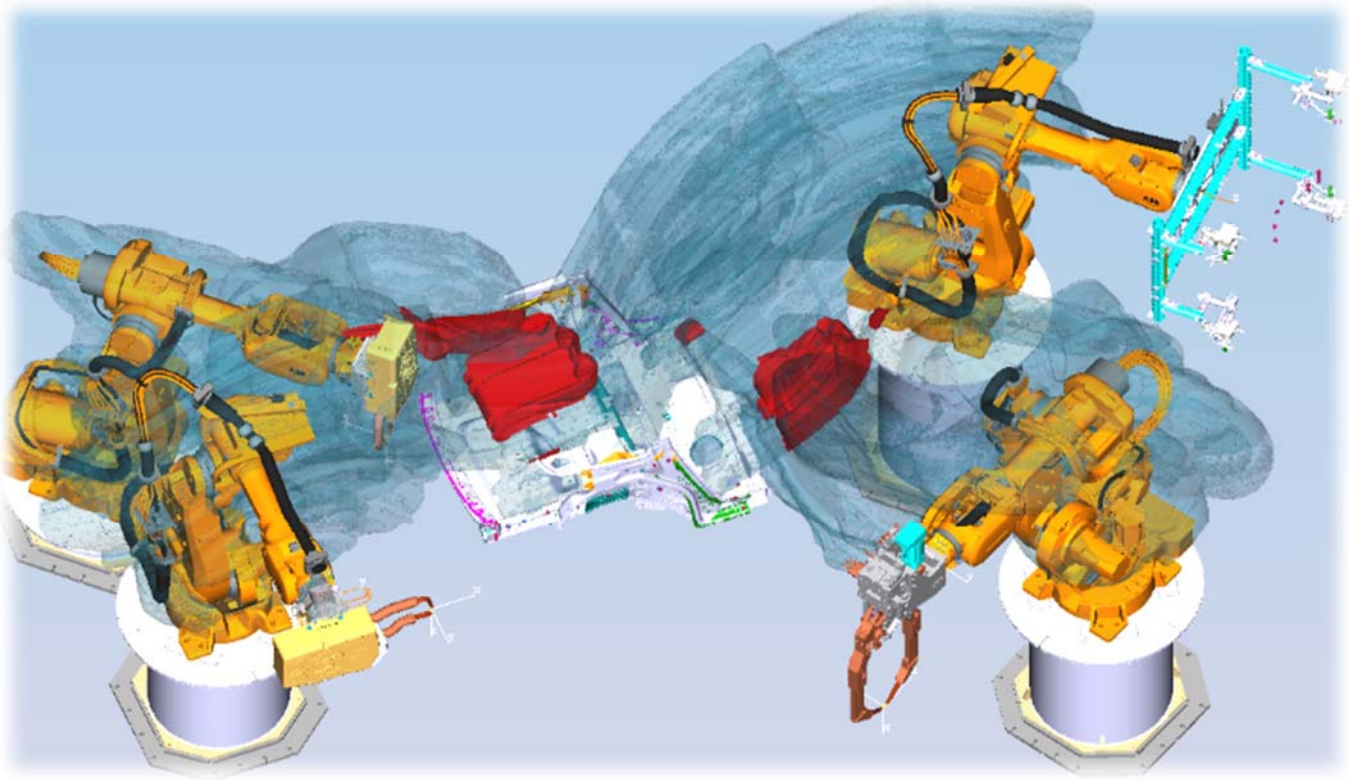
CHALMERS
UNIVERSITY OF TECHNOLOGY



Automated Robot Collision Avoidance
Using Virtual Commissioning and Formal Methods

Master's thesis in Production Engineering

LUCAS GARCIA ROMERO
KARL-OSCAR SKALBERG
Report no. EX079/2017

Department of Electrical Engineering, Division of Systems and Control
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Automated Robot Collision Avoidance using Virtual Commissioning and Formal Methods.

Lucas García Romero and Karl-Oscar Skalberg

Report no. EX079/2017
Division of Systems and Control
Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, 2017.

Cover: Screenshot from Process Simulate of four robots illustrating sweep and interference volumes in a robot cell.

Abstract

Some of the work tasks involved when creating a collision free robot cell could be conceived as boring, repetitive, and wasteful use of a simulation engineer. Due to this repetitiveness errors occur during programming, with unplanned stops of production as a result. Thus, the thesis aimed to automate parts of the workflow concerning creation and scheduling of interlocks in a robot cell and was conducted as a collaboration between Chalmers University of Technology (Chalmers) and Volvo Car Corporation (Volvo). The idea was to use the software Sequence Planner, developed at Chalmers, for scheduling of interlocks while creation of the interlocks was to be carried out in Process Simulate and be based on interference volumes. Thereby the engineer would be given greater freedom designing the paths and could reduce the time spent on routine tasks.

The steps in the workflow were identified from interviews with simulation engineers at Volvo and by following Siemens training material for "Intermediate Robotics" and "Virtual Commissioning" in Process Simulate. The developed plug-in creates interlocks by detecting at which checkpoints the robot configuration is closest to the interference volume and then writes signals and comments, which will be recognised by Volvo software, for allocation or release to these checkpoints.

Due to an oversight in what information is used to create schedules with Sequence Planner and limitations in the currently exposed functionality for Process Simulate, which prevented a fully automatic/one-click solution, the thesis was unable to meet its full aim. However, the code developed in this thesis shows potential as a basis for implementation of a recently published method as some of the method's current problems could likely be overcome or mitigated by using sweep volumes.

Keywords: Process Simulate, Sweep Volume, Interference Volume, Tecnomatix, Formal Methods, API.

Acknowledgements

Although the thesis could be considered the work of two persons the end result would not have been achieved without the insight and help from the following:

Professor Martin Fabian and Martin Dahl, PhD Student, at Chalmers University of Technology. Our thanks go to Martin Fabian as our examiner and supervisor – for giving us free reins when we wanted and directions when we needed. We would also like to thank Martin Dahl for his help with Tecnoviewer and Process Simulate.

Henrik Carlsson, company supervisor, and Henri Hansson, both Simulation Experts at Volvo Cars. Thank you, Henrik, for your help with getting us started at Volvo, (somewhat) bending Process Simulate to our will, and connecting us with the right people. One of those people, who we particularly want to mention is Henri Hansson. Thank you, Henri, for your valuable insights and attention to details that otherwise might have gone overlooked.

Naemi Jönsson and Johan Asklund, our thesis co-workers. Your help and assistance with Tecnoviewer and the configuration of Sequence Planner saved us many hours of troubleshooting.

Lucas would like to thank family and friends for keep on cheering up when the work was not progressing or when things did not go as expected.

Karl-Oscar would like to thank my girlfriend, Paulina, my family and friends for being there through good and bad times alike. My thanks also go to CIRC-2010, for giving me an interesting start at Chalmers, and last, but not least, to Pyrot, for all the fun and fireworks.

Definitions

- Process Simulate – 3D robot simulation software owned and developed by Siemens
- Sequence Planner – Software capable of generating optimized sequences based on certain input. Developed by Chalmers University of Technology.
- Tecnomatix – Collective term for software in Siemens PLM suite. Process Simulate is a part of this.
- SDK – Software Development Kit, typically a set of software development tools that allows the creation of applications for a certain software package.
- API – Application Programming Interface, a set of subroutine definitions, protocols and tools for building application software.
- IDE – Integrated Development Environment, a software application that provides comprehensive facilities for software development to computer programmers.
- OLP – Off-line programming (regarding robotics).
- GUI – Graphical User Interface.
- PLC – Programmable Logic Controller.
- Via-location – Point in the work space where the robot has specific joint values that is used to create the robot path.
- OLP command – Format/text sensitive command that is linked to a via-location. Used at Volvo for creating signals (zone allocation, tip dressing....)
- Interlocks – Set of signals that are implemented to avoid collision between robots.
- Sweep Volume – Volume generated by the movement of the robot when going from one via-location to another.
- Interference Volume – Volume resulting from the intersection of two sweep volumes from different robots.
- Shared Space/Zone – Space between via-locations of a robot path that is determined by an interlock.

Table of Contents

1 Introduction

This chapter aims to give an introduction to the thesis by providing more details of its background, purpose, and the research questions that were sought to be answered.

1.1 Background

In the field of manufacturing engineering there is a constant struggle to reduce the time from the product development or engineering phase to start of production. One of the practices currently being developed to accomplish this is Virtual Commissioning [1]. Virtual Commissioning encompasses methods to design, verify, and validate changes to a manufacturing process before the change has been implemented in the actual, physical, manufacturing system. Thereby is it possible to detect, among other things, scheduling errors leading to deadlocks, non-optimal resource utilization, and overlapping work space(s) before a problem has occurred, thus reducing unplanned costs and ramp-up time [2]. An example would be that service schedules sometimes have been forgotten when performing the collision avoidance check in a robot cell. This has resulted in a collision when the cell was run after it had been implemented.

This thesis will target the latter of the problems listed. Today it is possible to, from the robot simulation software, calculate the volumes that the robot sweeps while following its trajectory. However, in the case of multiple robots working near each other, verifying if the sweep volumes overlap is often done manually by the simulation engineer. Where such an overlap is found the simulation engineer then has to decide how access to this "shared resource" is distributed among the robots in order to avoid them trying to occupy the same physical space at the same moment in time (i.e. collide). The thesis will aim to reduce the manual labor needed while at the same time reduce the risk of code errors by automating this workflow [3].

The thesis was conducted at Chalmers University of Technology (Chalmers) in collaboration with Volvo Car Corporation (Volvo). At Volvo they have long experience of offline programming and using virtual methods to plan and validate changes to their production system [4], [5] . Since having a well-functioning production line is crucial for the company's survival they have an interest of making sure that their methods are up to date.

In addition there were two other theses under the VirtCom[1] project. The project aims to further advance and develop how Virtual Commissioning can be implemented in automotive industry. Additionally, the thesis should benefit manufacturing industry as a whole by making the progress made in the larger corporations accessible for small and medium sized enterprises. The other two theses, which were conducted during the same time as this thesis, focused on how to optimally schedule tasks in a manufacturing process [6] and how a Virtual Commissioning project should be specified, respectively [7].

1.2 Purpose

The thesis aims to create a plugin for the software Sequence Planner (SP) with the simulation software Process Simulate (PS) that can help coordinate robots working in a shared space. The plugin should use the sweep volume data from PS to generate resources, which are then fed to and prioritized by SP before being returned to the plugin which generates the corresponding code for access to the resources in PS. This should lead to fewer faults in the code and overall shorter lead time when designing robot cells.

[1] www.virtcom.se

Further, the project aims to examine whether there are other tasks that are now handled manually in Process Simulate which could be solved by the same or a similar plugin as the one developed for handling the sweep volumes. The overall goal would still be to reduce the repetitiveness that the task(s) currently hold and enable the engineer to use his or her talent in a more creative and value-adding manner [3].

1.3 Objective

The main objective of the thesis is to facilitate the design process of a robot cell by automating the code generation for access to shared resources, i.e. detection and designation of intersecting sweep volumes generated by two or more industrial robots. This improvement on the design process of the robot cell will be achieve by combining two existing functions on Process Simulate: Minimal distance between to objects (robot and interference volume) and writing OLP commands. This will generate a script that will run through all the interference volumes, find when the robot gets inside/outside of it and write it in the form of an OLP command.

Thereby enabling the engineer to focus on the optimisation of the behaviour of the individual robots, thus simplifying the creation of complex robot cells in manufacturing industry. As it is right now, the tasks of the simulation engineer are manual and repetitive, as such they sometimes end up in errors when implementing the shared resources. The solution will change the task of the engineer, from having to make the full implementation, to doublechecking the automatically generated code.

In addition to above objective the following research questions were formulated:

1. What is the result of having two or more robots generate the interference volume?
2. Are there other possibilities for Sequence Planner and Process Simulate interaction?

1.4 Delimitations

To keep the thesis focused and within reasonable limits the following delimitations are made:

- No investigation is made into how to generate sweep volumes or any further development of such methods.
- The development is primarily aimed towards facilitation of the current way of work and the integration of Process Simulate (version 13) and Sequence Planner.
- If a function or tool in Process Simulate is not publicly available the thesis will not seek to develop a similar function or tool based on available commands.
    - Example: At the time of this writing it is possible to create sweep volumes by using commands available through the Process Simulate's API. However, no similar functions for interference volume creation are made available.
- Only the models provided by Volvo will be used.
    - The model(s) provided was not without its own limitations, see Chapter 3.2.2.1 for details.

1.4.1 Conditions

This thesis could be seen as an adaption of [8], however, there are some key differences that separates this thesis from the previous. First, whereas the former thesis was using RobotStudio this thesis uses Process Simulate. Further, the former thesis had to develop the volume generation where this thesis can use the tools included in Process Simulate. Additionally, an end goal of the former thesis was to implement the solution in the robot cell available in the Production Systems Laboratory at Chalmers.

1.4.2 Clarification of terms

The code or plug-in developed throughout the thesis will be referred to as "ARCADE", short for "Automatic Robot Collision Avoidance". Tools already present in Process Simulate will be referred to as "tools". The term "functionality" will be used to describe the elements which make up a tool in Process Simulate, essentially that which the Siemens Developers has used to create the "tools". Examples of included functionality are most easily found in the Tecnomatix SDK (Chapter 2.4.1.1) whereas tools are found by running Process Simulate. As such, ARCADE will use and be based on included functionality but ARCADE will not be a tool (as it has been custom made and is not normally included in Process Simulate).

2 Theoretical background

This theory chapter goes into detail on some of the terms and theories used in virtual commissioning. The aim is to provide the reader with some of the fundamental knowledge required to comprehend the thesis.

2.1 Simulation

The act of simulating was described as "*the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system*" by Shannon [9]. Examples of simulation could, by this definition, be 3D-simulation or Discrete Event Flow Simulation [10]. The use of simulation will likely grow given the increased digitalization of production systems and simulations ability to reduce ramp-up time, contributing to create flexible production systems [11].

2.1.1 Formal Methods

The use of formal methods is a way to validate an entire model by proving logical properties of the model, often used in programming to validate mathematic models and later be able to implement them as code. Thus, it is possible to reduce the simulation and traditional testing as formal methods will help to verify the correctness of the methods and more importantly prove the absence of faults [12]. When implementing virtual commissioning, formal methods can be a tool to reduce time spent in the development phase [13].

2.1.2 Discrete Event Systems

To facilitate the application of formal methods (and thereby benefitting from a proven absence of errors) in a production system the system could be modelled as a Discrete Event System (DES). A DES is a system which, for both the system and its subsystems, could be represented as being in different states at specific (discrete) time intervals. These states are symbolic-valued, meaning that a subsystem (e.g. a machine, product, or operator) could be considered to be, for example, *Operating*, *Idle*, or *Out of Order*. This is different from modelling time-driven continuous systems, for example regulating temperature or the "cruise control" in a car, where a real-value is monitored and the process continuously controlled in relation to this [12].

A DES, even when continuously controlled, will transition between states as the result of events in the system. These events could in turn be modelled as resulting from transitions of other subsystems, "occurring asynchronously at discrete points in time" [12]. As such the traits of the system could be considered based on the chain of events, with respect to the order of the events included in each chain [12]. This allows a DES to be used as an abstraction of a real system, for which formal methods can be used to prove the correctness of the logical properties.

2.2 Shared space

One of the limitations and factors to optimise for when designing a robot cell is the physical space the cell occupies as well as the shared space between the robots. This is due to the considerable costs associated with expanding a building compared to the costs of relocating equipment as well as the savings realised by reducing travel time of products, material, personnel, and costs related to housing (e.g. heating and maintenance). Ideally, each robot should have unhindered access to the workpiece and as such be able to move along an optimal path without interruptions. In reality this will seldom, if ever, be the case. In reality it will be necessary to place robots in close proximity of one another, thus creating a situation where robots might collide due to overlapping workspaces.

2.2.1 Creating Zones

One method to avoid collisions is to create zones that have mutually exclusive access for the robots, thus acting as interlocks. A supervisor, generally a PLC, is then responsible for delegating access to one robot at a time. In this thesis the zones are based on interference volumes. An interference volume is defined as the intersection or overlap, of two sweep volumes. By using this definition, it is ensured that the shared zones are as small as possible, even though there ideally should not be any shared zones.

2.2.1.1 Deadlock

In the field of robot engineering the term deadlock is used for the situation where two or more robots get into a locked position, caused by the logic in the PLC not allowing them to continue on their set path. This is usually caused by a poor or incomplete implementation of the access to the shared zones. As the first and most important task is to avoid collision between robots, and this can sometimes lead to a deadlock, where robots will not collide but the cell will get paralyzed because of the logic. To avoid this the interlocks are generated.

2.2.1.2 Interlocks

The interlocks are a set of signals and logic that are both collision. This allows the creation of a robot cell that is fully functional. The only downside when implementing interlocks is that it requires longer time for implementation. However, the trade-off is (in the best case) having 0 errors once the implementation is done.

2.3 Robotics

This chapter aims to give a brief introduction to robotics, allowing the reader to more easily grasp the concepts later introduced in the thesis. Shown in Figure 1 are some basic concepts that will be used throughout the report. The “working envelope” is the volume which the robot can reach with its “end-effector”.

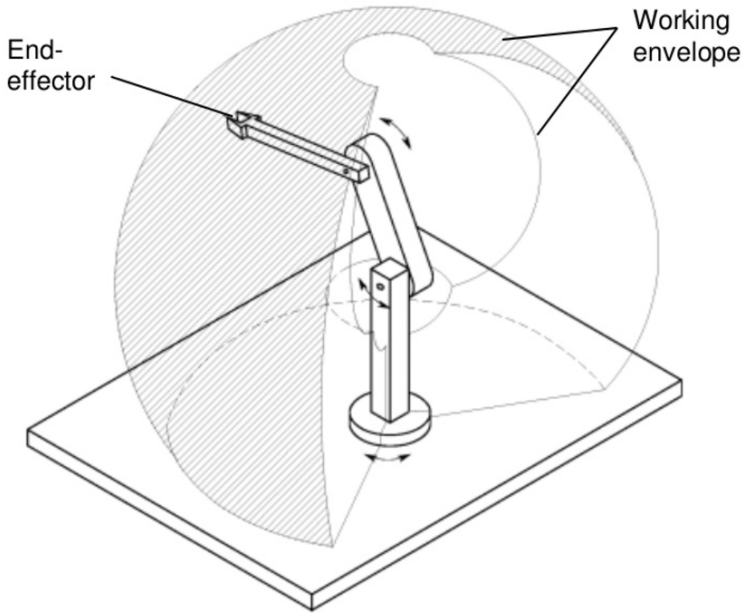


Figure 1. Basic robot concepts. [14]

2.3.1 Paths

The robot paths are generated by joining a group of via-locations among the reachable volume, the working envelope, of the robot. The via-locations are specified according to the position of the reference system of the robot, for example the Tool Centre Point (TCP). The TCP will then move between the locations following the assigned path, using either linear or joint movement depending on the user's programming. The TCP is often, but not always, programmed to be at the end-effector.

2.3.1.1 Zone Property

Another aspect of the robot movement and its path is how the robot reaches each via-location. Some of the via-locations could be far from where the collision might happen, which means that the robot does not need to be super precise going exactly through that point. On the other hand, when any signal needs to be read on the via-location the zone needs to be set at "fine" which means that the robot will stop to read the signal from the PLC and then continue if possible.

2.3.2 Controllers

The robot controllers are used to convert all electrical signals to the motion control (position, speed acceleration) of the machine. Different types of controllers will allow handling of more signals and therefore a more complex movement and positioning of the robot. A generic controller could be used when absolute accuracy is not necessary. However, when simulating a robot it is important to have the correct (usually developed by the manufacturer) controller to ensure that the motions you see in the simulation will be the same as when the program is loaded to the actual robot.

2.3.3 Sweep volume

When moving the robots through their paths, the volumes generated as the result of moving between the via-points are referred to as the "sweep volumes". This volume is used during the simulation to ensure the correct interaction between robots and other static parts of the cell, mainly used for checking pairs of colliding objects, to create and simulate a collision free cell.

2.3.4 Interference Volume

Interference volume is called to the overlap of two sweep volumes. This is used to easily picture in the simulation environment where robots might collide. On this project the interference volume is used to specify how close the robots are from it while moving. Once that is known access to the interference volume needs to be controlled, which is what interlocks are for. At the end the code must be such that it guarantees mutual exclusive access to the interference volumes, while at the same time avoiding the deadlocks.

2.4 Programming and software

The very nature of the thesis meant that programming was an integral part of it. Some software has been used to a larger extent and are therefore presented below. In addition to these, other widely adopted programs have been utilised, for example Microsoft Word, OneNote and Excel.

2.4.1 Process Simulate

The software Process Simulate is owned and developed by Siemens PLM [15]. Process Simulate can, among other things, be used to develop manufacturing processes virtually, thus being able to change the design of a product or layout of a production line before these are created physically. This gives benefits in a number of activities, including but not limited to,

ramp-up time, reduced number of trial runs, and collision avoidance in robot cells. The most interesting application to this thesis has been the ability to simulate robotics using the actual robot controllers, i.e. as a virtual commissioning tool.

To be able to create customised tools incorporating Process Simulate functionality the C# API available for development was accessed using Microsoft Visual Studio. This means that even though there was a lot of work within Process Simulate most of the tasks involved C#-programming in Visual Studio.

2.4.1.1 Tecnomatix SDK

Documentation on the commands available for Process Simulate was found through the Tecnomatix SDK. The SDK is available from within the installation folder of Process Simulate. In addition to browsing the SDK, questions regarding implementation and use of commands were posted on the Tecnomatix Developer Forum.

2.4.1.2 Calculation of sweep and interference volumes

One of the key dependencies of the thesis was Process Simulate's ability to calculate the volumes generated by the robot as it moves along its path (Figure 2). Due to Process Simulate only using one thread of the processor, calculation of multiple sweep volumes at the same time is not possible and this resulted in a time-consuming operation. This was not elaborated on since improving the calculation was not the objective of the thesis and validation of the tool in Process Simulate was deemed to be enough (i.e. comparing the motion of the robot to the calculated volume).

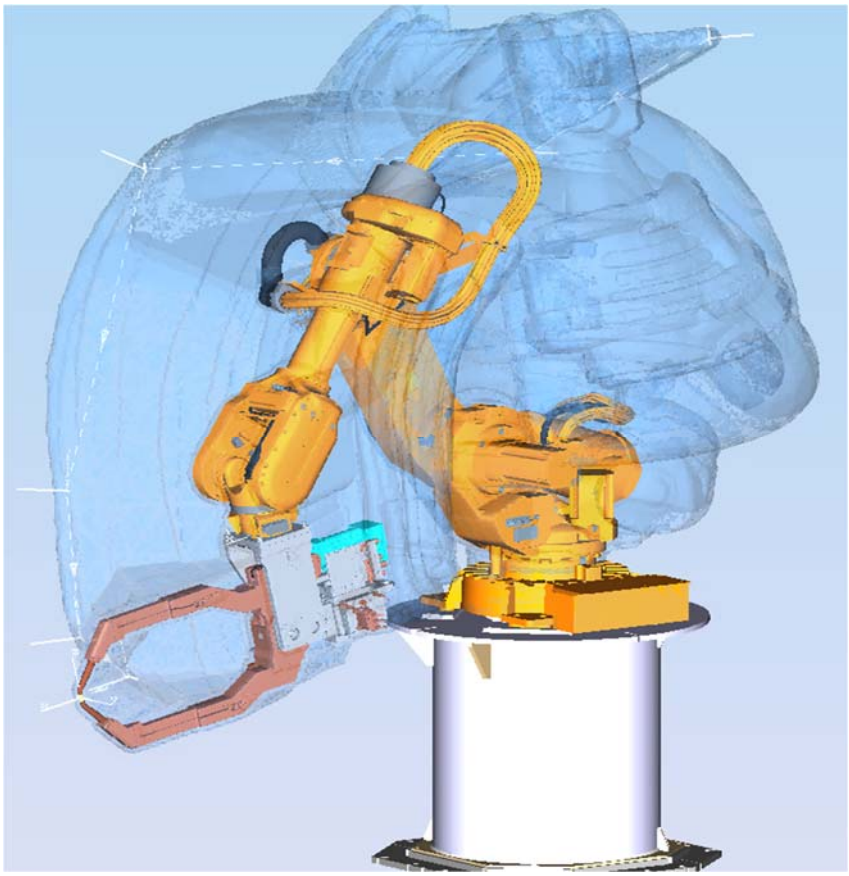


Figure 2. Robot with sweep volume. The white coordinate axes represent via-locations and the dashed white lines illustrate robot path and direction.

After calculating sweep volumes for the paths of two or more robots, Process Simulate is able to calculate the interference volume as the space where the sweep volumes intersect each other (Figure 3).

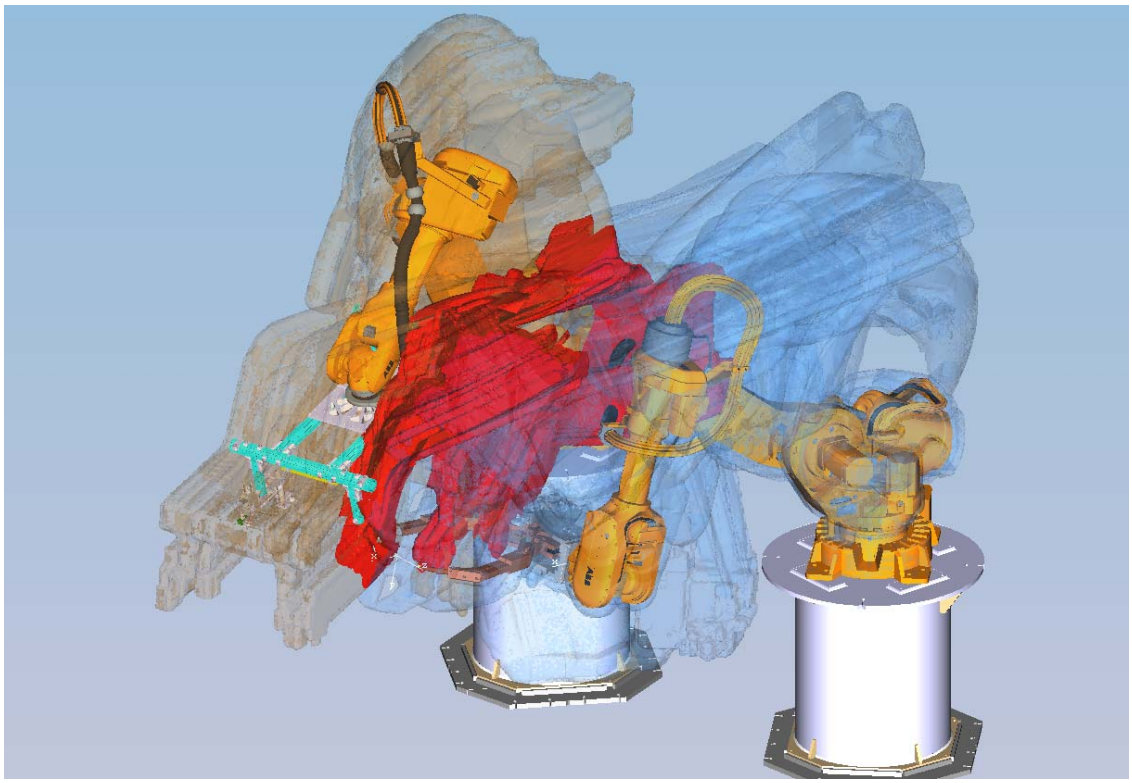


Figure 3. The red solids mark the intersections, i.e. interference volumes, of the sweep volumes (shaded).

2.4.2 Sequence Planner

Building on the experiences from Supremica, and incorporating its main functions, the software Sequence Planner is developed at Chalmers. Sequence Planner allows, as hinted by its name, the (optimised) scheduling of sequences while utilising control theory to avoid deadlocks.

Sequence Planner was developed as a tool to help with visualization of complicated sequences, as well as optimizing, verifying and simulating operation sequences on any solution. One of the strengths of Sequence Planner is its architecture and the way it is used for integrated virtual preparation and commissioning. Some examples of application of Sequence Planner go from Runtime control, energy optimization, online monitoring or Emergency department patient planning [16].

2.5 Ethics

As with almost all instances of automation there is a risk of taking away work opportunities. This is also the case for this project as it aims to automate parts of the current workflow. Therefore, there exists a risk of incurring *Ironies of Automation* [17]. However, this can be weighed against the usefulness provided by automating an otherwise error prone task, given that it is error prone due to the task being conceived as boring and repetitive [18].

There is also a risk of the ability being used to create more complex robot cells as means for "unnecessary" reduction of the number of operators in a plant. In addition to this there is a risk for using the simulation to find the minimum number of operators needed to operate the cells

(e.g. if the cycle time is increased due to lower demand). This risk might be acceptable given the consequences of having the plant closed due to high operating costs (of which personnel constitutes a considerable amount).

As such it can be concluded that automation is a solution that enables a company to remain competitive without having to resort to the actions mentioned above. Thus, again providing a compelling argument as to why automation is necessary.

2.6 Sustainability

The term "sustainability" has several definitions. One of these is commonly referred to as "triple bottom line" or "people, planet, profit", highlighting the need to consider more than one factor in order to truly be sustainable. The thesis affects these factors in the following way:

2.6.1 People

The "people" aspect refers to being socially sustainable. That is, simply put, to not have negative effects on the employees or the society at large. The results of the thesis can be considered to be beneficial to the employees since a potentially boring work task is automated, allowing them to make better use of their creativity. The risks first identified in the "Ethics" chapter apply here as well, as does the conclusion.

2.6.2 Planet

Environmental sustainability is captured under the "planet" factor of the triple bottom line. The aim of the thesis will result in an opportunity to create more complex robot cells and at the same time take energy optimization into consideration. Thus, reducing the effect of the manufacturing process. By improving the efficiency of virtual commissioning, it is possible to increase environmental sustainability by minimizing waste incurred by trial runs and design mistakes.

2.6.3 Profit

Last, is the necessary economical sustainability represented by the "profit" factor. For the thesis this factor benefits from the previous two in that they lead to a better use of the available resources. Thus, by doing more with less (or the same), the profit margin will be improved. In addition, the thesis will further increase the benefits brought by Virtual Commissioning, for example reduced ramp-up time, thereby ensuring the survivability of the company and ensuring that it can continue to contribute positively to the people and the planet.

3 Methods

The methodology chapter further describes what was done and why it was done in the thesis. The chapter is split into subchapters categorising the different actions according to if they could be considered to be more academic, practical or implementational in nature.

3.1 Academic

The thesis took inspiration from a methodology called "action research". The idea was to, at least initially, use the steps laid out in "*The Good Research Guide"* [19, Ch. 5] as a source of inspiration. Action research was deemed as a suitable method due to the nature of the problem (practical) and that it will allow for reflection of the process as it is carried out. In addition, the research was carried out in close proximity to the people affected by the results, and they were also able to give feedback as progress was made, which further suggests that action research is a suitable method.

3.1.1 Literature Review

In preparation of the project a literature review was carried out in order to establish an understanding of current progress in the area. To quickly assess the importance and scientific value of articles the Scopus Field-Weighted Citation Impact (FWCI) score was used for comparison. FWCI takes into account the year of publication, document type, and disciplines associated with its source. FWCI is calculated as the ratio of the article's citations to the average number of citations received by all similar articles over a three-year window. Each discipline contributes equally to the metric, thus eliminating differences in citation behaviour. A FWCI score greater than 1.00 means that the article is more cited than would be expected [20]. Since the knowledge on the studied area at the beginning was little, there was a need of relying on article metrics. In practice this helped sifting out the more impactful articles from the search results. The articles with a comparatively high score were also given more attention than those with a lower score.

3.1.2 Comparison with BoxSweeper

As stated in the Conditions (Chapter 1.4.1), this thesis could be considered a continuation of the work conducted by Bengtsson and Rutgerson [7]. As such it was deemed to be of interest to make a more thorough comparison of the two theses. In addition to the differences already mentioned (robot simulation software, volume calculation method, and end goal) a few others were identified and are discussed below.

To start, the previous thesis focused more on the development of a plug-in (BoxSweeper) whereas this thesis has integrated the functionality existing in Process Simulate into a tool. Additionally, BoxSweeper included work for creating the robot paths, which were pre-determined in this thesis. BoxSweeper also developed a PLC supervisor. The latter was not considered for this thesis as the goal was to be able to create "optimal" schedules through external software (Sequence Planner). The degree of involvement of the simulation software developers (ABB and Siemens, respectively) also seemed to differ between the theses, where the previous thesis seemed to be working in more close collaboration with the developers.

In the case of the previous thesis one of the goals was to create an environment to be used as part of a bachelor thesis. Thus, the work was carried out towards a cell which the (BoxSweeper) authors were able to influence. This was not the case for this thesis, as the robot cell against which ARCADE was developed and tested had not only already been designed but also installed and put into operation.

Since the volume generation is now based on high quality sweep volumes rather than simplified, although more easily calculated, boxes it is likely to believe that the mutually exclusive zones can be made smaller and more accurate than in the previous thesis. This would then enable more precise control of robots which could reduce energy consumption and cycle times. However, it should be noted that the previous thesis had its focus on if (the less computing intense) boxes could achieve the same level of functionality as sweep volumes and was able to automate the writing of RAPID-code for the robots.

3.2 Practical

Information on how the design and programming of a robot cell is conducted was gathered in two separate ways. The first way was through interviews with simulation engineers at Volvo [18]. The second way was by undertaking courses in “intermediate robotics” and “virtual commissioning” respectively. The courses are part of Siemens’ training material targeted at users of the Tecnomatix suite.

3.2.1 Process Steps

Based on the gathered information the following steps were used as basis for developing the code. "Step 0" is named as such as this is a prerequisite for any collision avoidance (no paths → no movement → no collisions).

0. Generate the robot path and operations.
    a. Path design is done by the simulation engineer.
1. Create sweep volumes of the path and operations.
    a. Using tools included in Process Simulate.
2. Calculate interference volumes (i.e. where sweep volumes intersect each other).
    a. Using tools included in Process Simulate.
3. Determine the last via-location before a robot enters and the first position after a robot exits an interference volume.
4. Use the via-locations to create conditions of when a robot can access the interference volume.
5. Feed the robot programs and conditions to Sequence Planner for scheduling.
6. Update the robot programs based on the prioritisation generated by Sequence Planner.

Explanations on how the result of each step would be validated are included in more detail at the end of each step.

3.2.2 Designing the model

The model on which ARCADE was tested and developed against was provided by Volvo, see Figure 4 for an overview of the cell. This is a model of an actual cell responsible for the production of 90-series cars in the body shop at the plant in Torslanda. The cell includes several robots working in close proximity to each other as well as several moving fixtures, including two turntable fixtures. The robots perform various operations, such as spot welding, pick and place, and glue application. A PLC is responsible for supervising the cell and controlling its different functions.

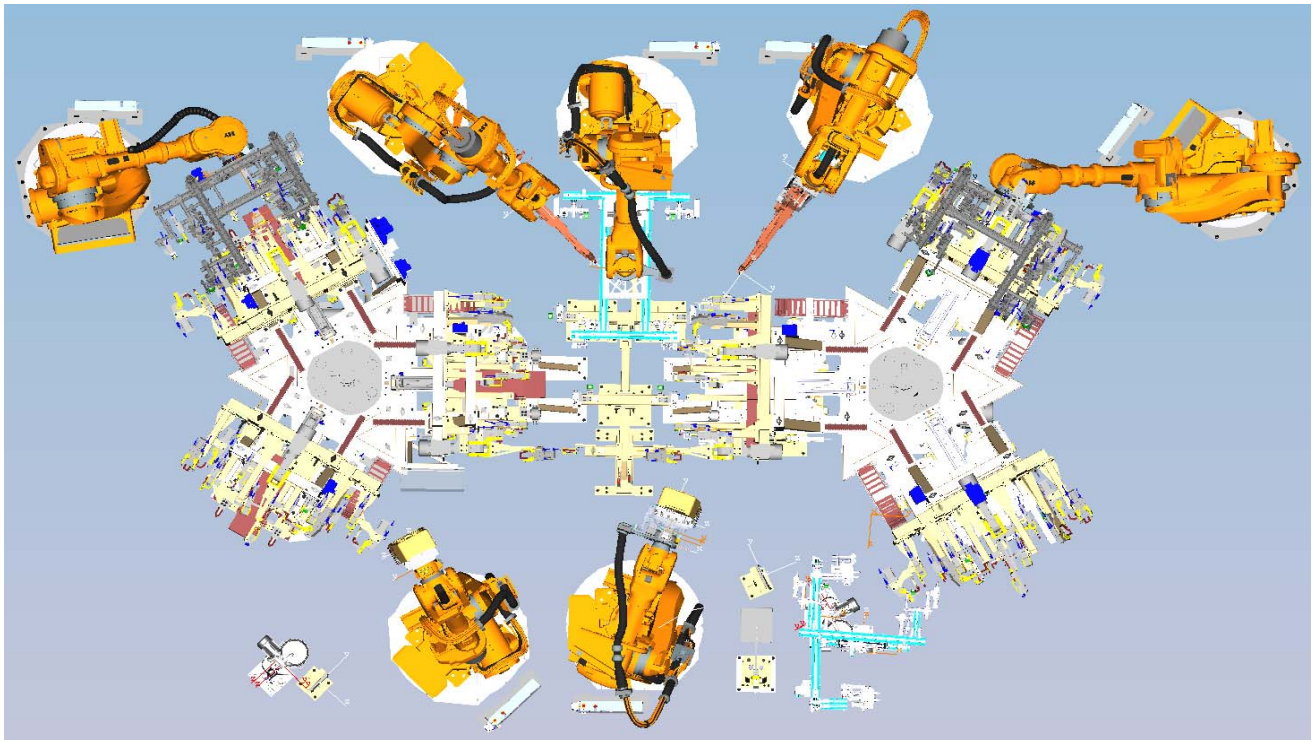


Figure 4. Top-down view of the central parts of the robot cell that was supplied by Volvo.

3.2.2.1 Limitations of the model

ARCADE will likely be most useful in the design phase when creating a robot cell. However, the model that was provided had not only already been designed but also installed. This meant that zones had already been generated and robots (including their programming and positioning) had been adjusted in a way that avoided collisions. Thus, a majority of the work was carried out on a model that in many cases had little room for improvement in the areas that the thesis will concern. For example, it was seen as good practice to try to place robots further apart, or change their paths, if their sweep volumes collided. By doing so one avoids having to create zones altogether. Due to the limitations mentioned in Chapter 2.2 this might not always be possible, hence the need for zones.

Further, the robot programs in the model included cases. This meant that a robot program, depending on the signals sent from the PLC, could perform very different operations. An example of this would be the possibility for a pick and place robot to place parts on a buffer or directly into the fixture. Due to the complexity of the levels in the operations tree and the cases not having a defined structure, which made it even more complicated to automate. That together with Sequence Planner only being able to handle straight sequences of operations (at the beginning of the thesis, alternative sequences were develop later) was the determinant to only focus on straight sequences for this thesis work.

3.2.3 Including customised commands

In order to add ARCADE to Process Simulate it was necessary to register it. In short, this meant moving the .dll-file generated from Visual Studio into a specific folder in Process Simulate's installation directory and then running the "CommandReg.exe". The steps followed were the same as is illustrated in the video available from Siemens PLM [21].

3.3 Development

The writing and development of ARCADE has been an iterative process. Siemens is still developing Process Simulate and therefore the functionality that is not fully developed is not available for users of the SDK. This was confirmed by Siemens when they were asked, per e-mail, about how to create interference volumes from code. As an example, writing a tool for creating a sweep volume is possible however using two sweep volumes to create an interference volume was not possible.

3.3.1 Step 0 – Generate robot path and operations

The task of creating paths and operations for the robot will still belong to the simulation engineer. In the provided model this had already been done. Since the objective of the thesis is to provide greater freedom during this step this was not further elaborated on. The step was included in the list to clarify the current working process.

Validation of Step 0

Step 0 was not validated as ARCADE assumes that the robot programs are correct. Performing optimization on robot programs that are not correct would most likely be a waste. As such, this data was assumed to be correct.

3.3.2 Step 1 – Create sweep volumes of the path and/or operations

Once the paths have been created it is possible to calculate the sweep volume generated by the robot as it travels along its path. Although the functionality to calculate sweep volumes is exposed through the Tecnomatix SDK it was decided to use the included "Automatic Interference Tool" (AIT) (described in more detail below, see section 3.3.2) since this also automatically calculates the interference volumes from the sweep volumes. The reason for this was that the functionality to create interference volumes (see next step) was not exposed through the SDK.

When using the AIT the "max error" can be set by the user as a way to prioritize between accuracy and calculation speed. A smaller error yields longer calculation times but might be necessary when there are tight tolerances in the model. For a faster computation the user can select a larger error at the expense of finer details, for example movement of the cables or the actions of the tool attached to the robot.

Ensuring volumes are calculated correctly

In spite of the fact that functionality for calculating sweep volumes is included in Process Simulate it was still necessary to ensure that the volumes were calculated correctly. From interviews it was identified that one source of error was when the path of a robot had been divided into smaller segments [18]. One reason for segmenting an operation is to avoid collisions between robots, thus this error could be added to the list of limitations of the model. Examples of such segments are shown in Figure 11.

The segmented operations caused the sweep volume calculation to miss parts of the robot's movement, thus the resulting volume was not representative of the actual movement. To remedy this, ARCADE was developed to copy via-locations from the last position of the previous segment to the first location in the following segment, as shown in Figure 5 below.

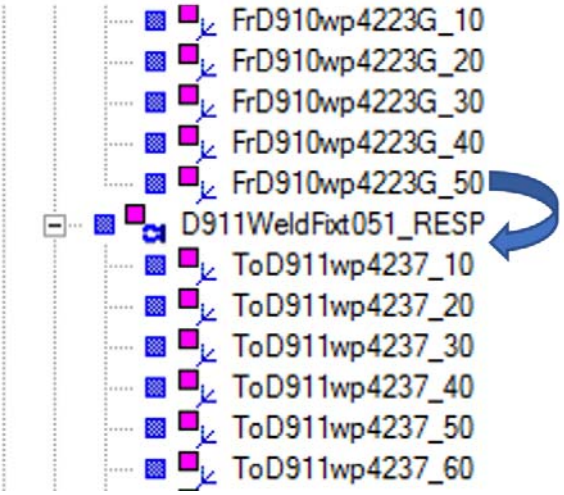


Figure 5. The via-location "FrD910wp4223G_50" should be copied and then pasted above "ToD911wp4237_10".

This step might not be needed if the robot program does not contain any segmented operations. However, since tests were only carried out on the provided model the step was necessary to achieve accurate results. Thus, by solving this special case ARCADE becomes more applicable as the general case, without segmented operations, is simpler.

Increasing calculation speed

From the interviews with the simulation engineers another interesting tweak surfaced, namely how to increase the calculation speed when calculating sweep volumes using Process Simulate [18]. Originally, the calculation tool was developed for an older software (Robcad), which has since been incorporated into Process Simulate, and the calculation tool therefore uses only one thread of the CPU. Hence, the process will be slow even on modern multi-core CPUs. One way to increase speed despite this is to have the operating system give greater priority to the process.

This can be done by, given a Windows environment, opening the Task Manager, selecting the tab "Processes", right-clicking "SweptVolumeEngine.exe", "Set Priority", and left-clicking "High" (Figure 6). However, this process is killed and restarted for every volume calculation, meaning that the procedure has to be repeated for each calculation. To remedy this a simple executable was created.

When run, the executable starts with asking the user how many volumes that are being calculated (Figure 7). The executable then proceeds to search for the process "SweptVolumeEngine.exe", which handles the sweep volume calculation. If the process is found, the executable changes its priority through a console command. If the process is not running, the executable will wait a number of seconds before searching again. This continues until either of a) the priority has been increased for the number of processes the user provided as input or b) 100 consecutive searches has failed to find the process. The latter ensures that the script stops running even if it misses a volume or the calculations are interrupted for some reason. Thus, allowing the simulation engineers to run their calculations at an increased speed without having to manually increase the priority of each calculation.

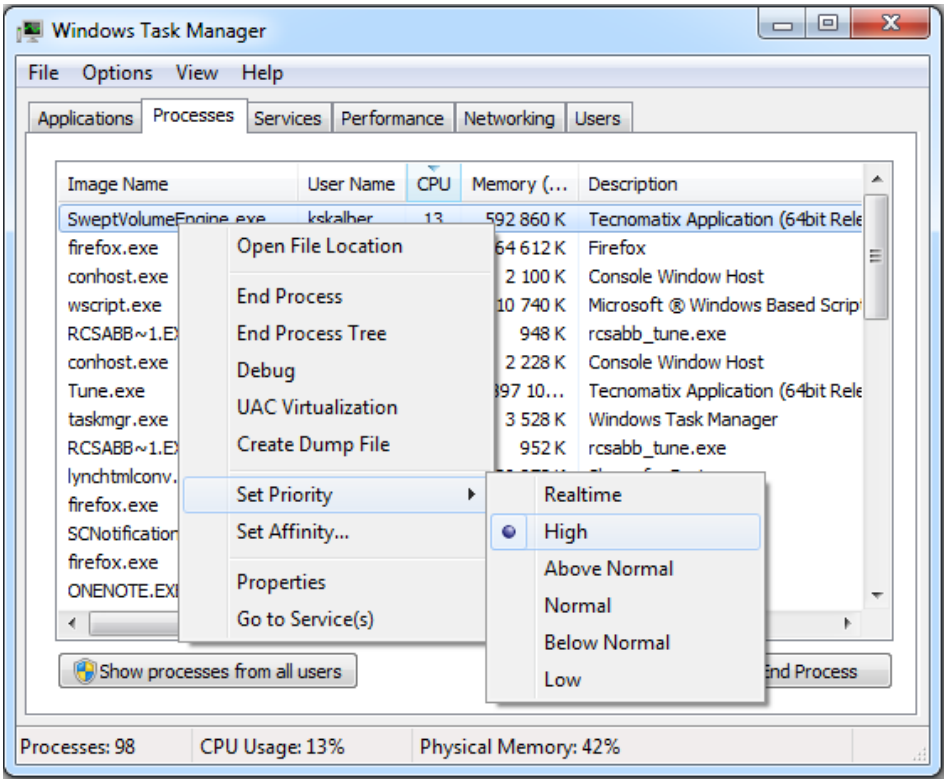


Figure 6. Manual method for raising priority of the process. Could also be used to verify that the program succeeds in raising the priority.

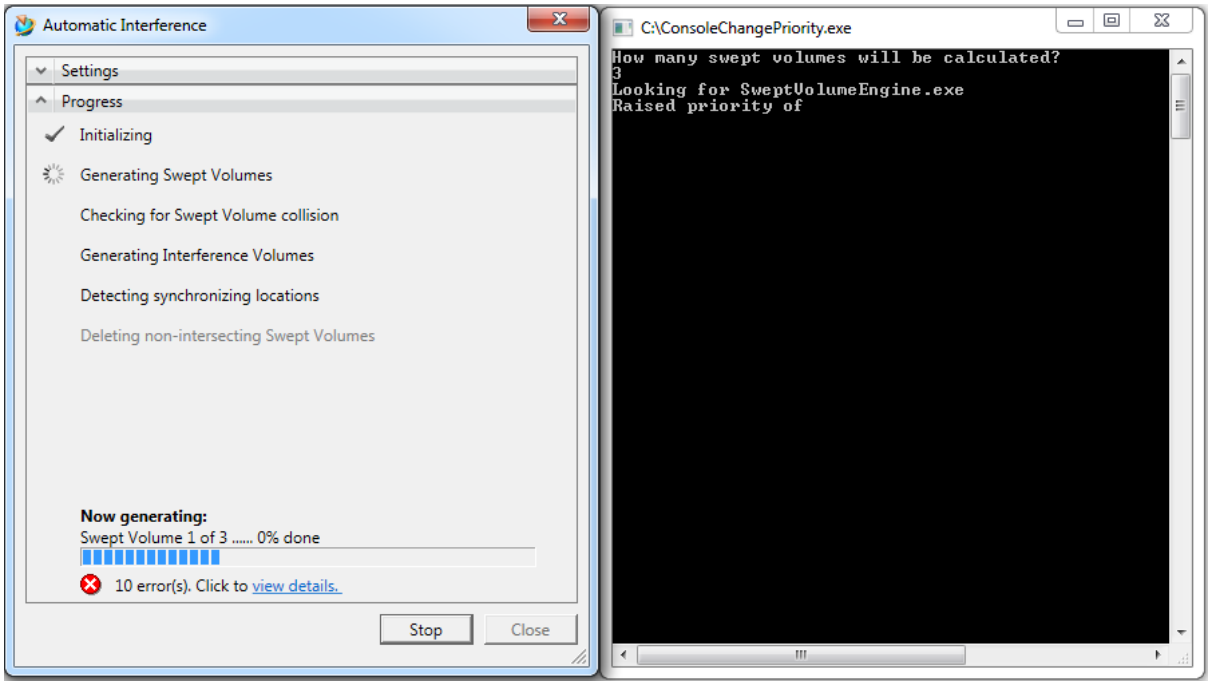


Figure 7. Left half shows the Automatic Interference Tool while running. Right half shows the program that raises the priority of the volume calculation process.

The increased calculation speed was tested by triggering the "Automatic Interference Tool" in Process Simulate eight times and raising the priorities of the processes for four of the runs. Before each run the computer was restarted to minimize the risk of the calculations affecting each other. The results are available in section 4.1.3.

Validation of Step 1
To validate Step 1 the motion of the robot will be compared to the generated sweep volume, using the function "Move Robot to" in Process Simulate. This function moves the robot continuously rather than the more direct "Jump Robot to". The latter function has some usefulness on its own, though, as will be elaborated on in chapter 4.3.

3.3.3 Step 2 – Calculate interference volumes
After having created the sweep volumes it is possible to calculate the interference volumes.

Automatic Interference Tool
Although the Automatic Interference Tool (AIT), which is readily available in Process Simulate, can be used to create sweep volumes its main function is to calculate where sweep volumes intersect. Figure 8, below, shows the user interface of the tool. Using the buttons in the top row it is possible to load operations or robot programs for which interference volumes will be calculated.

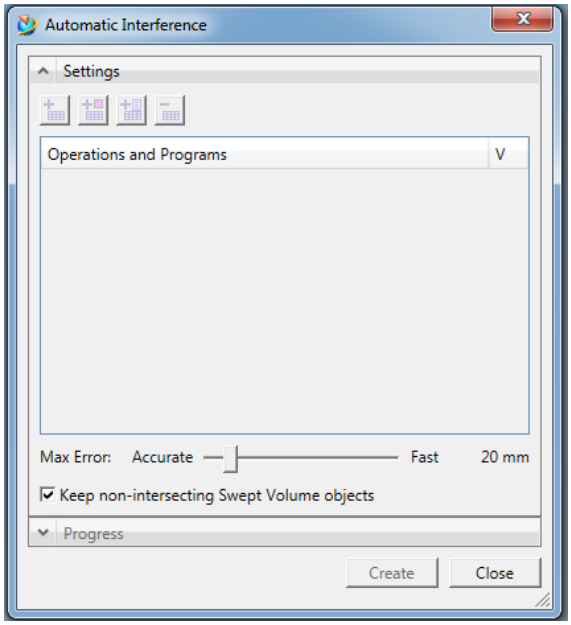


Figure 8. GUI of the Automatic Interference Tool available in Process Simulate.

Validation of Step 2
The generated interference volumes can be verified visually by displaying both of the sweep volumes that make up the interference volume and then comparing their intersection with the interference volume.

3.3.4 Step 3 – Determine entry and exit points
In order to define the shared resources, it is necessary to determine which via-locations are visited just before and after the interference volumes, also known as the entry and exit points of the interference volume. By developing a separate detection method it is possible to use the results of the already present AIT as a reference. If the user notices differences between where ARCADE and the AIT places entry and exit points there is likely a reason to be concerned. More importantly, a separate tool (ARCADE) needed to be developed in order to customize the output as AIT only wrote one of two generic strings of text as OLP-commands to the via-location.

The method to determine entry and exit points would use the included functionality for finding the minimal distance between two objects in Process Simulate. The objects between which the distance is measured would then be the robot (including all attachments) and the interference volume.

Validation of Step 3
Step 3 was validated by comparing the output generated by ARCADE with both the output generated by AIT and the zone allocation that was already implemented (by Volvo) in the cell. Thus, if ARCADE wrote the OLP-commands to the same via-locations as AIT it would be a sign that ARCADE was working as intended. Further, as the studied model already contained segments that had been implemented to avoid collisions, it would be seen as reassuring if the commands were similar to these segments.

3.3.5 Step 4 – Creation of conditions for access

After having defined the entry and exit points, the conditions for access to the interference volume were tied to the via-locations using OLP-commands. By default, AIT writes the OLP-commands "*Last location before Interference Volume*" or "*First location after Interference Volume*", respectively. In contrast, ARCADE should write commands such as "Allocate Zone*X*" and "Release Zone*X*". The signals should follow the standard used by Volvo at the time of writing.

OLP commands will then be added to these points in order to require access (allocate) or return access (release) the zone. AIT, detailed above, adds similar information when used. However, AIT is limited to predetermined output such as "*Last location before interference"* or "*Last location in interference"*. An example of what this could look like is shown below, see Figure 9 and Figure 10.

ToD910wp4209_10	
ToD910wp4209_20	!Allocate Zone 19 between 050R02 and 050R01;
ToD910wp4209_30	
ToD910wp4209_40	
D910wp4209	
D910wp4207G	
FrD910wp4209_10	
FrD910wp4209_20	
FrD910wp4209_30	!Release Zone 19 between 050R02 and 050R01;
FrD910wp4209_40	
FrD910wp4209_50	
FrD910wp4209_60	
FrD910wp4209_70	
FrD910wp4209_80	
FrD910wp4209_90	
HomeGun311	!Release Zone 5 between 050R02 and 050R01 !Release Zone 10 between 050R02 and 050
D911SchDefault	" - 'D911WeldFixt051_GEO'; " - 'D911WeldFixt051_RESP';

Figure 9. Example of OLP-commands assigned to via-locations in a segment.

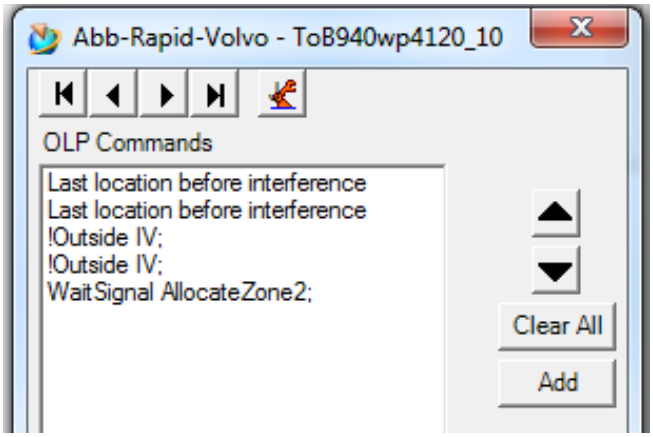


Figure 10. Example of OLP-commands when viewed through the Teach Pendant in Process Simulate.

Validation of Step 4
Since Step 4 to a large extent used the functionality from Step 3 the validation methods were similar. However, in Step 4 it was also necessary to check so that ARCADE wrote the correct type of commands depending on the via-location. That is, for an entry point ARCADE should write "WaitSignal AllocateZone*X*" and for the exit point it should write "ReleaseZone*X*".

3.3.6 Step 5 – Transfer of data to Sequence Planner
After the generation of the conditions the data should be transferred to Sequence Planner for treatment. There already existed a plugin, Tecnoviewer, for Process Simulate which was used for this task.

Validation of Step 5
Step 5 could be validated by seeing that data had been imported into Sequence Planner. The data was then manually compared with the information in the model, such as name of robots and segments, to ensure that the correct data that had been transferred.

3.3.7 Step 6 – Updating robot programs with results from Sequence Planner
The last step would be to update the robot programs according to the sequences generated by Sequence Planner. However, Sequence Planner reads the robots' programs in order to find the robot sequence (movement, alternatives, zone allocation, etc,), and ARCADE only modifies the OLP commands but not the robot programs. The reason for this being that Volvo creates their robot programs in this way (first placing via-locations, then finding entry and exit points to create segments which are used for the final robot programs) and the development of ARCADE aimed to follow in-house practice. Hence the connection between Process Simulate and Sequence Planner could not be finalised. It was possible to see that Sequence Planner could detect the data but in the current version of Sequence Planner there is no functionality to handle the data.

Validation of Step 6
Although it was not possible to update the program in the current model this had already been tested with RobotStudio, as was mentioned during the literature review. Further validation of Step 6 was not conducted.

4 Results and analysis

This chapter contains the results obtained in the thesis. The results have been split into sections according to at which step of the process they first occurred. This includes the results from experiments, discoveries, and the continuous development of the code.

4.1 Step 1 – Creating Sweep Volumes of the path and/or operations

The first step resulted in that the sweep volumes were calculated correctly and at an increased speed.

4.1.1 Ensuring volumes are calculated correctly by copying via-locations

When ARCADE successfully copies the last via-location from the previous segment to the first (top) position of the next segment. These are the via-locations shown in rectangles in Figure 11, seen below. The first via-location shown in a rectangle was copied from the previous segment, "B940WeldSeg2". This is realised by comparing the name of the via-location with the previous segment, "wp" in the name stands for "weld point" and the previous segment is indeed a weld segment ("WeldSeg").

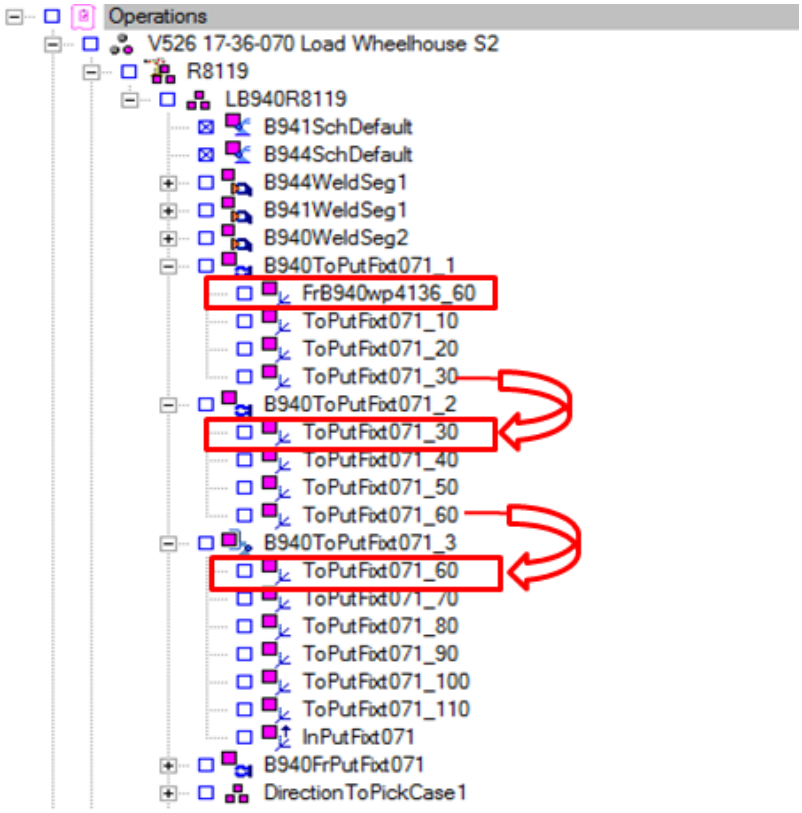


Figure 11. The via-locations, shown in rectangles, were copied to the subsequent segment.

4.1.2 Creation of sweep volumes

Calculating the sweep volumes of the segments with modified via-locations yielded the differences illustrated in Figure 12 (before) and Figure 13 (after). The calculations used the Sweep Volume tool already available in Process Simulate.

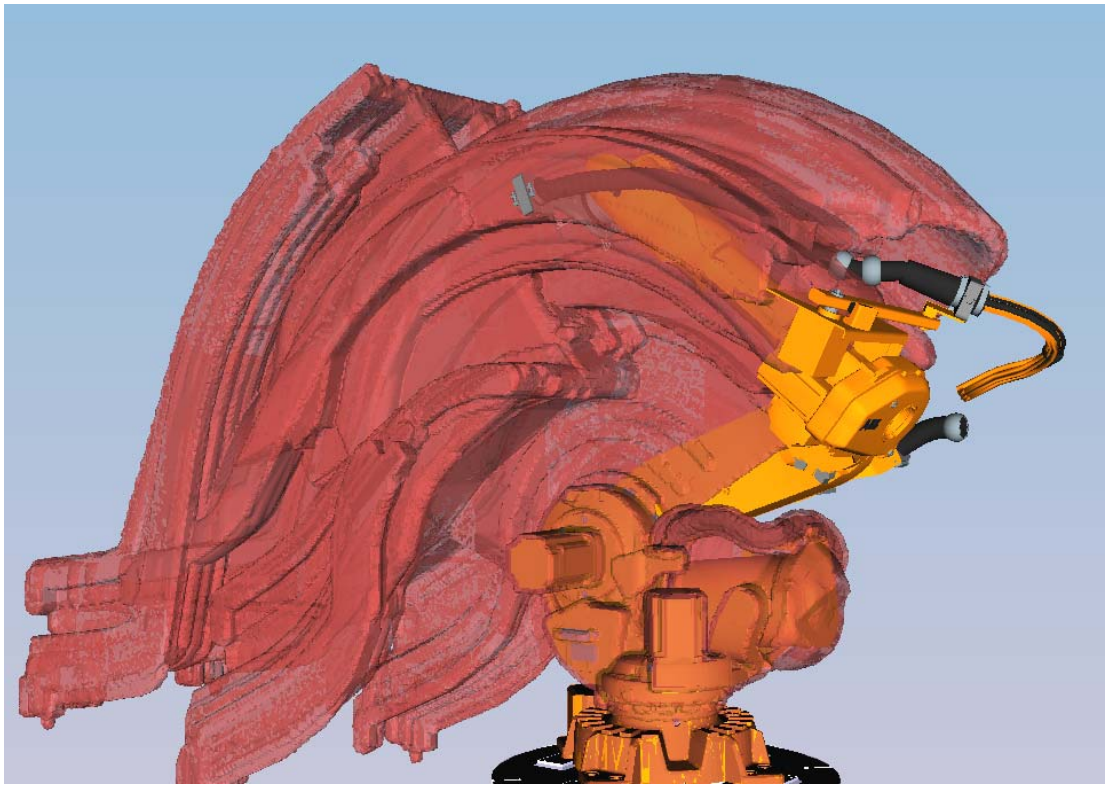


Figure 12. Sweep volume without adding the via-location from the previous segment.

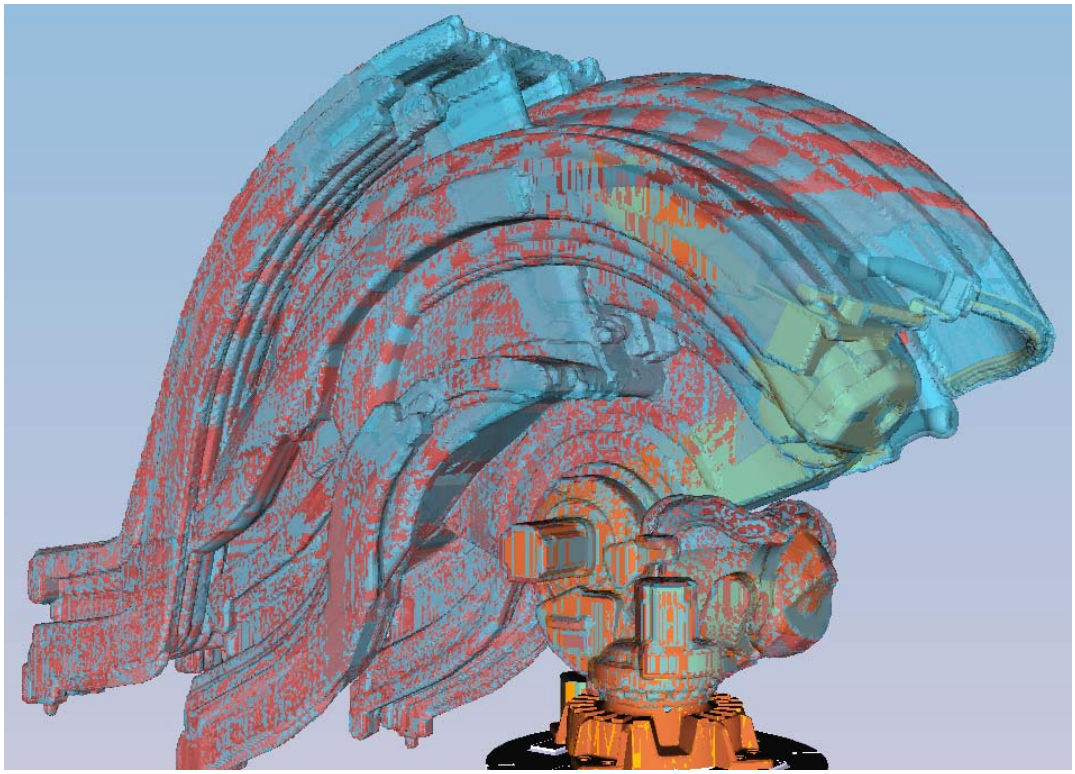


Figure 13. The blue sweep volume includes the last via-location from the previous segment. The red sweep volume is the same as in Figure 12.

4.1.3 Increased Calculation Speed

Using the developed script for increasing the priority of the process rendered the results shown in Table 1. "w. old volumes" means that any existing volumes were kept before calculating new sweep volumes whereas "w/o" meant deleting all previously generated volumes before calculating the new ones.

Table 1. Results from tests of volume calculation with and without raising the priority of the calculation process.

VALUES IN SECONDS, CALCULATED WITH 20 mm MAX ERROR		
Robots:	R8122 & R8123	R8124 & R8125
w. old volumes	612	975
w. old volumes + priority	584	934
Change	**-4,58 %**	**-4,21 %**
w/o Volumes	624	968
w/o Volumes + priority	595	937
Change	**-4,65 %**	**-3,20 %**

To summarize, increasing the priority of the process decreased the calculation time by 3,20 % in the worst case and 4,65 % in the best case. The remaining two cases saw a decrease of 4,21% and 4,58%, respectively.

4.2 Step 2 – Calculation of interference volumes

The resulting interference volumes from the Automatic Interference Tool were validated by using the path editor and moving the robots between each via-location with the "Move robot" command. Hence, the interference volumes were manually matched against the movement of each robot. See Figure 3 for an example of the interference volume between two robots in the cell.

4.3 Step 3 – Determining entry and exit points

The resulting entry and exit points were validated visually by jumping (using the "Jump Robot to" command available in Process Simulate) the robots to the locations determined by ARCADE and examining these for collision between interference volume and any part of each robot. Once the robot has been moved to the next location in its path ARCADE evaluates the distance between the robot (including tooling and some of its static parts) and the interference volume. Figure 14 shows an example of the distance between a via-location and an interference volume.

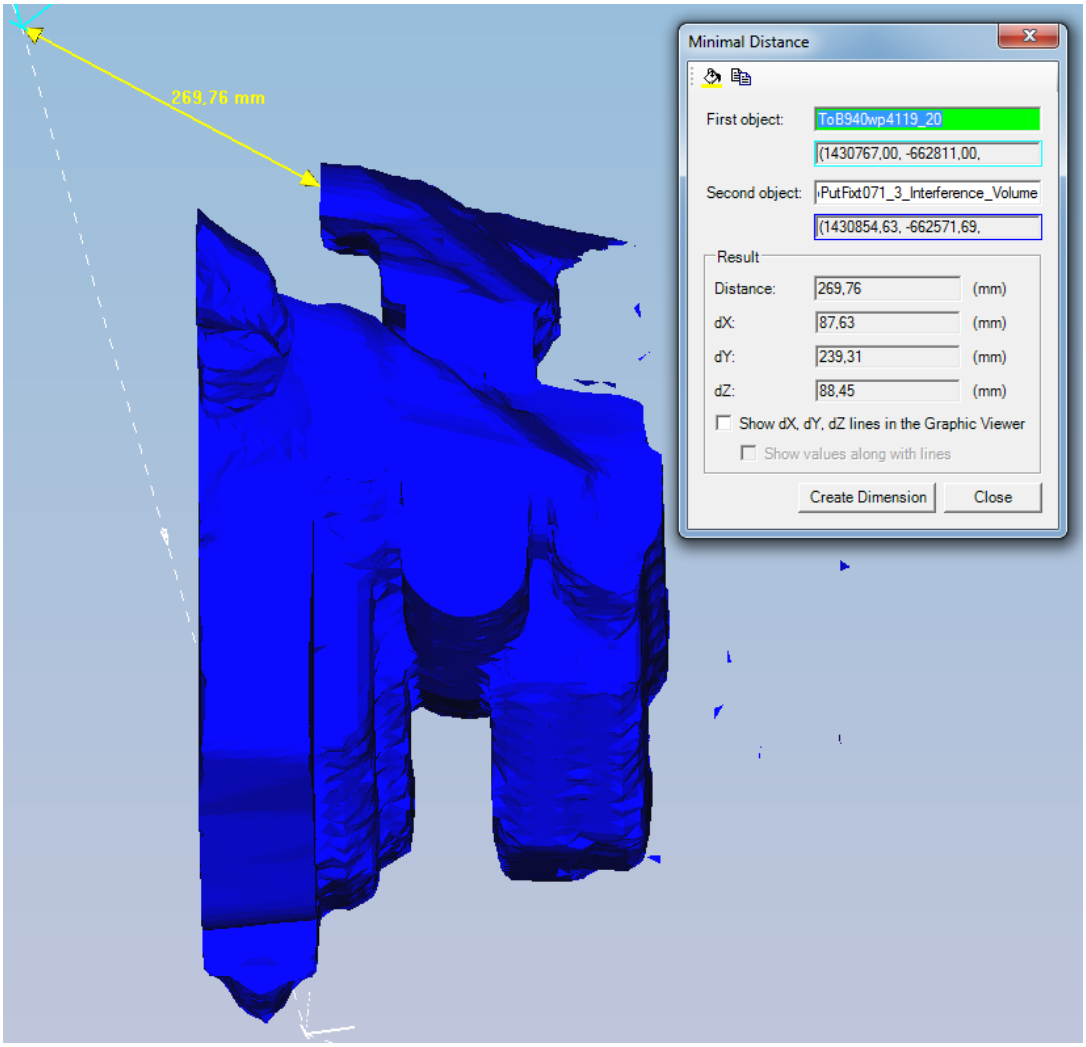


Figure 14. Using the included tool "Minimal Distance" on a via-location and an interference volume.

In the code below, Figure 15, the initialisation of the robot services can be seen from line 1 to line 7. Between lines 8 and 10 the jump is carried out. To be able to view the change after the movement the screen then needs to be updated, which is done using the "RefreshDisplay" method. This is only useful for the user, as Process Simulate will calculate the correct distance even if the display is not updated. While developing ARCADE it was seen that sometimes the result when using “JumpTo” was not the same as when the user did “Move to”. This happens because the “jump” command does not reproduce the real movement of the robot while the command “move” does. This did not affect the end result (zone generation) during testing.

```
1  TxOlpControllerUtilities utility = new TxOlpControllerUtilities();
2
3  ITxRoboticControllerServices services = utility.GetInterfaceImplementationFromController(Robot.Controller.Name, ...
4  typeof(ITxRoboticControllerServices), typeof(TxControllerAttribute), "ControllerName") as ITxRoboticControllerServices;
5
6  TxJumpToLocationData data = new TxJumpToLocationData();
7  services.Init(Robot);
8  services.JumpToLocation(ViaLoc as ITxLocationOperation, data);
9
10 TxApplication.RefreshDisplay();
```

Figure 15. Initialisation of robot and jump to location.

When the robot has been moved into position, the code of Figure 16 will be run. This section of code takes care of measuring the minimal distance between the robot (including all attachments) and the interference volume. Lines 1 and 2 select the robot arm and the above levels in the object tree until the selection includes the robot, its tooling, and the static parts (this is done using the "Collection" property). After this comes the measuring (line 3) between the interference volume (referred to as "IV_Location" in the code) and the robot, tooling, and static parts (referred to as "ViaPointLocation1"), this measurement is saved to "ActualDistance". Both parts of the code (Figure 15 and Figure 16) are run for every via-location.

```
1  ITxLocatableObject ViaPointLocation = Robot.Collection as ITxLocatableObject;
2  ITxLocatableObject ViaPointLocation1 = ViaPointLocation.Collection as ITxLocatableObject;
3  IV_Location.GetMinimalDistance(ViaPointLocation1, out ActualDistance, out PointOnIV, out PointOnViaPoint);
```

Figure 16. Code for retrieving the minimal distance between robot and interference volume.

For a more comprehensive view of the investigated via-locations all output is written to a .csv file (*Distance.csv)* which is saved to the user's "Documents" folder. The columns of the file include the name of the interference volume, the participating robots (Robot 1, Robot 2), the examined via-location, the OLP-command that is written, and last the zone number that is or would be used. Each interference volume is evaluated twice, first with the path of Robot 1 and the second time with Robot 2. See Figure 17 for an example of the output.

Interference Volume	Robot 1	Robot 2	Via-location	Status/Signal	Zone #
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToPutBuffer083_80	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToPutBuffer083_30	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToPutBuffer083_40	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToPutBuffer083_10	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToPutBuffer083_90	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToPutBuffer083_50	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToPutBuffer083_60	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToPutBuffer083_70	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	InPutBuffer083	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	FrPutBuffer083_10	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	FrPutBuffer083_20	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	FrPutBuffer083_30	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	FrPutBuffer083_40	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	FrPutBuffer083_50	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	FrPutBuffer083_60	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	HomeGripp421	!Outside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5211G_3	WaitSignal AllocateZone1;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5211G_5	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	B940wp5211G	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5199G_1	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5199G_2	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5199G_4	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5213_60	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	B940wp5199G	WaitSignal ReleaseZone1; @	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp4561G_2	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp4561G_3	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp4561G_4	WaitSignal ReleaseZone1;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp4561G_5	!Outside IV; @ WaitSignal A	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp4561G_6	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	B940wp4561G	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5423G_1	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5423G_3	WaitSignal ReleaseZone1;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5423G_4	!Outside IV; @ WaitSignal A	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5423G_5	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	ToB940wp5423G_6	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	B940wp5423G	!Inside IV;	1
IV_SV_R81-22-17-36-070	R81-22-17-36-07	R81-23-17-36-070F	B940wp5427	WaitSignal ReleaseZone1;	1

Figure 17. Screenshot of Distance.csv. Headings added for clarity. Due to page width restrictions some columns are not shown fully expanded.

4.4 Step 4 – Creation of conditions for access

Validation of the conditions created was done by comparing the results of the developed method with those of the "Automatic Interference Tool" (Figure 18 and Figure 19). The conditions are created by checking if the robot configuration, when jumped to a certain via-location, is "inside" or "outside" the interference volume. Once the "status" of the via-location is known the code continues to examine whether or not the pair of robots involved in that interference volume have any previously created zones and if so re-uses that zone number for the pair. Thus, the number of zones required is lowered. The code does not take into consideration any limit on the number of zones allowed. Currently, the code neither considers the zone type (e.g. "fine", "z1", "z2") of the via-location. In order to successfully read the command the zone type will have to be set to "fine". This can be done manually in Process Simulate or by implementing the "Zone Property" of the *TxRoboticLocationOperationMotionParameters* Class.

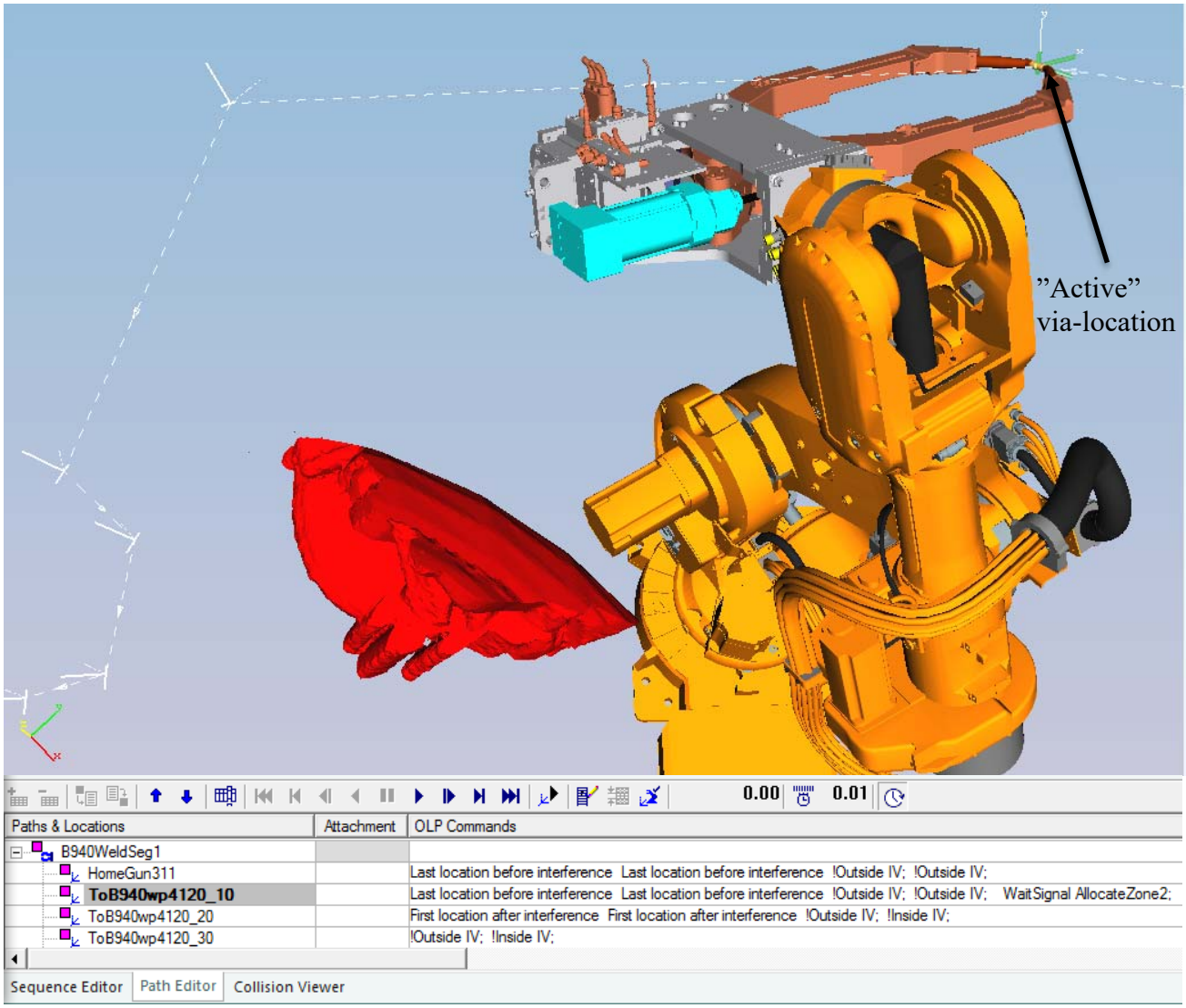
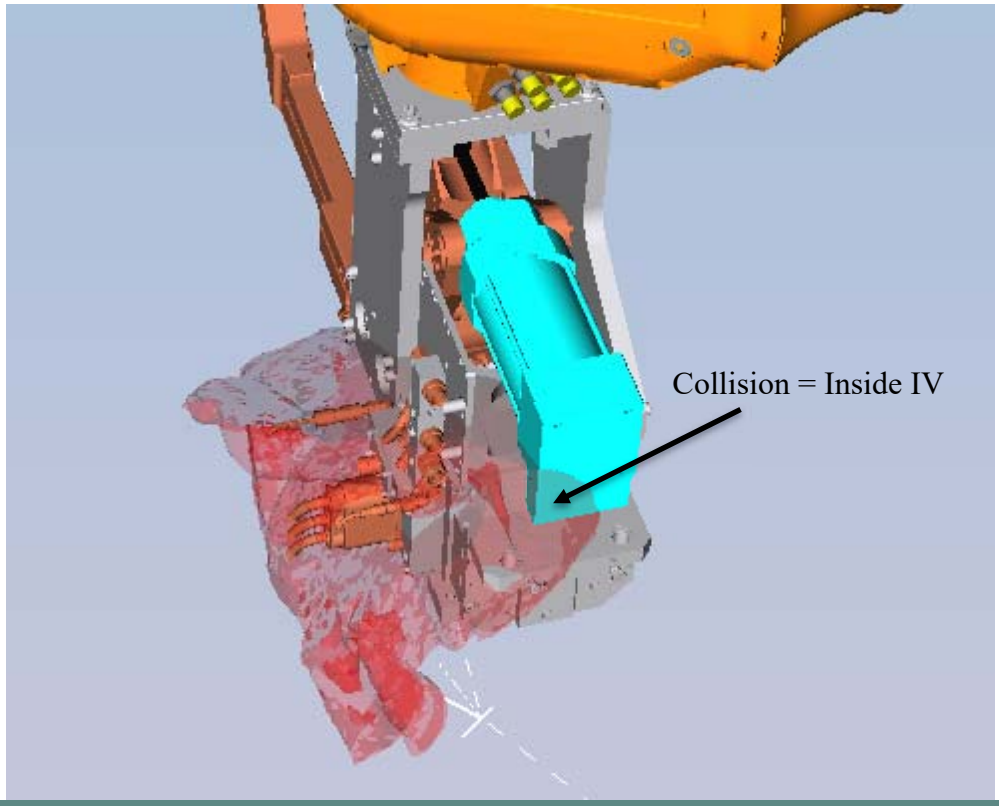


Figure 18. Robot R81-20-17-36-070R02 at via-location ToB940wp4120_10 (arrow). In the "OLP Commands" it can be seen that the robot would be waiting for the signal AllocateZone2.



Path Editor

0.00 0.01

Paths & Locations	Attachment	OLP Commands
B940WeldSeg1		
HomeGun311		Last location before interference Last location before interference !Outside IV; !Outside IV;
ToB940wp4120_10		Last location before interference Last location before interference !Outside IV; !Outside IV; WaitSignal AllocateZone2;
ToB940wp4120_20		First location after interference First location after interference !Outside IV; !Inside IV;
ToB940wp4120_30		!Outside IV; !Inside IV;
ToB940wp4120_40		Last location before interference Last location before interference !Outside IV; !Inside IV;
ToB940wp4120_50		First location after interference First location after interference !Outside IV; !Inside IV;
ToB940wp4120_60		Last location in interference Last location in interference !Outside IV; WaitSignal ReleaseZone2;

Figure 19. Robot R81-20-17-36-070R02, moved to the next via-location in the segment and now coliding with the interference volume.

From Figure 19 it can be seen that ARCADE writes OLP-commands to the same via-locations as the included "Automatic Interference Tool". It is also seen that commands for allocation and release of a zone is written to the same via-locations that, according to the tool, are either first or last in the interference volume. The reason that each via-location includes many OLP-commands is that the segment could be part of many interference volumes. Thus, the via-location will be subjected to many evaluations. The *Distance.csv,* Figure 17, could be used to confirm which interference volumes caused which OLP-commands in these cases.

4.5 Step 5 – Transfer of data to Sequence Planner

Sequence Planner was able to retrieve the data through Tecnoviewer, however the data is stored in a way that currently appears to be inaccessible for Sequence Planner. While ARCADE still holds practical applicability on its own this inability meant that the thesis would be unable to fully reach its aim.

4.6 Step 6 – Updating robot programs with results from Sequence Planner

Due to the issues with the previous step it was not possible to obtain any results from this step. However, automatically writing robot programs has been shown earlier [8] and the co-thesis [6] was able to update their programs with the results from Sequence Planner.

5 Discussion

The discussion of the results is split into two sections. These sections concern the results in general and a comparison with a recently published method, that tackles collision avoidance in a different manner.

5.1 General discussion

The ARCADE plug-in yields a collision-free environment based on the allocation of shared zones. As the zones are based on the interference volumes of the robots it is likely that the zones represent the best possible implementation of this method (without adding any extra requirements to the model). Since the sweep volumes matches the actual movement it would be possible to add additional via-points even closer to the volume to further minimize the sizes of the shared zones. As ARCADE identifies the same via-locations as the AIT in Process Simulate the results can be said to match current commercial options.

By introducing more zones it would be possible to have a high ”resolution” in the cell, as in having fine control of the movements of the robots. It might even be possible to slow down one of the robots before it reaches a shared volume and thus avoid having to start and stop as the zone becomes accessible [22]. Current practise, however, is to have as few zones as possible to avoid the waste associated with sending and waiting for signals (at Volvo, the rule of thumb was a 1 second increase of cycle time per signal) [18]. Therefore, as the thesis has not investigated this, it could be worthwhile to gather data on how many zones it takes before this problem arises or if the losses from having fewer zones is greater than the penalty of transmitting signals.

It cannot be excluded that the project could have gotten even further if the practitioners had been more experienced programmers. In addition, Process Simulate is not exactly an open-source software and has a steep learning curve. Hence, developers are at the mercy of the suite owner regarding the availability of certain commands and functions. Further development would thus likely benefit from a closer collaboration with Siemens PLM than was the case for this thesis.

The development has largely been adopted for implementation with the provided model. Therefore, it also contains elements that will be specific to how cell design is done at Volvo. This can be allowed since this has put further constraints on the programming and by solving for the special case a solution for the general case has also been achieved.

In general, the functionality that was sought to be developed is already present in the form of different tools. However, the tools are not as integrated with each other as the functionality included in ARCADE. The tools are also more general and in their current state the potential for zone creation/allocation and sequence planning cannot be fully utilised. The results of the thesis could be considered of limited applicability given that testing has only been carried out on one model and it is therefore possible that testing on another model will reveal programming errors or faults due to incorrect assumptions. This should not be the case, but without testing it cannot be guaranteed. On the other hand, ARCADE has not been tried out by anyone else but the developers, in order to have a proper validation it should be used by someone with less knowledge about the inner functions of ARCADE.

Another point of concern is naturally that the thesis was only partially able to meet its aim. On the other hand, the thesis did find the specifications necessary for continued development of

the Sequence Planner / Process Simulate interaction. At the start of the thesis the task was clear, however the preconditions were not and as the work has progressed it was realised what information should have been identified or provided from the beginning.

5.2 Comparison with "Sequence-modification based collision-free motion planning of multiple robots workcell"

An alternative and recently presented solution to collision-free motion planning has been proposed by Wu, Deng, Chen, Guan and Zhang [23]. In summary, the method uses the *A** search algorithm to generate an optimal, with regard to cycle-time, robot program. Without going to deep into details the method, for a two-robot case, is as follows:

1. Create a two-dimensional grid where each axis represents the operations of one of the included robots. The nodes of the grid will then represent a combination of operations from each robot program. The *(0,0)* point will be the starting point of each program and the point consisting of the last operations in each program will be the finish point or goal (in the top right corner, point *(N,M)*).
2. Nodes that result in a collision between the robots are then removed from the grid. The result will be a "map" of possible nodes with "islands" of forbidden nodes.
3. Using the *A** search algorithm, the shortest way from start to finish is found. This will result in the fastest, collision-free, program. An example is shown in Figure 20. The horizontal and vertical parts of the line (in the upper right corner) are indicative of one robot standing still while the other works.

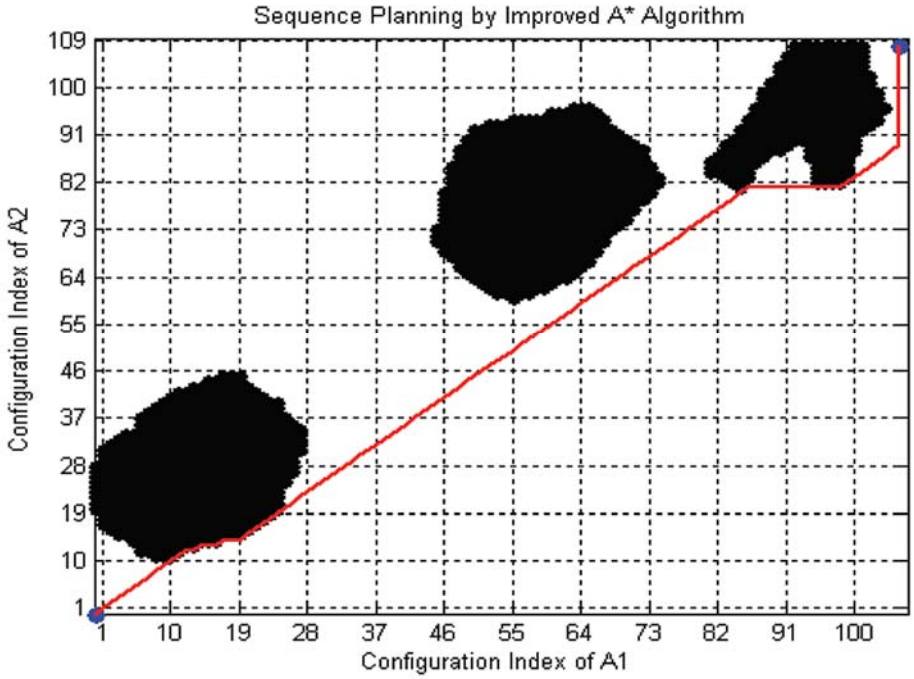


Figure 20. Illustration of an n = 2 robot case. Image retrieved from [23].

The authors then proceed to show that the method is valid for a case involving three robots and theoretically even *n* robots, see Figure 21. However, the authors mention some short-comings with the suggested method. Namely that calculation of the volumes and determining intersections needs to be improved. It is stated that the software Delmia was used to create the simulation as well as volumes and it could be possible that switching to Process Simulate would yield this performance boost. At the very least, ARCADE could be used to determine which via-locations would result in a collision.

To incorporate the method suggested by Wu et. al (for a 2 robot case) in Process Simulate the code developed in the thesis could be used with the following alterations:

1. Create an array with (*N*,*M*) rows and columns, where *N* and *M* is the number of via-locations in the respective robot programs. As the location (*0*,*0*) will be the starting point this node should represent the home position of each robot and location (*N*,*M*) their end locations.
2. Assuming interference volumes have been created, ARCADE determines whether or not a robot is inside the interference volume.
3. In the array, write information regarding the robots whereabouts to the respective node. This information is then used by *A** to determine whether or not the node is accessible, i.e. if those specific robot configurations can be allowed at the same moment in time.

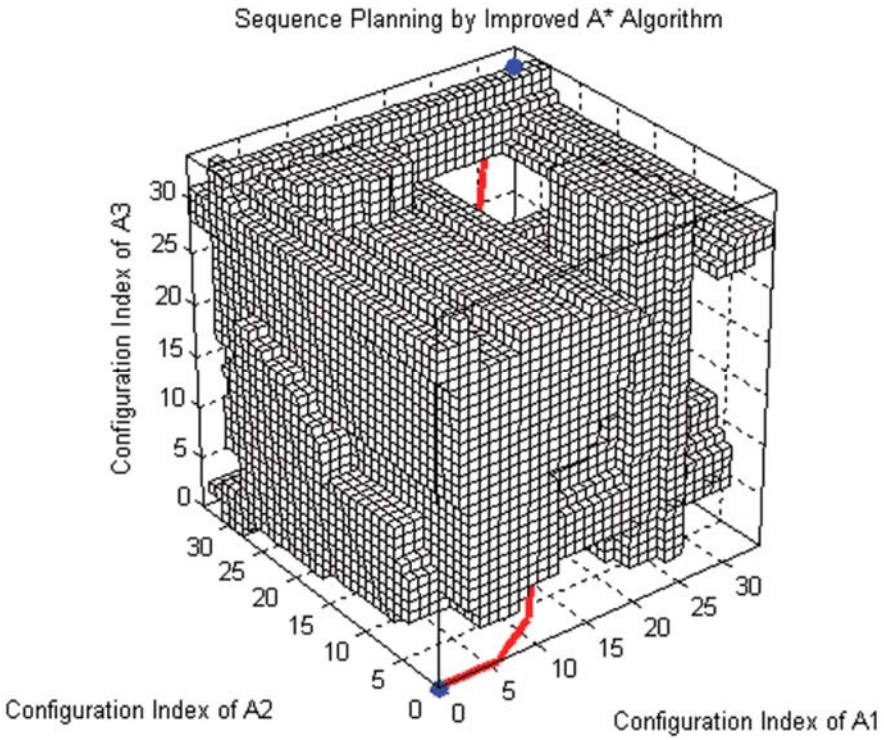


Figure 21. Illustration of the solution for an n = 3 robot case. Image retrieved from [23].

Thus, it might be possible to use the code developed in this thesis to implement the solution proposed in the article [23]. It should also be possible to use the results of the thesis for testing the article's solution with Process Simulate instead of Delmia V6.

6 Conclusion

To conclude the thesis, it can be said that all the planned steps were achieved save for integration with Sequence Planner. Some interesting discoveries were made and the main objectives were at least partially met, however the stipulated research questions remained unanswered. Although it could be argued that the results were “close, but no cigar” the progress made in the thesis could be useful for future work within the area. With this in mind the aim during the final stages was to facilitate any continued development. Currently, ARCADE writes information as OLP-commands, the handling of which is not supported by Sequence Planner. Possible solutions to this would be to re-write ARCADE to update the robot programs instead of generating the OLP-commands or enable Sequence Planner to handle information stored in OLP-commands. Writing information to OLP-commands was chosen based on information gained through interviews as this more closely resembled the way a simulation engineer would work with the programming. The limitations of Sequence Planner were unfortunately not realised at the time.

Regarding the research questions the following conclusions are made:

RQ1: *What is the result of having two or more robots generate the interference volume?*

After contacting the developers of Process Simulate it became clear that this research question would be left unanswered. The functionality for creating interference volumes is currently not exposed to the public. Regarding the possibility of having more than two robots generate an interference volume the idea was to *type cast* an interference volume as a sweep volume and then feed it back into the calculation (as the calculation takes two sweep volumes as input). However, according to the developers at Siemens, this would not be possible due to the properties of the different types.

RQ2: *What other possibilities exists for Sequence Planner and Process Simulate interaction?*

As the thesis was unable to achieve even the most "obvious" possibility of Sequence Planner/Process Simulate integration RQ2 could not be answered. Judging by the results of the co-thesis the possibilities for Virtual Commissioning are almost endless [7]. Therefore, integration of the software could be used not only for, for example, optimising cycle times or schedules but also for energy consumption.

6.1 Future work

As "future work" is left that which the authors considered would be promising but was omitted due to the limitations, that which the thesis did not fully investigate, or that which was fully investigated without reaching a conclusion.

ARCADE lacks integration with the PLC. This means that although zones are created automatically the necessary signals are not. However, given the integration of PLC-simulation into Process Simulate, solving this through software should be possible. Hence, the automatic generation of signals based on, or matching, OLP-commands could be an area for further investigation.

As worked progressed it was noted that Process Simulate has tools similar to what was developed already available. However, as stated earlier, the developments of the thesis are more specialised and focused towards the current usage at Volvo. Future work could include

further investigation of AIT as well as the "Automatic Interference Zone Tool". It could be that these are "good enough" for including in the current workflow.

Something that is considered promising but came to the authors knowledge too late in the thesis is the method involving the *A** search algorithm. Therefore, investigation into the possibilities of integrating that method is left as future work.

Additionally, the current code does not take into consideration if the via-location to which the OLP-command is written has the zone type “fine”. This is a prerequisite for the robot to be able to read the command, as this zone type then causes the robot to come to a stop at that via-location. This seems possible to implement by using the “Zone Property” of the “TxRoboticLocationOperationMotionParameters” class.

Further development could also aim to allow three robots to share a zone, as current zones are based on interference volumes based on the sweep volumes of two robots. To clarify, the solution can handle a cell with more than three robots but each zone is currently based on the pair of robots involved in the interference volume. Thus, it could be interesting to allow a third robot to share an already created zone instead of creating additional zones. However, this would negatively affect the possibilities for optimization (rather, having fewer zones allows for a less [time] optimal solution than having a larger number of zones do).

6.2 Recommendations

Despite the setback of not being able to return the schedules from Sequence Planner, further projects including integration of Process Simulate and Sequence Planner are recommended. This is due to the fact that ARCADE achieved the planned results up until the integration with Sequence Planner, hence the method could be applied to a new solution or Sequence Planner modified to handle OLP-commands.

As was hinted during the discussion, integration of the development done in the thesis with the discovery from Wu et al. [23] would be an interesting project. This could be related to RQ2 by using Sequence Planner for validation of the results.

References

[1] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Integrated Virtual Preparation and Commissioning: supporting formal methods during automation systems development," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1939–1944, 2016.

[2] W. Kühn, "Digital Factory - Simulation Enhancing the Product and Production Engineering Process," in *Proceedings of the 2006 Winter Simulation Conference*, 2006, pp. 1899–1906.

[3] R. Parasuraman and C. D. Wickens, "Humans: Still Vital After All These Years of Automation," *Hum. Factors*, vol. 50, no. 3, pp. 511–520, 2008.

[4] S. Axelsson, "Off-Line Programming of Robots at Volvo Cars – Floor shop preparation," in *Proceedings of the 33rd ISR (International Symposium on Robotics)*, 2002.

[5] T. Nordin, "Off-Line Programming of Robots at Volvo Cars – The technique," in *Proceedings of the 33rd ISR (International Symposium on Robotics)*, 2002.

[6] J. Asklund and N. Jönsson, "Sequence Planning and Optimization of Production Systems (Working Title)," Chalmers University of Technology, [Not yet published, report number EX097].

[7] S. Winther, "Virtual Commissioning of Production Process," Chalmers University of Technology, 2017.

[8] D. Bengtsson and C. Rutgersson, "Development of BoxSweeper and BoxSweeper PLC," Chalmers University of Technology, 2008.

[9] R. E. Shannon, *Systems Simulation - The Art and Science*. Prentice-Hall, 1975.

[10] R. G. Ingalls, "Introduction to Simulation," in *Proceedings of the 2011 Winter Simulation Conference*, 2011, pp. 1379–1393.

[11] W. Kühn, "Digital Factory - Simulation Enhancing the Product and Production Engineering Process," in *Proceedings of the 2006 Winter Simulation Conference*, 2006, pp. 1899–1906.

[12] M. Fabian, *Industrial Automation - Lecture Notes (SSY 065)*. Gothenburg: Chalmers Automation, Department of Signals and Systems, Chalmers University of Technology, 2006.

[13] M. Collins, "Formal Methods," *18-849b Dependable Embedded Systems*. [Online]. Available: https://users.ece.cmu.edu/~koopman/des_s99/formal_methods/. [Accessed: 27-Jun-2017].

[14] L. Sciaviccio and B. Siciliano, *Modelling and Control of Robot Manipulators*. London: Springer London, 2000.

[15] Siemens Product Lifecycle Management Software Inc., "Process Simulate: Siemens PLM Software." [Online]. Available: https://www.plm.automation.siemens.com/en_us/products/tecnomatix/manufacturing-simulation/assembly/process-simulate.shtml. [Accessed: 10-Jun-2017].

[16] M. Dahl, K. Bengtsson, P. Bergagård, M. Fabian, and P. Falkman, "Sequence Planner : Supporting Integrated Virtual Preparation and Commissioning," in *The 20th World Congress of the International Federation of Automatic Control, 9-14th July*, 2017.

[17] L. Bainbridge, "Ironies of automation," *Automatica*, vol. 19, no. 6, pp. 775–779, 1983.

[18] H. Hansson, "Simulation Expert - Volvo Cars Corporation," 2017.

[19] M. Denscombe, *The Good Research Guide for small-scale social research projects*, Second Edi. Maidenhead: Open University Press, 2003.

[20] Elsevier B.V., "What is Field-weighted Citation Impact (FWCI)? - Scopus Support." [Online]. Available: https://service.elsevier.com/app/answers/detail/a_id/14894/kw/fwci/supporthub/scopus/. [Accessed: 20-Feb-2017].

[21] O. Ohayon, "How to write a command - Tecnomatix .NET API General Training," 2016. [Online]. Available: https://community.plm.automation.siemens.com/t5/Tecnomatix-Developer-Knowledge-Base/Video-Tecnomatix-NET-API-General-Training-session-1-How-to-write/ta-p/348922. [Accessed: 10-Feb-2017].

[22] A. Kobetski and M. Fabian, "Velocity balancing in flexible manufacturing systems," in *Proceedings - 9th International Workshop on Discrete Event Systems, WODES' 08*, 2008, pp. 358–363.

[23] H. Wu, H. Deng, L. Chen, Y. Guan, and H. Zhang, "Sequence-modification based collision-free motion planning of multiple robots workcell," *2016 IEEE Int. Conf. Robot. Biomimetics, ROBIO 2016*, no. June 2017, pp. 1135–1140, 2017.

Appendix

A. Source Code: Main program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Tecnomatix.Engineering;
using Tecnomatix.Engineering.Ui;
using Tecnomatix.Planning;
using Tecnomatix.Engineering.DataTypes;
using Tecnomatix.Engineering.Olp;
using Tecnomatix.Engineering.Olp.OLP_Utilities;
using System.Collections;

namespace TecnomatixTesting1
{

    public class TxHelloWorld : TxButtonCommand
    {

        // Creation of constants
        static TxDocument TxStudy = TxApplication.ActiveDocument;
        static TxOperationRoot TxOpRoot = TxApplication.ActiveDocument.OperationRoot;
        static TxNoTypeFilter TxNoFilter = new TxNoTypeFilter();
        static TxTypeFilter OperationFilter = new TxTypeFilter(typeof(TxCompoundOperation));
        static TxTypeFilter LocationFilter = new TxTypeFilter(typeof(ITxRoboticLocationOperation));
        static TxTypeFilter IV_Filter = new TxTypeFilter(typeof(TxInterferenceVolume));

        //Path where the .CSV is placed (output)
        static string savePath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
        string OutputDirectory = savePath + @"\Distance.csv";

        public override string Category
        {
            get
            {
                return StringTable.CATEGORY;
            }
        }

        public override void Execute(object cmdParams)
        {

            #region MinimalDistance

            TxObjectList InterferenceVolumes =
TxApplication.ActiveDocument.PhysicalRoot.GetAllDescendants(IV_Filter);
            TxObjectList ViaLocations =
TxApplication.ActiveDocument.OperationRoot.GetAllDescendants(LocationFilter);
            TxObjectList ClosestViaLocation = new TxObjectList();
            TxObjectList Robots = TxApplication.ActiveDocument.PhysicalRoot.GetAllDescendants(new
TxTypeFilter(typeof(TxRobot)));
            var csv = new StringBuilder(); //Declare before loop
            csv.AppendLine("\"sep=;\""); //Specifies the separating sign in the .CSV-file

            TxObjectList Command = TxApplication.ActiveDocument.OperationRoot.GetAllDescendants(new
TxTypeFilter(typeof(ITxRoboticLocationOperation)));

            if (InterferenceVolumes != null)
```

```
            {
                int j = 0; //Counter for Interference Volumes
                int r = 1; //counter for Zones
                int a = 0; //Counter for ViaLocations
                int t = 0; //counter for writing ViaLocations on Process Simulate and .CSV
                string[,] Info = new string[Command.Count, 6]; //Array to save all generated data
(i,j), all VL "i" will have "j" properties assigned
                TxObjectList VLocation = new TxObjectList();
                foreach (TxInterferenceVolume Intervol in InterferenceVolumes)
                {

                    ITxDisplayableObject DisplayableInterferenceVolume =
(ITxDisplayableObject)InterferenceVolumes[j]; //Prevents error when calculating minimal distance by
making sure the interference volume is being displayed
                    DisplayableInterferenceVolume.Display();

                    // variables for generating the Minimal Distance measure
                    double ActualDistance = 0; //Unit: [mm]. Will be the return value from
GetMinimalDistance
                    TxVector PointOnIV = new TxVector();
                    TxVector PointOnViaPoint = new TxVector();
                    ITxLocatableObject IV_Location = Intervol as ITxLocatableObject;

                    TxObjectList InterferenceHoldingObjects = Intervol.HoldingObjects; //Gets the two
robots that generated the segment
                    // TODO: create a for loop to avoid having repetitive lines
                    ITxRoboticOrderedCompoundOperation Segment1 = InterferenceHoldingObjects[0] as
ITxRoboticOrderedCompoundOperation;
                    ITxRoboticOrderedCompoundOperation Segment2 = InterferenceHoldingObjects[1] as
ITxRoboticOrderedCompoundOperation;

                    ITxRobot Robot1 = Segment1.Robot;
                    ITxRobot Robot2 = Segment2.Robot;
                    TxObjectList TxORobot = new TxObjectList();
                    TxORobot.Add(Robot1);
                    TxORobot.Add(Robot2);

                    int q = 0; // counter for holding robots on a segment
                    foreach (ITxObjectCollection Segment in InterferenceHoldingObjects)
                    {
                        TxObjectList ViaLocation = Segment.GetAllDescendants(LocationFilter);

                        TxRobot Robot = TxORobot[q] as TxRobot;
                        q++;

                        TxObjectList RobotTx = new TxObjectList();
                        TxObjectList IV = new TxObjectList();
                        double[] Distance = new double[ViaLocation.Count];
                        string[] Text = new string[ViaLocation.Count];

                        TxRoboticCompositeCommandStringElement StringElement = new
TxRoboticCompositeCommandStringElement();
                        TxRoboticCompositeCommandCreationData CommandCreationData = new
TxRoboticCompositeCommandCreationData();
                        ITxRoboticLocationOperation VLocCommand;
                        string[] Separator = { "@" };

                        int i = 0;
                        var InIV = 0;

                        foreach (ITxLocationOperation ViaLoc in ViaLocation)
                        {
```

```
                            TxOlpControllerUtilities utility = new TxOlpControllerUtilities();
                            ITxRoboticControllerServices services =
utility.GetInterfaceImplementationFromController(Robot.Controller.Name,
typeof(ITxRoboticControllerServices), typeof(TxControllerAttribute), "ControllerName") as
ITxRoboticControllerServices;
                            TxJumpToLocationData data = new TxJumpToLocationData();
                            services.Init(Robot);
                            services.JumpToLocation(ViaLoc as ITxLocationOperation, data);

                            TxApplication.RefreshDisplay();

                            ITxLocatableObject ViaPointLocation = Robot.Collection as
ITxLocatableObject;
                            ITxLocatableObject ViaPointLocation1 = ViaPointLocation.Collection as
ITxLocatableObject;
                            IV_Location.GetMinimalDistance(ViaPointLocation1, out ActualDistance, out
PointOnIV, out PointOnViaPoint);

                            //checks for pairs of robots same as the current selection, if not find
anything then new zone is created
                            for (int x = 0; x < a; x += 1)
                            {
                                if (Info[x, 1] == Robot1.Name && Info[x, 2] == Robot2.Name)
                                {
                                    Info[a, 5] = Info[x, 5];
                                    break;
                                }
                                else if (Info[x, 1] == Robot2.Name && Info[x, 2] == Robot1.Name)
                                {
                                    Info[a, 5] = Info[x, 5];
                                    break;
                                }
                            }
                            if (Info[a, 5] == null)
                            {
                                if (a == 0)
                                {
                                    Info[a, 5] = r.ToString();

                                }
                                else
                                {
                                    r++;
                                    Info[a, 5] = r.ToString();
                                }
                                //break;
                            }

                            Info[a, 0] = Intervol.Name.ToString();//Interference Volume
                            Info[a, 1] = Robot1.Name.ToString();//ROBOT1
                            Info[a, 2] = Robot2.Name.ToString();//ROBOT2
                            Info[a, 3] = ViaLoc.Name.ToString();//Via Location evaluated
                            VLocation.Add(ViaLoc);

                            TxObjectList Delt = new TxObjectList();

                            if (ActualDistance <= 0)
                            {
                                if (InIV == 0)
                                {
                                    InIV = 1;
```

```
                                    if (i == 0)
                                    {
                                        Info[a, 4] = "WaitSignal AllocateZone" + Info[a, 5] +
";";//first location in IV
                                    }
                                    else
                                    {
                                        Info[a - 1, 4] = Info[a - 1, 4] + " @ " + "WaitSignal
AllocateZone" + Info[a, 5] + "; ";
                                        if (i == ViaLocation.Count - 1)
                                            Info[a, 4] = "WaitSignal ReleaseZone" + Info[a, 5] +
";";//last location in IV
                                        else
                                            Info[a, 4] = "!Inside IV;";
                                    }

                                    //TODO: Evaluate if this if worth to do.
                                    VLocCommand = ViaLoc as ITxRoboticLocationOperation;
                                    Delt = VLocCommand.Commands;
                                    if (Delt.Count > 0)
                                    {
                                        foreach (TxRoboticCommand Cmmnd in Delt)
                                        {
                                            TxRoboticCommand Del = Cmmnd as TxRoboticCommand;
                                            //Del.Delete();
                                        }
                                    }
                                }
                                else
                                {
                                    if (i == ViaLocation.Count - 1)
                                    {
                                        Info[a, 4] = "WaitSignal ReleaseZone" + Info[a, 5] +
";";//last location in IV
                                    }
                                    else
                                        Info[a, 4] = "!Inside IV;";
                                }
                            }
                            else
                            {
                                if (InIV == 1)
                                {
                                    InIV = 0;
                                    Info[a, 4] = "WaitSignal ReleaseZone" + Info[a, 5] + ";";


                                    VLocCommand = ViaLoc as ITxRoboticLocationOperation;
                                    Delt = VLocCommand.Commands;
                                    if (Delt.Count > 0)
                                    {
                                        foreach (TxRoboticCommand Cmmnd in Delt)
                                        {
                                            TxRoboticCommand Del = Cmmnd as TxRoboticCommand;
                                            //Del.Delete();
                                        }
                                    }
                                }
                                else
                                {
                                    Info[a, 4] = "!Outside IV;";
                                }
```

```
                            }

                            i++;//DEL
                            a++;
                        }
                        //Loop for writing to file (.csv) and OLPcommands

                        for (int y = t; y < a; y += 1)//Needs to be aout of the main loop x=b from x
until a
                        {
                            //FILE

                            var newLine =
string.Format("\"{0}\";\"{1}\";\"{2}\";\"{3}\";\"{4}\";\"{5}\"", Info[y, 0], Info[y, 1], Info[y, 2],
Info[y, 3], Info[y, 4], Info[y, 5]);
                            csv.AppendLine(newLine);

                            VLocCommand = VLocation[y] as ITxRoboticLocationOperation;

                            //OLP
                            string[] Word = Info[y,4].Split(Separator,
StringSplitOptions.RemoveEmptyEntries); //separates wait wignals that are written in the same line
for same ViaLocation

                            for (int s = 0; s < Word.Length; s++) // writes OLP commands included in
text[t], usually only 1
                            {
                                ArrayList ArrList = new ArrayList();
                                StringElement.Value = Word[s];
                                ArrList.Add(StringElement);
                                CommandCreationData.Elements = ArrList;
                                VLocCommand.CreateCompositeCommand(CommandCreationData);
                            }
                            t++;
                        }
                        var newLine1 = string.Format("\" \";");
                        csv.AppendLine(newLine1);
                        t=a;
                    }
                    j++;
                }
                //copy here the writing file
            }

            //Declare after loop - Actual writing of output to file
            System.IO.File.WriteAllText(OutputDirectory, csv.ToString());

            //Information window telling that process has ended and the path in which the file has
been saved.
            MessageBox.Show("Script finished. Output written to " + savePath, "MinimalDistance",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        public override string Name
        {
            get
            {
                return StringTable.MODEL_OBJECTS_COMMAND_NAME;
            }
        }
    }
}
            #endregion
```

B. Source Code: ConsoleChangePriority.exe

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
using System.ComponentModel;

namespace ConsoleChangePriority
{

    class Program
    {


        static void Main(string[] args)
        {
            Console.WriteLine("How many swept volumes will be calculated? ");
            string strPriorityCommand = "/C wmic process where name=\"SweptVolumeEngine.exe\" CALL
setpriority \"high priority\""; //"SweptVolumeEngine.exe" could be replaced with another process
which priority you want to change
            string consoleCalcInfo = "Raised priority of swept volume calculation {0} out of {1}";
            string strUserInput = Console.ReadLine();
            int intUserInput;
            int intEndofRun = 0;
            int intMaxNumberOfRuns = 100;

            if (Int32.TryParse(strUserInput, out intUserInput))
            {
                intUserInput = Int32.Parse(strUserInput);
            }
            else
            {
                Console.WriteLine("String could not be parsed, restart the application and enter an
integer when prompted");
                Console.ReadKey();

            }

            for (int i = 1; i <= intUserInput; i++)
            {
                try
                {

                    Console.WriteLine("Looking for SweptVolumeEngine.exe");
                    Process[] SWE = Process.GetProcessesByName("SweptVolumeEngine");
                    if (SWE[0] != null)
                    {
                        var observation = System.Diagnostics.Process.Start("CMD.exe",
strPriorityCommand);
                        Console.WriteLine("Raised priority of ");
                        SWE[0].WaitForExit();
                        intEndofRun = 0;
                        Process[] CMD = Process.GetProcessesByName("CMD");
                        CMD[0].Kill();
                        Console.WriteLine(consoleCalcInfo, i, intUserInput);
                        System.Threading.Thread.Sleep(1000 * 4); //Input in milliseconds. Waits 4
seconds before making a new attempt
                    }
```

```
                }
                catch (IndexOutOfRangeException)
                {
                    i--;
                    intEndofRun++;
                    Console.WriteLine("SweptVolumeEngine.exe is currently not running");
                    if (intEndofRun > intMaxNumberOfRuns)
                    {
                        break;
                    }

                    System.Threading.Thread.Sleep(1000 * 3); //Input in milliseconds. Waits 3 seconds
before making a new attempt
                }
                catch
                {
                    Console.WriteLine("Program encountered unexpected error and will now close");
                    i = intUserInput;
                    break;


                    // System.Threading.Thread.Sleep(1000 * 3); //Input in milliseconds. Waits 3
seconds before making a new attempt
                }
            }

                Console.WriteLine("Search ended, did not find any SweptVolumeEngine.exe with {0}
attempts", intMaxNumberOfRuns);
                Console.WriteLine("Therefore approximately " + intMaxNumberOfRuns * 3 / 60 + "
minutes since last SweptVolumeEngine.exe finished");
                Console.WriteLine("Press any button to exit application");
                Console.ReadKey();

        }
    }
}
```

C. Source Code: Copy-Paste via-locations

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Tecnomatix.Engineering;

namespace TecnomatixTesting
{

#region ButtonCommand
       public class TxHelloWorld : TxButtonCommand
       {

              public override string Category
              {
                     get
                     {
                            return TecnomatixTesting1.StringTable.CATEGORY;
                     }
              }

              public override void Execute(object cmdParams)
              {
             //TxRoboticViaLocationOperation SavedOp = new TxRoboticViaLocationOperation();

             // TODO DECLARE i for each of the "foreachs" loops
             //typeof(TxCompoundOperation)
             TxObjectList Level1 = TxApplication.ActiveDocument.OperationRoot.GetDirectDescendants(new
TxNoTypeFilter());
             int i1 = 0;
             foreach (TxCompoundOperation Level1Op in Level1)
             {
                 TxObjectList Level2 = Level1Op.GetDirectDescendants(new TxNoTypeFilter());
                 foreach (TxCompoundOperation Level2Op in Level2)
                 {
                     TxObjectList Level3 = Level2Op.GetDirectDescendants(new TxNoTypeFilter());
                     int i3 = 0;
                     foreach (ITxOperation if3 in Level3)
                     {
                         TxCompoundOperation Level3Op = Level3[i3] as TxCompoundOperation;
                         TxObjectList Level4 = new TxObjectList();

                         Level4 = Level3Op.GetDirectDescendants(new TxNoTypeFilter());

                         i3++;
                         int i4 = 0;
                         ITxOperation SavedOp = null as ITxOperation;
                         ITxOperation SavedOp1;
                         foreach (ITxOperation Op4 in Level4)
                         {
                             TxCompoundOperation If4 = Level4[i4] as TxCompoundOperation;
                             if (If4 != null)//enter here when we have cases
                             {
                                 #region Compound Operations
                                 TxCompoundOperation Level4Op = Level4[i4] as TxCompoundOperation;
```

```
                                TxObjectList Level51 = Level4Op.GetDirectDescendants(new
TxTypeFilter(typeof(TxCompoundOperation)));//TODO fix this to be able to work with Directin cases
ROB22
                                int i5 = 0;
                                foreach (ITxOperation Op5 in Level51)
                                {

                                    TxCompoundOperation Level51Op = Op5 as TxCompoundOperation;
                                    if (Level51Op != null)
                                    {
                                        TxObjectList Level6 = Level51Op.GetDirectDescendants(new
TxNoTypeFilter());
                                        int i6 = 0;
                                        foreach (ITxOperation compOp5 in Level6)
                                        {
                                            TxGenericRoboticOperation GenRobOp = Level6[i6] as
TxGenericRoboticOperation;
                                            TxWeldOperation WeldOp = Level6[i6] as TxWeldOperation;
                                            TxObjectList Level7;
                                            ITxOperation Level6Op;

                                            //if (GenRobOp != null)
                                            //{ TxObjectList Level7 =
GenRobOp.GetDirectDescendants(new TxNoTypeFilter()); }
                                            if (WeldOp != null)
                                            {   Level6Op = WeldOp;
                                                Level7 = WeldOp.GetDirectDescendants(new
TxNoTypeFilter());
                                            }
                                            else
                                            {   Level6Op = GenRobOp;
                                                Level7 = GenRobOp.GetDirectDescendants(new
TxNoTypeFilter());
                                            }

                                            SavedOp1 = SavedOp;
                                            if (Level7.Count != 0)
                                            {
                                                if (i5 == Level51.Count)
                                                { SavedOp = Level7[Level7.Count - 1] as
TxRoboticViaLocationOperation; }
                                            }
                                            else { SavedOp = null; }

                                            if (i4 != 0 && SavedOp1 != null) //i3 means the index
below level 3, to see when the frist operation of the segment hapens. might move this to the copy
method
                                            {   PasteMethod(Level6Op, SavedOp1);
                                            }
                                            i5++;
                                            i6++;
                                        }
                                    }
                                    else
                                    {
                                        TxGenericRoboticOperation GenRobOp = Op5 as
TxGenericRoboticOperation;
                                        TxWeldOperation WeldOp = Op5 as TxWeldOperation;
                                        TxObjectList Level7;
                                        ITxOperation Level6Op;

                                        //if (GenRobOp != null)
```

```
                                        //{ TxObjectList Level7 = GenRobOp.GetDirectDescendants(new
TxNoTypeFilter()); }
                                        if (WeldOp != null)
                                        {   Level6Op = WeldOp;
                                            Level7 = WeldOp.GetDirectDescendants(new
TxNoTypeFilter());
                                        }
                                        else
                                        {   Level6Op = GenRobOp;
                                            Level7 = GenRobOp.GetDirectDescendants(new
TxNoTypeFilter());
                                        }

                                        SavedOp1 = SavedOp;
                                        if (Level7.Count != 0)
                                        {
                                            if (i5 == Level51.Count)
                                            { SavedOp = Level7[Level7.Count - 1] as
TxRoboticViaLocationOperation; }
                                        }
                                        else { SavedOp = null; }

                                        if (i4 != 0 && SavedOp1 != null) //i3 means the index below
level 3, to see when the frist operation of the segment hapens. might move this to the copy method
                                        { PasteMethod(Level6Op, SavedOp1);
                                        }
                                        i5++;
                                    }

                                }

                            }
#endregion
                            else // Enter here when the having a generic robotic operation at level 4
                            {

                                TxObjectList Level52;
                                ITxOperation Level4Op;
                                TxGenericRoboticOperation GenRobOp = Level4[i4] as
TxGenericRoboticOperation;
                                TxContinuousRoboticOperation ContOp = Level4[i4] as
TxContinuousRoboticOperation;
                                TxWeldOperation WeldOp = Level4[i4] as TxWeldOperation;

                                //TODO revise this as the same statement is repited start and end
                                if (GenRobOp != null)
                                {   Level4Op = GenRobOp;
                                    Level52 = GenRobOp.GetDirectDescendants(new TxNoTypeFilter());
                                }

                                else if (ContOp != null)
                                {   Level4Op = ContOp;
                                    Level52 = ContOp.GetDirectDescendants(new TxNoTypeFilter()); }

                                else if (WeldOp != null)
                                {   Level4Op = WeldOp;
                                    Level52 = WeldOp.GetDirectDescendants(new TxNoTypeFilter()); }

                                else
                                {   Level4Op = GenRobOp;
                                    Level52 = GenRobOp.GetDirectDescendants(new TxNoTypeFilter()); }
```

```
                                SavedOp1 = SavedOp;
                                if (Level52.Count != 0)
                                { SavedOp = Level52[Level52.Count - 1] as
TxRoboticViaLocationOperation; }
                                else { SavedOp = null; }

                                if (i4 != 0 && SavedOp1 != null)
                                {
                                    PasteMethod(Level4Op, SavedOp1);
                                }
                            }
                            i4++;
                        }
                    }
                }
            }
            #region Old code
            //TxTypeFilter IV_Filter = new TxNoTypeFilter();
            /*TxObjectList InterferenceVolumes1 =
TxApplication.ActiveDocument.OperationRoot.GetDirectDescendants(new TxNoTypeFilter());

            TxCompoundOperation show = InterferenceVolumes1[1] as TxCompoundOperation;
            TxObjectList InterferenceVolumes2 = show.GetDirectDescendants(new TxNoTypeFilter());
            TxCompoundOperation show1 = InterferenceVolumes2[5] as TxCompoundOperation;
            TxObjectList InterferenceVolumes3 = show1.GetDirectDescendants(new TxNoTypeFilter());
            TxCompoundOperation show2 = InterferenceVolumes3[0] as TxCompoundOperation;
            TxObjectList InterferenceVolumes4 = show2.GetDirectDescendants(new TxNoTypeFilter());
            TxCompoundOperation show3 = InterferenceVolumes4[1] as TxCompoundOperation;
            TxObjectList InterferenceVolumes5 = show3.GetDirectDescendants(new TxNoTypeFilter());
            ITxOperation show4 = InterferenceVolumes5[0] as ITxOperation;

            TxTypeFilter Sel_Filter = new TxTypeFilter(typeof(TxCompoundOperation));
            TxObjectList Selection_1 =
TxApplication.ActiveDocument.OperationRoot.GetAllDescendants(Sel_Filter);

            TxRoboticViaLocationOperation[] lastViaList = null ;
            TxObjectList Preview = new TxObjectList();
            TxObjectList Preview1 = new TxObjectList();
            int i = 0;


            foreach (TxCompoundOperation compOp in Selection_1)
            {
                TxCompoundOperation CompOp = Selection_1[i] as TxCompoundOperation;
                TxObjectList weldList = CompOp.GetDirectDescendants(new
TxTypeFilter(typeof(TxWeldOperation)));
                int i2 = 0;
                foreach (TxWeldOperation compOp1 in weldList)
                {
                    TxWeldOperation CompOp1 = weldList[i2] as TxWeldOperation;
                    TxObjectList weldList1 = CompOp1.GetDirectDescendants(new
TxNoTypeFilter());//think about putting and exclude for compounds

                    if (weldList.Count > 0)
                    {
                        //TxRoboticViaLocationOperation lastViaLoc = weldList1[weldList1.Count - 1]
as TxRoboticViaLocationOperation;
                        Preview1.Add(weldList1[weldList1.Count - 1]);
                    }
                }
                //lastViaList.SetValue(lastViaLoc, i);
                i++;
```

```
            }


           /* foreach (TxCompoundOperation compOp in InterferenceVolumes)
            {
                TxObjectList weldList = compOp.GetAllDescendants(new
TxTypeFilter(typeof(TxCompoundOperation)))  ;

            }
            TxOperationRoot operartionRoot = TxApplication.ActiveDocument.OperationRoot;
            TxRoboticViaLocationOperation op = InterferenceVolumes[0] as
TxRoboticViaLocationOperation;
            TxObjectList operationsToCopy = new TxObjectList();
            operationsToCopy.Add(op);

            TxObjectList operationsCopies = null;
            if (operartionRoot.CanPasteList(operationsToCopy))
                operationsCopies = operartionRoot.Paste(operationsToCopy);
            */


            /* int index = 0;
            foreach (ITxObject InterfVol in InterferenceVolumes)
            {

                string InterferenceVolName = string.Format("Interference volume: {0}", index + 1);
                // Console.WriteLine(InterferenceVolName + "\n");

                ++index;
            }

            var sb = new StringBuilder();

            /*if (InterferenceVolumes.Count < 1)
            {
                MessageBox.Show(string.Format("Number of Interference Volumes: {0}",
InterferenceVolumes.Count), Name, MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
            }
            else
            {
                int index = 0;
                foreach (ITxObject InterfVol in InterferenceVolumes)
                {

                    MessageBox.Show(string.Format(MessageFormat, InterferenceVolumes[index].Name),
Name, MessageBoxButtons.OK, MessageBoxIcon.Information);
                    ++index;
                }
            }
            MessageBox.Show(string.Format(MessageFormat, show.Name), Name, MessageBoxButtons.OK,
MessageBoxIcon.Information);
            MessageBox.Show(string.Format(MessageFormat, show1.GetType()), Name,
MessageBoxButtons.OK, MessageBoxIcon.Information);
            MessageBox.Show(string.Format(MessageFormat, show1.Name), Name, MessageBoxButtons.OK,
MessageBoxIcon.Information);
            MessageBox.Show(string.Format(MessageFormat, show2.GetType()), Name,
MessageBoxButtons.OK, MessageBoxIcon.Information);
            MessageBox.Show(string.Format(MessageFormat, show2.Name), Name, MessageBoxButtons.OK,
MessageBoxIcon.Information);
            MessageBox.Show(string.Format(MessageFormat, show3.GetType()), Name,
MessageBoxButtons.OK, MessageBoxIcon.Information);
```

```
            MessageBox.Show(string.Format(MessageFormat, show3.Name), Name, MessageBoxButtons.OK,
MessageBoxIcon.Information);
            MessageBox.Show(string.Format(MessageFormat, show4.GetType()), Name,
MessageBoxButtons.OK, MessageBoxIcon.Information);
            MessageBox.Show(string.Format(MessageFormat, show4.Name), Name, MessageBoxButtons.OK,
MessageBoxIcon.Information);
            */
#endregion
              }

        //this method gets info about the parent and SavedOp to make a copy of the vialocation in the
next segment.
        public void PasteMethod(ITxOperation ParentOp, ITxOperation SavedOp1)
        {

            TxGenericRoboticOperation GenericParent = ParentOp as TxGenericRoboticOperation;
            ITxWeldOperation WeldParent = ParentOp as ITxWeldOperation;
            ITxContinuousOperation ContParent = ParentOp as ITxContinuousOperation;

            TxRoboticViaLocationOperation SavedViaLoc = SavedOp1 as TxRoboticViaLocationOperation;
            TxWeldLocationOperation SavedWeldLoc = SavedOp1 as TxWeldLocationOperation;

            if (SavedViaLoc != null)
            {
                if (GenericParent != null)
                {
                    TxObjectList operationsToCopy = new TxObjectList();
                    operationsToCopy.Add(SavedViaLoc);
                    TxObjectList CopyOperation = null;
                    if (GenericParent.CanPasteList(operationsToCopy))
                    {
                        CopyOperation = GenericParent.Paste(operationsToCopy);
                        ITxObject CopyOperationOP = CopyOperation[0] as ITxObject;
                        // this list is used for when there is no location on the operation and the
copied operation cant be moved anywhere
                        TxObjectList ListParent = GenericParent.GetDirectDescendants(new
TxNoTypeFilter());
                        if (ListParent.Count > 1)
                        { GenericParent.MoveChildAfter(CopyOperationOP, null); }

                    }
                }
                else if (ContParent != null)
                {
                    TxObjectList operationsToCopy = new TxObjectList();
                    operationsToCopy.Add(SavedViaLoc);
                    TxObjectList CopyOperation = null;
                    if (ContParent.CanPasteList(operationsToCopy))
                    {
                        CopyOperation = ContParent.Paste(operationsToCopy);
                        ITxObject CopyOperationOP = CopyOperation[0] as ITxObject;
                        ContParent.MoveChildAfter(CopyOperationOP, null);
                    }
                }
                else
                {
                    TxObjectList operationsToCopy = new TxObjectList();
                    operationsToCopy.Add(SavedViaLoc);
                    TxObjectList CopyOperation = null;
                    if (WeldParent.CanPasteList(operationsToCopy))
                    {
                        CopyOperation = WeldParent.Paste(operationsToCopy);
```

```
                    ITxObject CopyOperationOP = CopyOperation[0] as ITxObject;
                    WeldParent.MoveChildAfter(CopyOperationOP, null);
                }
            }
        }
        else
        {
            if (GenericParent != null)
            {
                TxObjectList operationsToCopy = new TxObjectList();
                operationsToCopy.Add(SavedWeldLoc);
                TxObjectList CopyOperation = null;
                if (GenericParent.CanPasteList(operationsToCopy))
                {
                    CopyOperation = GenericParent.Paste(operationsToCopy);
                    ITxObject CopyOperationOP = CopyOperation[0] as ITxObject;
                    GenericParent.MoveChildAfter(CopyOperationOP, null);
                }
            }
            else if (ContParent != null)
            {
                TxObjectList operationsToCopy = new TxObjectList();
                operationsToCopy.Add(SavedWeldLoc);
                TxObjectList CopyOperation = null;
                if (ContParent.CanPasteList(operationsToCopy))
                {
                    CopyOperation = ContParent.Paste(operationsToCopy);
                    ITxObject CopyOperationOP = CopyOperation[0] as ITxObject;
                    ContParent.MoveChildAfter(CopyOperationOP, null);
                }
            }
            else
            {
                TxObjectList operationsToCopy = new TxObjectList();
                operationsToCopy.Add(SavedWeldLoc);
                TxObjectList CopyOperation = null;
                if (WeldParent.CanPasteList(operationsToCopy))
                {
                    CopyOperation = WeldParent.Paste(operationsToCopy);
                    ITxObject CopyOperationOP = CopyOperation[0] as ITxObject;
                    WeldParent.MoveChildAfter(CopyOperationOP, null);
                }
            }
        }


    }

      public override string Name
      {
            get
            {
                  return TecnomatixTesting1.StringTable.MODEL_OBJECTS_COMMAND_NAME;
            }
      }
}
   #endregion

#region VolumeRetrieve
/*public class TxVolumeRetrieve : TxDocument
{
```

```
        //Class members:
        //Interference object. Include as list?
        TxObjectList List_of_objects = TxApplication.ActiveDocument.GetObjectsByName("Volume");

        public void
        string objectType2 = List_of_objects[0].ToString();



    } */

    #endregion
}
```