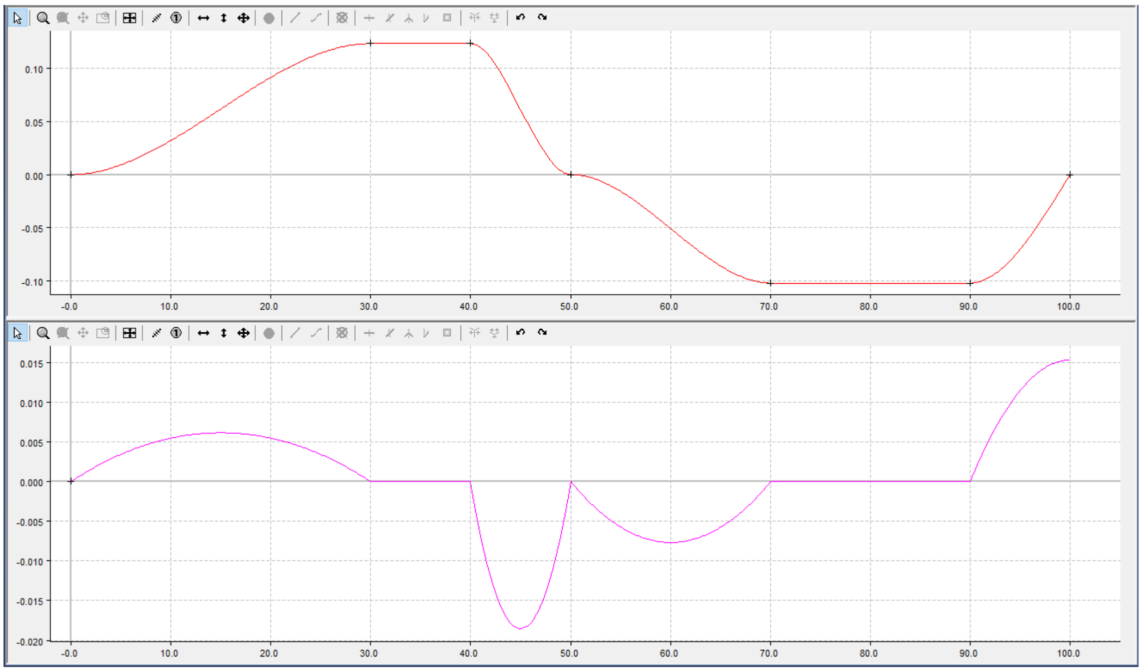


Figure 68: This figure shows a motion profile where the jerk is non-zero at the end of the deceleration decrease motion in TwinCAT. The acceleration is in red, and jerk is in pink

In figure 69 the derivative of the jerk is zero at the beginning of the acceleration increase, at the end of the acceleration decrease motion, at the beginning of deceleration increase and at the end of deceleration decrease.

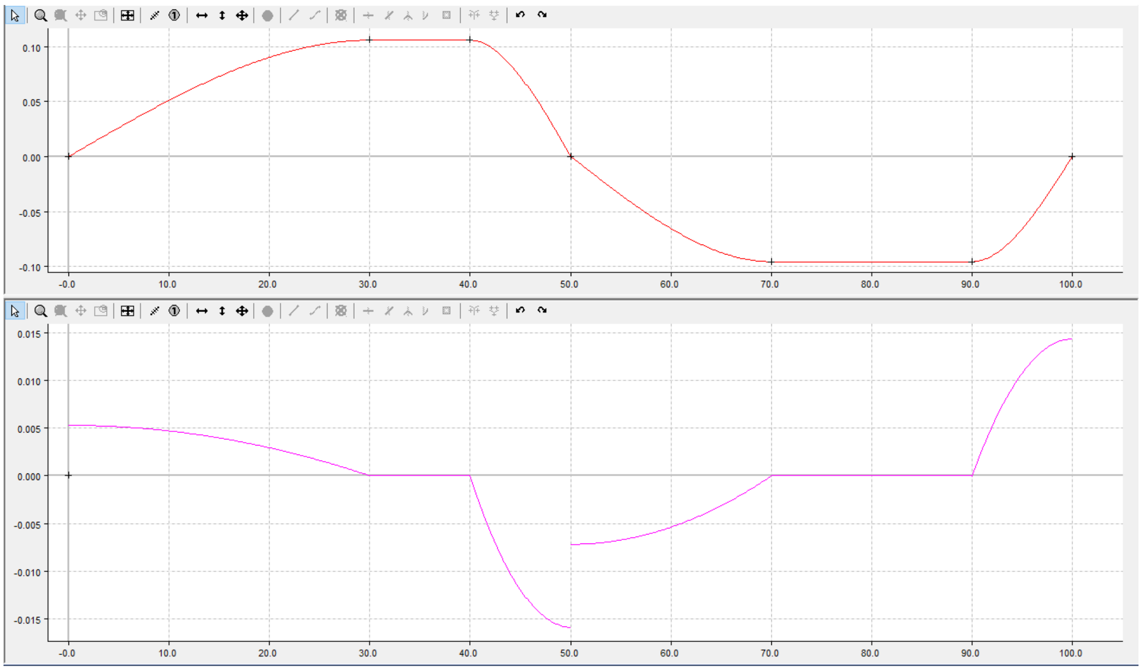


Figure 69: This figure shows a motion profile where all the four possible positions for jerk is non-zero. The acceleration is in red, and jerk is in pink

5.2.3 Velocity Increase and Decrease

Another aim was to implement a feature in the software that could calculate a motion profile which had a defined start velocity, start position and a defined end velocity. The results are shown in figures 70 and 71, where figure 70 shows the motion profile Velocity Increase and figure 71 shows the Velocity Decrease. For Velocity Increase the initial velocity has to be smaller than the final velocity and the opposite, initial velocity must be bigger than the final Velocity, is true for Velocity Decrease.

Figure 70 shows a motion profile where the velocity goes from 0 to 2 and figure 71 shows a motion profile where the velocity goes from 2 to 0.

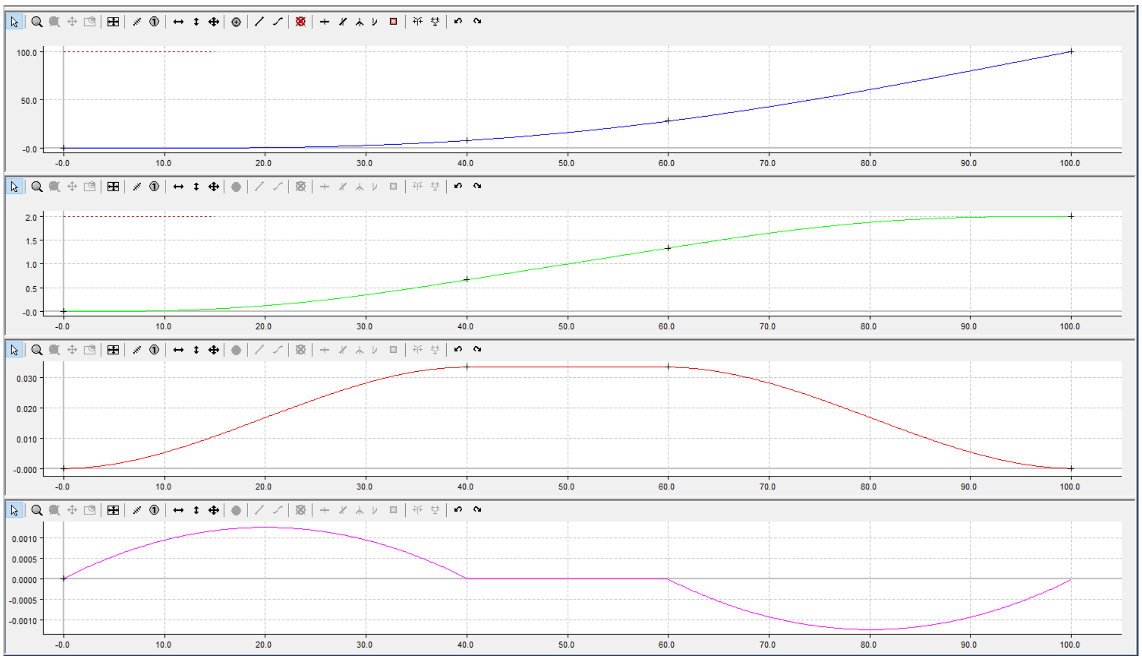


Figure 70: This figure shows a motion profile for position (in blue), velocity (in green), acceleration (in red) and jerk (in pink) for a Velocity Increase motion type in TwinCAT

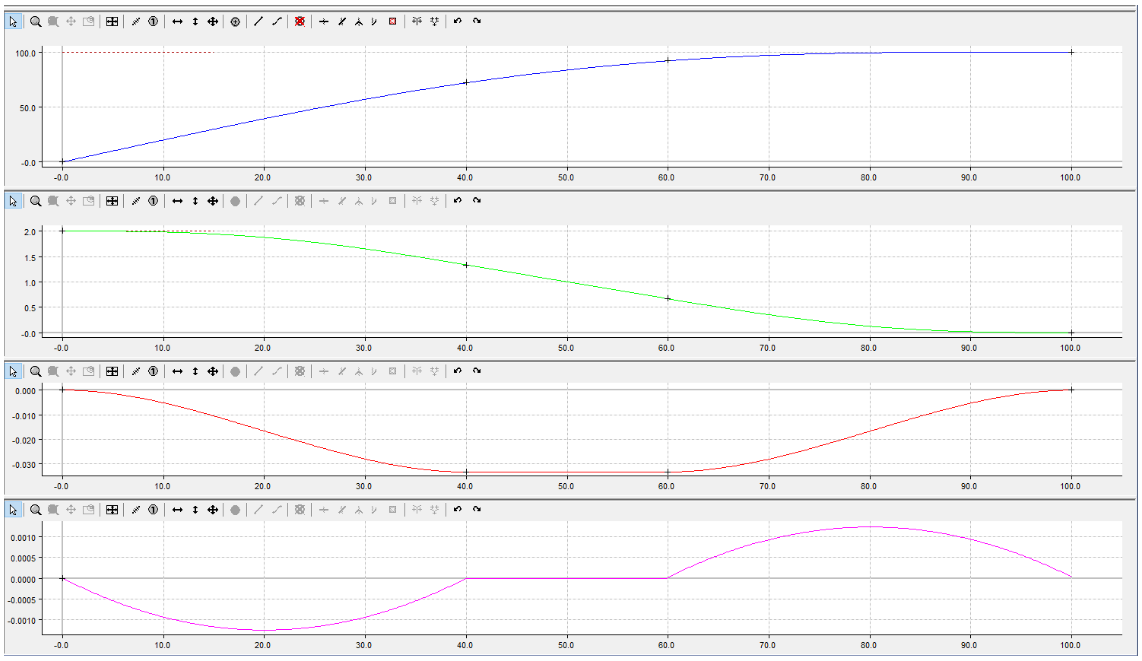


Figure 71: This figure shows a motion profile for position (in blue), velocity (in green), acceleration (in red) and jerk (in pink) for a Velocity Decrease motion type in TwinCAT

5.2.4 Position Increase and Decrease

An additional requirement was to calculate a motion profile, Position Increase, with a predefined start position, start velocity, end position and an acceleration behaviour. For Position Increase the end position must be larger than the start position. The result of this can be seen in figure 72, where the position has been increased from 0 to 180.

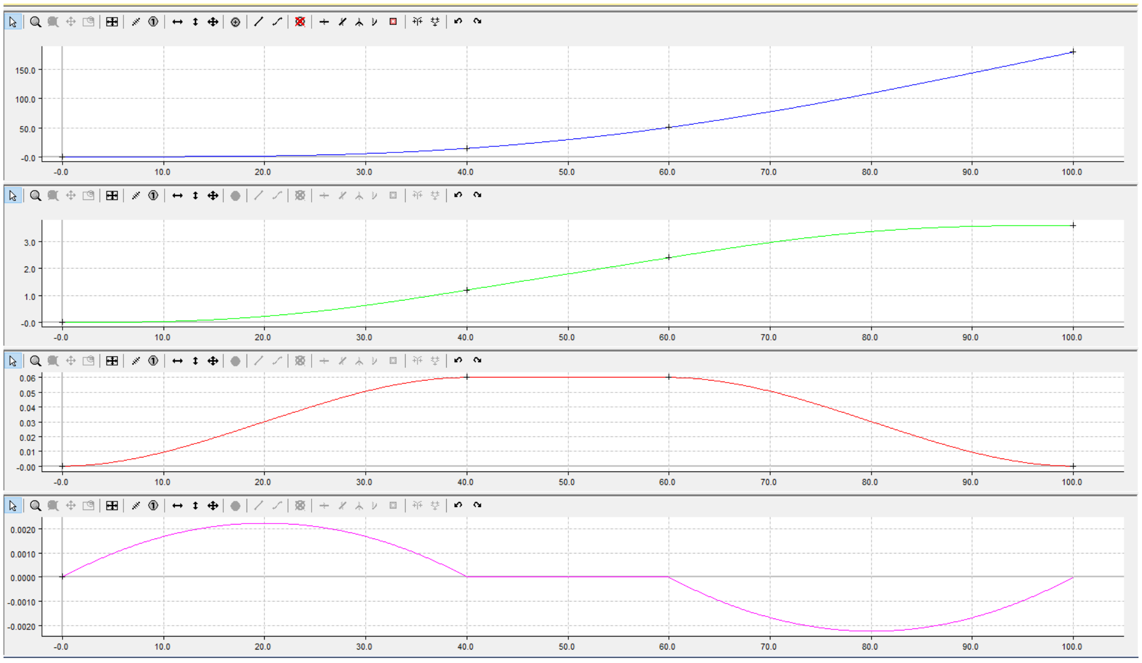


Figure 72: This figure shows a motion profile for position (in blue), velocity (in green), acceleration(in red) and jerk (in pink) for a Position Increase motion type in TwinCAT

Another additional requirement was to calculate a motion profile, Position Decrease, with a predefined start position, start velocity, end position and an acceleration behaviour. For Position Decrease the end position must be smaller than the start position. The result of this can be seen in figure 73. In this figure the position has been decreased from 180 to 0.

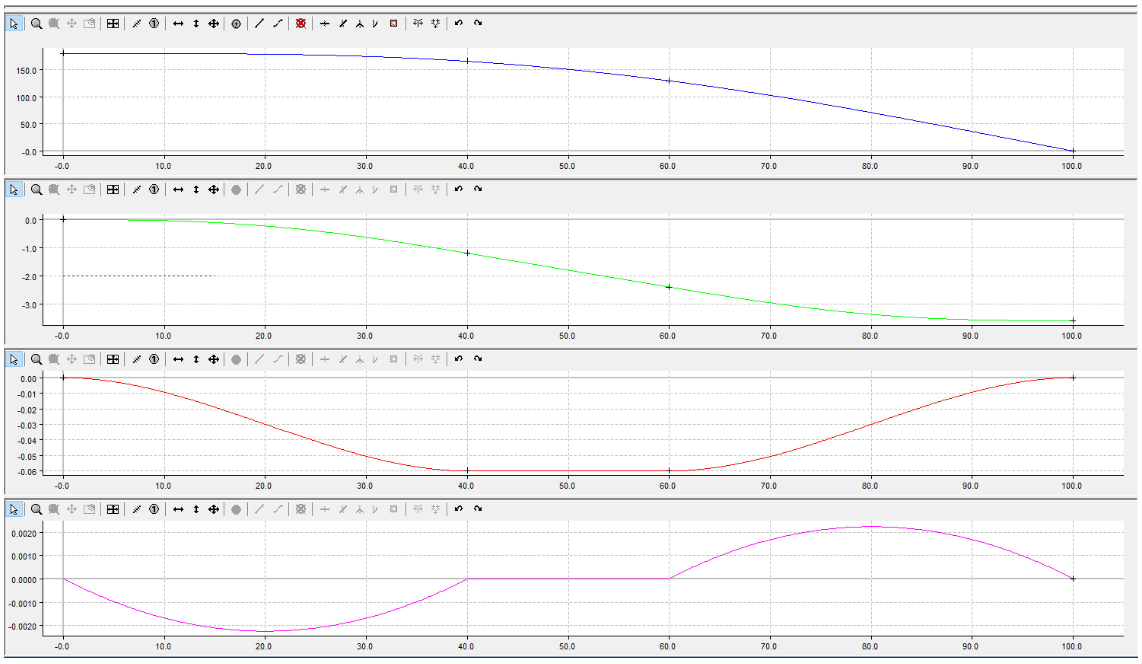


Figure 73: This figure shows a motion profile for position (in blue), velocity (in green), acceleration(in red) and jerk (in pink) for a Position Decrease motion type in TwinCAT

5.2.5 Dwell and Cruise

Figure 74 shows a motion profile for Dwell where the initial velocity and end velocity is zero. Since the velocity is constant the acceleration and jerk values are also zero. Figure 75 shows an example of a motion profile for Cruise where the initial velocity and the end velocity is the same value, in this case 2. Since the velocity is constant here as well the acceleration and jerk values are also zero.

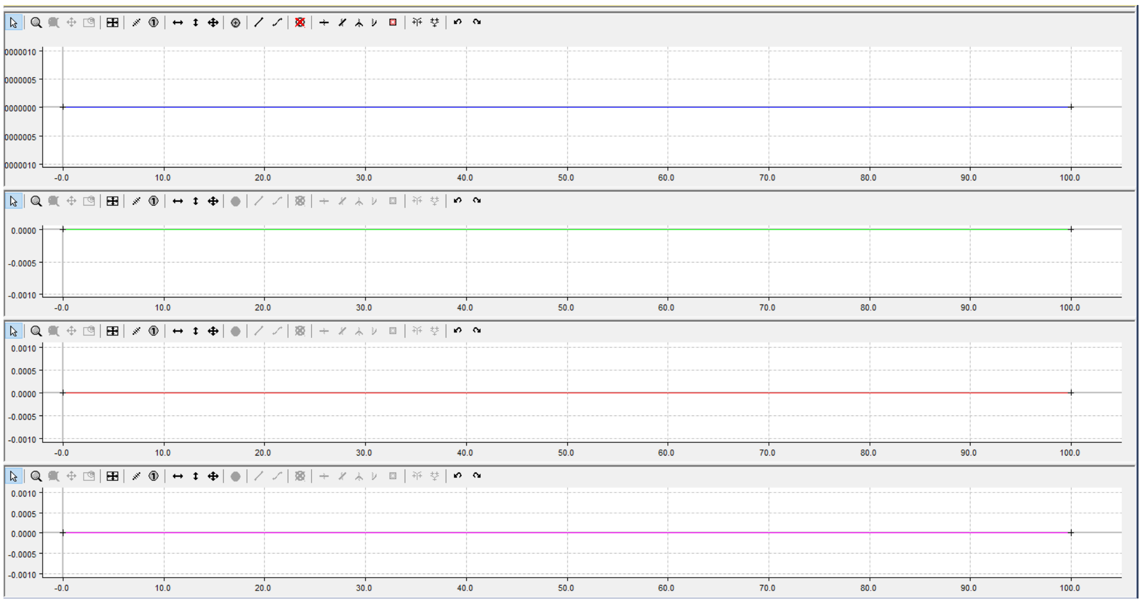


Figure 74: This figure shows a motion profile for position (in blue), velocity (in green), acceleration(in red) and jerk (in pink) for a Dwell motion type

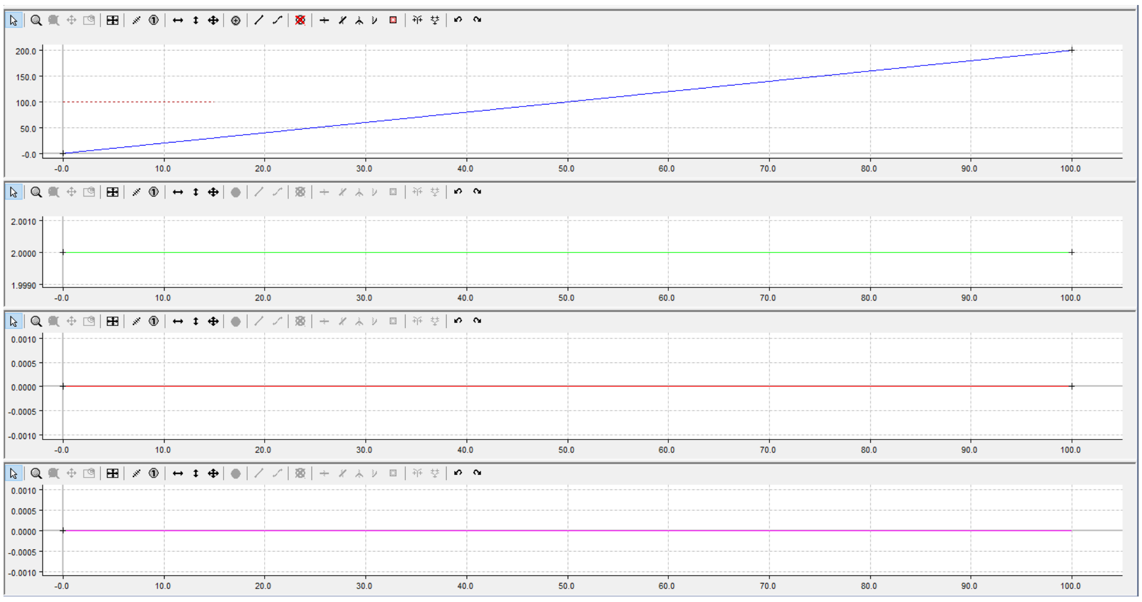


Figure 75: This figure shows a motion profile for position (in blue), velocity (in green), acceleration(in red) and jerk (in pink) for a Cruise motion type

5.2.6 Real Time Implementation

Figure 76 shows the result of running the new version of TwinCAT version of CAMTool. The program has been rewritten to reduce the amounts of exceeds to zero as can be seen in the exceed counter in the bottom of figure 76. As stated in chapter 3, the exceed counter ticks when the program occupies too much of the CPU, hence other routines as input and output actions can not take place. That is the reason why it is important to make sure the exceed counter stays at zero.

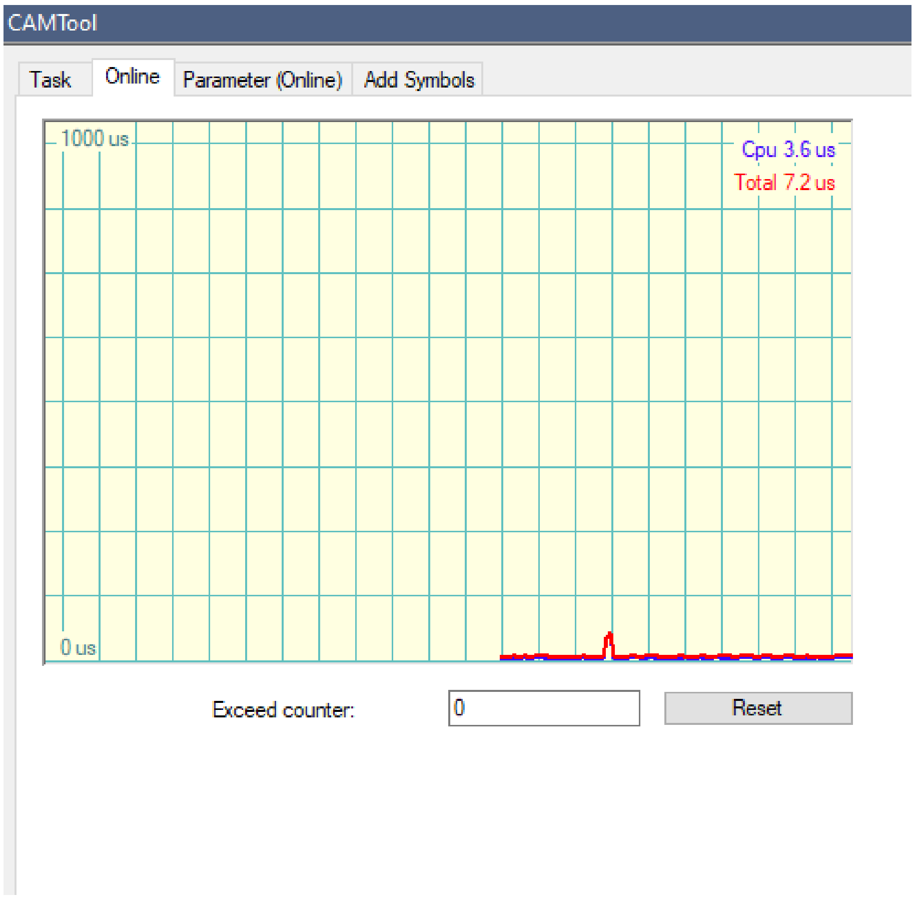


Figure 76: This figure shows the exceed counter when running the new version of CAMTool in TwinCAT

5.3 Verification with Servo Motor

Is it not possible to verify if CAMTool turned out to be better than Beckhoff's CAM Design Tool without comparing the outcome of the two, as stated in section 4.3. The first part of the result is shown in figure 77, which shows one possible operate path created with the new version of CAMTool. The parameters for this motion profile can be seen in figure 78. The second part of the result is shown in figure 79, which shows the operate path created with Beckhoff's CAM Design Tool for creating motion profiles. If one compare the operating-paths the advantage of using CAMTool is shown since the operating path is inside the limits of the servo motor, which is not the case when Beckhoff's CAM Design Tool is used. The third part of the result is shown in figure 80, which has the same input parameters as figure 77 but with another symmetry. The input parameters can be seen in figure 81. This is used for instance when a system has a lot of friction or if a system working against gravity.

The overall motion of the servo motor in all three cases is to go from one point to another, and then back again. Both the forward motion and the reverse motion takes the exact same time to run, and the distance is also exactly the same. The only thing that differs in each case is how the servo motor is accelerating and decelerating.

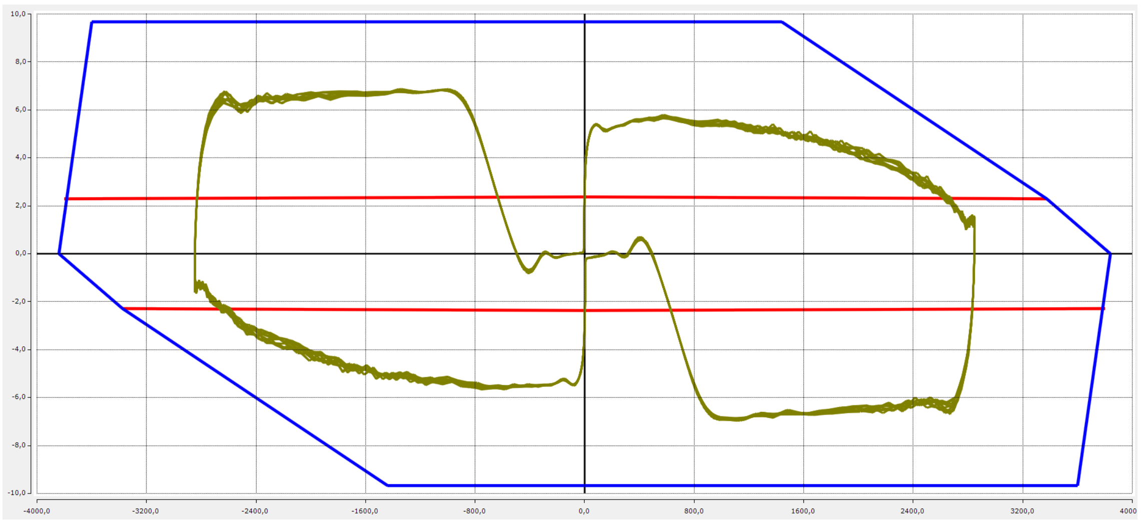
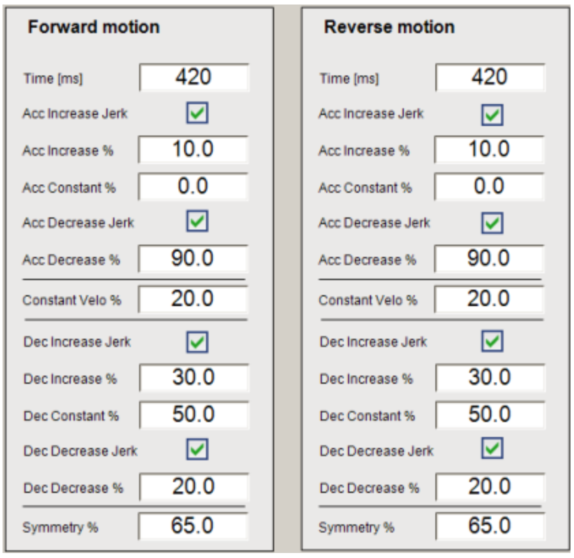


Figure 77: This figure shows one possible operating path, in green, created by the CAMTool. The limit for the servo motor is in blue, and the $T_{rms-limit}$ is coloured red. It is no danger to operate outside of the $T_{rms-limit}$ a short time of period, as the figure shows. Y-axis is the torque in Nm, while x-axis is the speed in rpm



Forward motion		Reverse motion	
Time [ms]	420	Time [ms]	420
Acc Increase Jerk	✓	Acc Increase Jerk	✓
Acc Increase %	10.0	Acc Increase %	10.0
Acc Constant %	0.0	Acc Constant %	0.0
Acc Decrease Jerk	✓	Acc Decrease Jerk	✓
Acc Decrease %	90.0	Acc Decrease %	90.0
Constant Velo %	20.0	Constant Velo %	20.0
Dec Increase Jerk	✓	Dec Increase Jerk	✓
Dec Increase %	30.0	Dec Increase %	30.0
Dec Constant %	50.0	Dec Constant %	50.0
Dec Decrease Jerk	✓	Dec Decrease Jerk	✓
Dec Decrease %	20.0	Dec Decrease %	20.0
Symmetry %	65.0	Symmetry %	65.0

Figure 78: In this figure the input parameters for figure 77 are shown

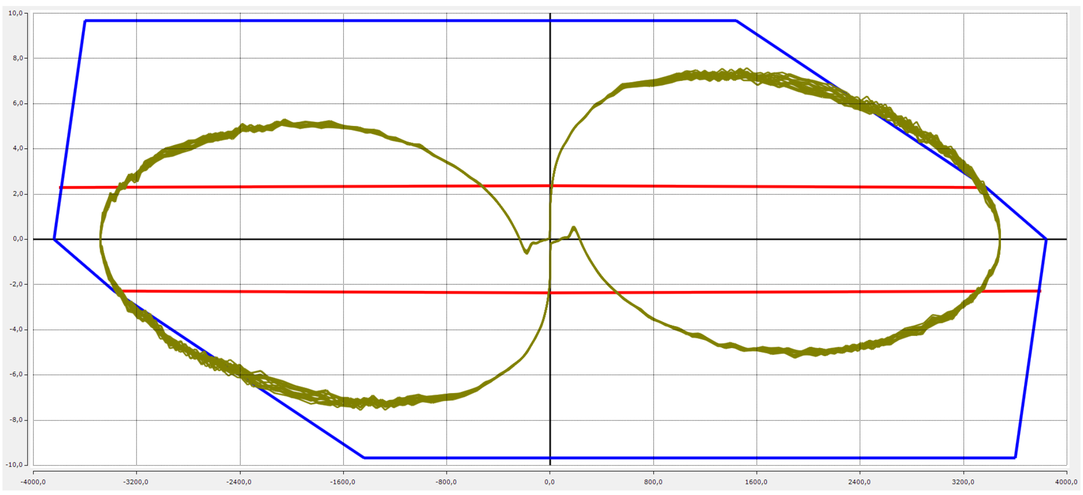


Figure 79: This figure shows the operating path, in green, created by Beckhoffs CAM Design Tool. The limit for the servo motor is in blue, and the $T_{rms-limit}$ is in red. It is no danger to operate outside of the $T_{rms-limit}$ a short time of period, as the figure shows. Y-axis is the torque in Nm, while x-axis is the speed in rpm

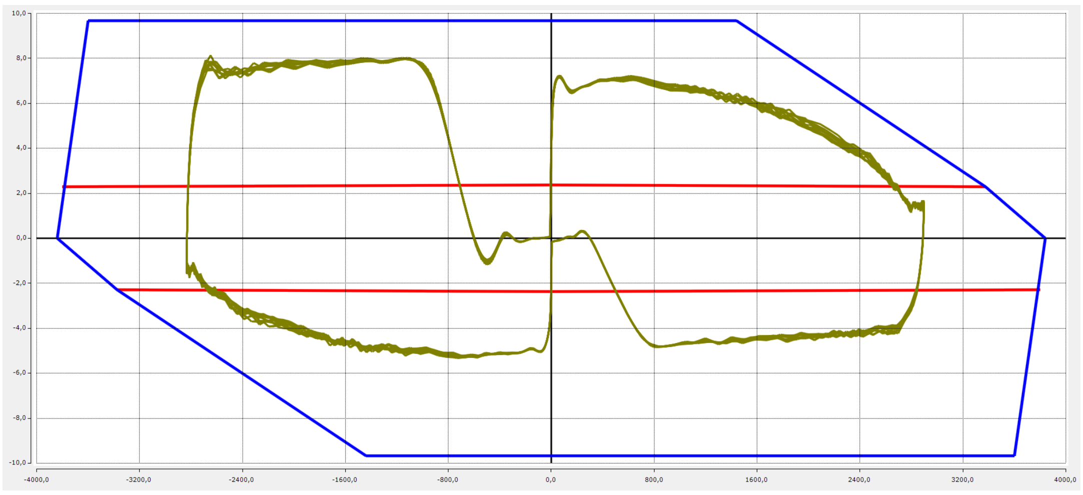
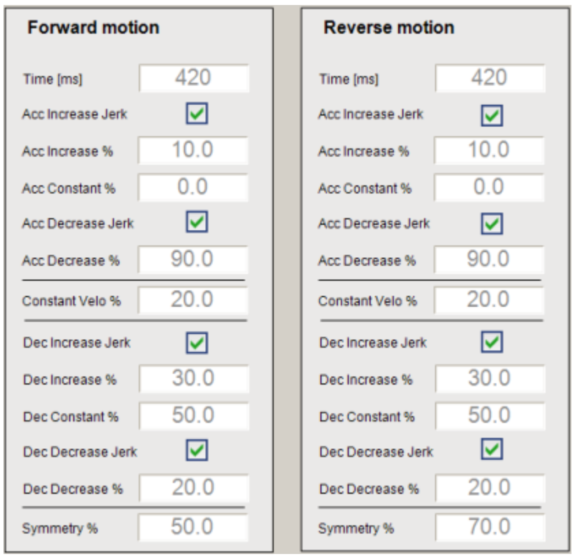


Figure 80: This figure shows one possible operating path, in green, created by the CAMTool. It is the same as in figure 77 but with another symmetry. The limit for the servo motor is in blue, and the $T_{rms-limit}$ is coloured red. It is no danger to operate outside of the $T_{rms-limit}$ a short time of period, as the figure shows. Y-axis is the torque in Nm, while x-axis is the speed in rpm



Forward motion

Time [ms]	420
Acc Increase Jerk	✓
Acc Increase %	10.0
Acc Constant %	0.0
Acc Decrease Jerk	✓
Acc Decrease %	90.0
Constant Velo %	20.0
Dec Increase Jerk	✓
Dec Increase %	30.0
Dec Constant %	50.0
Dec Decrease Jerk	✓
Dec Decrease %	20.0
Symmetry %	50.0

Reverse motion

Time [ms]	420
Acc Increase Jerk	✓
Acc Increase %	10.0
Acc Constant %	0.0
Acc Decrease Jerk	✓
Acc Decrease %	90.0
Constant Velo %	20.0
Dec Increase Jerk	✓
Dec Increase %	30.0
Dec Constant %	50.0
Dec Decrease Jerk	✓
Dec Decrease %	20.0
Symmetry %	70.0

Figure 81: This figure shows the input parameters for figure 80

6 Discussion and Conclusions

This chapter discusses the results of this thesis. In the first section, Enhancement, the updated version is compared with the old one. There is also a discussion if the requirements have been fulfilled. The next section, Verification, features a discussion around the verification of our results. After that a discussion regarding the use of polynomial functions for describing the motion in the implementation instead of sine, constant and cosine functions like in SCCA will follow in the section Polynomial Motion Functions. The conclusions drawn from this project is then presented and lastly a section describing future work is presented, introducing new possible ways to improve the tool further and what new features to implement.

6.1 Enhancement

Since the tool has been developed in many aspects, each one will be discussed in this section to determine if it was a good change or not. Firstly, the GUI changes will be described, followed by the motion types. This section ends with a discussion on if the goals were achieved and if they were, how successful were they.

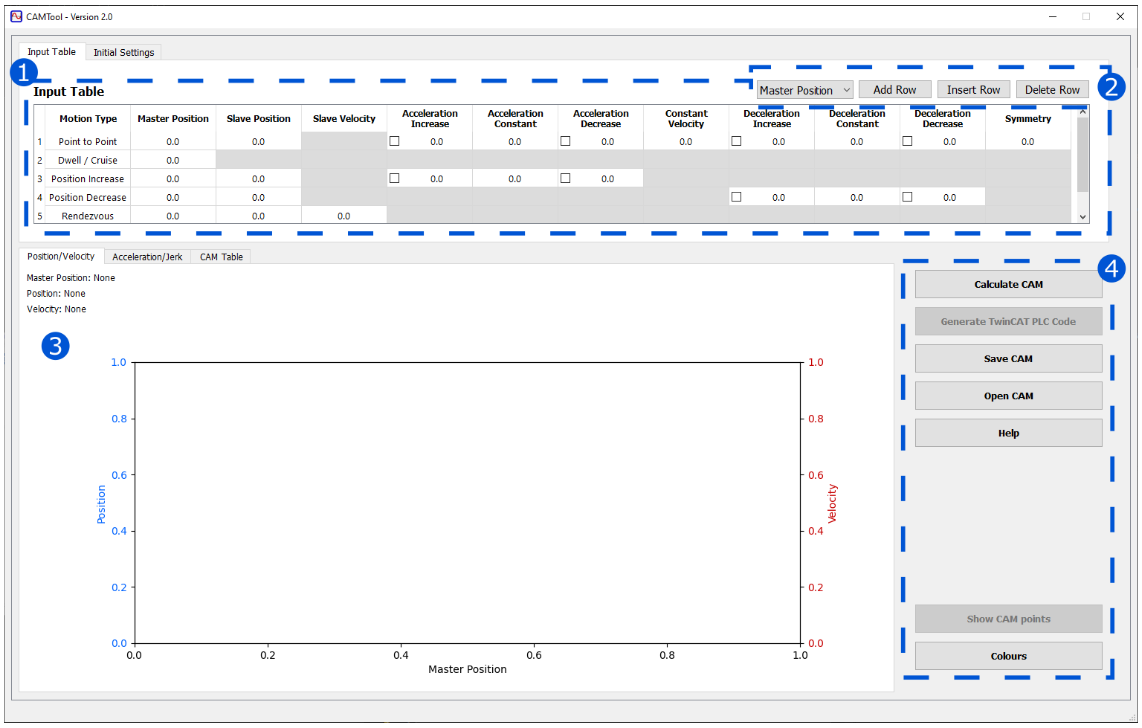
6.1.1 Graphical User Interface

The first observation one can make when one opens the application is the new graphical interface. Compared to figure 13 all boxes in figure 82 have one or more changes. Firstly, the input table in box one has been extended with two columns. One of them is motion type, which is necessary to display when the application now has several motion types. The other column is slave velocity, which the operator uses to specify when the design of a Velocity Decrease, Velocity Increase or Rendezvous type of motion profile takes place. Box number two has also got a minor upgrade in terms of text instead of a sign. The reason why is Insert, which is an additional button. This makes it possible for the designer to insert a row above the selected one in the input table wherever, which was not possible with the old version. The Add button can also be used wherever in the input table, but adds a row below the selected one. To keep these two buttons apart text was a better way to go instead of signs. Box two has also got a new location due to the extended input table and the aim to keep the application window not too wide.

Boxes three and four have also got some changes in figure 82 compared with figure 13. Box three has got a very requested update, namely the value of the graph where the mouse pointer points. This feature includes both x and y value, and are displayed in the top left corner. When the mouse pointer is not in the graph, or when no graph has been calculated, the values are none, which one can see in the figure. The last and final box in figure 82, box number four, has been extended from five to seven buttons. Some has got new names, but some functions are added. Instead of left click in the graph to be able to change colour of the graph, there is now a button. In that way the user knows there is a function to change the colour, instead of having been told or to figure it out on his/her own. This button creates a pop-up window, as shown in figure 42. Compared to the old colour palette, shown in figure 17, the user has less different colours, but more shades. This is a good feature for those who are colourblind. The second added button is to display the acceleration knots, and the corresponding position, velocity and jerk knots. This is a useful feature since the user will have a good overview of the motion profile in terms of non-zero jerk and where each motion profile sections starts and ends.

Under the Initial Settings tab there is also some updates. Since the Slave Velocity is taken into account in some motion types as mentioned earlier, the initial value of it should be able to be edited by the designer. Therefore there is a new cell under Starting Values where one can enter the starting value of Slave Velocity, which corresponds to box five in figure 83. The middle section of the Initial Settings has got a new input line, where the designer can enter the name of the reference of the CAM when it is generated. This feature is located in box six in figure 83. This middle section has also got a title as well as subtitles to the input lines.

As can be seen in the input table in figure 82 there are already some motion types added. This is not default, but to be able to discuss the result of how different motion types are implemented when added. When the user adds a Point to Point motion to the input table, the Slave Velocity should not be included in the calculations and therefore it is not able to write anything in that cell. When adding a Dwell/Cruise motion, nothing but Master Position should be able to be filled in. The next motion type in figure 82 is Position Increase, where the user should fill in Master Position, Slave Position and all acceleration cells. The same is valid for Position Decrease, but for all deceleration cells. One motion type that is not shown in the input table is Velocity Increase, which will have the same cells available as Position Increase but with one difference, Slave Velocity will be available instead of Slave Position. The other motion type that is not shown in the input table is Velocity Decrease, but it will have the same cells available to fill in as Position Decrease but instead of Slave Position it will have Slave Velocity available. The last motion type in the input table in figure 82 is Rendezvous, which should only have the three first cells available to fill in for the engineer.


CAMTool - Version 2.0

Input Table | Initial Settings

1

Input Table

Master Position | Add Row | Insert Row | Delete Row

2

	Motion Type	Master Position	Slave Position	Slave Velocity	Acceleration Increase	Acceleration Constant	Acceleration Decrease	Constant Velocity	Deceleration Increase	Deceleration Constant	Deceleration Decrease	Symmetry
1	Point to Point	0.0	0.0		☐ 0.0	0.0	☐ 0.0	0.0	☐ 0.0	0.0	☐ 0.0	0.0
2	Dwell / Cruise	0.0										
3	Position Increase	0.0	0.0		☐ 0.0	0.0	☐ 0.0					
4	Position Decrease	0.0	0.0						☐ 0.0	0.0	☐ 0.0	
5	Rendezvous	0.0	0.0	0.0								

Position/Velocity | Acceleration/Jerk | CAM Table

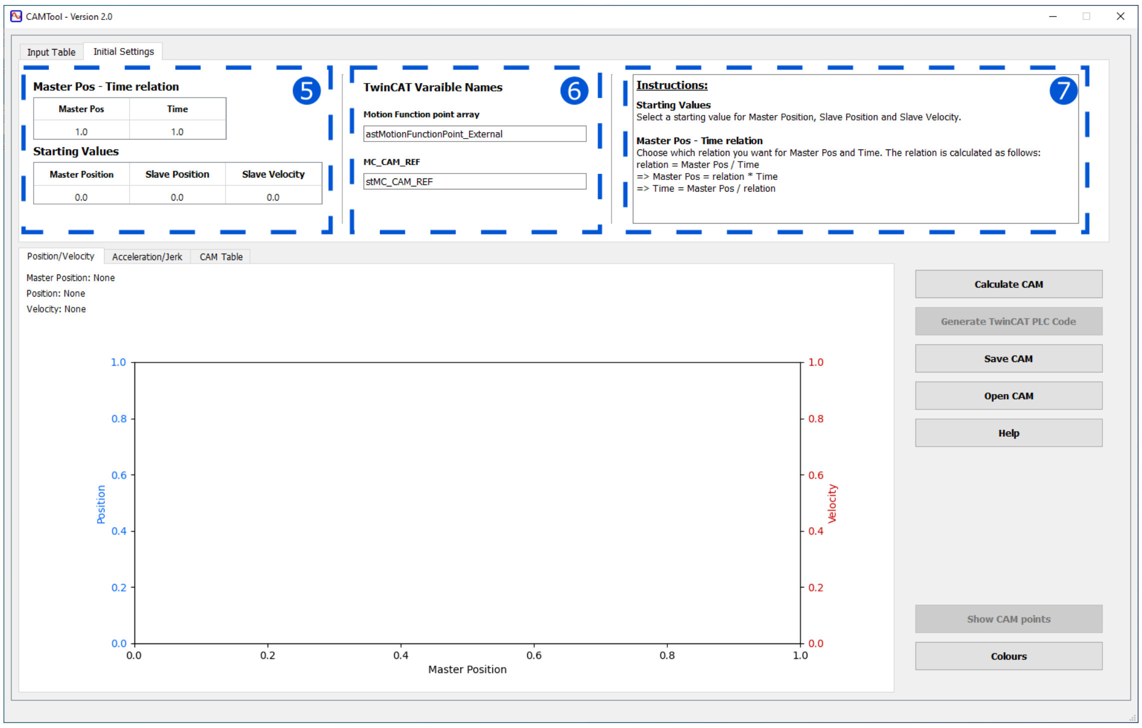
Master Position: None
Position: None
Velocity: None

3

4

Calculate CAM
Generate TwinCAT PLC Code
Save CAM
Open CAM
Help
Show CAM points
Colours

Figure 82: This figure shows the new GUI including the Input Table tab. The picture is divided into smaller parts, numbered one to four


CAMTool - Version 2.0

Input Table | Initial Settings

Master Pos - Time relation

5

Master Pos	Time
1.0	1.0

Starting Values

Master Position	Slave Position	Slave Velocity
0.0	0.0	0.0

TwinCAT Varaible Names

6

Motion Function point array
astMotionFunctionPoint_External

MC_CAM_REF
stMC_CAM_REF

Instructions:

7

Starting Values
Select a starting value for Master Position, Slave Position and Slave Velocity.

Master Pos - Time relation
Choose which relation you want for Master Pos and Time. The relation is calculated as follows:
relation = Master Pos / Time
=> Master Pos = relation * Time
=> Time = Master Pos / relation

Position/Velocity | Acceleration/Jerk | CAM Table

Master Position: None
Position: None
Velocity: None

Calculate CAM
Generate TwinCAT PLC Code
Save CAM
Open CAM
Help
Show CAM points
Colours

Figure 83: This figure shows the new GUI including the Initial Settings tab. The picture is divided into smaller parts, numbered five to seven

To generate a table with the motion profile values, the user will enter the tab CAM Table. Under this tab there is a small but requested update, as can be seen in figures 84 and 85. Boxes numbered eight and ten in the figures have no change compared to the same boxes in figure 15 and 16. But the updates have been made in boxes numbered nine and eleven in figure 84 and 85, respectively. The button that was called Save Table in the old GUI is now named Save CAM Table, and a new button is added called Save Motion Designer CAM. The difference between the buttons is what they are saving from the generated table. The first one saves everything while the second one just saves the first and third columns of the generated table as the name tells.

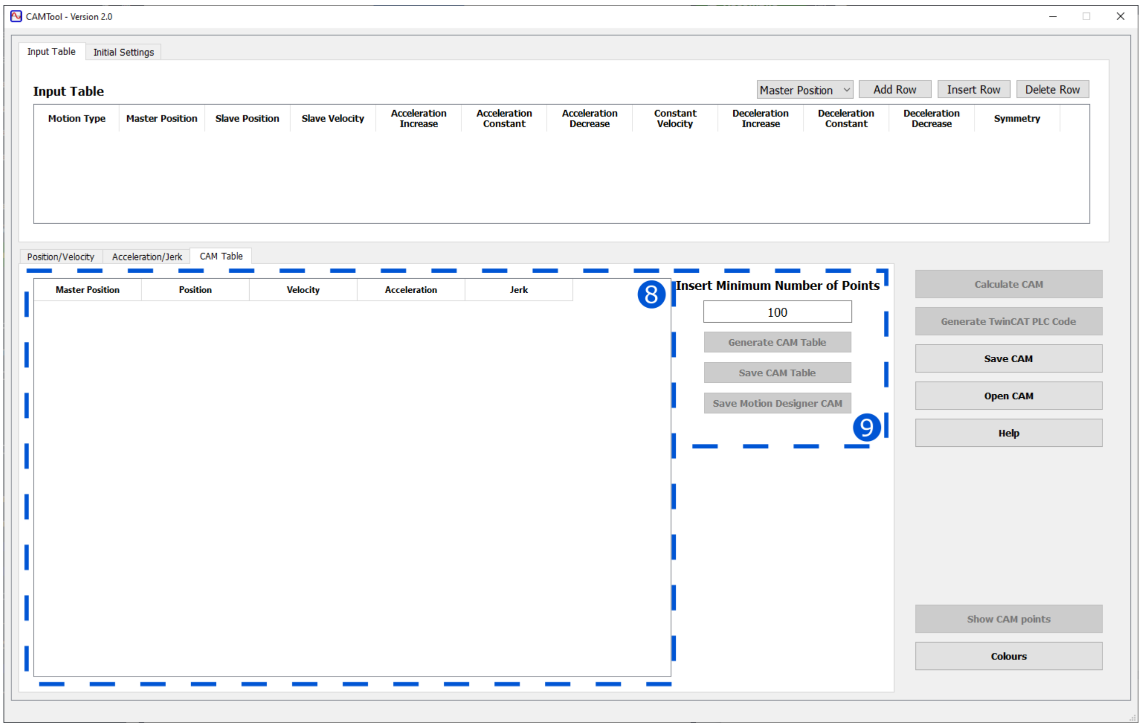
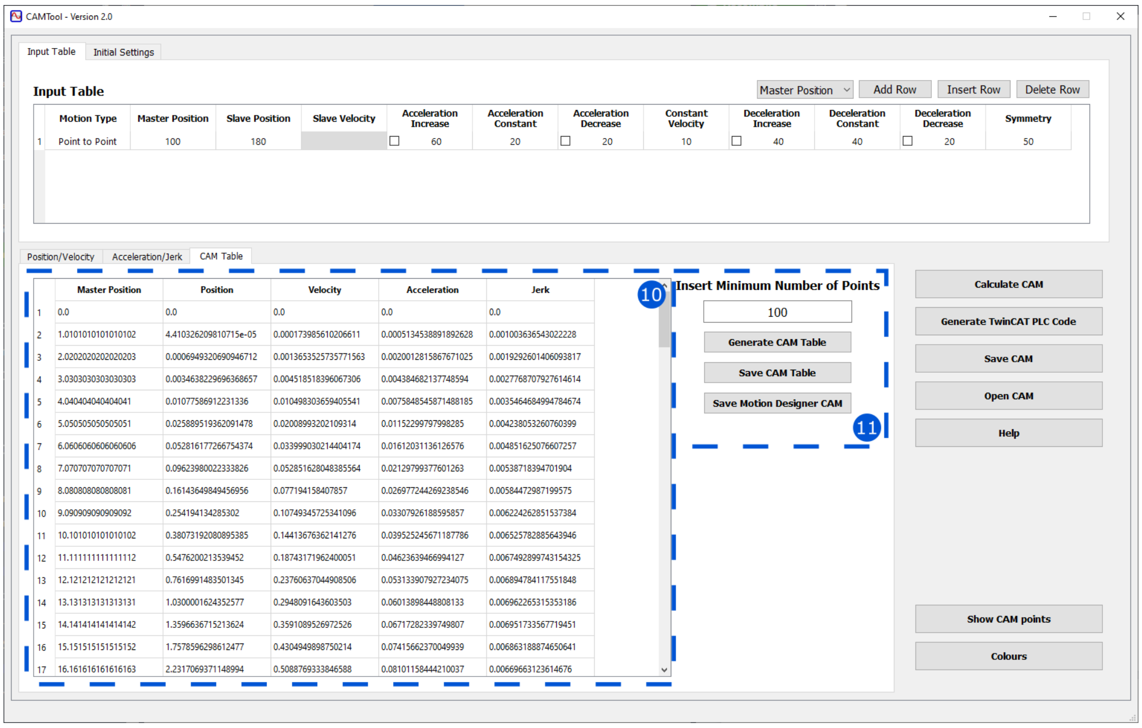


Figure 84: This figure shows the new GUI including the CAM Table tab. The picture is divided into smaller parts, numbered eight to nine



	Motion Type	Master Position	Slave Position	Slave Velocity	Acceleration Increase	Acceleration Constant	Acceleration Decrease	Constant Velocity	Deceleration Increase	Deceleration Constant	Deceleration Decrease	Symmetry
1	Point to Point	100	180		☐ 60	20	☐ 20	10	☐ 40	40	☐ 20	50

	Master Position	Position	Velocity	Acceleration	Jerk
1	0.0	0.0	0.0	0.0	0.0
2	1.0101010101010102	4.410326209810715e-05	0.000173985610206611	0.0005134538891892628	0.001003636543022228
3	2.0202020202020203	0.0006949320690946712	0.0013653525735771563	0.0020012815867671025	0.0019292601406093817
4	3.0303030303030303	0.0034638229696368657	0.004518518396067306	0.004384682137748594	0.0027768707927614614
5	4.040404040404041	0.01077586912231336	0.010498303659405541	0.0075848545871488185	0.0035464684994784674
6	5.050505050505051	0.025889519362091478	0.02008993202109314	0.01152299797998285	0.004238053260760399
7	6.0606060606060606	0.052816177266754374	0.033999030214404174	0.01612031136126576	0.004851625076607257
8	7.070707070707071	0.09623980022333826	0.052851628048385564	0.02129799377601263	0.00538718394701904
9	8.080808080808081	0.16143649849456956	0.077194158407857	0.026977244269238546	0.00584472987199575
10	9.090909090909092	0.254194134285302	0.10749345725341096	0.03307926188595857	0.006224262851537384
11	10.101010101010102	0.38073192080895385	0.14413676362141276	0.039525245671187786	0.006525782885643946
12	11.111111111111112	0.5476200213539452	0.18743171962400051	0.04623639466994127	0.0067492899743154325
13	12.121212121212121	0.7616991483501345	0.23760637044908506	0.053133907927234075	0.006894784117551848
14	13.131313131313131	1.0300001624352577	0.2948091643603503	0.06013898448808133	0.006962265315353186
15	14.141414141414142	1.3596636715213624	0.3591089526972526	0.06717282339749807	0.006951733567719451
16	15.151515151515152	1.7578596298612477	0.4304949898750214	0.07415662370049939	0.006863188874650641
17	16.161616161616163	2.2317069371148994	0.5088769333846588	0.08101158444210037	0.00669663123614676

Figure 85: This figure shows the new GUI including the CAM Table tab. The picture is divided into smaller parts, numbered ten to eleven

One big advantage with the GUI of the Python version of CAMTool is the transparency between the TwinCAT version of CAMTool. To move from working in Python with designing CAMs to do the same work in TwinCAT have to be easy, which it is. The same holds for moving in the other direction as well. If one compare the input table in figures 78 and 38, it is observed they have the exact same input parameters.

6.1.2 Application

The overall software architecture was kept intact during the development of the CAMTool, and the new features were implemented on top of the existing ones. This could be done since all motion types are based on the same mathematical expressions and ideas with polynomial functions.

The main difference in the TwinCAT version of the application and the Python version is the use of Recalculate. The need of Recalculate came from the integral calculation of the acceleration, that turned out not to be accurate enough to calculate a correct amplitude of the motion profile. This will be discussed more in depth in section 6.6. The recalculate algorithm is iterative, which means it iterates by moving the acceleration points up or down until the motion reaches the end goal, whether it is an end velocity or an end position, as can be seen in the algorithms 1, 2 and 3. The condition for running the algorithm can be seen in the while statements in the algorithms presented as algorithms 1, 2 and 3, and breaks the while loop when the difference between the current value and the end value is smaller than 10^{-8} or when the algorithm has run for 1000 iterations. Running for 1000 iterations should indicate that an error with the code has occurred.

The calculation of the new acceleration or deceleration amplitudes is done by calculating the amplitude needed to reach the required position, see equations 20, 21, 22 and 23. First a unit acceleration motion is calculated where the max amplitude of the acceleration is one. The total distance or velocity change is then divided by either the double integral or integral of this acceleration motion. The quota is then multiplied by the starting acceleration amplitude to gain the correct acceleration amplitude. This way, a motion profile with an initial velocity can be calculated since it can not be scaled to reach the target. The initial velocity is accounted for when calculating the position distance, which can be seen in the equations 22 and 23.

Compared to what was presented in subsection 3.3.1 about non-zero jerk and what was presented in subsection 5.1.3, the application can calculate motion profiles with non-zero jerk in the four specified positions that fulfils the requirements presented in the beginning of this thesis project. The motion profiles presented in the results, figures 45, 46, 47 and 48 as well as figures 65, 66, 67 and 68 are similar to those presented in the background as well as the goal of the application, figures 23, 24, 25 and 26. This means that both the Python application and the TwinCAT application achieved their goals.

Doing a similar comparison between the requirements stated in subsection 3.3.2 about Velocity Increase and Velocity Decrease and shown in the figures 27, 28, 29 and 30 and the figures presented in subsection 5.1.4 of the result, figures 50, 51, 52 and 53. Likewise the same comparison can be made between the requirements presented previously and the result in TwinCAT, figures 70 and 71. It is possible to see that the requirements have been fulfilled and the motion profiles behaves as expected and it is possible to achieve a predetermined velocity assuming a specified acceleration behaviour, both in the TwinCAT and in the Python application.

Similarly, the same comparison can be made between the requirements stated in subsection 3.3.3 about Position Increase and Position Decrease and shown in figure 31, 32, 33 and 34 and the figures presented in subsection 5.1.5, figures 54, 55, 56 and 57. Likewise doing the same comparison with the figures presented in the TwinCAT section, figure 72 and 73. From this comparison it shows that the results fulfil the requirements and behaves as expected and it is possible to achieve a set position assuming a specified acceleration behaviour, both in the Python application and in TwinCAT.

Another important aspect to consider is the similarities in the results between the Python version of CAMTool and the TwinCAT version of CAMTool. When comparing the figures for non-zero jerk between the Python version, figures 45, 46, 47, 48 and 49 with the corresponding figures for TwinCAT, figures 65, 66, 67, 68 and 69, one can see that the profiles have the same shape and amplitude. This is important since the transition between the two version should be smooth and one should be able to move between the two without any obstacle.

The same comparison can be made between the figures for Velocity Increase and Velocity Decrease. Comparing figures 50, 51, 52 and 53 for the Python version and for the TwinCAT version which is the figures 70 and 71 show that the motion profiles produced are similar, which means that the transition between the two versions are smooth.

A similar comparison can be made for the position figures in Python, figures 54, 55, 56 and 57 and for TwinCAT, figures 72 and 73. This shows that the motion types Position Increase and Position Decrease produce the same acceleration behaviour and amplitude in both versions which make transition between the two seamless.

Two subsections that are presented in the results, even though they were not a goal with the project, are subsections 5.1.7 and 5.2.5. To make a Dwell or Cruise is important in motion control, since one might want the servo motor to have a constant speed or to stand still a longer time of period. Those two motion types fell out in a naturally way of the calculations, hence they are presented in the result as well.

6.1.3 Rendezvous

The Rendezvous function was decided not to be implemented in TwinCAT because it lacked the control of the acceleration behaviour in contrast with the other motion types and thus losing some of the strength with this application. However, it was decided to keep this feature in the Python software since it enables more advanced motion profiles to be designed.

However, it would require to implement the calculations of the coefficients for a fifth-degree polynomial. But much of this work builds on the same principle as the calculations used for calculating the coefficients for a third-degree polynomial, presented in subsections 2.4.1 and 2.4.2. The equations necessary are also presented in subsection 3.3.4.

6.1.4 Real-Time Implementation

The real-time implementation was done by converting the code from purely sequential to state machine based with support from the theory presented in subsection 2.7.1. The advantage of this is that the software may be interrupted in a state and other parts of the machine software can continue running, for example any safety functions or similar. The importance of making it real-time compatible is to ensure that it can run on the same PLC as the servo motor so that the operator can update the motion profile on site and not have to go back to the Python program to adjust the motion profiles.

6.1.5 Goals

All the goals that were set up during the planning and the ones added during the project have been achieved, except that of implementing a motion profile where the start velocity and start position were predetermined and there was an end velocity and end position to achieve in TwinCAT. The reason for this has been discussed in previous subsections. The motion profiles that were implemented reached the specific target set in the requirements. The motion profiles implemented does also work both in Python and in the TwinCAT environment which was essential for this thesis project.

Some of the goals, however, were adjusted as the work progressed since the original plan did not fit into the implementation at that stage of the CAMTool or the result was not satisfactory. For instance the decision to not implement Rendezvous in TwinCAT because the lack of control of the acceleration behaviour as that is the primary purpose of this tool, to be able to design motion profiles with a determined acceleration behaviour.

The Point to Point motion profile was verified against a motion profile made with Motion Profile Designer provided by Beckhoff and the results showed that it is possible to more freely design a motion profile and being able to adapt the motion profile to the application. The result that we present in figures 77 and 80 shows that we are able to use the current servo motor to perform the motion compared to figure 79 where the motion is outside the motor servo specification.

6.2 Verification

It is quite easy to observe that the CAMTool is able to adapt the motion profile better to the application compared to Beckhoffs CAM Design Tool for making motion profiles. As can be seen in figure 77 the operating path decided by the motion profile of the CAMTool is inside the servo motor limit, and even has some margins to it. If one takes a look at figure 79 instead, the operating path decided by the motion profile of Beckhoffs CAM Design Tool is at some points outside the limits for the servo motor. In this way, the opportunity for the design engineer to create a motion profile, which will make an operating path inside the servo motor limit, is important and necessary. Without CAMTool the user is not able to move to one point and back again at a specific time as in section 5.3, which verifies the application.

Further, the symmetry of the motion profile has a big impact on the operating path. As can be seen in figures 78 and 81 the symmetry is not equal, which means the corresponding operating path in figures 77 and 80 differ. The accelerating peak value of the forward motion is around six and seven, respectively, while the deceleration peak value of the forward motion is around negative seven and negative five, respectively. The same can be observed for the reverse motion, and both directions of the motion is concluded in tables 1 and 2. By changing the symmetry, one can observe the operating path will be moved up or down depending of how the symmetry is changing. This means the user can give either acceleration or deceleration motion more or less time, which is important when the application works against gravity or has significant mechanical friction. The belt and pulley application that was used to test the motion profiles is such an example. When the servo motor gets more time to works against gravity or mechanical friction, the it do not need to work as hard and may be inside the limits of its operation. This functionality is complicated to achieve in the Beckhoffs CAM Design Tool.

Table 1: The approximate torque peak values of the operating path for figure 77

Approximate Torque Peak Values	Forward Motion	Reverse Motion
Acceleration	6	-6
Deceleration	-7	7

Table 2: The approximate torque peak values of the operating path for figure 80

Approximate Torque Peak Values	Forward Motion	Reverse Motion
Acceleration	7	-5
Deceleration	-5	8

One undesired behaviour of the operating path exists in the result figures of the verification in section 5.3. It emerges when the torque is almost zero close to stand still. This flat area of the operating path is not desired, and this behaviour is due to several factors including non-rigidness in the mechanical drive train, not optimal tuning of the servo system as well as friction. As stated in section 2.2 backlash in the gearbox is necessary in many applications, but when the servo motor is changing direction frequently it creates the unwanted behaviour seen in the figures. Further testing when tightening the belt reduced the flat area. Tightening the belt reduced the friction when driving the belt and thus reducing the friction in the overall application.

6.3 Polynomial Motion Functions

In this section the use of a polynomial motion function will be discussed. In chapter 2 two types of models for describing motion profiles were introduced, the SCCA model and the polynomial model. In this software we chose to work with polynomial functions for modelling of jerk, acceleration, velocity and position. Polynomial functions have two big advantages over the SCCA-method. Firstly, they introduce a more intuitive way of modelling and describing the motion functions. If one were to look at the function for acceleration and the function for position in equations 9 and 16 the first three coefficients are the same. Thus the polynomial equations give an intuitive way of understanding how the motion profiles and different functions depend on each other. One additional aspect to notice is how the position is solely dependent on the acceleration behaviour, starting velocity and starting position. This is easy to spot when looking at the position equation 16 but could be rather confusing when looking at the corresponding motion equations in the SCCA model, equations 2, 3, 4, 5, 6 and 7.

Second, as shown in subsection 2.5.2 is that there is extensive library support for spline interpolation built on cubic splines. This is not the case for the SCCA model and not only does the libraries support the development of applications with polynomial functions, it actually hinders the development with another functions, as for example the SCCA. However this is not necessarily a problem since the SCCA functions can be approximated using polynomial functions.

However, one advantage when using the SCCA motion functions is that the cosine and sine functions have infinite continuous derivatives. Polynomial functions does not have this property since the exponent is reduced by one every time the function is differentiate, as can be seen when going from velocity to acceleration in equation, 8. This means that if the position is defined by a third-degree polynomial, then the jerk value will be constant and not continuous between different splines, which may lead to worse performance. But since the cosine and sine functions have infinite continuous derivatives this performance issue will never occur.

6.4 Special Case in Position Increase and Position Decrease

This application builds on the concept of defining a motion by defining its acceleration or deceleration behaviour. However, when defining a motion with a high initial velocity, whether positive or negative, when moving a short distance in a relative long time the acceleration will be opposite to what is defined.

For example, a motion where the application is supposed to move from position zero to five when the Master Position goes from zero to five and the initial velocity is four. Then with no acceleration the end position would be 20 and thus the software presents a deceleration profile instead of an acceleration profile as one might expect when looking at the GUI. This is not wrong per say, since the end position is correct, and since the initial velocity is more than enough to reach the end velocity the motion has to be of deceleration movement to reach the correct end position. But it can be confusing when you enter the parameters for an acceleration behaviour.

The same phenomenon happens when defining a Position Decrease with a high initial backwards velocity. The software will produce a motion with acceleration although the user have defined a deceleration through the GUI. This could be confusing, but it is not wrong.

To limit the Position Increase or Position Decrease motion profile and exclude these cases is not an option since it would limit the tool too much, especially since there is no fault in the application. One should instead take extra caution when designing motion profiles for motions with a high initial velocity.

6.5 Conclusions

By using splines and polynomial functions for acceleration, jerk, velocity and position more advanced motion profiles can be designed and then used to drive servo motors in real-world applications. As can be seen in figure 45, 46, 47, 48 and 49 which shows motion profile with non-zero jerk, at all the possible points points. Figure 49 shows the acceleration behaviour when the option for non-zero jerk is applied at all possible points.

The motion profiles for Velocity Increase, Velocity Decrease, Position Increase and Position Decrease builds on the same concept as a Point to Point and can be just as flexible which, is a strength with this tool.

As can be seen in figures 77 and 80, compared with figure 79 these more advanced motion profiles can use smaller motors to achieve the same end result, which is to be art a specific position at a specific time. There is also a flexibility to design the motion profile according to the motor performance and not the opposite.

All the goals that were set up during the planning and then added during this project have been achieved, except that of implementing a motion profile where the start velocity and start position were predetermined and there was an end velocity and end position to achieve in TwinCAT.

6.6 Future Work

This tool will now be evaluated and hopefully used at Tetra Pak and as the use increases, more cases will be explored. Hopefully, the features that we have implemented will be enough for now but there are a few areas where this tool could be improved and more features could be added. This section aims to outline those areas.

6.6.1 Rendezvous in TwinCAT

As mentioned previously, this feature was deemed unnecessary to implement in TwinCAT since there is no possibility to control the acceleration behaviour and thus not relevant for this application. However, it was left in the Python application and depending on how it is used in the future it can be relevant to implement it in TwinCAT.

How that implementation will look is difficult to say but it would probably not be built on the previous implementation since the position builds on quintic splines and not cubic splines. However, the rest of the calculations should follow the implementation in the Python program thus that can be used as a guide for the overall application as has been done for the other features. Additionally, the same ideas for getting the constraints as is implemented for the acceleration curves can be utilised when implementing the calculations for the quintic splines. Instead of using the jerk constraints the velocity constraints would be used and instead of the acceleration constraints the position constraints would be used, as is presented in the equations 17, 18 and 19 in subsection 3.3.4.

6.6.2 Numerical Integration in Real Time Environment

Due to the inaccuracy of the position integration the function block recalculate is necessary otherwise the end result would not be correct. This inaccuracy leads to extra execution time compared to for example a Point to Point calculation and thus the program becomes slower.

The difficulty here is in implementing a fast but at the same time resource effective algorithm that ensures the necessary accuracy. Since the algorithm in the Python version of CAMTool uses recursion and this function is to be implemented on a PLC where memory and computing power is limited this would not be a good way to do it.

Here are opportunities for future work. Investigate relevant algorithms for calculating an integral numerically and compare the result with the current way to do it.

6.6.3 Other features

Other features that could be implemented is to use superposition of one motion profile on top of another or to be able to add points on an already calculated motion point to change the behaviour in that segment. This is to introduce minor changes to an otherwise suitable motion profile or manipulate the acceleration behaviour in a small section.

There is also a request for a function where it is possible to compare two CAM profiles. In this feature one CAM could be a continuous line and the other CAM profile could be a dashed line. This to be able to show other people who are not so familiar with the concept and theory behind the different CAM profiles the difference between two CAM profiles and be able to easily reason why the one is better than the other.

Another possible feature to be implemented in the future is to make support for motions not based around a fifth-degree polynomial for position. The version of CAMTool as it is now is made specifically to support TwinCAT's fifth degree polynomial when describing the position in a motion. But there might be other manufactures that uses third- or seven-degree polynomials. Support for those could be a possible way to further develop this tool.

There could also be a need for other motion profiles not mentioned in this thesis. The authors hope that the applications is built in such a way that it will be easy to implement those on top of those motion profile options already implemented.

References

[1] R.L. Norton. *Cam Design and Manufacturing Handbook.* Industrial Press, 2009. ISBN: 9780831133672.

[2] A.B. Shahriman, A.K.M. Syafiq, M.S.M. Hashim, D. Hazry, Z.M. Razlan, K.Wan, R. Daud, E.M. Cheng, S.K. Zaaba, and Azizi Azizan. “Improvement of cam performance curve using B-Spline curve”. In: *Journal of Physics: Conference Series* 908 (Oct. 2017), p. 012043. DOI: 10.1088/1742-6596/908/1/012043. URL: https://doi.org/10.1088/1742-6596/908/1/012043.

[3] H.A. Rothbart. *Cam design handbook [Electronic resource].* New York: McGraw-Hill, 2004. ISBN: 0071433287.

[4] H. Barghi Jond, V.V. Nabiyev, and R. Benveniste. “Trajectory Planning Using High Order Polynomials under Acceleration Constraint”. In: *Journal of Optimization in Industrial Engineering* 10 (Dec. 2016), pp. 1–6. DOI: 10.22094/joie.2016.255.

[5] Beckhoff. *AM8041-wDyz.* URL: https://www.beckhoff.com/sv-se/products/motion/rotary-servomotors/am8000-servomotors/am8041-wdyz.html. (accessed: 14.02.2022).

[6] Beckhoff. *Technical terms.* URL: https://infosys.beckhoff.com/english.php?content=../content/1033/am8100/1333724939.html&id=. (last accessed: 2022.04.26).

[7] Beckhoff. *Technical data.* URL: https://infosys.beckhoff.com/english.php?content=../content/1033/am8100/1333724939.html&id=. (last accessed: 2022.04.26).

[8] Circuit Globe. *Four Quadrant Operation of DC Motor.* URL: https://circuitglobe.com/four-quadrant-operation-of-dc-motor.html. (accessed: 07.02.2022).

[9] Apex Dynamics. *Why do we combine a servo motor with a gearbox.* URL: https://www.apexdyna.nl/en/servomotor-gearbox. (accessed: 22.04.2022).

[10] R Keith Mobley. “39 - Gears and Gearboxes”. In: *Plant Engineer’s Handbook.* Ed. by R.K. Mobley. Woburn: Butterworth-Heinemann, 2001, pp. 629–637. ISBN: 978-0-7506-7328-0. DOI: https://doi.org/10.1016/B978-075067328-0/50041-0. URL: https://www.sciencedirect.com/science/article/pii/B9780750673280500410.

[11] B. Einarsson. “Approximation med ri-funktioner”. In: *Nordisk Matematisk Tidskrift* 21.4 (1974), pp. 145–151. ISSN: 00291412. URL: http://www.jstor.org/stable/24525180 (visited on 05/10/2022).

[12] S. Weston. *An introduction to the mathematics and construction of splines.* URL: https://www.academia.edu/5566796/An_introduction_to_the_mathematics_and_construction_of_splines. (accessed: 02.03.2022).

[13] E. Kreyszig and H. Kreyszig. *Advanced engineering mathematics.* 10. ed., International student version. Previous ed.: 2006. Hoboken, N.J. : Wiley ; 2011.

[14] D. Eager, A-M Pendrill, and N. Reistad. “Beyond velocity and acceleration: jerk, snap and higher derivatives”. In: *European Journal of Physics* 37.6 (2016), p. 065008. DOI: 10.1088/0143-0807/37/6/065008. URL: https://doi.org/10.1088/0143-0807/37/6/065008.

[15] Python. *What is Python? Executive Summary.* URL: https://www.python.org/doc/essays/blurb/. (accessed: 07.03.2022).

[16] Python. *General Python FAQ.* URL: https://docs.python.org/3/faq/general.html#what-is-python. (accessed: 07.03.2022).

[17] Qt Company. *Qt Designer Manual.* URL: https://doc.qt.io/qt-5/qtdesigner-manual.html. (accessed: 07.03.2022).

[18] Qt Company. *User Interfaces.* URL: https://doc.qt.io/qt-5/topics-ui.html#important-concepts-in-qt-widgets. (accessed: 13.04.2022).

[19] Qt Company. *Qt Widgets.* URL: https://doc.qt.io/qt-5/qtwidgets-index.html. (accessed: 13.04.2022).

[20] Qt Company. *Application Main window.* URL: https://doc.qt.io/qt-5/mainwindow.html. (accessed: 13.04.2022).

[21] Qt Company. *Desktop integration.* URL: https://doc.qt.io/qt-5/desktop-integration.html. (accessed: 13.04.2022).

[22] Qt Company. *Layout Management.* URL: https://doc.qt.io/qt-5/layout.html. (accessed: 13.04.2022).

[23] Qt Company. *Dialog Windows.* URL: https://doc.qt.io/qt-5/dialogs.html. (accessed: 13.04.2022).

[24] Qt Company. *Rich Text Processing.* URL: https://doc.qt.io/qt-5/richtext.html. (accessed: 13.04.2022).

[25] Qt Company. *Drag and Drop.* URL: https://doc.qt.io/qt-5/dnd.html. (accessed: 13.04.2022).

[26] Qt Company. *Interatnionalization.* URL: https://doc.qt.io/qt-5/internationalization.html. (accessed: 13.04.2022).

[27] Qt Company. *Model/View Programming.* URL: https://doc.qt.io/qt-5/model-view-programming.html. (accessed: 13.04.2022).

[28] Numpy Community. *Numpy user Guide.* URL: https://numpy.org/doc/stable/numpy-user.pdf. (accessed: 11.03.2022).

[29] C.R. Harris, K.J. Millman, S.J. van der Walt, R.Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, M. Picus, S. Hoyer, M.H. van Kerkwijk, Brett M., Haldane A. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Merchant, K. Sheppard, T. Reddy, W. Wecksesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. "Array programming with NumPy". In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[30] T.E. Oliphant. *A guide to NumPy.* Trelgol Publishing USA, 2006.

[31] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burkovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, I. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Hneriksen, E.A. Quintero, C.R. Harrisa, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbergt, and SciPy 1.0 Contributors. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nat Methods* 17 (2020), pp. 261–372. DOI: 10.1038/s41592-019-0686-2.

[32] SciPy. *scipy.interpolate.CubicSpline.* URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html. (last accessed: 2022.04.20).

[33] SciPy. *scipy.interpolate.make_interp_spline.* URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.make_interp_spline.html. (last accessed: 2022.04.20).

[34] J. Kiusalaas. *Numerical Methods in Engineering with Python 3.* 3rd ed. Cambridge University Press, 2013. DOI: 10.1017/CBO9781139523899.

[35] Beckhoff. *TwinCAT 3: the flexible software solution for PC-based control.* URL: https://download.beckhoff.com/download/Document/Catalog/Beckhoff_TwinCAT3_e.pdf. (last accessed: 2022.04.22).

[36] Beckhoff. *TE1510—TwinCAT3 CAM Design Tool.* URL: https://www.beckhoff.com/sv-se/products/automation/twincat/te1xxx-twincat-3-engineering/te1510.html. (last accessed: 2022.05.05).

[37] R. Williams. *Real-time Systems Development.* Computer Science. Elsevier Butterworth-Heinemann, 2006. ISBN: 9780750664714.

[38] Beckhoff. *TwinCAT 3 Real-Time.* URL: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_system/html/tcsysmgr_systemnode_subnodes_realtime.htm&id=. (last accessed: 2022.04.20).