International Research Journal of Advanced Engineering and Science
ISSN (Online): 2455-9024

Trajectory Control of Computerized Numerical Control Machines from External Python Program Using OPC UA

Sara Menetrey[1], Holger Schlegel[1], Martin Dix[1]
[1]Department Production Systems and Processes, Chemnitz Technical University, Chemnitz, Germany

Abstract*—Following the trend towards digitization, Computerized numerical control (CNC) machines are lagging behind due to the lack of compatibility between different manufacturers. The recently developed Open Platform Communications – Unified Architecture (OPC UA) is supposed to change that. The aim of this work is to find the limits of this tool for trajectory planning. It sends the coordinates of the path the machine must follow from an external source while operating. The communication between a DMU 75 monoBlock machine and a Python program is performed using the Python "opcua" library to create an OPC UA Client and the inbuilt OPC UA Server from the Siemens SINUMERIK 840D sl control. Speed measurements have been made monitoring the tool tip position via OPC UA and SINUMERIK trace function. The results were used to determine the boundary conditions for the feed rate and the minimum distance between two trajectory coordinates to obtain a viable solution. The structure of the trajectory itself was also analyzed to find suitable working conditions. After comparing these results with the theory, a rule was set up to ensure the machine's correct operation by guiding it through an external system with OPC UA.*

***Keywords*—** *CNC machining, Industry 4.0, OPC UA*

I. INTRODUCTION

"Industry 4.0", announcing the fourth Industrial Revolution, is a recurring term in scientific papers over the past fifteen years [1]. It contains several concepts, including the digitization of manufacturing processes and the creation of "smart factories", i.e. machines equipped with sensors that are intended to obviate the need for human supervision [2]. The digitization of the industrial production is becoming increasingly interesting with the rise of big data, Artificial Intelligence and the Internet of Things [3]. Incorporating these new technologies to a production system can help to improve the safety and efficiency of a production process, reduce the energy costs, prevent and predict maintenance and maximize the product quality [4]. For this purpose, communication interfaces between machine, sensors and the data processing algorithms are required. One example is the communication standard Open Platform Communications - Unified Architecture (OPC UA).

Released in 2008 OPC UA is an international standard (IEC 62541 [5]) for the platform independent secure transfer of data in the automation field [6]. With the Client-Server communication, it works as a request-respond mechanism, but can also directly be used as a data and event notifier via the publish-subscribe method.

The aim of this study is to investigate what has been achieved with OPC UA up to now in the field of Computerized Numerical Control (CNC) machining and to extend this knowledge to examine its limits for real-time machine trajectory adaptation by using flexible G-code programming. Applications currently under development, which will not be presented in detail in this article, are the communication with an external AI module or an external force sensor. The first one uses image recognition to guide the machine in front of an unspecified work-piece. The second one regulates the radial depth of cut of the tool to extend its service life and improve the general efficiency of the process.

II. RELATED WORK

In the literature, the digitization of CNC machines is mainly used for process monitoring. In [7]–[11], OPC UA servers are created for the various components of experimental setups so all data can be monitored via a client and all elements of the setup can communicate with each other even if the controllers are from different manufacturers. Gui et al. [12] created a monitoring system for FANUC30i and SIEMENS 840D CNC systems using Visual Basic language for the FANUC and OPC for the SIEMENS system. A connectivity framework for Siemens 840D based on web service technologies and OPC UA called UNIFIK is presented in [4]. It enables the acquisition and display of programmable logic controller (PLC) variables.

Some publications also mention the use of digitization to control the machine. Reiser et al. [13] have performed drilling tasks with a modular robot system. Each module had an OPC UA server, including a robot arm with CNC control and a virtual twin. G-code programs were uploaded to the controller using OPC UA which were then executed. The speed or accuracy of the OPC UA transfer is not addressed. Measurements were not presented. In the project of Lin et al. [14], motion commands from a cloud were converted into G-code and forwarded to the controller of a Stäubli robot via OPC UA. Similarly, Madhyastha et al. [15] used OPC UA to remotely control a 3D printing machine. Shicong et al. [16] used OPC to connect a monitoring computer to a SINUMERIK 840D. They developed a tool and program management system to optimize the production task scheduling using genetic algorithms. The common element is the use of the PLC to import external code into the control, and then execute it. The code is not altered once it has been sent to the control and the execution cannot be stopped from external devices once it has been started. The trajectory itself is not implemented during the program execution, which excludes any online adaptation from an external environment.

Sara Menetrey, Holger Schlegel and Martin Dix, "Trajectory Control of Computerized Numerical Control Machines from External Python Program Using OPC UA," *International Research Journal of Advanced Engineering and Science*, Volume 9, Issue 3, pp. 30-35, 2024.

Philip Samuel et al. [17] have developed a system that converts a two-dimensional drawing into G-code, saves the data in an Excel file and sends it to a PLC via OPC UA (OPC Expert). This can be achieved using the file transfer and program selection methods of a Siemens OPC UA Server [18]. The process time has been monitored using Wireshark. Recorded time between write request from computer and write response from controller was 0.5 to 0.9 ms. For the program to be executed by the machine, there must either be an operator starting it or a function block added in the PLC enabling the start by writing a variable. However, with a CNC machine that was commissioned several years ago, it is sometimes complicated or even dangerous to access the internal system and use it to create new function blocks. In the case of Siemens products, a computer would have to be connected to the control and the correct PLC project would have to be found, adapted and re-uploaded. However, if the PLC project does not match the former project that was already loaded on the controller, it may cause the entire machine to stop working properly. The programs are not designed by the manufacturers to be rewritten by an external user. On the other hand, if the machine has to be started manually every time after a new program had been uploaded, it makes the system less appropriate for trajectory adaptation during the operation. So, it would be interesting to directly control the axes by writing CNC variables linked to a running flexible G-code programmed with synchronous actions. The program would have to be started only once.

III. System Design

The aim of this work is to adapt the trajectory of a CNC machine from an external program using OPC UA. The data exchange model is described in Fig. 1. The used CNC machine for the experiments is a 5-axes machine tool (DMU 75 monoBLOCK from DMG MORI) with a Siemens SINUMERIK 840D sl control which has an integrated OPC UA server collecting the machine's data (current position, machine parameter, user data, etc.). The client, which will read and write variables of the machine's server, is written in Python using the "opcua" library and is running on a computer with Windows 10 operating system and Visual Studio 2022.

The G-code running on the CNC machine is called flexible because of its unconventional usage of while-, for-loops, if-conditions and goto-functions that enable to jump back or forward to a different line in the code. The machine is running in a continuous loop until it gets the start signal from the client. Once it is outside the loop, it reads the coordinates from the R-parameters and moves to these points until the client sends the stop signal. Start and stop signals are transmitted by writing and reading R-parameters. Algorithm I summarizes the functioning of the G-code Program. R-parameters are readable and writable variables used to compute data in Siemens and Beckhoff controls. (Other manufacturers may name them differently.) They can be assigned to the spindle speed, the feed rate and any axis value.

As there is a limited amount of available R-parameters (99 per default, up to 999 by changing 28050 MM NUM R PARAM in the system settings), they must be constantly rewritten once the machine reaches their location. For the client to know when to overwrite the R-parameters there are two possibilities:

1) The client observes constantly the current position of the machine and compares it with the point to be overwritten. This can be achieved reading the current actual position (node Id: /Channel/GeometricAxis/aaAcsRel[1,2,3]) or the current remaining distance (node Id: /Channel/GeometricAxis/aaDtew [1,2,3]). In either case, the distance to the next point is computed and when it is small enough, the point is considered reached and can be rewritten. The nodes can be read using request-respond or by creating a subscription. The limiting distance is evaluated in section V.

ALGORITHM I. Flexible programming method

```
1:   Label1
2:   while not R_start do
3:       goto Label1
4:   end while
5:   Label2
6:   G1 X=R11 Y=R12 Z=R13
7:   G1 X=R21 Y=R22 Z=R23
8:   G1 X=R31 Y=R32 Z=R33
9:   ...
10:  if not R_stop then
11:      goto Label2
12:  end if
```

2) The G-code sets another R-parameter to indicate that the point has been passed. The client constantly reads this parameter or even creates a subscription to get informed about the change and overwrites the corresponding parameters. This can also be applied for longer lists of coordinates as long as at the end of this list there is an indicating R-parameter to signal that all points of the list have been passed.

The advantage of the first method lies in its simplicity. The G-code does not need to write parameters at run-time. The system may therefore run more smoothly. The advantage of the second method is its reliability, lead in writing the parameters and requirement to read one parameter instead of three in each loop. Rather than needing to pursue the current position, it reflects the current line that has been written and can write several parameters at the same time. Therefore, it sends larger messages but at a lower frequency.

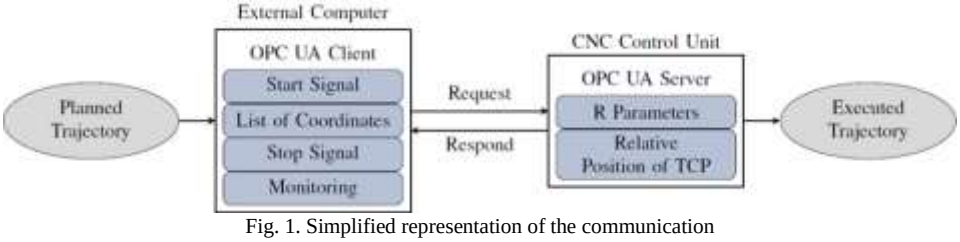


Fig. 1. Simplified representation of the communication

Sara Menetrey, Holger Schlegel and Martin Dix, "Trajectory Control of Computerized Numerical Control Machines from External Python Program Using OPC UA," *International Research Journal of Advanced Engineering and Science*, Volume 9, Issue 3, pp. 30-35, 2024.

International Research Journal of Advanced Engineering and Science
ISSN (Online): 2455-9024

IV. Up Front Analysis

A. Speed Tests

The speed of OPC UA is highly variable depending on the speed of the network in which the different participants of the connection are installed. This may be a reason why it has not often been measured and discussed in the scientific community before. Cavalieri et al. [19] and Veichtlbauer et al. [20] have indeed evaluated the performance of OPC UA. Both found communication times of several milliseconds and highlighted the lack of real-time capabilities of this connection.

When sending a trajectory, consecutive points can sometimes be very close to each other. Depending on the machine's feed rate, the time between two points can be below one millisecond, thus ideally requiring real-time communication. As this is not the case, the client must be able to maintain the rhythm, otherwise it will not be able to update the parameters on time and the machine will follow the wrong trajectory. This could lead to a fatal error, where the operator could be harmed and the work-piece, the tool and the machine itself could be damaged.

In order to get an idea of the processing time of both methods, a temporal recording was made on the machine, in which the value of a real variable, coded on 32 bits, was recorded. This can be done via the trace function of the SINUMERIK 840D sl control. During the recording, the client rewrites the variable ten times without pause. The sequence of values is the following: 1.0, 0.0, 2.0, 0.0, 3.0, 0.0, 4.0, 0.0, 5.0, 0.0. The resulting graph is shown in Fig. 2. It can be seen that each new value was written for the same duration of four milliseconds. This corresponds to the set interpolation cycle of the control, which represents the lower limit of the communication cycle duration.

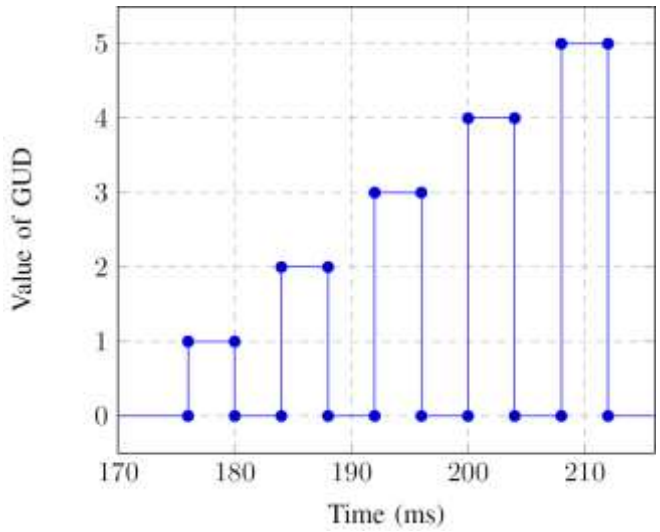


Fig. 2. Recording of a Global User Data during the test with trace function.

To verify and refine these measurements the duration of reading and writing different R-parameters (also 32 bits) have been measured. For the results of table I, one, twenty and hundred R-parameters were read and written. The duration was recorded using the Python library "time". To write and read multiple R-parameters at once, two functions were used. The first (V1) ran a loop to read/write each R-parameter individually. The second (V2) used the inbuilt function called get values from the Python opcua library to read (respectively set values to write) a list of parameters.

V1 is between 14 and 30 % slower, with an average of over 5 ms per parameter, than V2, which average is 4.3 ms per parameter. The standard deviation of the measurements were all under 6 %, so the duration of 4.3 ms will be taken as reference duration in the following.

TABLE I. Time Measurements Results for Reading and Writing.

Reading					
No. Parameters	**1**	**20**		**100**	
Function	V1	V2	V1	V2	V1
Minimum (ms)	0	78.0	83.2	396	493
Maximum (ms)	10.25	115	114	474	562
Average (ms)	4.35	86.9	99.3	428	537
STD (ms)	3.44	7.26	7.46	22.2	15.4
Median (ms)	5.15	83.4	99.1	422	538

Writing					
No. Parameters	**1**	**20**		**100**	
Function	V1	V2	V1	V2	V1
Minimum (ms)	0	72.6	103	412	489
Maximum (ms)	15.95	94.6	125	479	593
Average (ms)	5.67	86.6	111	433	567
STD (ms)	6.15	4.59	4.76	16.9	17.9
Median (ms)	0	88.3	110	424	571

B. Subscriptions

Via subscriptions, parameters can be monitored, not written. The server publishes the information for any client who is interested in it, but it does not receive any information. The R-parameter will still have to be written by request-respond, but the signaling R-parameter or the current position can be observed by subscription. To evaluate the duration difference, a similar speed test has been performed subscribing an R-parameter via Python and running a G-code setting that parameter to different values.

To create the subscription, an object from the class "SubHandler" was generated and used in the function create subscription, but this type of communication depends on the given publishing interval. Lowering this interval only works if the sampling rate of the parameter to be subscribed is lower. Thus, per definition, it is not possible for the subscription communication to be faster than the interpolation cycle of the machine.

During the test, the parameter was set to ten different values consecutively. When the subscription recognized a change in its value, it printed the value to the Python prompt and the time at which it occurred. After repeating this procedure twenty times, the results showed that only two of the ten values were printed to the prompt (maximum four). Regardless of the publishing interval, he subscription was at most able to register two changes with a temporal delay of minimum 10 ms. The others were skipped. This does not make it more reliable than the request-respond system and will therefore not be considered further in the experiments.

Sara Menetrey, Holger Schlegel and Martin Dix, "Trajectory Control of Computerized Numerical Control Machines from External Python Program Using OPC UA," *International Research Journal of Advanced Engineering and Science*, Volume 9, Issue 3, pp. 30-35, 2024.

International Research Journal of Advanced Engineering and Science
ISSN (Online): 2455-9024

V. EPERIMENTAL PROTOCOL

A. Settings

For the experiments, a reference G-code has been generated using the computer-aided manufacturing software ESPRIT. For simplicity reasons, it only uses G1 (linear interpolation) commands. In order to implement multiple forms of G-commands (G0, G2, etc.), other parameters could be written to define which form each line is supposed to be. To get an initial idea of feasibility, this option is first taken out of the equation.

A Python script has been written to extract the coordinates of that generated G-code and put them in a list. Both methods presented in section III are also coded in Python. They take the list and write the coordinates to the right R-parameters at the right time which is either when the Tool Center Point (TCP) is considered close enough (defined later in (2)) to the point to reach next (method 1) or when the signaling R-parameter has changed its value (method 2). The key element of each method is shown as pseudocode in algorithms II and III.

ALGORITHM II. Method 1

Input: $list,\ N$

$index \leftarrow N$
while $index < list\ length$ **do**
  if $\text{—}\ aaDtew\text{—} < \delta$ **then**
    $Rparameter \leftarrow list[index]$
    $index = index + 1$
  end if
end while

ALGORITHM III. Method 2

Input: $list,\ N$

$index \leftarrow N$
$indexGcode \leftarrow 0$
while $index < list\ length$ **do**
  $read\ R0$
  if $R0 = 0$ **then**
    $Rparameter \leftarrow list\left[index : index + \left\lceil\frac{N}{2}\right\rceil\right]$
    $index = index + \left\lceil\frac{N}{2}\right\rceil$
    $R0 \leftarrow 2$
  end if
  if $R0 = 1$ **then**
    $Rparameter \leftarrow list\left[index + \left\lceil\frac{N}{2}\right\rceil : N\right]$
    $index = index + N - \left\lceil\frac{N}{2}\right\rceil$
    $R0 \leftarrow 2$
  end if
end while

In parallel, the machine is running a G-code that loops on reading the R-parameters and moving to the respective points. To compare the results, the TCP position is continuously read and saved by the Siemens integrated trace function.

The influence of the following variables will be analyzed:

- the machine feed rate (F in mm/min),
- the smallest distance between two consecutive points (d_{min} in mm) for method 1,
- the smallest distance between $\left\lceil\frac{N}{2}\right\rceil$ consecutive points (d^N_{min} in mm) for method 2,
- the limiting distance between the TCP and the next point to be evaluated as passed (δ in mm),
- the length of the set of R-parameter coordinates (N).

B. Theory

a) Method 1: In every loop, the position is read and occasionally three R-parameters are written. So the average duration of a loop (Δt in ms) in method 1 is expected to be between 12.9 and 25.8 ms. For the algorithm to have enough time to write all the coordinates and recognize the passed points, the following conditions from (1) and (2) must be met.

$$\frac{F \times \Delta t}{60000} < d_{min} \tag{1}$$

$$\frac{F \times \Delta t}{60000} < 2 \times \delta \tag{2}$$

b) Method 2: In every loop, the R0 parameter is read and occasionally written. Three other R-parameters times $\left\lceil\frac{N}{2}\right\rceil$ are also written from time to time, so the looping time should be between 4.3 and $4.3 \times \left(3\left\lceil\frac{N}{2}\right\rceil + 1\right)$ ms. As the functioning condition is $d^N_{min} > \left\lceil\frac{N}{2}\right\rceil \times d_{min}$, (3) is sufficient.

$$\frac{F \times 4.3 \times \left(3\left\lceil\frac{N}{2}\right\rceil + 1\right)}{60000} < \left\lceil\frac{N}{2}\right\rceil \times d_{min} \tag{3}$$

To verify these equations, the following experiments were conducted.

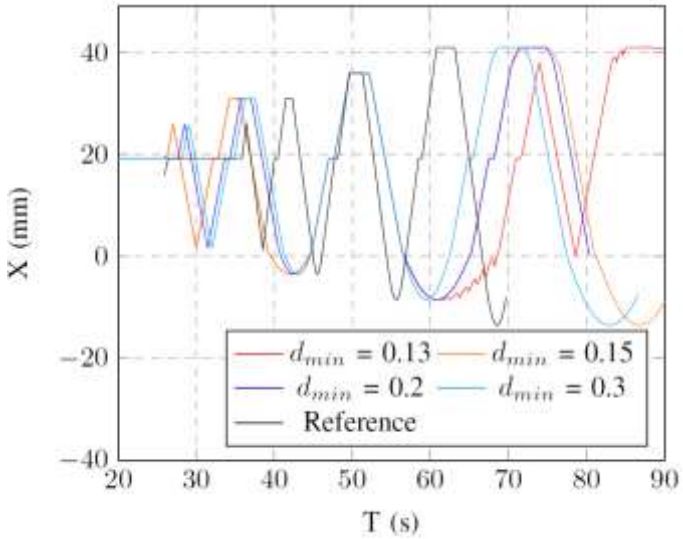
VI. EXPERIMENTS

For each method, the same trajectory list is tested. In order to observe the feed rate's influence, the programs with δ equal to d_{min} (method 1) and N equal to 7 (method 2) are executed for different feed rates, with values of d_{min} starting from 0.5 mm. Then, this value is decreased until failure in order to get the limiting d_{min} for each feed rate. Examples of the resulting trajectory graphs are shown in Fig. 3.

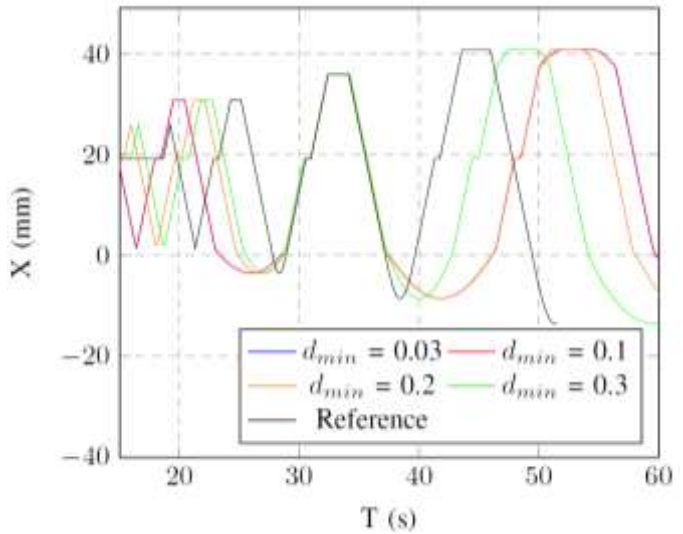
Every recording shows short periods of time (on average 40 ms) in which the feed of the machine drops to zero. Thus, the tool comes to a standstill repeatedly. This is caused by the required synchronization, between each G-code loop, programmed with the command STOPRE. It stands for "Stop Prediction" and deactivates the Look-ahead of the control. Since the R-parameters are read in the pre-run and not in the main run, their values are not updated in the loop without synchronization. The frequency of these stops depends on the length of the G-code loop. The longer the loop, the smaller the frequency of stops. When the frequency of the interruptions is too high or when their placement is inconvenient, especially while working on the work-piece, these fractures in the trajectory can damage the tool and reduce its lifetime.

Sara Menetrey, Holger Schlegel and Martin Dix, "Trajectory Control of Computerized Numerical Control Machines from External Python Program Using OPC UA," *International Research Journal of Advanced Engineering and Science*, Volume 9, Issue 3, pp. 30-35, 2024.

International Research Journal of Advanced Engineering and Science
ISSN (Online): 2455-9024



(a) Method 1 with $F = 500$ mm/min



(b) Method 2 with $N = 7$ and $F = 700$ mm/min

Fig. 3. Trajectory of the X-axis during the d_{min}/feed rate influence test of method 1 (a) and 2 (b).

A. *Results*

As predicted, repetitions of the same trajectory parts occur when the feed rate is too high compared to the distance between two points of the path. Method 1 demonstrates a limiting value of 0.15 mm for d_{min} when the feed rate equals 500 mm/min. Fig. 4 displays the measured dependencies for method 1. The curve presents a standard deviation of 0.053 mm to the theoretical curve. This may be due to the limited number of measurements made for each of the combinations. As demonstrated earlier in the speed test, transmission times are variable. A longer time on a passage, where the consecutive points are very close may cause an error. This explains the variability of the errors observed.

Method 2 does not exhibit any repetitions in the trajectory no matter how small the distance between the points is. The experiments were interrupted when arriving to a minimum distance of 0.001 mm at a feed rate of 600 mm/min because the smallest distance between two points in the G-code from which the list had been generated, had been underbid. Testing with a smaller d_{min} would in reality not change the list of coordinates. In this concrete example, the program must write the coordinates of four points during a period of time determined by the minimum distance between three consecutive points. Looking at the trajectory, this distance d_{min}^N is actually equal to 0.1 mm and not $3 \times d_{min} = 0.003$ mm. With the additional time of 40 ms caused by the synchronization pause, the time available is 50 ms, which is enough to write four points. As all curves produced under the conditions of (3) are valid, this inequality is verified, but it is possible to considerably improve trajectory accuracy by knowing the real value of d_{min}^N and taking into account the synchronization time.

Finally, to verify (2), a similar test has been executed varying δ with $d_{min} = 0.2$ mm and $F = 400$ mm/min. All resulting curves with δ smaller than d_{min} show errors. Only after a value larger than 0.15 mm the trajectory was sure to be correct. For security reasons, it is therefore advisable to add a safety coefficient of at least 1.5 on the lower side of the inequality in (2) in order to expect a viable outcome.

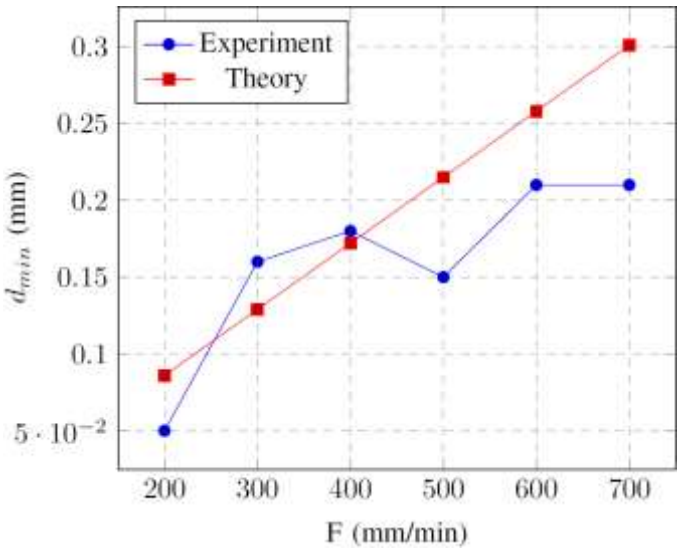


Fig. 4. Dependency between F and d_{min} for method 1.

VII. Discussion

Even if the machine is now able to cover all the points given to it in the right order and without repetition, there remains the problem due to the pause time. Unfortunately, "STOPRE" is the only way to actualize the R-parameter while the program is running. Another solution suggested by Siemens is to use the online tool compensation ($TOFF). Simultaneously to the movement, the tool size would be modified without having to synchronize the program with "STOPRE". Thus, the trajectory would be completely smooth, but the basic trajectory would have to be set from the beginning and the linear interpolation from point to point would no longer be given. Instead, there is an unclear movement in the period between two different tool lengths. Similarly, online adaptation has been successfully achieved writing the $AA OFF variable, which creates a superimposed movement in whatever axis is selected. The only difference to $TOFF, is that the generated offset does not take

Sara Menetrey, Holger Schlegel and Martin Dix, "Trajectory Control of Computerized Numerical Control Machines from External Python Program Using OPC UA," *International Research Journal of Advanced Engineering and Science*, Volume 9, Issue 3, pp. 30-35, 2024.

into account the orientation of the tool. It compensates along the machine axes X, Y and Z. To do so, a synchronous action command, like the following, has to be added to the existing G-code:

IDS = 1 WHENEVER Condition DO $AA OFF[Y] = $R0

This is a viable solution to achieve for instance small adaptations in purpose of optimization, but it is not a feasible option when the goal is to define the entire trajectory while operating. Its disadvantages are the same as for the tool offset: no more real control over the trajectory, no interpolation, complex determination of the axis in which the offset is desired.

Another approach could be to use the file transfer method of the Siemens OPC UA server in order to import subprograms that would always have the same name but different content and to have the G-code again running in a loop, but this time executing the subprograms instead of reading the R-parameters. However, to actualize the subprogram, a "STOPRE" command must again be integrated into the G-code loop and the problem would be the same. Therefore, this is only a presentation variation of the already existing methods that have been tested, although its speed limitation could be different and would need to be analyzed.

To avoid excessive usage of the tools or even accidents, it is conceivable to adapt the trajectory so that the stops are at times when it is not dangerous to have them, for example, after the tool retracts. Additional retraction could even be implemented to circumvent this problem, but this would elongate the manufacturing time.

VIII. Conclusion

This work presents a successful way to control the trajectory of a CNC machine from an external source while moving and thus enables online adaptations. The external python code was able to communicate with the Siemens control using OPC UA. R-parameters and current positions were read and written via request-respond in order to generate the requested path. The speed of the OPC UA communication was measured and found to be close to the duration of the machine's interpolation cycle (4 ms/parameter). Writing several parameters at the same time using the inbuilt function of the Python "opcua" library was proved to be up to 23 % faster than writing each parameter one after the other. Requirements for faultless implementation involving the machine feed rate and the minimal distance between points of the trajectory have been established and verified by testing it with a SINUMERIK 840D sl CNC control. The system requires synchronization pauses of about 40 ms in order to read the modified R-parameter values. During this time, the feed rate falls down to zero. Depending on the requirements of the applications, there are possibilities to get round the problem by positioning the pauses thoughtfully. In application where this does not pose a problem, the presented solution is viable and allows on-line adaptation of the trajectory.

Acknowledgment

The authors acknowledge the support of the project "AI-supported, partially automated disassembly of traction batteries (KaDoTE)" that is funded by the German Federal Ministry for Economic Affairs and Climate Protection.

References

[1] D. Horváth and R. Z. Szabó, "Driving forces and barriers of industry 4.0: Do multinational and small and medium-sized companies have equal opportunities?," Technological forecasting and social change, vol. 146, pp. 119–132, 2019.

[2] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," Business & information systems engineering, vol. 6, pp. 239–242, 2014.

[3] O. K. Marc Bayer, "From individual solutions to an integrated concept: How companies can implement the digital factory," Industry 4.0: The importance of an overarching digitalization strategy, 2021.

[4] C. Martínez Ruedas, F. J. Adame-Rodríguez, and J. M. Díaz-Cabrera, "A low-cost'plug and play'connectivity and integration system for sinumerik cnc machines to join industry 4.0," Available at SSRN 4334474.

[5] "Opc unified architecture," international standard, International Electrotechnical Commission, TC 65/SC 65E - Devices and integration in enterprise systems, 2020.

[6] O. Foundation, "Opc foundation unified architecture." https:// opcfoundation.org/about/opc-technologies/opc-ua/.

[7] A. Martins, J. Lucas, H. Costelha, and C. Neves, "Cnc machines integration in smart factories using opc ua," Journal of Industrial Information Integration, p. 100482, 2023.

[8] G. Martinov, P. Nikishechkin, A. Al Khoury, and A. Issa, "Control and remote monitoring of the vertical machining center by using the opc ua protocol," in IOP Conference Series: Materials Science and Engineering, vol. 919, p. 032030, IOP Publishing, 2020.

[9] M. André, L. João, C. Hugo, and N. Carlos, "Developing an opc ua server for cnc machines [j]," Procedia Computer Science, vol. 180, 2021.

[10] D. Mourtzis, N. Milas, and N. Athinaios, "Towards machine shop 4.0: a general machine model for cnc machine-tools through opc-ua," Procedia CIRP, vol. 78, pp. 301–306, 2018.

[11] J. Rodrigues and E. Ribeiro, "Virtualization and optimization of processes in industry 4.0," in International Conference of Progress in Digital and Physical Manufacturing, pp. 197–205, Springer, 2021.

[12] L. Gui, T. Y. Ruan, Z. Z. Wang, A. C. Sun, and M. Xu, "Cnc online monitoring system based on internet of things," Advanced Materials Research, vol. 1079, pp. 672–678, 2015.

[13] R. Reiser, B. Thiele, T. Bellmann, P. Koch, and C. Walter, "Real-time simulation and virtual commissioning of a modular robot system with opc ua," in ISR Europe 2022; 54th International Symposium on Robotics, pp. 1–8, VDE, 2022.

[14] H.-I. Lin and Y.-C. Hwang, "Integration of robot and iiot over the opc unified architecture," in 2019 International Automatic Control Conference (CACS), pp. 1–6, IEEE, 2019.

[15] H. V. Madhyastha and C. Okwudire, "Remotely controlled manufacturing: A new frontier for systems research," in Proceedings of the 21st International Workshop on Mobile Computing Systems and Applications, pp. 62–67, 2020.

[16] P. Shicong, W. Guocheng, and T. Fuqiang, "Design and realization of cnc machine tool management system using internet of things," Soft Computing, vol. 26, no. 20, pp. 10729–10739, 2022.

[17] A. K. Philip Samuel, A. Shyamkumar, and H. Ramesh, "Industry 4.0-connected drives using opc ua," in Industry 4.0 and Advanced Manufacturing: Proceedings of I-4AM 2019, pp. 3–12, Springer, 2021.

[18] SIEMENS, "Sinumerik 840dsl/828d sinumerik access mymachine / opc ua configuration manual," SINUMERIK, vol. 146, pp. 119–132, 2018.

[19] S. Cavalieri and F. Chiacchio, "Analysis of opc ua performances," Computer Standards & Interfaces, vol. 36, no. 1, pp. 165–177, 2013.

[20] A. Veichtlbauer, M. Ortmayer, and T. Heistracher, "Opc ua integration for field devices," in 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), pp. 419–424, IEEE, 2017.