Verification and visualization of equipment access on offshore platforms

Marius Hansen Røed

Master of Science in Engineering and ICT
Submission date: June 2014
Supervisor: Bjørn Haugen, IPM

Norwegian University of Science and Technology
Department of Engineering Design and Materials

THE NORWEGIAN UNIVERSITY
OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF ENGINEERING DESIGN
AND MATERIALS

MASTER THESIS SPRING 2014
FOR
STUD.TECHN. MARIUS RØED

VERIFICATION AND VISUALIZATION OF EQUIPMENT ACCESS ON OFFSHORE PLATFORMS

Verifikasjon og visualisering av tilkomst av utstyr på offshoreplattformer

Aker Solutions provides engineering services, from feasibility studies to designing the world's most advanced floating facilities, to the global oil and gas market. For more than 40 years Aker Solutions has developed industry-leading expertise on topsides, flow assurance, field architecture and a full range of floating production units.

The Aker Solutions Piping & Layout department is responsible for the design of piping, equipment, access routes and small outfitting structures. This design has to be according to the requirements of different standards like for instance NORSOK. They use the Plant Design Management System (PDMS), the most common CAD-system in Norwegian offshore industry, as the design tool. A key activity is to perform multidiscipline coordination with Process, Structural and Electrical departments in PDMS and seek to fulfill other discipline requirements.

The Material Handling group is a sub group of the Piping & Layout department. Their main responsibility is to handle all equipment and valves weighing more than 25 kg. Lifting equipment is designed in the 3D model and an obstruction volume is reserved for transport of the equipment and valves. Examples of equipment are pumps, motors, coils, heaters, coolers, compressors and gearboxes. One of the main deliveries is a description of how to handle and transport equipment. The equipment is moved onto a trolley and then transported to the required destination.

The procedure of validating the transport from its start point to its destination is done in PDMS or another 3D tool. Today this is a manual process done by the engineer. He/she simply moves the 3D object representing the trolley and equipment along the access routes in the 3D model. This process is comprehensive and time consuming. When there are changes of the equipment size or structural changes affecting the access way, the manual process has to be repeated from scratch.

The goal of the master thesis is to use dynamic relaxation to create a software application which helps the layout engineer to automatically check the feasibility to transport the equipment. This could potentially save a lot of time and add value by enabling the ability to run several iterations with a low cost.

This master thesis is a continuation of the project assignment Maris Røed wrote the fall 2013; "Verification and visualization of personnel access on oil platforms". Here, dynamic relaxation was used in a software application to verify the transport of a stretcher in a stairtower. Dynamic relaxation is a numerical method that aims to solve a linear or non-linear system of equations by looking at the steady-state solution of a pseudo-dynamic problem.

Some of the tasks involved are:

1. Describe the current process and challenges of equipment and access way modeling
2. Define the input parameters for the application to be developed
3. Define a new work flow for the modeling process with the software tool that is to be developed
4. Develop a .NET application which visualizes and verifies access of equipment and access ways
5. Validate the results of the new work flow (3) and the new application (4) against the current process (1)

Three weeks after start of the thesis work, an A3 sheet illustrating the work is to be handed in. A template for this presentation is available on the IPM's web site under the menu "Masteroppgave" (http://www.ntnu.no/ipm/masteroppgave). This sheet should be updated one week before the Master's thesis is submitted.

Performing a risk assessment of the planned work is obligatory. Known main activities must be risk assessed before they start, and the form must be handed in within 3 weeks of receiving the problem text. The form must be signed by your supervisor. All projects are to be assessed, even theoretical and virtual. Risk assessment is a running activity, and must be carried out before starting any activity that might lead to injury to humans or damage to materials/equipment or the external environment. Copies of signed risk assessments should also be included as an appendix of the finished project report.

The thesis should include the signed problem text, and be written as a research report with summary both in English and Norwegian, conclusion, literature references, table of contents, etc. During preparation of the text, the candidate should make efforts to create a well arranged and well written report. To ease the evaluation of the thesis, it is important to cross-reference text, tables and figures. For evaluation of the work a thorough discussion of results is appreciated.

The thesis shall be submitted electronically via DAIM, NTNU's system for Digital Archiving and Submission of Master's thesis.

The contact person are:	Christoffer Lange,	Aker Solutions (KBeDesign) and
	Thomas Mørk,	Aker Solutions (Piping & Layout)

[illegible]

Torgeir Welo
Head of Division

Bjørn Haugen

Bjørn Haugen
Associate Professor/Supervisor

NTNU
Norges teknisk-naturvitenskapelige universitet
Institutt for produktutvikling og materialer

Preface

This Master's thesis is a result of the work done during spring 2014 at the Department of Engineering Design and Materials (IPM) at Norwegian University of Science and Technology (NTNU) in Trondheim. The Master's thesis is the final diploma thesis at the program of study; Engineering and ICT at NTNU. The thesis is written in collaboration with Aker Solutions' KBeDesign and Piping and Layout departments. The academic supervisor from NTNU is Associate Professor Bjørn Haugen, and the supervisors from Aker Solutions are Christoffer Lange (Developer, KBeDesign), Thomas Mørk (Engineer, Design, Layout & Material Handling) and Alan Roach (Specialist Engineer, KBeDesign).

The Master's thesis is a continuation of the project work "Verification and visualization of personnel access on oil platforms" conducted during the fall 2013. The mathematical techniques and principles from the project are used in the Master's thesis, but the application is implemented this spring.

The thesis' goal is to develop a software application which can help the layout engineers in Aker Solutions to automatically check whether an equipment transport route on offshore platforms is feasible using dynamic relaxation.

Marius Hansen Røed

Marius Hansen Røed
10.06.2014
Trondheim

Acknowledgement

I would like to express my sincere gratitude to my advisor Associate Prof. Bjørn Haugen at NTNU for the continuous support of my Master's thesis, for his patience, motivation, enthusiasm and immense knowledge. His guidance helped me all the time during the development and implementation of the dynamic relaxation algorithm both during the project work and the Master's thesis.

Further, I would like to thank my advisors at Aker Solutions; Christoffer Lange (KBeDesign), Thomas Mørk (Piping & Layout) and Alan Roach (KBeDesign), for their encouragement, insightfully comments and response. Special thanks to Christoffer Lange. Without his enthusiasm, the result would not be as it is.

I will also give my sincere gratitude to the following people;

Olivier Doucet, KBeDesign Aker Solutions
Frans Erstad, KBeDesign Aker Solutions

for their use of time to help with technical problems, answers to necessary questions about the thesis and proofreading.

Lastly, I would like to thank my fellow students for their help with small, general problems during the thesis.

Abstract

The Master's thesis discusses verification of equipment transport on offshore platforms. The goal of the thesis is to use the form finding method dynamic relaxation to create a software application, which helps the layout engineers to automatically check whether an equipment transport route on offshore platforms is feasible using dynamic relaxation. This could potentially save a lot of time and add value by enabling the ability to run several iterations of the transport route with little effort. The thesis presents a new workflow, based on using the new application, and discusses advantages and disadvantages between the new and current workflow for transport route verification.

By using Autodesk Navisworks and its .NET and COM API, a plug-in application was created for Navisworks, to automatically verify movement of an object through a given access way. This was done using Navisworks clash detective feature through the .NET API. The plug-in application required input parameter such as a validation object, a boundary geometry and a path. The plug-in application is divided into two phases; one path optimizing phase, and one verification phase. The optimizing phase is a pre-phase for the verification phase, to optimize run-time speed. The verification phase uses dynamic relaxation to try to move the validation object through the given path. If the validation object clashes with the boundary geometry, the dynamic relaxation will try to find a point of equilibrium where there is no clash.

The plug-in application uses WPF (Windows Presentation Foundation) as a presentation system in .NET, with the MVVM (Model-View-ViewModel) architecture. The visualization of the validation object's movement, and the verification result, is done via Navisworks .NET API, where the movement is presented in Navisworks' own UI. The plug-in application's UI will present information about the status of a verification. The UI gives the opportunity for the user to monitor the velocity, acceleration and applied force in the dynamic relaxation method during verification using chart controls.

Sammendrag (Norwegian)

Denne masteroppgave omhandler verifisering av transport av utstyr på offshore plattformer. Oppgavens mål er å bruke form findings metoden "dynamic relaxation" til å utvikle en applikasjon, som kan hjelpe layout ingeniørene til å automatisk sjekke muligheten til å transportere utstyr. Dette kan potensielt medføre sparing av tid, ved å gi mulighet til å gjennomføre flere iterasjoner med lav kostnad. Oppgaven presenterer en ny arbeidsflyt, basert på den nyutviklede applikasjonen, og ser på fordeler og ulemper mellom den nye og tidligere arbeidsflyten for å transportere utstyr.

Ved å bruke Autodesk Navisworks sitt .NET og COM API, ble det utvikler en plugin applikasjon for Navisworks til å gjennomføre den automatiske verifiseringen av utstyrstransport. Dette ble gjort ved å bruke Navisworks sin innebygde funksjonalitet for "clash detection" via .NET API'et. Plugin applikasjonen krever spesifikke input parametre som verifikasjonsobjektet, grensegeometrien og en gitt følgebane. Plugin applikasjonen er delt inn i to hovedfaser; En optimaliseringsfase for banen, og en verifikasjonsfase. Optimaliseringsfasen er en for-fase til verifikasjonsfasen for å optimalisere kjøretiden til verifikasjonen. Verifikasjonsfasen bruker dynamic relaxation til å prøve å flytte på verifikasjonsobjektet gjennom den gitte følgebanen. Hvis verifikasjonsobjektet krasjer med grensegeometrien, vil dynamic relaxation metoden prøve å finne et likevektspunkt hvor verifikasjonsobjektet ikke kolliderer.

Plugin applikasjonen bruker WPF (Windows Presentation Foundation) som presentasjonssystem i .NET, med MVVM (Model-View-ViewModel) som arkitektur. Visualiseringen av verifikasjonsobjektets bevegelse, samt verifikasjonsresultatet, blir gjort via Navisworks .NET API, hvor bevegelsen vises gjennom Navisworks sitt eget UI. Plugin applikasjonens UI vil presentere informasjon om verifiseringen sin status, og gir muligheten for brukeren å overvåke fart, akselerasjon og kraft i dynamic relaxation metoden under verifikasjonen ved bruk av grafer/ trender.

Contents

Nomenclature

.NET	Software framework developed by Microsoft for Microsoft Windows.
AI	Artificial Intelligence
AML	Adaptive Modeling Language
API	Application Program Interface
CAD	Computer Aided Design
CAE	Computer Aided Engineering
COM (API)	Component Object Model
DR	Dynamic Relaxation
HDD	Hard Disk Drive
HHT	Hilber-Hughes-Taylor α damping
HSE	Health, Security and Environment
ICT	Information Communication Technology
KBE	Knowledge Based Engineering
MVC	Model View Controller
MVVM	Model-View-ViewModel
NORSOK	Norwegian Continental Shelf Competitive Position (Norsk sokkels konkurranseposisjon)
ODE	Ordinary Differential Equations
OS	Operating System
PDMS	Plant Design Management System
PHP	Php Hypertext Preprocessor
SDK	Software Developer Kit
SSD	Solid State Drive
UI	User Interface

WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

List of Figures

Chapter 1

Introduction

In this chapter the problem outline, objectives, and scope of the thesis will be presented. Since the thesis is written in collaboration with Aker Solutions KBeDesign and Piping & Layout, a short introduction will be given to these departments. This will give the reader a better understanding of why these departments wanted to collaborate on the thesis.

1.1 Problem outline, objectives and scope

The Material Handling Group in Aker Solutions has the responsibility to handle all equipment and valves weighing more than 25 kg. The lifting equipment is designed in the 3D model, and an obstruction volume is reserved for transport of the equipment and valves. Examples of equipment are pumps, motors, coils, heaters, coolers, compressors and gearboxes. One of the main deliveries is a description of how to handle and transport equipment. The equipment is moved onto a trolley and then transported to the required destination.

The procedure of validating the transport from its start point to its destination is done in PDMS or another 3D tool. Today this is a manual process done by an engineer. He/she simply moves the 3D object representing the trolley and equipment along the access routes in the 3D model. This process is comprehensive and time consuming. When there are changes of the equipment size or structural changes affecting the access way, the manual process has to be redone every time.

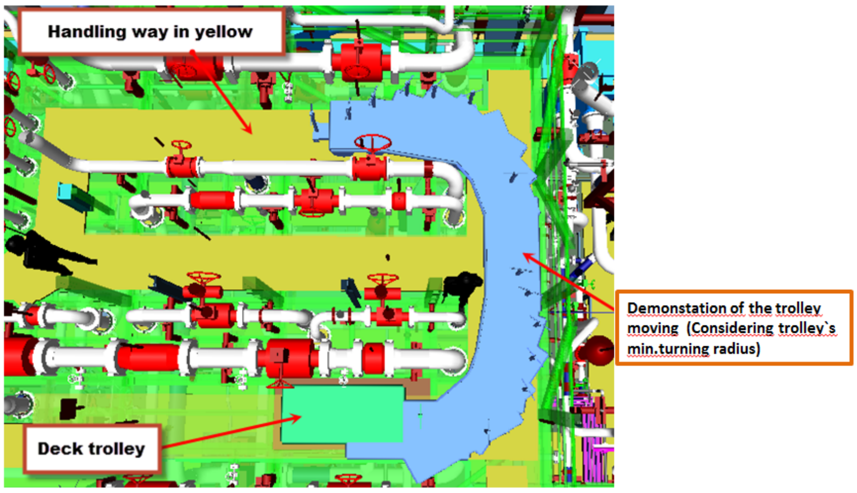

Handling way in yellow
Demonstation of the trolley moving (Considering trolley`s min.turning radius)
Deck trolley

Figure 1.1: The figure shows an example on the movement of a trolley on a offshore platform in the 3D model.

The goal of the Master's thesis is to use dynamic relaxation to create a software application, which helps the layout engineer to automatically check whether the equipment transport route is feasible. This could potentially save a lot of time and add value by enabling the ability to run several iterations of the transport route with little effort.

The Master's thesis is a continuation of the project assignment written fall 2013; "Verification and visualization of personnel access on oil platforms". Here, dynamic relaxation was used in a software application to verify the transport route of a stretcher in a stair tower. Dynamic relaxation is a numerical method that aims to solve a linear or non-linear system of equations by finding the steady-state solution of a pseudo-dynamic problem.

The KBeDesign team have, together with the Material Handling Group, set a list of task, which should be in the final delivery:

1. Describe the current process and challenges of equipment and access way modelling.
2. Define the input parameters for the application to be developed.
3. Define a new workflow for the modelling process with the software tool that is to be developed.
4. Develop a .NET application which visualizes and verifies access of equipment and access ways.
5. Validate the results of the new workflow (3) and the new application (4) against the current process (1).

The first two sub-tasks are only intended to give an introduction to how the problem is solved in current projects, and to show which parameters that are needed. The first sub-tasks are also subject to confidentiality, and has therefore less focus. The assignment's focus is therefore on the last three sub-tasks, where the forth sub-task is the main task with the primary focus in the thesis. This sub-task is also the most time consuming and have been given most priority during the 20 weeks.

1.2 Aker Solutions

Aker Solutions provides engineering services, from feasibility studies to designing the world's most advanced floating facilities, to the global oil and gas market. For more than 40 years Aker Solutions has developed industry-leading expertise on topsides, flow assurance, field architecture and a full range of floating production units. Many expertises are needed to achieve this, and departments as KBeDesign and the Piping and Layout are some important departments with the needed knowledge. Next, a short introduction will be given to the two departments and their knowledge expertise.

1.2.1 KBeDesign - Knowledge Based Engineering

Knowledge Based Engineering (KBE) is the strategic use of computerized engineering knowledge to automate design and engineering of variants [1]. KBE tools enable engineers to write dedicated applications that can perform complex and repetitive tasks in a much more efficient and accurate manner than what an ordinary engineer can do. KBE applications combines object-oriented programming and logic and reasoned systems (AI) with CAD / CAE systems. [1]

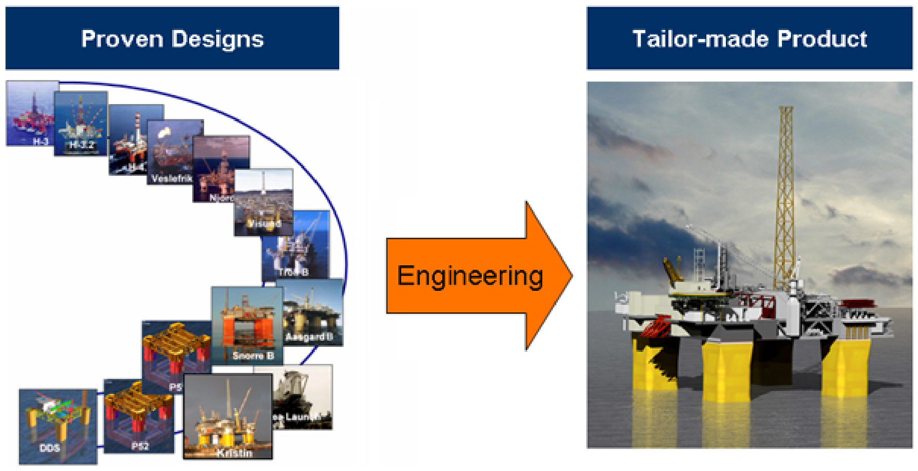


Figure 1.2: KBeDesign tool for automation of complex design. Figure from [4], © 2014 Aker Solutions

Aker Solutions KBeDesign team is leading pioneers in using KBE tools to create knowledge based software for engineers in the oil and gas industry. The KBeDesign application has improved quality and reduced cost and time hours for Aker Solutions for many years, and has become an important asset when competing for contracts. [1]

" - KBeDesign is not basically about the IT-tool, it is about utilizing the knowledge and experience that we as a company have gained over decades,"

- Department manager Jon Østmoen [4]

This thesis addresses just that. By exploiting prior knowledge, an application will be developed, which can help the layout engineers in Aker Solutions to automatically check the feasibility of equipment transport routes on offshore platforms.

1.2.2 Piping and Layout, Material Handling group

The Aker Solutions Piping & Layout department is responsible for the design of piping, equipment, access routes and small outfitting structures. This design has to be according to the requirements of different standards like for instance NORSOK. They use the Plant Design Management System (PDMS), the most common CAD-system in Norwegian offshore industry, as their design tool. A key activity is to perform multi discipline coordination with Process, Structural and Electrical departments in PDMS and seek to fulfil other discipline requirements.

The Material Handling group is a subgroup of the Piping & Layout department with main responsibility to handle all equipment and valves weighing more than 25 kg.

Chapter 2

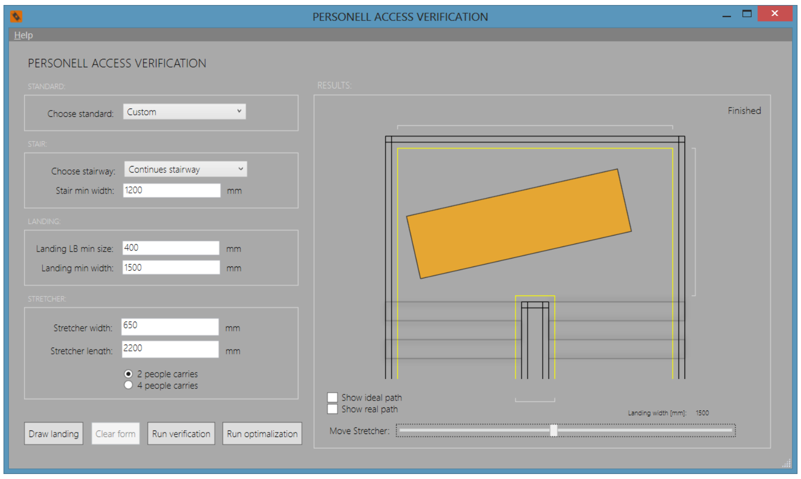
Background and motivations

During the summer and fall 2013 a collaboration with Aker Solutions and KBeDesign were well established, through a summer internship and the project work. This became the foundation for a continuing collaboration during the Master's thesis. The study program Engineering and ICT focus on combining computer science and engineering task in one overall study. The KBeDesign departments work area is a good example of such a combination, and they were therefore a good collaborator for this Master's thesis.

2.1 Project work - Personnel access verification

The project work "Verification and visualization of personnel access on oil platforms" was done during the fall 2013. During a summer internship at KBeDesign the same year, an introduction to an idea was given for the project work and the programming technology to be used (.NET, WPF, MVVM). A task draft was basically already created by one of the developers, and together with the supervisor at NTNU and a responsible developer at KBeDesign, the task was formed.

2.1.1 Short introduction



PERSONELL ACCESS VERIFICATION

STANDARD:
Choose standard: Custom

STAIR:
Choose stairway: Continues stairway
Stair min width: 1200 mm

LANDING:
Landing LB min size: 400 mm
Landing min width: 1500 mm

STRETCHER:
Stretcher width: 650 mm
Stretcher length: 2200 mm
[✓ 2 people carries]
4 people carries

RESULTS:
Finished
Show ideal path
Show real path
Landing width [mm]: 1500
Move Stretcher:

Figure 2.1: Screenshot of the final application from the project work

The project work discusses HSE in stairs on platforms, specifically verification of transport of injured personnel with stretcher in stair tower and hull. The project scope was to create an application that could verify and visualize this, based on input parameters. A mathematical algorithm/ formula was to be created for calculating the minimum landing size (the width of the stair landing/ response). After some consulting from the supervisor at NTNU, the verification algorithm was made using the form finding method Dynamic Relaxation (DR) , and geometric vector calculations. The application was implemented in .NET with C#. The visualization of the calculations in the application was implemented using WPF and the MVVM architecture.

2.1.2 The project work continuation to Master thesis

The actual application from the project work was not to be continued in the Master's thesis, but the algorithm and technology were going to be used. The application in the Master's thesis was still going to be solved as a 2D problem, but was going to be visualized in 3D. The dynamic relaxation technique was going to be used, as did the architecture (MVVM), the framework (.NET) and the UI presentation system (WPF).

2.2 Current process and challenges

The current process of equipment transport is very comprehensive and time consuming. As mentioned earlier, the engineer has to manually move the 3D object representing the trolley and equipment along the access routes in the 3D model. Possible collisions between the trolley object and walls, pipes, etc. must be seen with his or her own eyes. It is also impossible to give the trolley a realistic movement, and the verification can therefore not be one hundred percent correct. This is one of the main challenges for the engineers. Many of the trolleys have axle boundaries either in front or in the rear of the trolley. This means that the trolley only can move sideways either in front or in the back. This makes a large challenge when trying to get a realistic movement for the trolley.

2.2.1 Manual verification with Navisworks

The Material Handing group uses Autodesk Navisworks Simulate or Freedom to verify the equipment transport through the access ways. Navisworks enables engineers to holistically review integrated models and data with stakeholders to gain better control over project outcomes. [9]. Navisworks is only a tool to review models, modelled in PDMS (or in another 3D tool, ex. AutoCAD). Navisworks can't it self create any geometry. More of this in chapter 4.

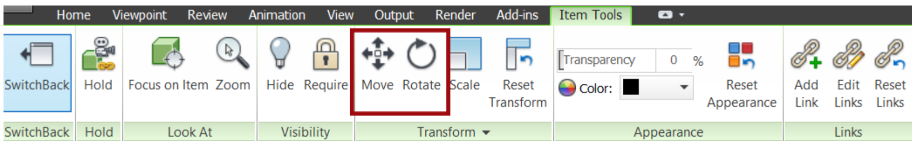


Figure 2.2: Screenshot of the features used in Navisworks to verify equipment transport today.

The engineers uses the two features *"Move"* and *"Rotate"* (Fig: 2.2) to move the trolley in a most realistic way as possible. This is very time consuming, since the engineers only can do one of the two simultaneously. If an engineer wants the trolley to go around a corner, he or she has to move it in x-, y- or z- direction, rotate it, move it again, rotate, move, etc. It will take a lot of move- and rotate steps just to get around a corner. If the transport path is very long, it will take a lot of time just to move it. In addition the engineer has to look for clashes with the boundary construction for each move. This is a very time consuming process.

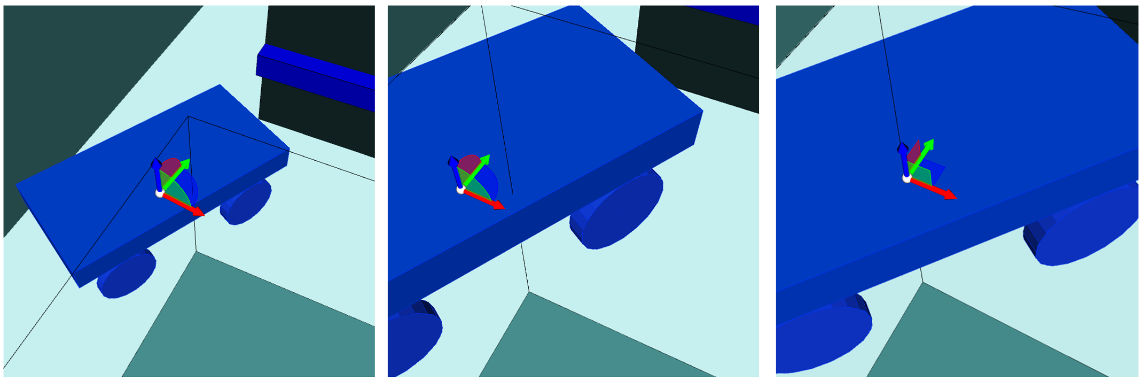


Figure 2.3: Screenshot of the verification process used in Navisworks today. It takes very many steps, just to get around a corner.

As shown in the figure (Fig: 2.3) it takes the engineer a lot of steps just to get around a corner, especially if it is very little space. It will also be more difficult to verify if there are many pipes or other components, instead of just a vertical wall.

2.2.2 Today's workflow

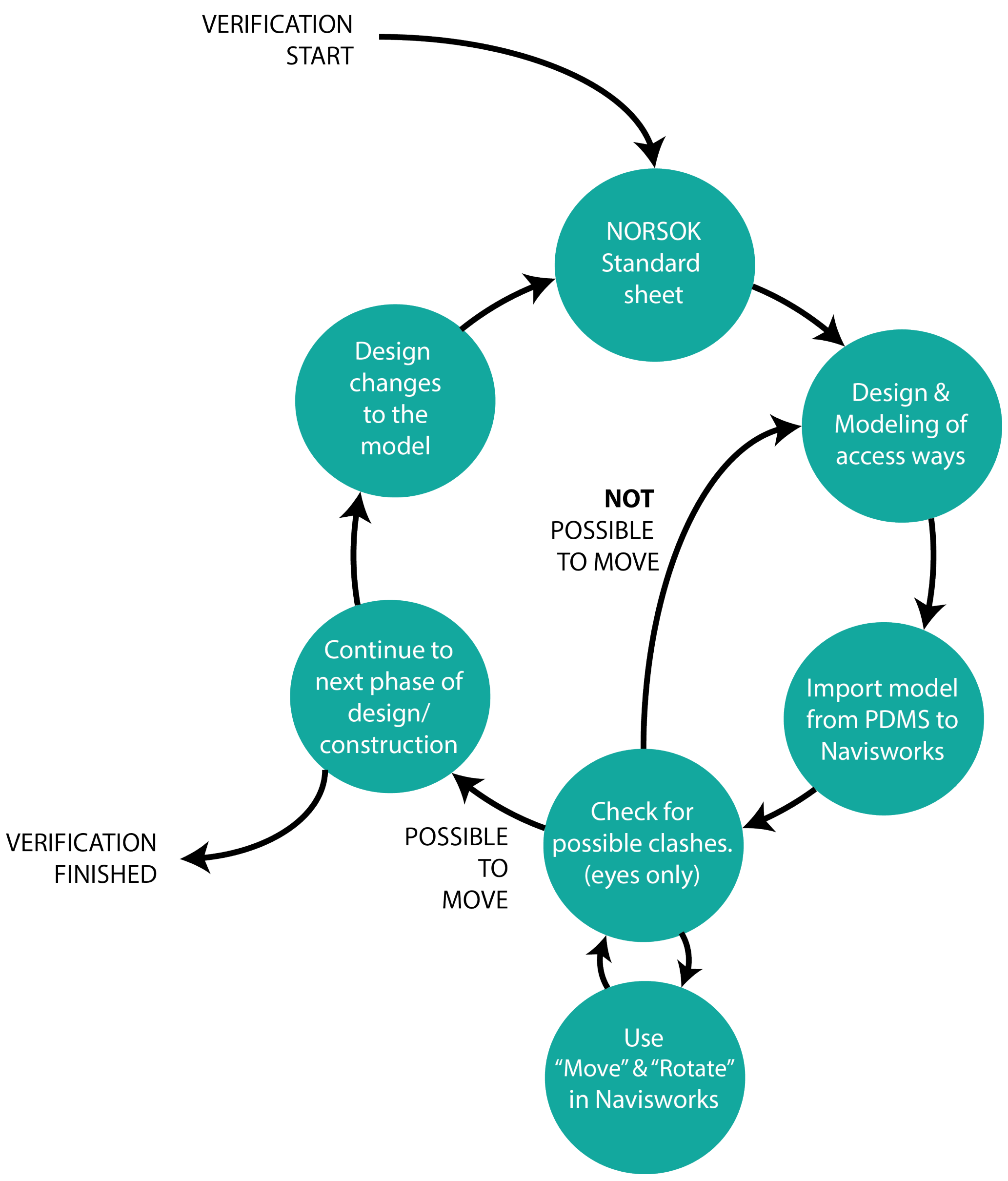

VERIFICATION START
NORSOK Standard sheet
Design & Modeling of access ways
Import model from PDMS to Navisworks
Check for possible clashes. (eyes only)
Use "Move" & "Rotate" in Navisworks
NOT POSSIBLE TO MOVE
POSSIBLE TO MOVE
Continue to next phase of design/ construction
Design changes to the model
VERIFICATION FINISHED

Figure 2.4: Illustration of the current workflow for verification og modelling of access ways.

As told, today's workflow is very time consuming, mostly because of the way the engineers have to create the movement of the trolley. In figure 2.4 today's verification work flow is described.

The engineer starts by finding necessary requirements for the access way based on NORSOK standards. See B.1. Then, based on these requirements, they design and model the required access ways. Unfortunately, this is not enough to secure that transport through the access ways is possible. The model is therefore imported to Navisworks from PDMS for review. By using the "Move" and "Rotate" features as mentioned earlier, the trolley's movement is created and the engineer can look for clashes using his/her eyes only. If movement is not possible, they need to go back to the design and modelling phase. If the movement constraints are very small, they may apply for deviation. If the transport is possible, the engineer can continue to the next phase of the design process. If nothing changes to the model during this phase, the verification is finished. Note that there is very little documentation done today. If there are changes of the equipment size, or structural changes affecting the access way, the work flow process has to be repeated.

Chapter 3

Workflow and input parameters

In this chapter the new workflow developed during the Master's thesis and the required input parameters for the application will be addressed. This chapter will cover the following two sub-tasks from the assignment text:

- Define the input parameters for the application to be developed.
- Define a new workflow for the modelling process with the software tool that is to be developed.

The last section in this chapter, discusses the choice between a standalone application, or a plug-in application to Navisworks, and why this choice was made.

3.1 Input parameters

One of the goals of the application was that it would be as generic as possible. This means that the application should work for every type of model object that is supposed to follow an access way and that any kind of model geometry could be the boundary (the geometry that the object shouldn't clash with). Therefore three main input parameters became necessary: The verification object (the object that is supposed to follow the access way, e.g. a trolley or stretcher), the boundary geometry (walls, pipes, etc.) and the path (the access way).

Each of these input parameters have to be defined as *Selection Sets* in Navisworks. A selection set is a selection of different model items from the current document. By using selection sets, it is much more easier to keep track of the different model item collections. As e.g. a trolley consists of many different components, it is easier to select a set rather than search for all the components in a large model tree during every verification. (Fig: 3.1 and Fig: 3.2)

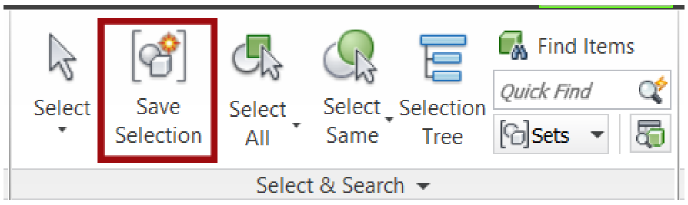
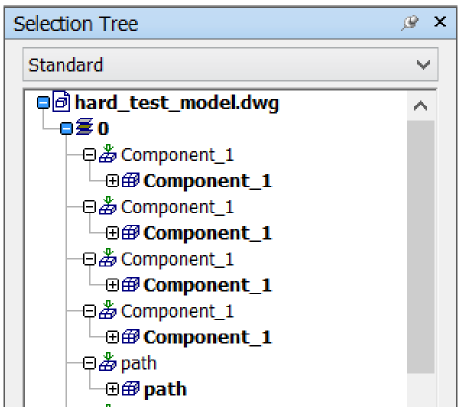
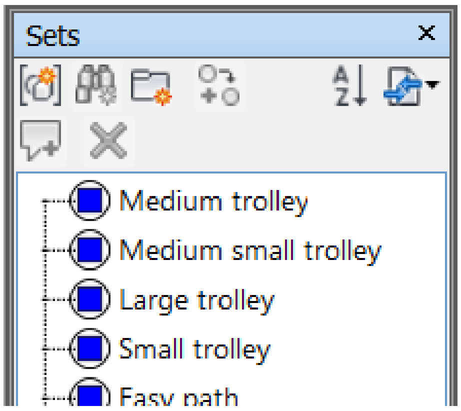


Figure 3.1: Screenshot of the selection menu, where to use the feature *Save Selection*.



```
hard_test_model.dwg
  0
    Component_1
      Component_1
    Component_1
      Component_1
    Component_1
      Component_1
    Component_1
      Component_1
    path
      path
```



```
Medium trolley
Medium small trolley
Large trolley
Small trolley
Easy path
```

Figure 3.2: Screenshot of the "Selection tree" window (t.l.), where the engineer has to find all needed components to e.g. a trolley, and then save them as a selection set, using the *Save Selection* feature in the menu. All the saved selection sets are shown in the "Sets" window. (t.r.)

More detailed information about how to do this in Navisworks, will be given in the Navisworks chapter.

3.1.1 Path

Dynamic Relaxation demand a defined path, which the validation object is supposed to follow. Since geometry can't be created in Navisworks, the path has to be defined at a earlier stage, i.e. in PDMS, AutoCAD etc. The path must therefore be an input parameter to the application.

The path has to be a continuous path, with a start and an end point (not a circle). It is very important that there is no gaps in the path, as the application will then fail. The corners should be rounded, using arc segments with a large radius. A half circle should be used if possible, see fig. 3.3. There are not any specific rules to where the path should be drawn in the model. A good basis is to use the centre line of the access way, and create arc with a reasonable radius for corners. A more sensible line placement should be chosen, in cases where the centre line obviously will cause a clash.

A selection set containing all needed line sets/ segments and arcs for representing the path must be created. The application will only find the created selection sets and not model items from the selection tree.

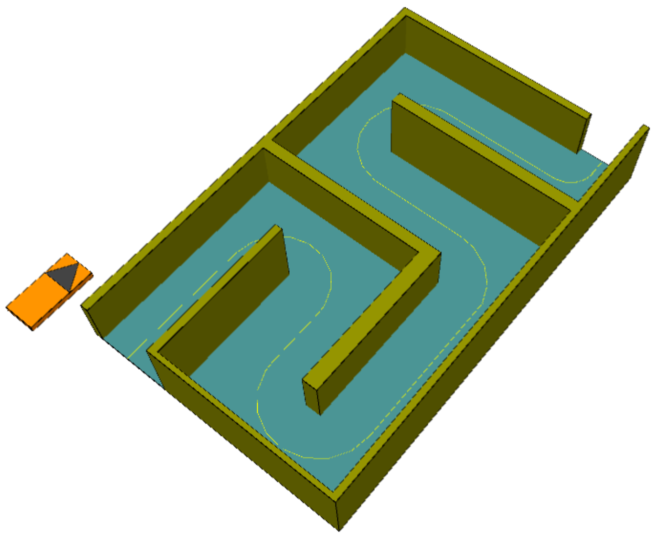


Figure 3.3: Screenshot of an example path in a corridor. The path is marked in yellow.

3.1.2 Boundary geometry

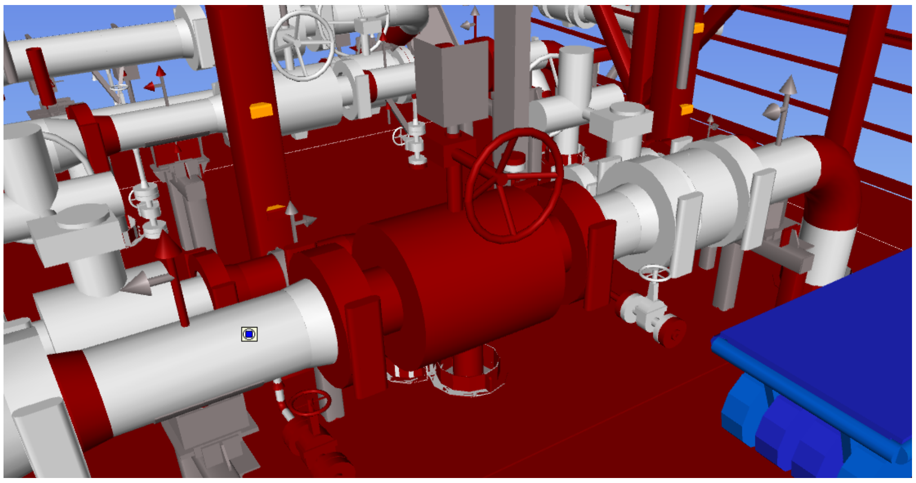


Figure 3.4: Screenshot of an example of boundary geometry from a offshore installation. Pipes with valves, fans, pumps etc.

The boundary geometry defines the model geometry that the validation object is supposed to be verified up against. This can be walls, pipes, racks, stairs, railings, pumps etc. (components that can be found on an offshore platform.) Since the application is generic in the choice of validation object, the boundary geometry can also be walls in e.g. a building. A selection set containing all needed polyface meshes from the model must be created.

3.1.3 Verification geometry

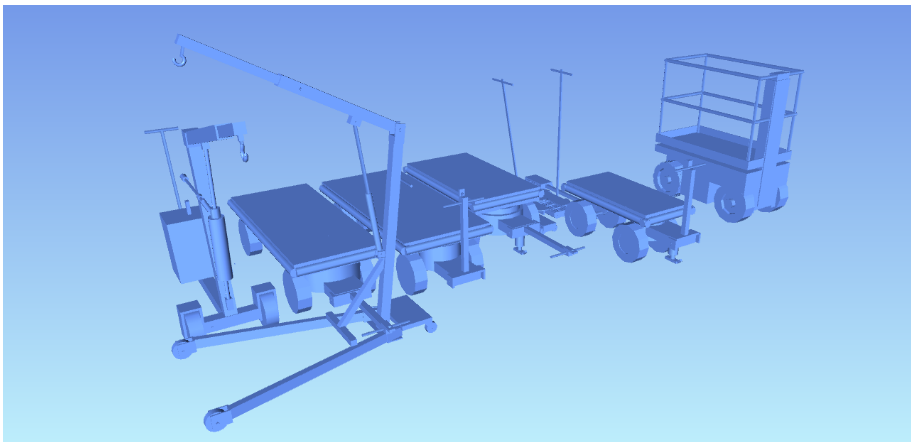


Figure 3.5: Screenshot of examples of different trolley validation objects. Size and type of trolley are decided based on need and NORSOK standard requirements.

The verification object is the model object which the user wants to use to verify transport movement on. It can be a trolley, a stretcher etc. In most cases (in this thesis), it would be a trolley based on NORSOK requirements. A selection set containing all needed polyface meshes and line set from the model needed to represent the trolley must be created.

3.2 New workflow

A new workflow had to be developed before implementation of the application. The new work flow (Fig: 3.6) is very similar to the old one and the only difference is the verification. Instead of doing manually clash verification with the "Move" and "Rotate" features, the engineers runs the Dynamic Clash Verification Plug-in, and runs the verification after setting the required settings. The workflow when using the plug-in application is shown in its own workflow schema (Fig: 3.7).

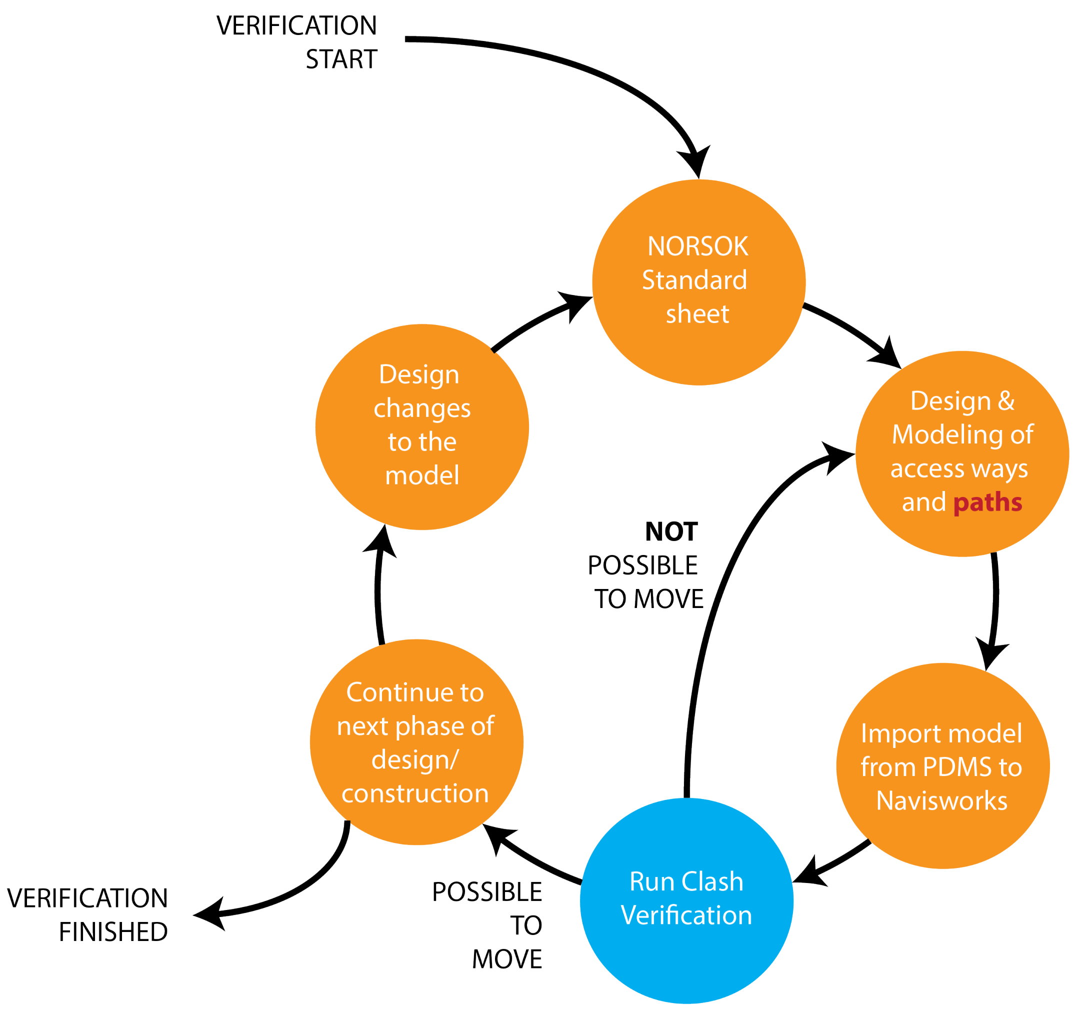

VERIFICATION START
NORSOK Standard sheet
Design & Modeling of access ways and paths
Import model from PDMS to Navisworks
Run Clash Verification
NOT POSSIBLE TO MOVE
POSSIBLE TO MOVE
Continue to next phase of design/ construction
Design changes to the model
VERIFICATION FINISHED

Figure 3.6: The new workflow, illustrated in a new schema. One of the new activities during the design and modelling of access way, is the creation of needed path(s). The creation of the path is explained in section 3.1.1. The blue circle, with "Run Clash Verification" represents the totally new activity in the work flow, and are explained in detail in its own work flow schema (Fig: 3.7). This activity will be much more time saving than the optional from the old workflow. During the verification the engineer can do other activities, since the calculations doesn't require full supervision.

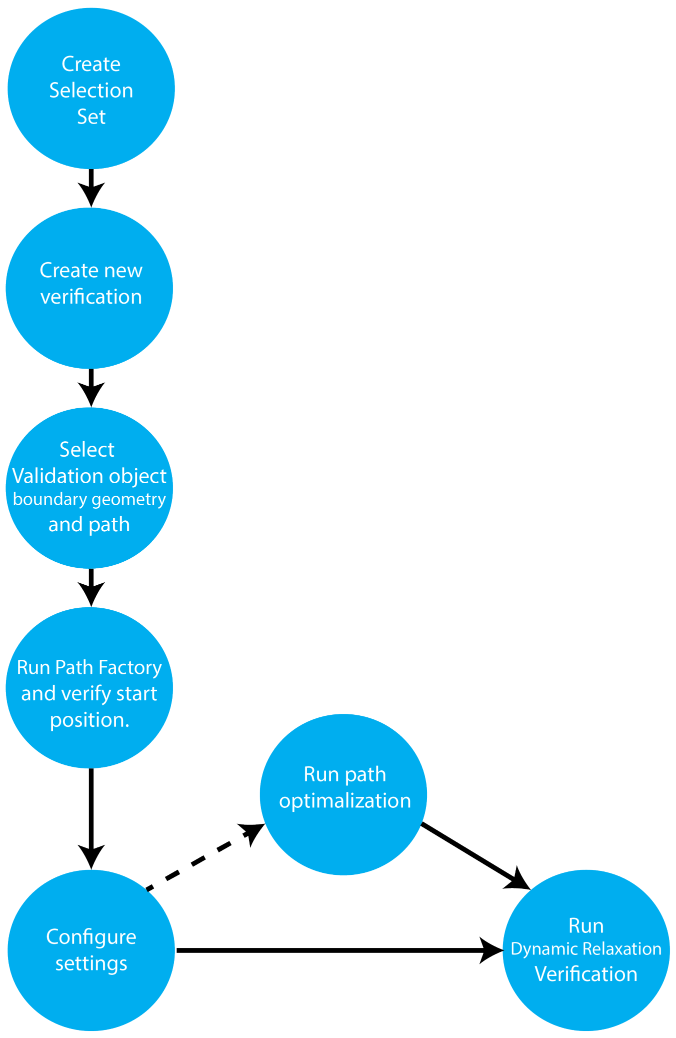

Create Selection Set
Create new verification
Select Validation object boundary geometry and path
Run Path Factory and verify start position.
Run path optimalization
Configure settings
Run Dynamic Relaxation Verification

Figure 3.7: The detailed workflow schema for the "Run Clash Verification" activity from Fig: 3.6. The schema explains how to go through the application on a high level.

3.3 Standalone v.s. plug-in application

A choice which had to be made before starting of implementation was which application type that should be made. Should the application be a standalone application, like in the project work, or should it be a integrated plug-in application to Navisworks? As the application had to use Navisworks .NET API and COM API, and a visualization would be easier to implement in a plug-in application, the choice became a integrated Navisworks plug-in. This would also give the users/ engineers the opportunity to use a already known software, i.e. less training on the new application, easier access together with other reviewing work on the model etc.

Chapter 4

Autodesk Navisworks

An introduction to Autodesk Navisworks will be given in this chapter. Knowledge of Navisworks is useful to see advantages and disadvantages of the software when developing the application in the Master's thesis and to fully get an understanding on how to use the software together with the application.

4.1 Introduction to Navisworks

The Navisworks software is a "project reviewing software" that enables architects, engineers, or rest of the entire project team to holistically review integrated models and data with stalk holders in order to gain better control over project outcomes. The software includes features for integration, analysis and communication to help project teams to coordinate between each discipline, to resolve conflicts and to plan projects before construction or renovation begins (From [9]).

The Navisworks family consists of three products:

- Freedom
- Simulate
- Manage

The three are only upgrades of each other, where Freedom is the simplest one, and Manage the most advanced one. Aker Solutions uses Freedom and Simulate in their projects, much because Manage's high price. On the other hand, the Manage version has a functionality for clash detection. As this functionality is usable through the API, and since Aker Solutions wasn't aware of this feature, it would be impractical to not use it for the thesis. Therefore Navisworks Manage will be the software of choice in this thesis.

Navisworks Manage is a very easy to use. As it is only a reviewing tool, it doesn't support functionality to create or change geometry. It is however enable to import a lot of different 3D-files and convert them to its own. There are shown some important features that the user needs to know to use the Master's thesis' own application on the following figures (Fig: 4.1, 4.2, 4.3).

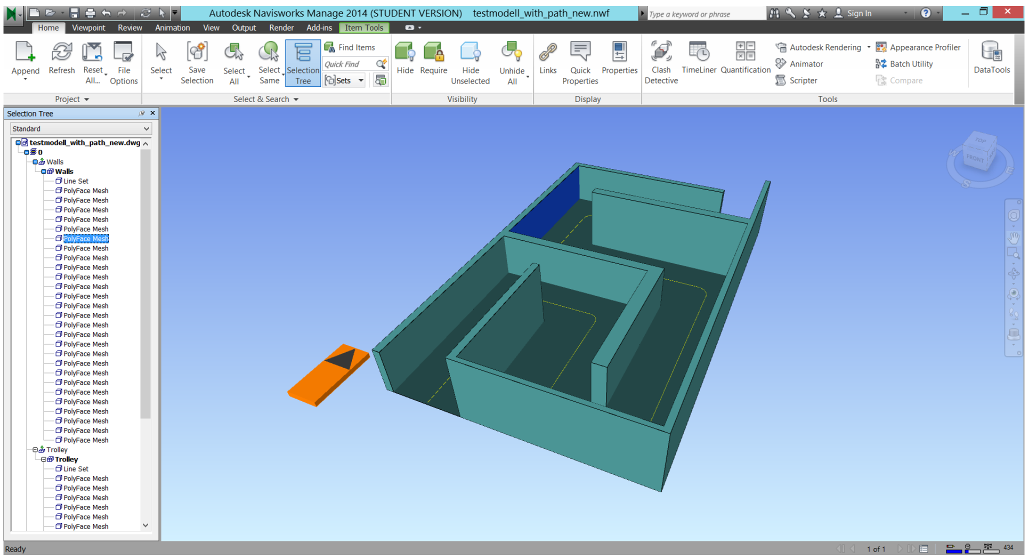


Figure 4.1: Screenshot of the main window in Navisworks after a file is opened. The current model shown in Navisworks is a test model for the application.

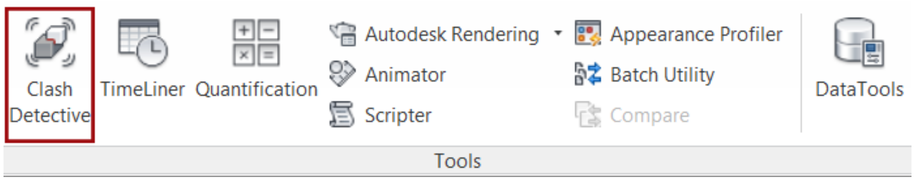


Figure 4.2: Screenshot of the toolbar, where to find Navisworks Manage Clash detective feature. An introduction to this feature will be given in the next section.

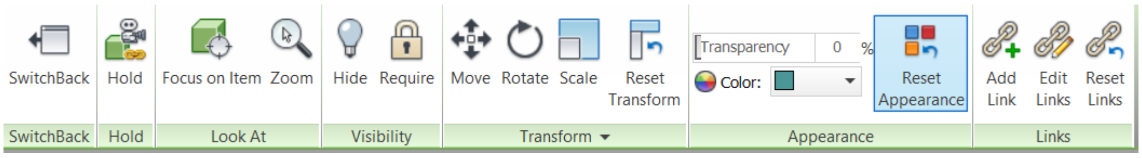


Figure 4.3: Screenshot of the item toolbar. This is the location of the "Move" and "Rotate" features used in the verification today. This can still be useful, since you need to rotate the validation object into the right direction. It is impossible for a application to know what is front and what is back of the validation object. This will be addressed later. The option to reset the original transform of the object which has been moved or rotated is also a feature.

4.2 Clash detection feature in Navisworks Manage 2014

The clash feature in Navisworks checks if two model objects are intersecting each other. If they do, Navisworks gives result data about clash distance and clash point for each polyface mesh surface. It will also give a report, and create viewpoints/ visualizations of the clash. The clash detection feature has four different clash tests, with a tolerance[1] as a input.

- Hard clash test
- Hard conservative clash test
- Duplicate clash test (will not be addressed in this thesis)
- Clearance clash test

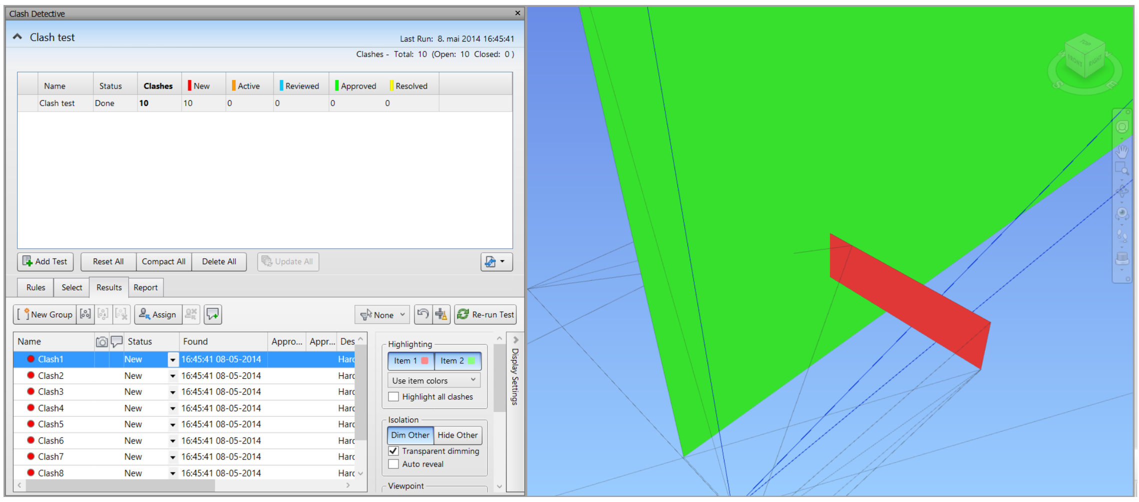
The hard clash test reports a clash in which the geometry of the first selection intersects the second selection by a distance of more than the tolerance input. If the hard conservative clash test is selected, it doesn't only show all clashes shown by the hard clash, but reports also all items that might clash. A clearance clash test is a test where the first selection may intersect with the second selection, but comes within a distance of less than the tolerance input[12].

Because of the way the dynamic relaxation method (which will be used in the clash application) works, the clearance clash test is the only test that will be used in the application. This will be explained in section 5.2.

A clash test in Navisworks applies a *Normal Intersection Method.* This method tests for intersections between any triangles[2] defining the two sections that are being tested. This can lead to missed clashes between objects that none of the triangles intersects. Example, if two pipes are exactly parallel and are overlapping each other at their ends. The pipes will intersect, yet none of the triangles which defines the geometry are clashing. This is the reason of using hard conservative clash testing, since it will report clashes that might clash. However, the conservative test can give false results of items that are not clashing. The clearance test uses the same method as the hard (standard) method, but adds a clearance distance to the intersection check.

[1]The tolerance controls the severity of the clashes reported and the ability to filter out negligible clashes, which can be assumed to be worked around on site. Any clashes found that are within this tolerance will be reported,whereas clashes outside of this tolerance will be ignored.[12]

[2]All geometry in Navisworks are composed of triangles.



Last Run: 8. mai 2014 16:45:41
Clashes - Total: 10 (Open: 10 Closed: 0)

Name	Status	Clashes	New	Active	Reviewed	Approved	Resolved
Clash test	Done	**10**	10	0	0	0	0

Name	Status	Found	Appro...	Appr...	Des
Clash1	New	16:45:41 08-05-2014			Harc
Clash2	New	16:45:41 08-05-2014			Harc
Clash3	New	16:45:41 08-05-2014			Harc
Clash4	New	16:45:41 08-05-2014			Harc
Clash5	New	16:45:41 08-05-2014			Harc
Clash6	New	16:45:41 08-05-2014			Harc
Clash7	New	16:45:41 08-05-2014			Harc
Clash8	New	16:45:41 08-05-2014			Harc

Figure 4.4: Screenshot of the clash detective feature in Navisworks. On the left side the clash detective window are shown, and on the right hand side is a visualization of the occurred clash.

4.3 Navisworks .NET API and COM API

Autodesk has offered a .NET API for Navisworks since 2011. Since then a lot of new features have been added to the API. In this application the Navisworks 2014 SDK is being used. During the work with the thesis the 2015 SDK was released. There are therefore possibilities for optimizing the application, or adding new features with the new SDK as further work. All necessary assemblies, documentation and code samples are available in the install folder for Navisworks (Manage and Simulate only).

Using the Navisworks .NET API, custom plug-ins to Autodesk Navisworks products can be created by running Navisworks from outside the GUI and automate certain tasks. It is also possible to utilize the .NET controls by embedding them in new applications. E.g. to create a document viewer for Navisworks. [14] The .NET API gives access to application and model/ document information. It can perform simple operations on documents as opning and saving files and running plug-ins without fully running the main application. It also offers the possibility to interoperate with the COM API.

The .NET API for Navisworks can be used in three ways [14] (Fig: 4.5):

- Plug-in: Makes it possible to create additional features for extending the Navisworks product.
- Automation: Makes it possible to drive products from outside to automate different assignments and invoke crucial plug-ins.
- Controls: Makes it possible to embed an Navisworks file viewer into a standalone application, or to open Navisworks documents without the need to run the full Navisworks program.

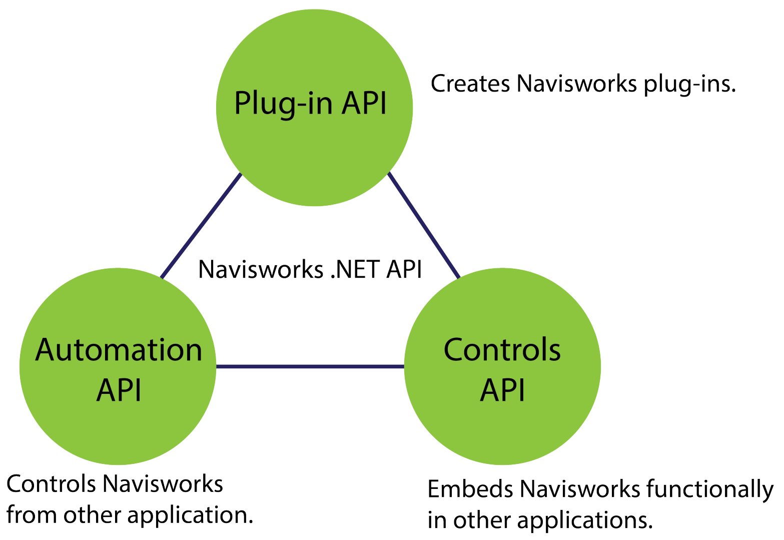

Plug-in API
Creates Navisworks plug-ins.
Navisworks .NET API
Automation API
Controls API
Controls Navisworks from other application.
Embeds Navisworks functionally in other applications.

Figure 4.5: The different Navisworks .NET API components. Illustration (in slide 4) based on illustration from [13]

The Autodesk Navisworks .NET API contains the following assemblies:

- `Autodesk.Navisworks.Api.dll` - This is the core API assembly, for use in plug-ins and control application development. This is the assembly used in this application.
- `Autodesk.Navisworks.Automation.dll` - For the automation API.
- `Autodesk.Navisworks.Controls.dll` - For the control API.

If the .NET API should interoperate with the COM API, the following assemblies must also be included:

- `Autodesk.Navisworks.Api.Interop.ComApi.dll` - Includes all COM API interfaces.
- `Autodesk Navisworks.ComAPI.dll` - Contains the`ComApiBridge` class, the bridge between COM and .NET API.

For a more detailed introduction to the Navisworks .NET API and COM API, see Navisworks API Referance and Developer Guide (NET API.chm) either from the web, or from the Master thesis' attachments.

Chapter 5

Dynamic Relaxation

The thesis introduces a dynamic method for solving the verification problem numerically. This chapter gives an introduction to the method and how it has been implemented in the thesis.

Dynamic relaxation (DR) is a numerical method that aims to solve a linear or non-linear system of equations by looking at the steady-state solution[1] of a pseudo-dynamic problem. It is modelled by a system of ordinary differential equations (ODE) in pseudo time t. The system is solved by numerical integration, where only the solution $t \to \infty$ is of interest. [2]

5.1 Form finding using Dynamic Relaxation

One of the many things DR can be used for is form finding [3]. The method aims to find a geometry where all forces are in equilibrium. All the nodes in the geometry are assigned a mass, and the stiffness is set to define the relationship that exists between the nodes. The system starts to oscillate about an equilibrium position under the influence of loads. One can simulate a dynamic process where the geometry is updated at each step in the iteration by carrying out an iterative process, based on the effect of the influencing loads.

Damping may be introduced to the system to make the DR more efficient (by reducing the number of iterations). There are two ways to do this, either viscous and kinetic damping. In this case, viscous damping has been chosen. A viscous force between the nodes may be assumed between the nodes when using this damping. The advantage of using viscous damping is that the representation of the system becomes more realistic. In principle a drag cable between the nodes is added.[3]

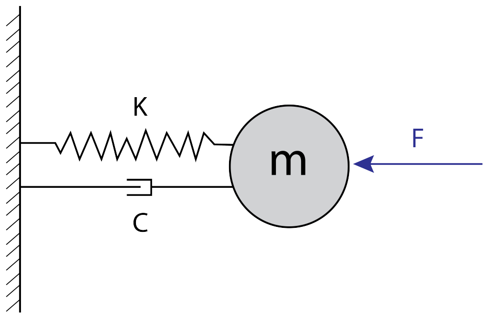
[1]**Steady-State:** A system that is in steady-state, has a number of features which are consistent over time. This means that for the features on the system, the partial derivatives with respect to time equal zero

5.1.1 The method

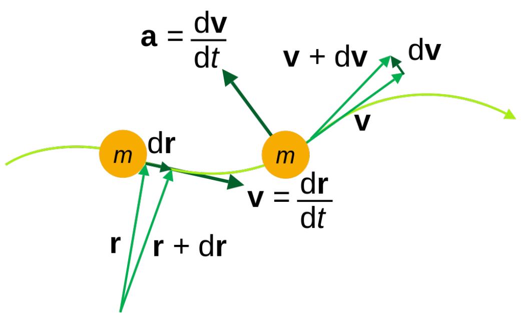
The equation the method (DR) originated from, is the equation of motion 5.1 (fig: 5.1b), which for a discretized[2] system is:

$$\mathbf{P}_{ji} = [\Sigma K\delta]_{ji} + \mathbf{M}_{ji}\ddot{\delta}_{ji} + C\dot{\delta}_{ji} \tag{5.1}$$

where ji refers to the jth node in the ith direction in a discretized system. $\mathbf{P}_{ji}$ is the external load vector acting on the node. $[\Sigma K\delta]_{ji}$ is the internal loads vector where K is the node stiffness and δ is the displacements. C is the viscous damping coefficient, and $\ddot{\delta}_{ji}$ and $\dot{\delta}_{ji}$ are the nodal acceleration and nodal velocities respectively. [3]



(a) A node with mass m, is connected to a spring and a damper. The damping coefficient is represented by C. F is external load.



(b) The principle of the motion equation. Mass m, position r (δ), velocity v ($\dot{\delta}$), acceleration a ($\ddot{\delta}$). Fig. 1 from [25].

By introducing the residual force $\mathbf{R}_{ji}$ as the difference between the external an internal forces, and combine it with 5.1, the result becomes:

$$\mathbf{R}_{ji} = \mathbf{P}_{ji} - [\Sigma K\delta]_{ji} \tag{5.2}$$

$$\mathbf{R}_{ji} = \mathbf{M}_{ji}\ddot{\delta}_{ji} + C\dot{\delta}_{ji} \tag{5.3}$$

The acceleration and velocity are then calculated at each iteration in order to predict the displacement at time $n+1$:

$$\ddot{\delta}^{n+1}_{ji} = \frac{1}{\mathbf{M}_{ji}}(\mathbf{R}^{n}_{ji} - C\dot{\delta}^{n}_{ji}) \tag{5.4}$$

[2]**Discrete system:** A system with a finite countable number of states.

$$\dot{\delta}_{ji}^{n+1} = \dot{\delta}_{ji}^{n} + \ddot{\delta}_{ji}^{n}\Delta t \tag{5.5}$$

$$\delta_{ji}^{n+1} = \delta_{ji}^{n} + \dot{\delta}_{ji}^{n}\Delta t \tag{5.6}$$

The iterative process continues by using 5.4, 5.5 and 5.6 until the residual forces are close to zero.

One needs to assume necessary values for the mass, damping coefficient, stiffness and time increment to let the iterative method above work realistically, . This might be a "trial and error" approach and is not always straight forward as we will see later. In this case the damping coefficient is set to the value for critical damping[3]:

$$C_{critical} = 2\sqrt{mK}$$

In the section 5.2, the use of DR in this thesis will be explained. There are some differences between conventions and names which are used in this thesis and the ones explained above. There are also made some special adaptations. The method above is intended as a brief summary of the method. Refer to [2] and [3] for further reading.

5.1.2 HHT - Hilber-Hughes-Taylor α damping correction

To increase the accuracy and stability in the dynamic relaxation method, a simplified Hilber-Hughes-Taylor method (also called the α-method) is introduced. This method is an implicit method for solving the transient problem, which attempts to increase the magnitude of numerical damping present without degrading the order of accuracy. (From [23]) The equation 5.2 becomes:

$$\mathbf{R}_{ji} = \alpha \cdot \mathbf{R}_{j+1,i} - (1-\alpha)(\mathbf{P}_{ji} - [\Sigma K\delta]_{ji}) \tag{5.7}$$

where α is set to 0.1.

5.2 Use of Dynamic Relaxation in this Master's thesis

DR is used in the application to find a possible form of the path, which the validation object is supposed to follow without clashing with the boundary geometry. (Fig.

[3]**Critical damping:** The damping of a system can be described as critical damped. This means that the system returns to equilibrium as quickly as possible without oscillating.

5.1) The validation object's centre point (which is the point that follows the path) is decided by the value of the bounding box centre of the model item. If additional information about the validation object is selected, the centre is moved into correct position. E.g. if the validation object represents a trolley with axles, on which only one of the two axles have the possibility to rotate, the centre is moved to the rotating axle. (Fig: 5.2)

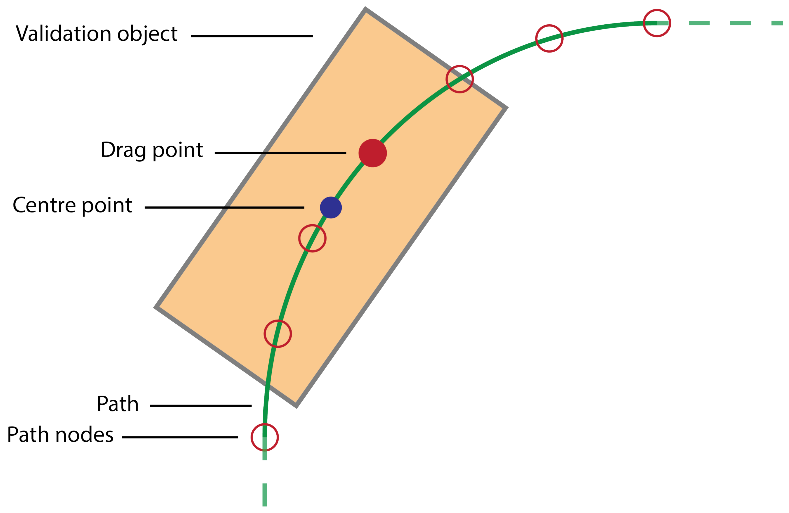


Figure 5.1: Illustration overview of the different important elements in the verification. The different elements are described in the text below.

The path consist of many path nodes which are used as drag nodes in the algorithm. The step size between each drag node is decided by the movement precision, selected in the application during optimizing. If optimizing is not done, there will only be nodes in the edges of the line and arc segments from Navisworks. If optimizing is done before verification, the movement precision can be altered before running the verification. Altering the movement precision after optimizing will cause a better performance for the verification, but less accurate. If the step distance between two drag nodes are smaller than the movement precision, the algorithm will just skip to the next drag point. This will continue until the movement size is equal or smaller than the distance between the current centre node to the drag node.

The angle of the validation object is decided based on the vector direction of the vector between the current node and the drag node. The smaller step size, the more accurate is the validation object's angle, and the movement becomes more realistic.

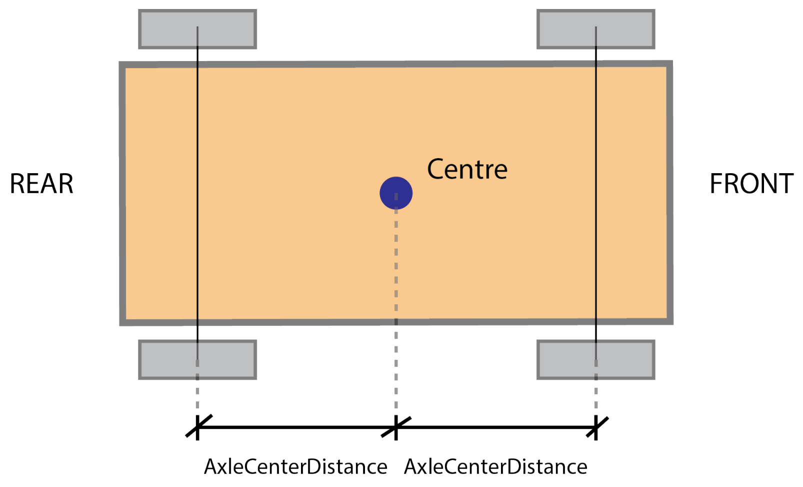


Figure 5.2: The figure illustrates the axle situation if the validation object is a trolley. The user needs to apply which of the two axles that are rotating, and the distance from the centre of the trolley to the axle.

The drag node moves to the next path node as the algorithm iterates forward. Between the validation object's centre node and the drag node, there is a viscous damper. (Fig: 5.3) The distance that occur between the centre node and drag node (δ), together with the damper that exists between them, results in a velocity ($\dot{\delta}$) and acceleration ($\ddot{\delta}$) for the validation object. Δt is set to 0.8 during ordinary movement, and to 0.1 during clashes. Δt can be tuned to get the most optimized run-time for the algorithm. These values are the best values found in this thesis by "trail and error".

With a certain number of time steps, the damper will cause the validation object to move towards the drag node. The drag node will continue to the next path node as soon as the system is in equilibrium (when the resultants are approximately zero, 1×10^{-6}).

The validation object has also been given an reasonable value for the mass and stiffness.

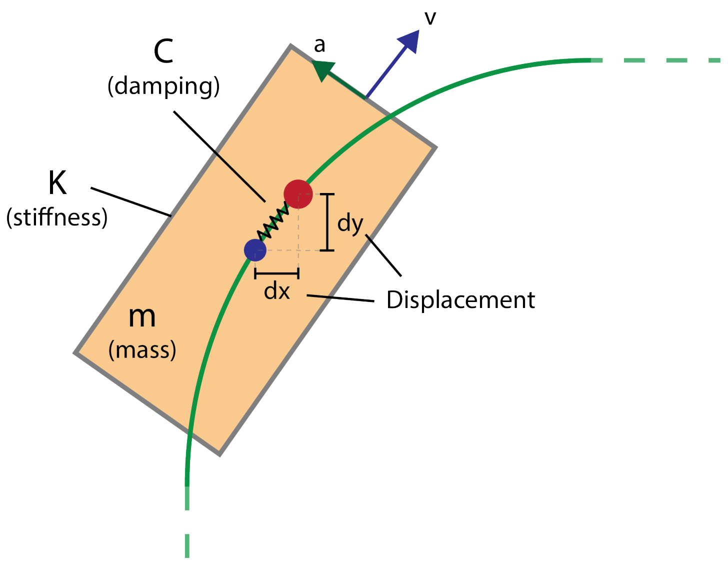


Figure 5.3: Figure describing what happens when the validation object moves against a new drag node.

When a clash is registered, a force is applied on the validation object to force it back inside the boundary. (Fig: 5.4) The size of the force is based on the distance "clash distance" outside the boundary and the boundary geometry's stiffness ($K_{boundary}$). The stiffness of the boundary is much larger than the stiffness (K_{object}) of the validation object. The applied force will create unbalance in the system, and the algorithm will try to find equilibrium. When it does, the drag node will move on to the next path node. If it doesn't, the algorithm will return false and break. The iteration tolerance is set to 500 iterations. If the system uses more than 500 iterations between two nodes on the path, it is not possible to get into equilibrium, and the validation object fails to get through.

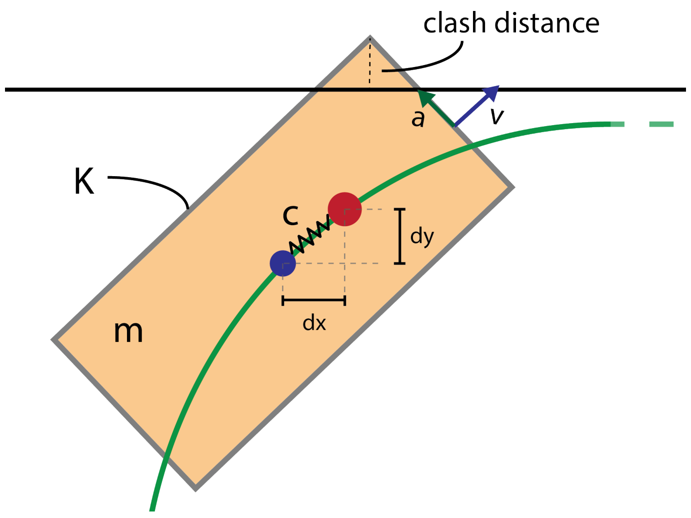


Figure 5.4: Figure describing what happens when the validation object clashes into the boundary geometry.

Necessary values for the mass, damping, stiffness and time increment needs to be assumed, as mentioned earlier. These values were chosen based one the values used in the project work. In the project work the "trial and error" method had been used to find some good values. The only criteria are that $K_{wall} \gg K_{stretcher}$ and $C = 2\sqrt{M \sum K}$. The following values have been set:

Object mass, m	2
Stiffness boundary, K_{wall}	8
Stiffness object, $K_{stretcher}$	1
Object damping, C	$2\sqrt{m \cdot \sum K}$

Figure 5.5 and 5.6 shows the force, damping and velocity on the way into and out of the boundary geometry. The stiffness during clash becomes:

$$K_{clash} = \sum K = K_O + K_B \tag{5.8}$$

where:

O - Object

B - Boundary

The damping during clash is:

$$C > C_{cr} = 2\sqrt{m(K_O + K_B)} \tag{5.9}$$

$$\begin{aligned} C &= C_O + C_B \\ C_O + C_B &= 2\sqrt{m(K_O + K_B)} \\ C_B &= 2\sqrt{m(K_O + K_B)} - C_O \\ C_B &= 2\sqrt{m(K_O + K_B)} - 2\sqrt{m \cdot K_O} \\ C_B &= 2\sqrt{m} \cdot (\sqrt{K_O + K_B} - \sqrt{K_O}) \end{aligned} \tag{5.10}$$

where C_B is the damping applied during clashing.

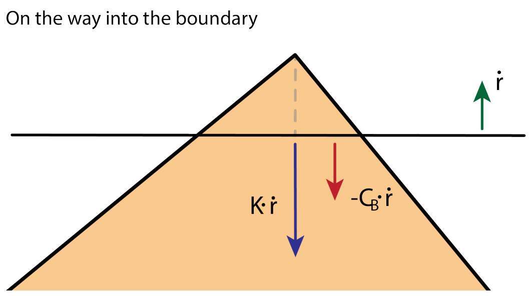


Figure 5.5: The figure shows the validation object in the moment it clashes, with velocity direction ($\dot{r}$) into the boundary geometry. The damping and the applied force have the same direction, directed against the validation object.

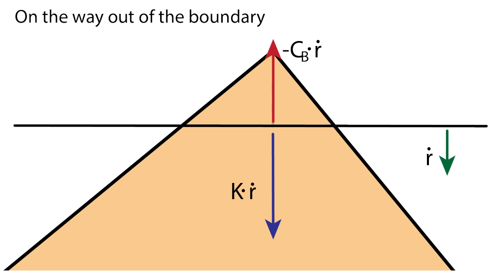


Figure 5.6: The figure shows the validation object during clashing, with velocity direction ($\dot{r}$) out of the boundary geometry. The damping and the applied force have the opposite directions.

When the resultant is closed to zero, the validation object has found equilibrium. The current point during equilibrium is registered, before the algorithm goes on to the next drag node. (Fig: 5.7)

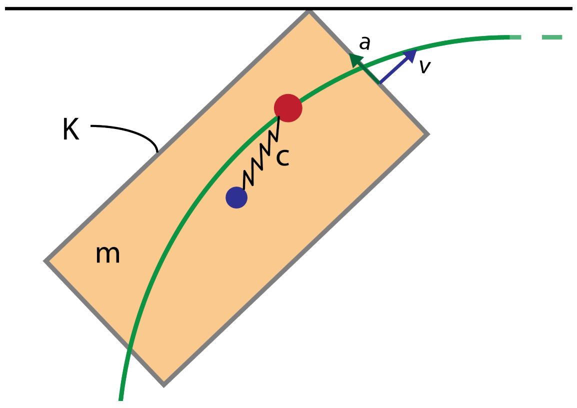


Figure 5.7: Illustration of the moment where the validation object is in equilibrium.

Dynamic relaxation demands a certain intrusion in the boundary geometry to get in equilibrium. Since the clash detection feature in Navisworks will not allow such an intrusion will a clearance clash test be used. This means that the clash detective feature will detect every clash within 100 *mm* from the boundary geometry. This causes that the DR method to allow an intrusion of 100 *mm*.

5.2.1 The final algorithm

In this sub section, the dynamic relaxation algorithm from the application is shown in pseudo code, in a general way. The method `SolveDynamicRelaxtion()` is the starting point of the algorithm and iterates and finds the next drag node. It also performs the rotation of the validation object. It sends the next drag node as a parameter to the `NextIteration()` method, which applies dynamic relaxation on the node. The pseudo code is based on a object orientated programming language, where methods for rotating, moving, and clash checking already exists.

```
function SolveDynamicRelaxation
    Init validation object properties.
    Init boundary geometry properties.

    Add first node to RealPathList

    for all nodes in path do
        if Distance: current node and drag node < MovementPrecision then
            continue
        end if
        Rotate the validation object based on direction vector.
        if NextIteration(node) then
            continue
        else
            return false
        end if
    end for
    return true
end function
```

```
function NextIteration(Node node)
    Set all dynamic variables equal zero
    while ($\mathbf{R} < 1 \times 10^{-6}$|| firstTime) do
        Set force and moments equal zero
        $\delta_x^n = N^n$.X - node.X
        $\delta_y^n = N^n$.Y - node.Y
        RunClashTest()
        Set value of $\Delta t$
        for all clash do
            $\mathbf{D}^n$ = GetClash()
            $F_x^n \mathrel{+}= K_{x,boundary} \cdot D_x$
            $F_y^n \mathrel{+}= K_{y,boundary} \cdot D_y$
        end for

        alpha = 0.1
        $\mathbf{R}^n = \mathbf{R}^n \cdot alpha + (1.0 - alpha)(\mathbf{F^n} - \mathbf{K}_{\text{object}} \cdot \delta^n - \mathbf{C}_{\text{object}} \cdot \dot{\delta}^n)$
        $\ddot{\delta}^{n+1} = \frac{1}{\mathbf{M}} \cdot \mathbf{R}_{ji}$
        $\dot{\delta}^{n+1} = \dot{\delta}^n + \ddot{\delta}^n \Delta t$
        $\delta^{n+1} = \delta^n + \dot{\delta}^n \Delta t$

        Node newNode = new Node($\delta_x^{n+1}$,$\delta_y^{n+1}$)
        MoveValidationObjectTo(newNode)
        $N^n$ = newNode
        if tolerans > 500 then
            return false
        end if
    end while
    RealPath.Add($N^n$)
    return true
end function
```

Chapter 6

System design of plug-in application

In this chapter the system design of the plug-in application will be presented. There will be given an introduction to the different technologies, framework, techniques and architectures used. The use of Autodesk Navisworks .NET and COM APIs in the plug-in application will be addressed.

6.1 Microsoft .NET technologies and architectures

As Navisworks has a .NET API and that the project work was done in .NET, the choice of framework for the Master thesis was obvious. .NET 4.0 and the same architectural pattern and presentation system were used in the project work assignment, .NET 4.0 was selected. As WPF and the MVVM pattern are central in the application, a short introduction to these will be given.

6.1.1 Windows Presentation Foundation

The Windows Presentation Foundation (WPF) is a presentation system for building Windows client applications. The core of WPF is that it is a resolution-independent and vector-based rendering engine, which is built to take advantage of modern graphics hardware. WPF extends the core with a extensive set of development features, which includes the Extensible Application Markup Language (XAML) , controls, data binding, layout, 2-D and 3-D graphics, animation, styles, templates, documents, media, text and typography. (From [6].)

The plug-in application also uses the WPF Toolkit for creating charts. The WPF Toolkit is a large collection of WPF features and components that has being made available outside of the normal .NET Framework ship cycle. [15]

6.1.2 Model-View-ViewModel architecture

The Model-View-ViewModel (MVVM) is an architectural pattern used in software engineering that originated from Microsoft. Largely based on the Model-View-Controller pattern (MVC), MVVM is a specific implementation targeted at UI development platforms which support the event-driven programming in Windows Presentation Foundation (WPF) and Silverlight on the .NET platforms using XAML and .NET languages. (From [8].)

The plug-in application uses the MVVM Light Toolkit. The toolkit's purpose is to accelerate the creation and development of MVVM applications in WPF. The `ViewModelBase` interface is the main part from the toolkit that has been used. It's being used to raise property change to the view, and for message passing between the different view-models.

Data bindings

Data bindings provides a simple way for Silverlight-based applications to display and interact with data. The way data is displayed is separated from the management of the data. A connection, or binding, between the UI and a data object allows data to flow between the two. When a binding is established and the data changes, the UI elements that are bound to the data can reflect changes automatically. Similarly, changes made by the user in a UI element can be reflected in the data object. (From [7].)

Data bindings have been used in all parts of the application. The "code-behind" has not been used. This is a one of many benefits by using data bindings. The figure below shows have the different parts of the application are connected and where data bindings are used.

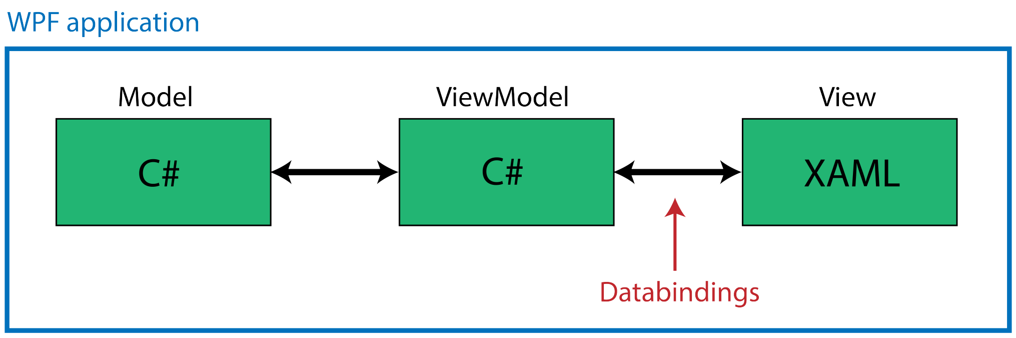


Figure 6.1: The MVVM pattern. The figure shows where data bindings are used.

6.2 Plugin application architecture

The plug-in application has an extended MVVM architecture, with Views, View-Models, Models and classes for connecting the application to Navisworks as a plug-in/ add-in, as figure 6.2 shows. The application also uses the observer pattern for updating the GUI with information about progress and current results.

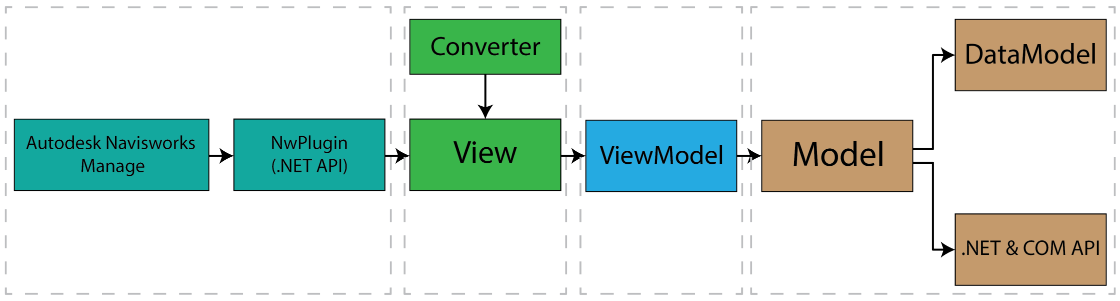


Figure 6.2: Illustration of the overall architecture of the plug-in application. It shows how Autodesk Navisworks is connected to classes in the NwPlugin namespace, which is connected to the View, then the ViewModels, Models, DataModels and the .NET & COM API. The different sections are explained in the following figures; 6.3, 6.4, 6.5 and 6.6.

6.2.1 Navisworks Plugin (NwPlugin)

The NwPlugin namespace consists of two classes and are the link to Navisworks; `DynamicClashVerificationPlugin` and `DynamicClashVerificationAddin`. The add-in plug-in class creates a generic plug-in for Navisworks, called through the GUI. The dock pane plug-in class makes the add-in plug-in to be a dockable pane, and adds it to Navisworks GUI[17]. The plug-in is loaded when Navisworks starts, and unloaded when it stops. The plug-in is copied into the designated folder, which usually is the "/Plugin" folder in the "/Autodesk/Navisworks Manage 2014/" from "AppData".

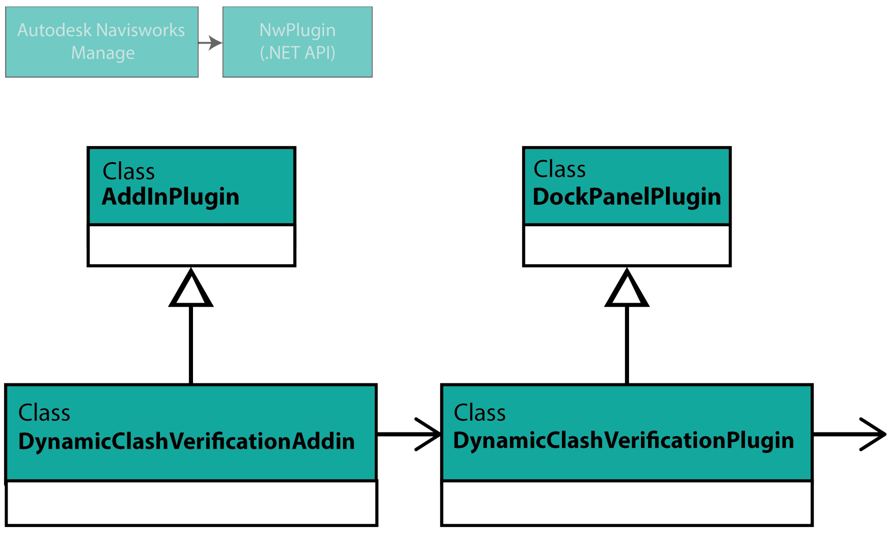

Autodesk Navisworks Manage
NwPlugin (.NET API)
Class AddInPlugin
Class DockPanelPlugin
Class DynamicClashVerificationAddin
Class DynamicClashVerificationPlugin

Figure 6.3: Class diagram over the NwPlugin namespace. The `AddInPlugin` and `DockPanelPlugin` are from Navisworks' .NET API.

6.2.2 The Views

The view consists of three user controls; `DynamicClashVerificationView`, `ResultView` and the `ControlView`. The `DynamicClashVerificationView` uses the static resource class `UniversalValueConverter`, which inherits from the `IValueConverter` interface[1]. The `UniversalValueConverter` is used to convert properties from a View-Model, which is bounded to a property of an element in the user control and converting it to the right type. E.g. from `string` to `PathGeometry` for the icons.

The plug-in user control window are separated into two main parts; the dynamic clash verification view and the control view. The `DynamicClashVerificationView` contains of elements for creating, deleting and reviewing all clash verification sessions. The view makes it possible to create different verification sessions, for example for different validation objects, paths etc. It will also give information about status of the different sessions. The `DynamicClashVerificationView` is a part of the `ControlView`, but has its own `ViewModel`.

The `ResultView` is a part of a `TabItem` in the `ControlView` and shows graphs/ trends of the velocity, acceleration and applied force for the validation object. It uses the WPF Charting Toolkit, with a custom styling of the graphs.

The `ControlView` is the main plug-in window. In addition to include the `DynamicClashVerificationView` and the `ResultView`, it consist of elements for selection validation object, boundary geometry, path, settings, running the validation and path optimization.

Almost all of the elements in the views have custom styling. This to get a consistency between the UI design in Windows 7 and Windows 8.

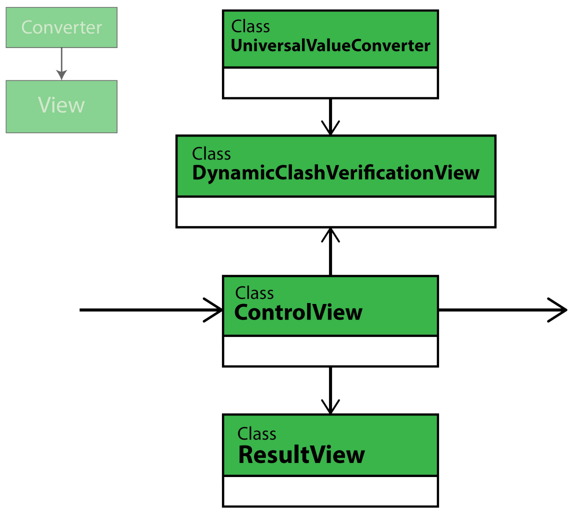


Figure 6.4: Class diagram of the different views. Properties and fields are not included. The "code-behind" classes are not being used, due the MVVM pattern.

[1]A **converter** that implements the `System.Windows.Data.IValueConverter` interface can change data from one type to another, translate data based on cultural information, or modify other aspects of the presentation (From [18]). See [18] for more info.

6.2.3 The ViewModels

The ViewModel classes are corresponding to the different views. The `DynamicClash-VerificationViewModel` contains needed properties and method for handling needed bindings to the `DynamicClashVerificationView`. It includes methods for handling the current selected verification session, and communicates this to the `ControlViewModel` through message passing via MVVM Light, which handles the communication with the model.

The `ResultViewModel` contains properties and method for handling the the elements of the `Result- View`. It implements the ResultListener interface, and listen for changes in velocity, acceleration and force during the dynamic relaxation verification. It communicates with the `ControlViewModel` with message passing via MVVM Light.

The `ControlViewModel` includes properties and method for all of the elements in the control view. This includes elements as validation object, boundary geometry and path selection, settings, progress bar and verification. The `ControlViewModel` communicates with the `DynamicClashVerification` class. The `DynamicClashVerification` class can be seen as a controller which controls all needed user input from the View-Model and creates required objects from the model.

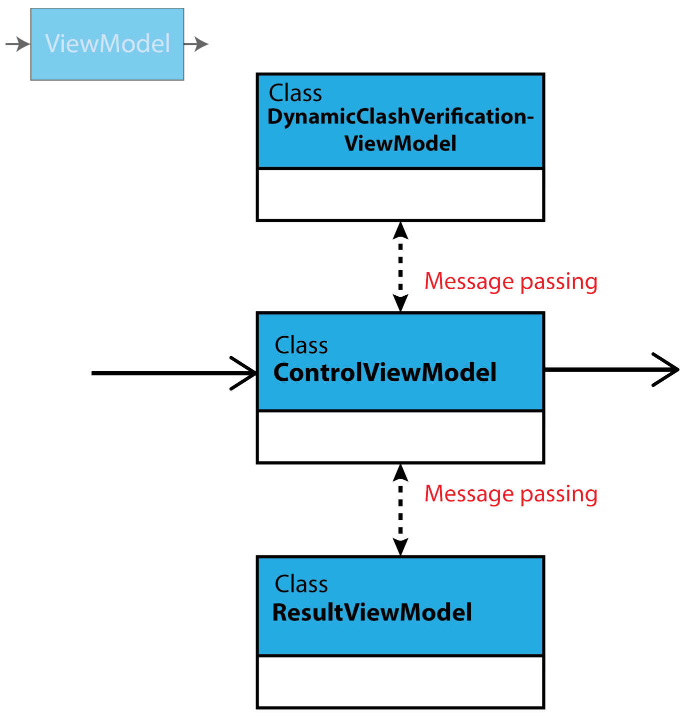


Figure 6.5: Class diagram of the ViewModels and their communication between each other. The implemented interfaces are not shown in the figure, since they aren't important in this context. The ResultListnener interface will be adressed in section 6.2.5.

6.2.4 The Models

The main model is the `DynamicClashVerification`. As mentioned, this class is the connection between the `ControlViewModel` and all the model classes. The model classes are divided into "ordinary" model classes, and data model classes. The data model classes are classes for holding data, while the other model classes contains methods using the data models.

The model classes:

- `DynamicClashTest` class
- `DynamicMovement` class
- `DynamicRelaxationSolver` class
- `CallBackGeomListener` class

The `DynamicClashTest` class contains properties and methods for creating a new dynamic clash test. The class uses the Navisworks Clash API. Unlike a clash test in Navisworks, a dynamic clash test contains thousands of Navisworks clash tests. It will create a new Navisworks clash test for each step the validation object does. A clash test consists of many 2D clashes, one for each triangle surface in the model item that is clashing. Based on these 2D clashes, the `DynamicClashTest` class creates a 3D clash (for each clash the validation object does), whose properties are calculated based on all the 2D clashes from the Navisworks clash test. In short, one can might say that a 3D clash is the same as a Navisworks clash test.

The `DynamicMovement` class contains properties and methods for creating a dynamic movement based on a set validation object and path. Based on a parameter to the class' constructor, it can create a movement with, or without clash testing.

The `DynamicRelaxationSolver` class contains properties and methods for running the dynamic relaxation algorithm on a validation object, on a path with a set boundary geometry.

The `CallBackGeomListener` class uses the `COMApi.InwSimplePrimitivesCB` interface from the COM API, for getting the primitives (line, point, triangle, snap point) from a model item fragment. More about this in section 7.1.

The data models:

- `ValidationObject` class
- `Path` class
- `Clash3D` class
- `Line3D` class

The `ValidationObject` class contains data about the validation object model item. It also includes method for transforming the position of the validation object. The `Line3D` represents a line in 3D and `Clash3D` represents a clash in 3D. The `Path` class includes properties and method for storing and creating a path. It includes methods for creating a path, from the .NET API from a `ModelItemcollection` and for optimizing a path based on clashes. More details about the creation of the path will be addressed in section 7.2.

The following classes from the Navisworks .NET API are used. See the Navisworks documentation for detailed information about them:

- `Document` class
- `DocumentClash` class
- `DocumentClashTest` class
- `ModelItemCollection` class
- `ModelItem` class
- `Point3D` class
- `Vector3D` class
- `Rotation3D` class

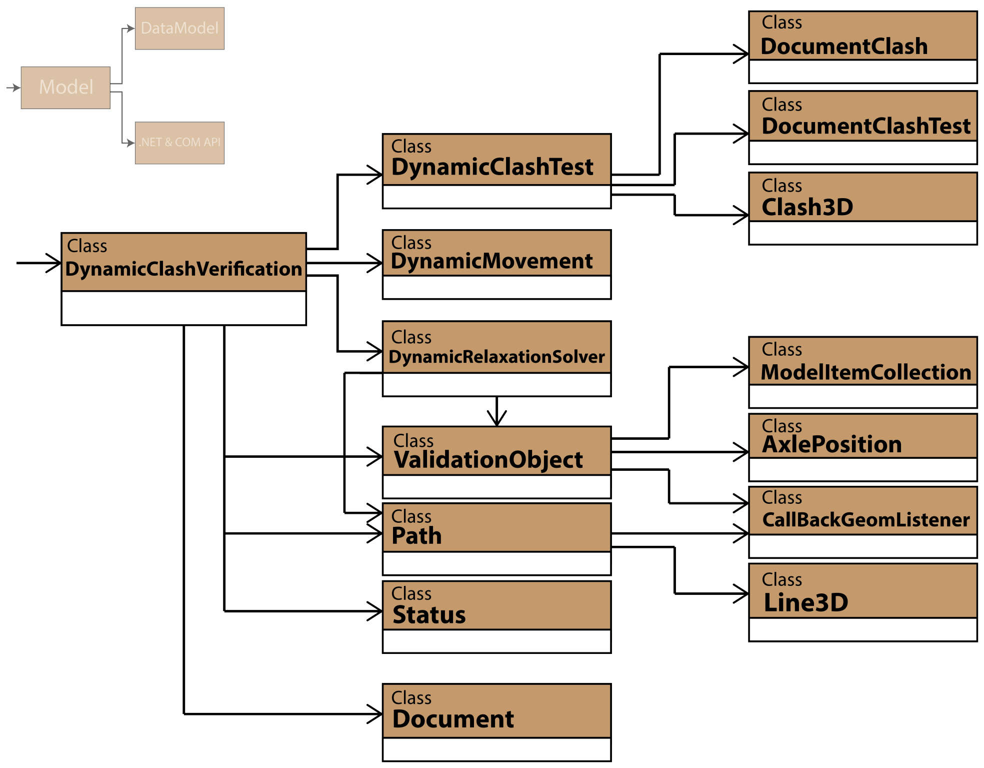


Figure 6.6: A simplified class diagram of the model classes. See documentation for full class diagram.

6.2.5 The Observer design pattern

The observer design pattern enables listeners to subscribe to an observable provider to receive notifications when changes happen on observable objects. The pattern defines a observable object and zero, one or more observers. Each observer is added as listeners within the observable object. Whenever a predefined change happens, the observable object will automatically fire a notification change message to all of its listeners. The listeners can then do changes on itself, based on the observable's changes. The listeners have to implement a defined interface for the method that the observable is going to call. [24]

The plug-in application uses this pattern to observe the progress during the path factory, path optimization and during the dynamic relaxation verification. The `ControlViewModel` is added as a progress listener to the `DynamicRelaxationSolver`, which inherits from the abstract `Observable- Progress` class. The progress is presented with a progressbar in the `ControlView`. In addition, the `ResultViewModel` is added as a result listener to the `DynamicRelaxationSolver`, which also is an `ObservableResult`. This gives information about the current velocity, acceleration and applied force of the validation object and is presented in two chart controls in the `ResultView`.

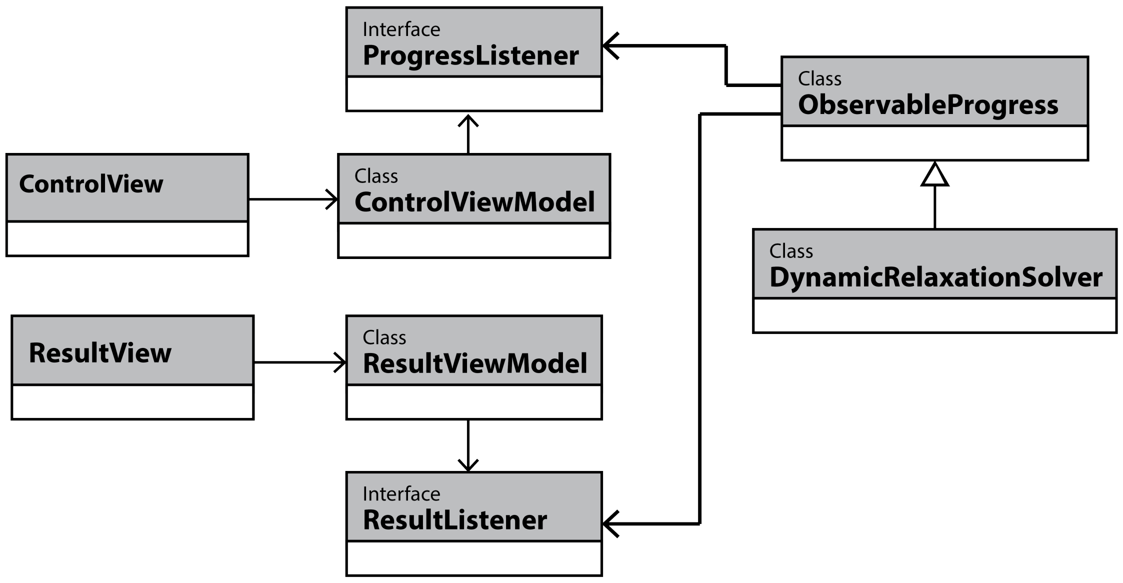


Figure 6.7: Class diagram including the classes in the observer design pattern. Note that the `ObservableProgress` class also includes methods for `ObservableResult`, since a class only can inherit from one super class. See source code documentation for method definitions.

Chapter 7

Implementation of the plug-in application

In this chapter the different mathematical implementations will be addressed such as the path generation and path optimization, the implementation of the main algorithm, and the use of .NET API and COM API. The two main phases of the application will be addressed, and the flow to some impotent sequential operations. There will also be given a short introduction to the matrix translation used together with the COM API.

7.1 Use of Navisworks .NET and COM APIs

There are two main functionalities from the .NET and COM API in Navisworks used in the implementation. Besides the classes for creating a Navisworks plug-in (`NwPlugin`-namespace), the transform features from .NET API and features for getting primitives of models from the COM API that are used. There are also some data model classes in use, such as `Point3D` and `Vector3D`.

7.1.1 Getting primitives from COM API

Navisworks .NET API doesn't have features for getting detailed info about the geometry of a model. It can give information about which type of primitives a model item consists of (points, lines, triangles), give the centre position of the bounding box and each maximum and minimum position values (the point on the model item which is closest and farthest from the origin). The positions of these items are given in the local coordinate system of the item and no translation method or matrix is able to convert it to global system.

However, the COM API can give the value of the primitives. These values will also only be given in its local coordinate system. The COM API has methods for converting to the global coordinate system according to its documentation. These methods are unfortunately very difficult to use. Since the developers at Autodesk had problems using them, they had to be developed manually by using the COM API method `GetLocalToWorldMatrix()`. This methods returns a array representing a 4x4 matrix.

The 4x4 transformation matrix is defined like this [20]:

$$\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{S} \\ \mathbf{p}^T & U \end{bmatrix} \tag{7.1}$$

where

- **R** - (3x3) direction cosines
- **S** - (3x1) translation vector
- $\mathbf{P}^T$ - (3x1) perspective vector
- U - The uniform scaling factor

Only the translation vector is important since the path has neither been scaled nor rotated or had its perspective altered. The global coordinates are therefore defined as:

$$\mathbf{P}_{\text{Global}} = \mathbf{P}_{\text{Local}} \cdot \mathbf{S} = \begin{bmatrix} x_{\text{local}} \\ y_{\text{local}} \\ z_{\text{local}} \end{bmatrix} \cdot \begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} = \begin{bmatrix} x_{\text{global}} \\ y_{\text{global}} \\ z_{\text{global}} \end{bmatrix} \tag{7.2}$$

This gives that the elements with index 13, 14 and 15 are the three elements in the S-vector from the array returned from `GetLocalToWorldMatrix()`.

The following code snipped shows how to get the primitives from a model item (`oModelColl`). The code goes through each `InwOaPath3` (3D path) from the model and from each path it goes through each `InwOaFragment3` (3D fragment). It gets its primitives with the `GenerateSimplePrimitives` method. A fragment can e.g. be a point, a line or a triangle.

```
//convert to COM selection
COMApi.InwOpState oState = ComBridge.State;
COMApi.InwOpSelection oSel = ComBridge.ToInwOpSelection(oModelColl);

// create the callback object. the false parameter is only to tell the
    callback object that
// it is not used to generate a path.
CallBackGeomListener callbkListener = new CallBackGeomListener(true);

foreach (COMApi.InwOaPath3 path in oSel.Paths())
{
    foreach (COMApi.InwOaFragment3 frag in path.Fragments())
    {
        //Getting the transformation matrix. See report for more
           information.
        COMApi.InwLTransform3f3 localToWorld =
           (COMApi.InwLTransform3f3)(object)frag.GetLocalToWorldMatrix();

        Array array_v1 = (Array)(object)localToWorld.Matrix;
        double x = (Double)array_v1.GetValue(13);
        double y = (Double)array_v1.GetValue(14);
        double z = (Double)array_v1.GetValue(15);

        Vector3D S = new Vector3D(x, y, z);
        // Setting the translation vector, to be used to generate the
           primitives.
        callbkListener.SetTranslationVector(S);
        frag.GenerateSimplePrimitives(COMApi.nwEVertexProperty.eNORMAL,
           callbkListener);
    }
}
```

The `GenerateSimplePrimitives` method calls on the correct methods via the `COMApi.-InwSimplePrimitivesCB` interface, which is implemented in the `CallBackGeomListener`. After all `InwOaPath3` in the model item have been handled, it returns a list of all the points the model item consists of. This list is then used to generate the path further. [19]

```
public void Line(COMApi.InwSimpleVertex v1, COMApi.InwSimpleVertex v2);

public void Point(COMApi.InwSimpleVertex v1);

public void SnapPoint(COMApi.InwSimpleVertex v1);

public void Triangle(COMApi.InwSimpleVertex v1, COMApi.InwSimpleVertex
   v2, COMApi.InwSimpleVertex v3);
```

See application souce code and Navisworks COM API documentation for more information.

7.1.2 Transformation of Validation object position

The transformation features through the .NET API enables the developer to override the transformation of a model item in Navisworks. The transformation is only an override and Navisworks will always remember the original position an object had when it was imported to Navisworks for the first time. Then using this feature, the position of the object will always move in the UI of Navisworks. This means that during the dynamic relaxation calculations, the user will see the validation object move inside the Navisworks UI.

The method used to do translation transformation is:

```
public void MoveValidationObject(Vector3D newDirection)
{
    //Creates a transformation matrix, based on the translation vector.
    Transform3D oNewOverrideTrans =
       Transform3D.CreateTranslation(newDirection);

    //Overrides the current position sith the new transform. See NW doc
    //to read why you need to override permanent, and what this mean.
    doc.Models.OverridePermanentTransform(Model, oNewOverrideTrans,
       true);
}
```

The method takes a vector as parameter, and moves the Model based on the vector. This vector is the same as the **S** vector described in the previous section. Based on the translation vector, a transformation matrix is created and used to override the transformation of the model. According to .NET API documentation, it should also be possible to create the rotation matrix and use it in the same transformation matrix as the **S** vector. Unfortunately this isn't working. The API is only doing the rotation and not translation if the matrices are put in the same override function together. Therefore is the rotation done separately.

The method for doing rotation transformation is:

```
public void RotateValidationObject(Rotation3D rotation, Point3D
   rotationPoint)
{
    //Creates a opposite vector based on the position, to
    //move the validation object to origo, for rotation.
    Vector3D vec = rotationPoint.ToVector3D().Multiply(-1.0);

    //Move to origo
    MoveValidationObject(vec);

    //Do transformation, and override transform.
    Transform3D oNewOverrideRotationTrans = new Transform3D(rotation);
    doc.Models.OverridePermanentTransform(Model,
       oNewOverrideRotationTrans, true);

    //Move back to original position.
    MoveValidationObject(vec.Multiply(-1.0));
}
```

This method takes a `Rotation3D` (an angle represented in a Navisworks .NET API object) and a rotation point and rotates the model the given angle around the point. The rotation point is usually the centre of the model item. Because of the way the transformation method works through the API, the model has to be translated to its origin be rotated, and than moved back into position.

7.2 Path factory

The path factory takes geometry data via Navisworks' COM API, and generates a `List<Point3D>` to be used in the optimizing and verification. As the geometry data only consists of line sets and arcs in a random order, the geometry have to go through phases with sorting and selection before it can be used.

The COM API (as described in the previous section) generates a list of all the primitives (lines) of the path in a random order. The lines must therefore be sorted based one there position on the path, so that all points will be in correct order from start to end. This sorting is done by looking at similar points on different lines. If two lines share a point, they should connect on the path, and must be in sequence. The path generation is finished when all lines are sorted based on this technique and all duplicate points have been removed. The sorting algorithm is shown in following pseudo-code. This is most likely not the best way of sorting the list based on run-time, but since the process itself doesn't take significant time hasn't an optimization been prioritized. The `RemoveDuplicationsInPath` has not been included here because of its obviousness. See application code for more.

```
function SortCurrentPath(List <Line> lineList)
    1. Init new sortedList.
    2. Add the first line to the sortedList from lineList.
    3. Remove the first line from lineList
    length = Path.Count()
    counter = 0
    while sortedList.Count() != length && lineList.Count() > 0 do
        for all line in lineList do
            if Diff(sortedList.First(), line) < 5 then
                Add line to sortedList
                Remove it from lineList
            end if
        end for
    end while
    RemoveDuplicationInPath()
end function
```

7.3 The two phases

The application consist of two phases; The optimization of path using clash logging, and the verification using dynamic relaxation. The first phase is intended to be a pre-phase for the verification. Its main task is to optimize the path created in another software and generated by the path factory to optimize the run-time speed of the verification. The optimization phase doesn't need to run to get the verification phase to run. If an optimizing has been performed, the application will use the optimized path during the dynamic relaxation verification, otherwise it would just use the path generated by the path factory.

7.3.1 Phase 1: Optimization of path using clash logging

The optimization of the path is done by moving the validation object through the path generated by the path factory. During the movement, the application will check for clashes with use of the .NET API and register all clashes that occur. After finishing the movement, the path will be altered based on the clash distance for each clash. The optimized path should then be a better alternative than the factory generated path, since the validation object most likely would experience fewer clashes during the dynamic relaxation verification.

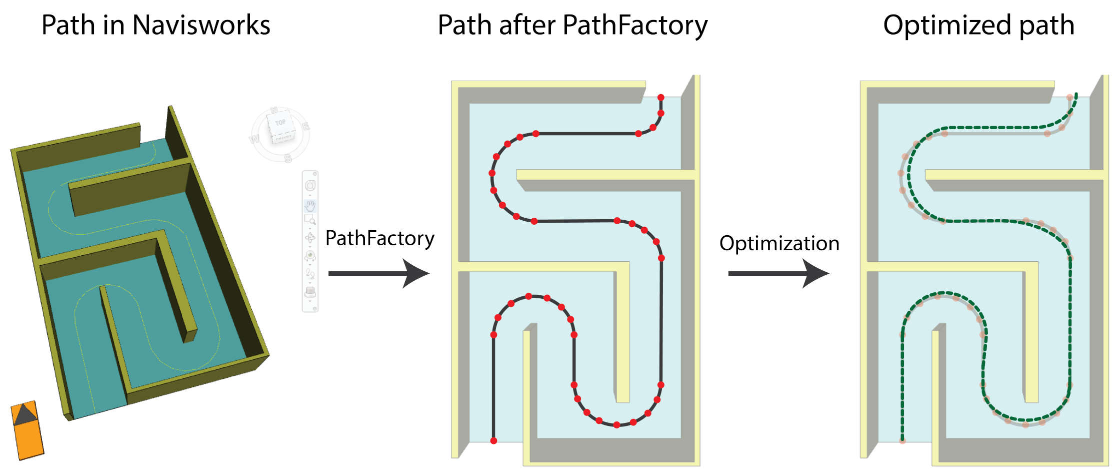


Figure 7.1: Illustration of the path generation from line sets and arcs in Navisworks, to a path list of points generated by the path factory, to an optimized path.

The optimizing phase uses the `MoveValidationObjectThroughPath`-method in `DynamicMovment` to move the validation object through the path. The method creates required clash tests and runs them for each step in the movement. (much the same as in DR). All clashes which occur during the dynamic movement are stored as `Clash3D` object in a list, which later is returned and used by the `Path`-class to optimize the path. See the sequence diagram 7.2.

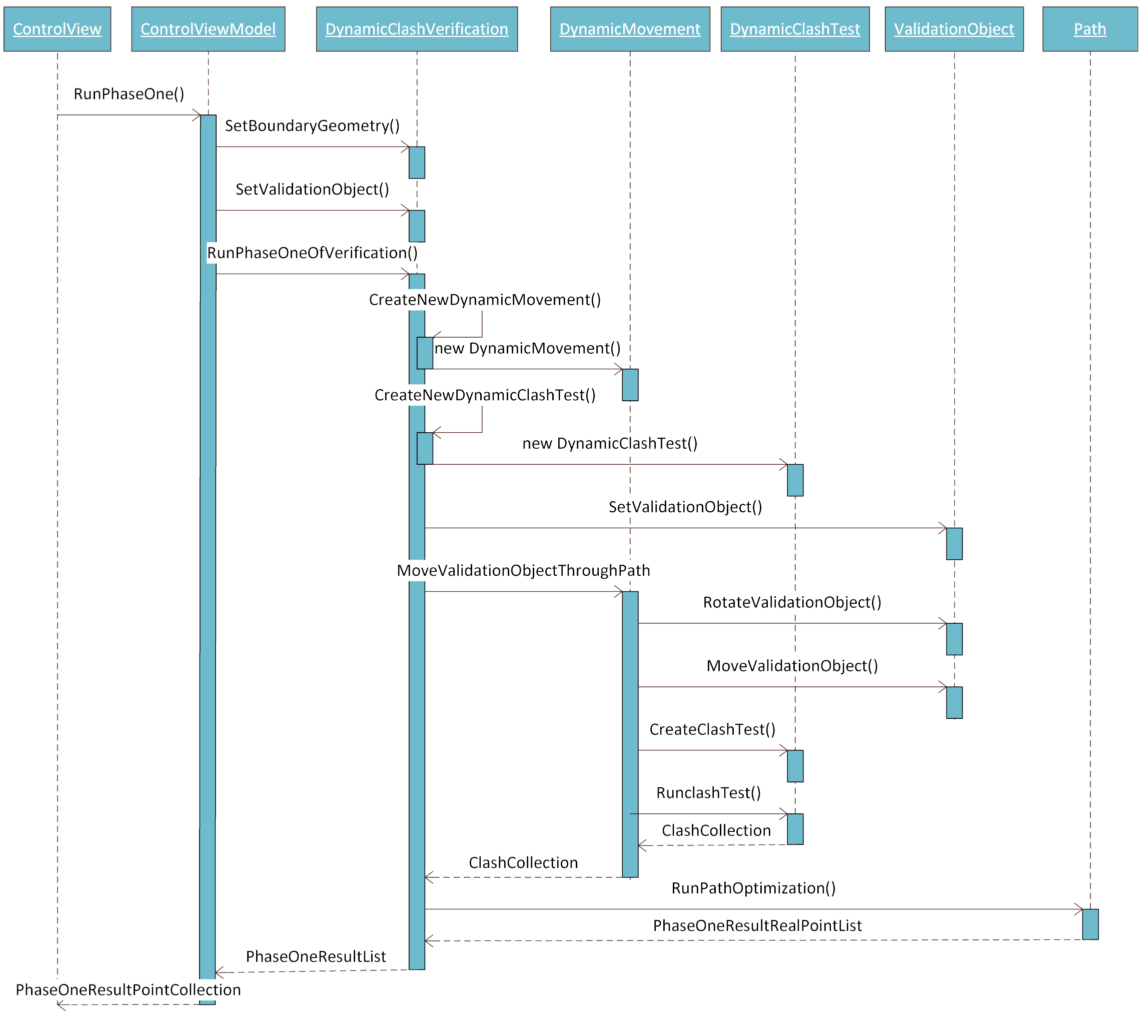


Figure 7.2: Sequence diagram of the first phase in the application; The optimizing phase.

7.3.2 Phase 2: Verification using dynamic relaxation

The last phase is the verification phase. When running this phase, the application tries to use Dynamic Relaxation to move the validation object through the path without clashing. It tries to find equilibrium before it continues if a clash occurs. The application will give a response to the user of the result, either by a completed verification with green symbols, or a fail on with a red symbol and progressbar.

During the verification, the user can monitor the velocity, acceleration and applied force in the "DR Analysis" tab. This can happen when the path is not created the proper way, or the geometry becomes to difficult.

The verification with dynamic relaxation is done by the `SolveDynamicRelaxation`-method in the `DynamicRelaxationSolver`-class. See sequence diagram 7.3.

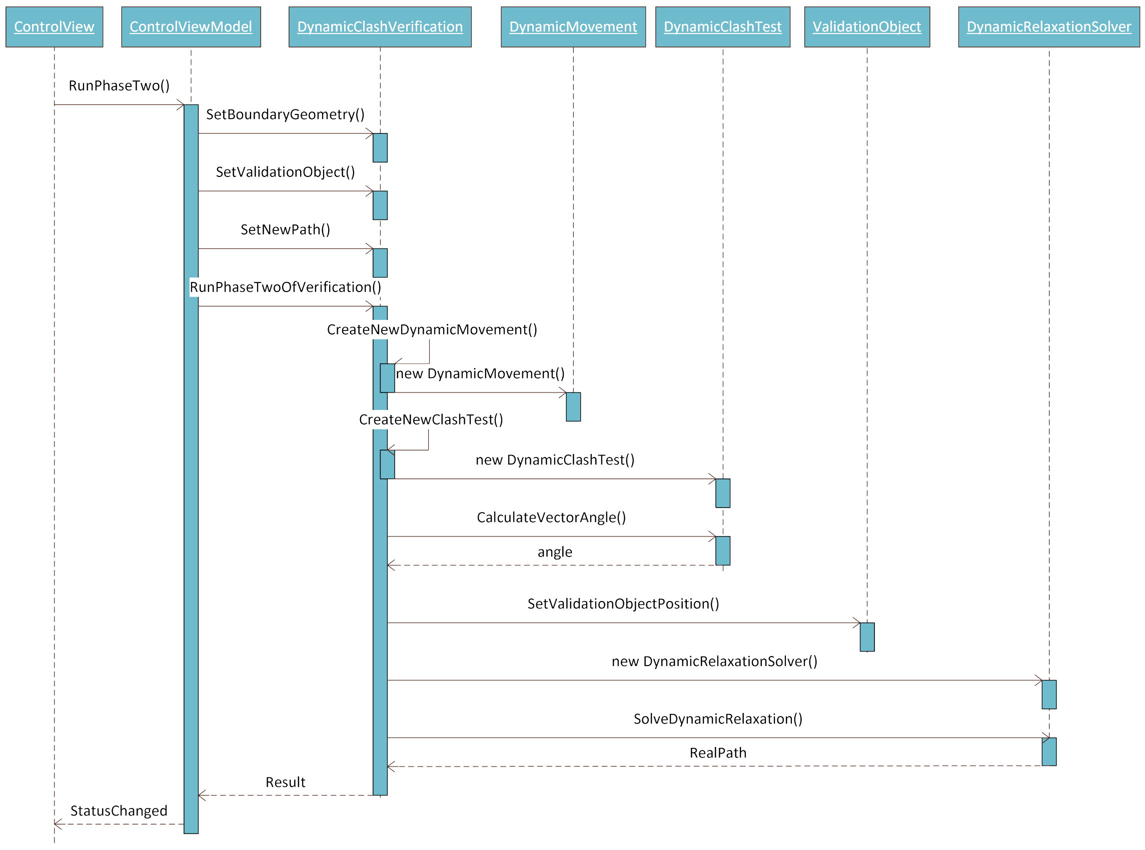


Figure 7.3: Sequence diagram of the verification phase using dynamic relaxation.

7.4 Implementation of Dynamic Relaxation algorithm

For each drag node in the path, the `NextIteration`-method is called to move the validation object to the drag point by using DR and clash testing. The result is returned. See sequence diagram 7.4. See 5.2.1 for the algorithm in pseudo code.

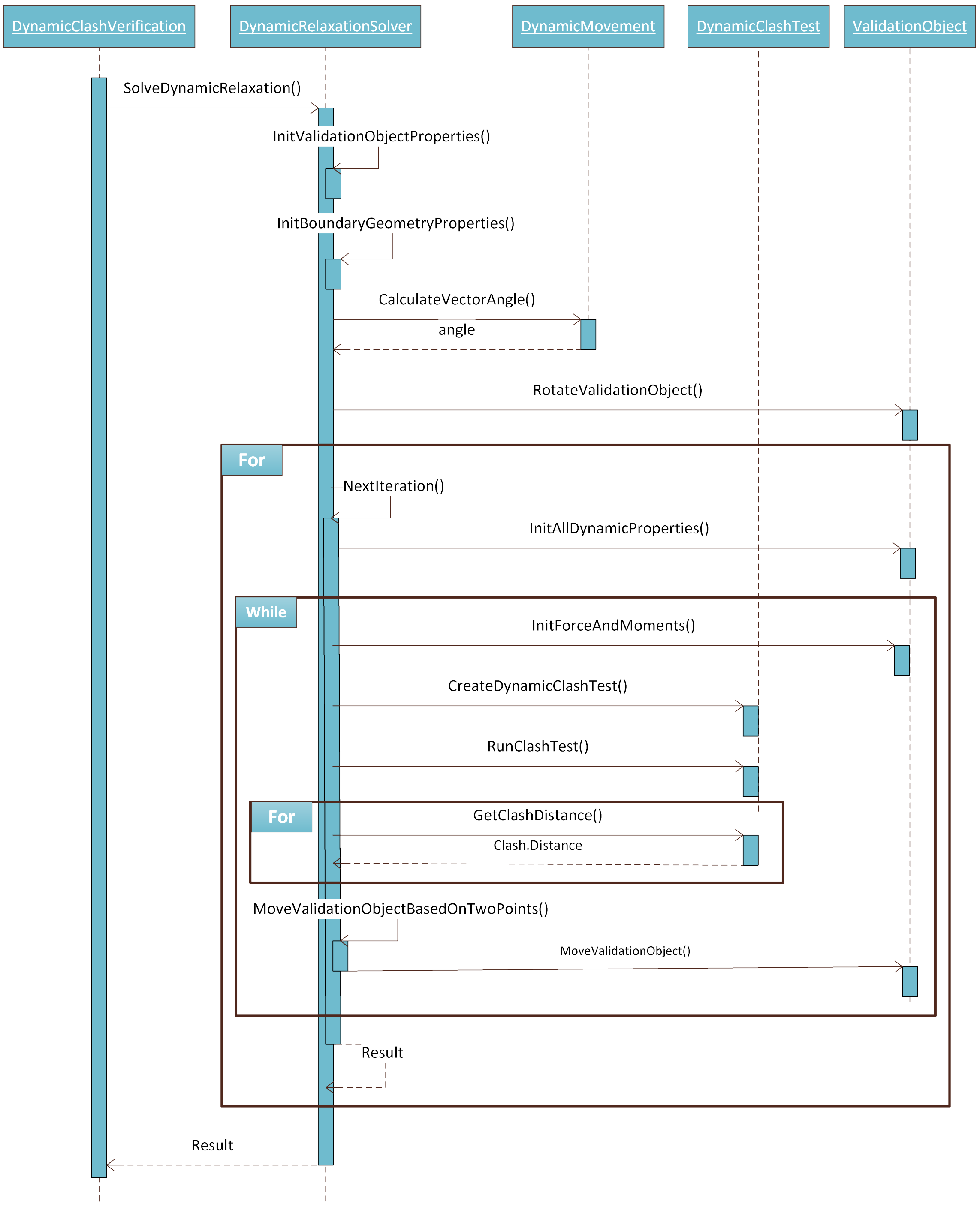


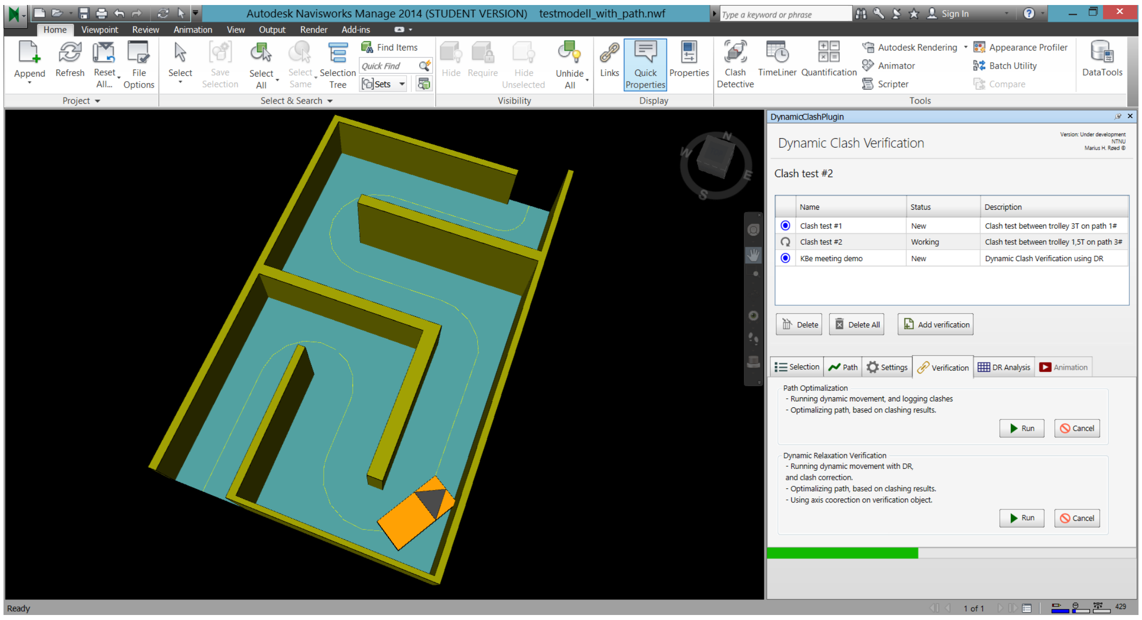
Figure 7.4: Sequence diagram of the verification process in the `DynamicRelaxationSolver`-class.

Chapter 8

The final plug-in application

In this chapter a short presentation will be given to the final plug-in application. A full introduction on how to use the application will not be given, but details can be found in the user manual in appendix A. There will also be given an introduction to the techniques in human-computer-interaction used for the user interface, the run-time and some run-time optimization and details about the plug-in application documentation.

8.1 Short presentation



	Name	Status	Description
●	Clash test #1	New	Clash test between trolley 3T on path 1#
	Clash test #2	Working	Clash test between trolley 1,5T on path 3#
●	KBe meeting demo	New	Dynamic Clash Verification using DR

Path Optimalization
- Running dynamic movement, and logging clashes
- Optimalizing path, based on clashing results.

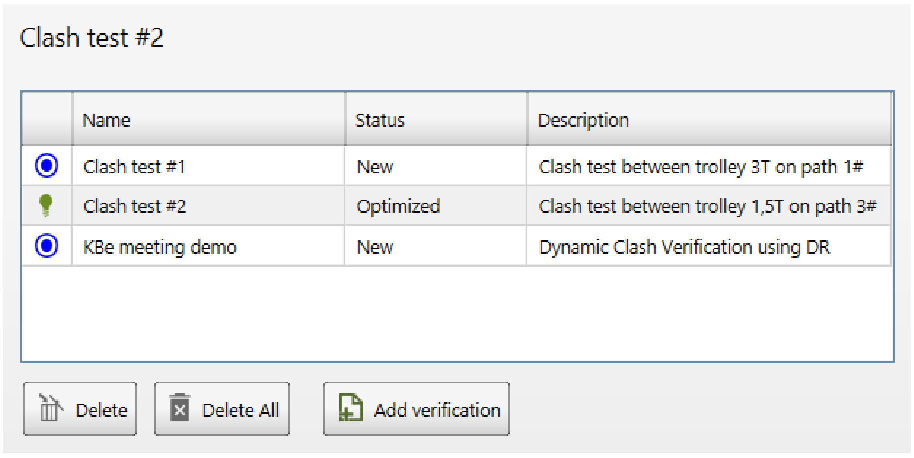
Dynamic Relaxation Verification
- Running dynamic movement with DR,
and clash correction.
- Optimalizing path, based on clashing results.
- Using axis coorection on verification object.

Figure 8.1: Screen shot of the plug-in application in use in Navisworks.

The final plug-in application (fig. 8.1)includes functionality for creating a path from a Navisworks model item, optimizing the path based on clash logging, and verifying movement of a validation object through path with use of dynamic relaxation. The application can have multiple verification sessions for a model item, but can't at this time save and load verification sessions to file. The application also includes chart controls for DR analysis during verification. An animation tab is included, but is disabled, due to an unfinished implementation. However, a lot of example code has been prepared in addition to methods in the source code of the application for further development.

In the next sub sections the different user controls and tab items in the application will briefly be introduced. Se appendix A for more detailed information.

8.1.1 The verification sessions view



Clash test #2

	Name	Status	Description
●	Clash test #1	New	Clash test between trolley 3T on path 1#
●	Clash test #2	Optimized	Clash test between trolley 1,5T on path 3#
●	KBe meeting demo	New	Dynamic Clash Verification using DR

Figure 8.2: Screen shot of dynamic clash verification sessions control view. Features for viewing, adding and deleting sessions of different verifications.

The verification session control viewer (fig: 8.2) enables the user to review, add and delete different clash verification sessions during the same Navisworks run-time session. The data grid showing the sessions informations, includes name, description and status information. The icon in the left most column represents the current status and is connected to the status column.

8.1.2 The verification control view

The verification control view contains a tab control for setting required information/ settings for running optimization and verification. The view is divided into six tabs, which the user is supposed to follow sequentially from left to right. The different tabs are explained below. In addition there is a progress bar at the bottom, which shows the progress during path generation, optimizing and verification. The progress bar becomes red if the verification fails. See appendix A for descriptions of the different icons.

Selection tab

The user selects the wanted validation object and boundary geometry from the list of selections sets in the **Selection** tab (fig: 8.3). The selection sets have to be created in Navisworks. See section 3.1.

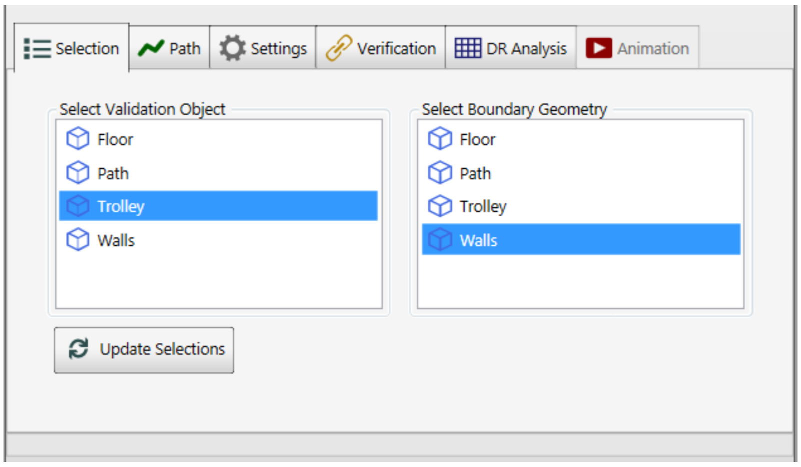
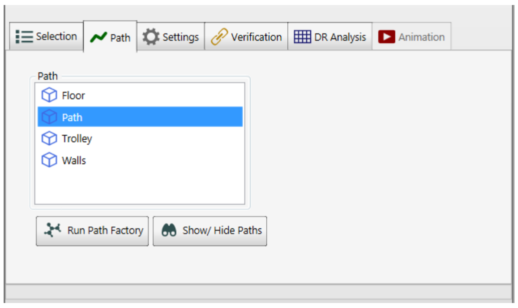


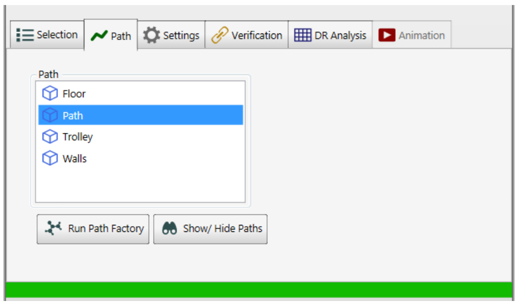
Figure 8.3: Screenshot of the selection tab in the control view.

Path tab

In the **Path** tab (Fig: 8.4a and 8.4b) the user must select the desired path from the list of selection sets. After selection the user must run the *Path Factory* to convert the path from a model item in Navisworks to a path in the application.



(a) Screenshot of the path tab before *Path Factory* has been run.



(b) Screenshot of the path tab after *Path Factory* has been run. The progress bar shows that the path generation is finished.

The *Path Factory* will randomly select a start point on the path (one of the two end points). In the path popup view (fig: 8.4) the user can change the start point, as well as review the generated path. The user can also go back to the popup view after optimizing and verification and see the resulting paths. The button *"Show/Hide Paths"* displays the path popup view.

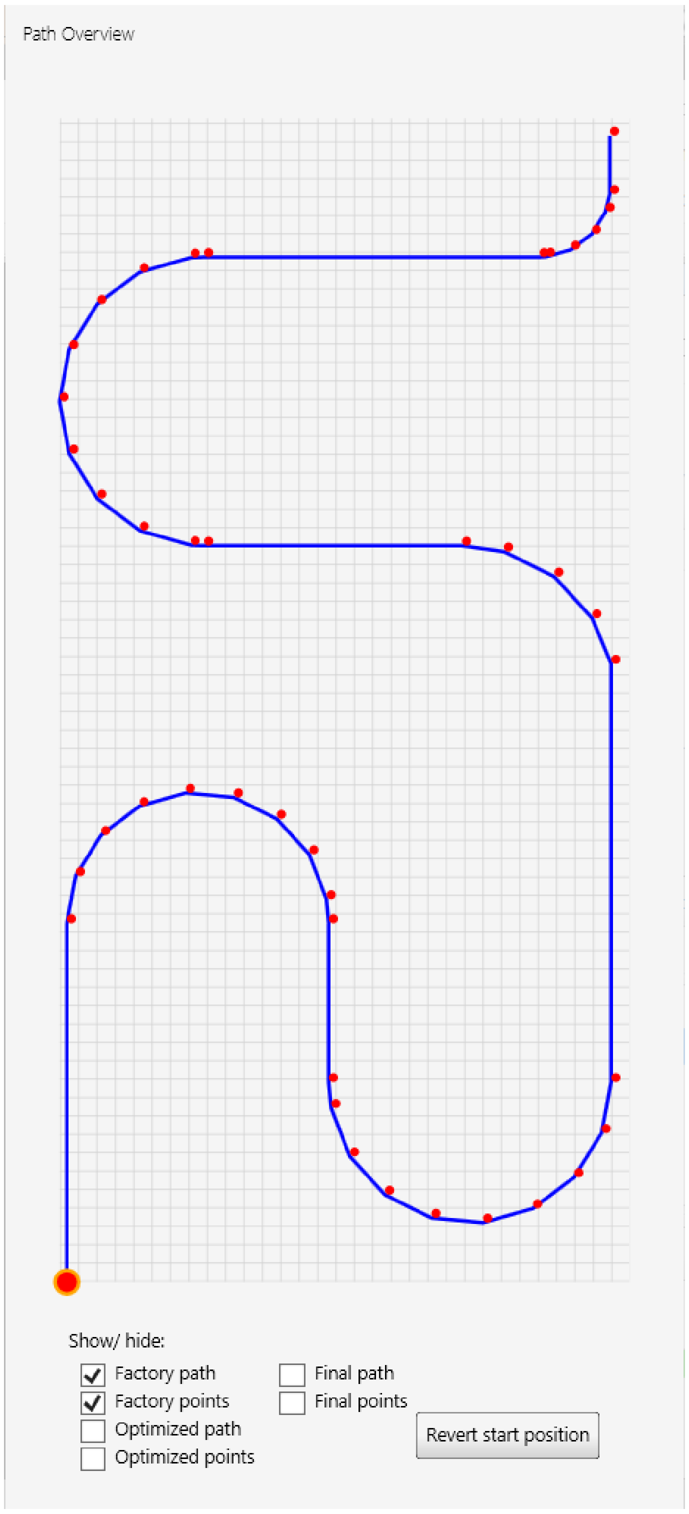


Figure 8.4: Screenshot of the path popup view. The different path results are shown, as well as features for reverting start point and visibility for the different paths and points.

Settings tab

The user can set wanted step precision and axle settings in the **Settings** tab (fig: 8.5) t Note that the step precision can be changed between optimization and verification. The application should have a high step precision during optimizing and less precision during verification to improve the run-time speed and achieve the best result possible.

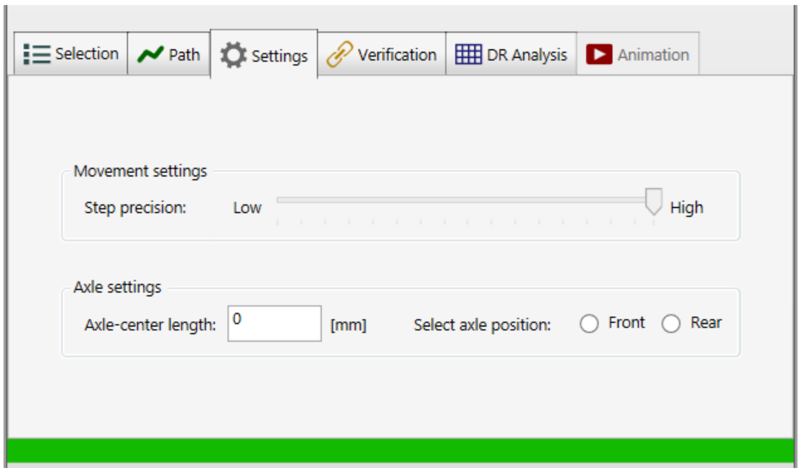
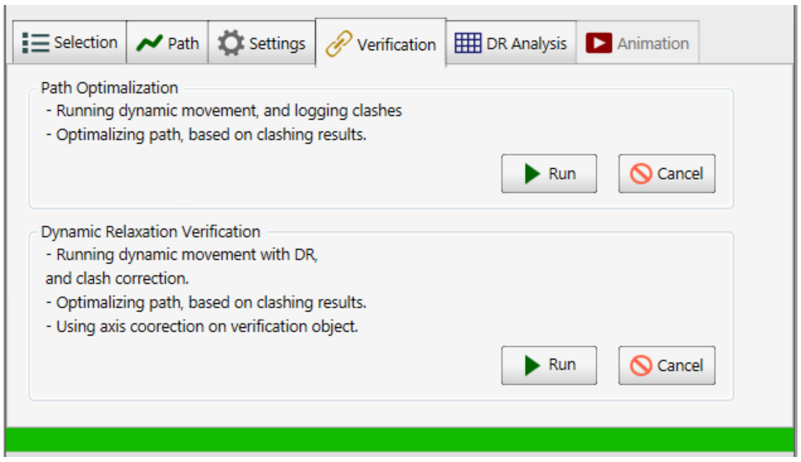


Figure 8.5: Screenshot of the settings tab view, where the user can set desired step precision and axle settings.

Verification tab

The user can run and cancel path optimization and DR verification in the **Verification** tab (fig: 8.6). Note that it is possible to run the DR verification without running the optimization first, as long as the path factory has run. The "Visualize result" button is being visible when the verification is finished (Fig 8.7).


Selection | Path | Settings | Verification | DR Analysis | Animation

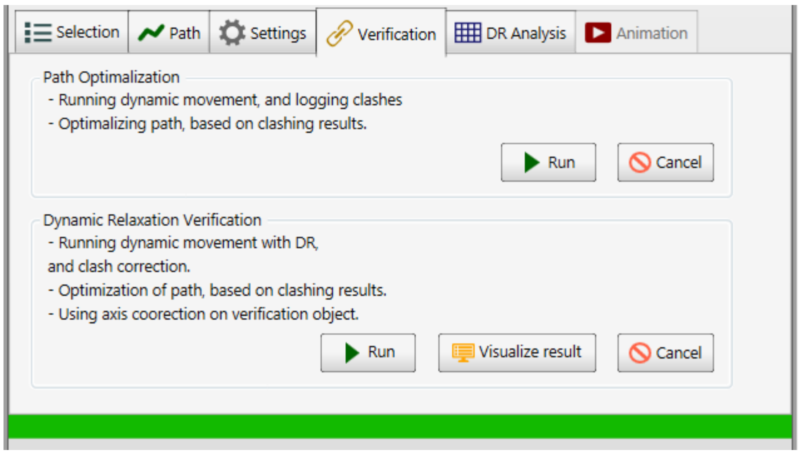
Path Optimalization
- Running dynamic movement, and logging clashes
- Optimalizing path, based on clashing results.

Run | Cancel

Dynamic Relaxation Verification
- Running dynamic movement with DR,
and clash correction.
- Optimalizing path, based on clashing results.
- Using axis coorection on verification object.

Run | Cancel

Figure 8.6: Screenshot of the verification tab view, where the user can start the optimizing and verification.



Path Optimalization
- Running dynamic movement, and logging clashes
- Optimalizing path, based on clashing results.

Dynamic Relaxation Verification
- Running dynamic movement with DR,
and clash correction.
- Optimization of path, based on clashing results.
- Using axis coorection on verification object.

Figure 8.7: Screenshot of the verification tab view after the verification. The user can now visualize the result.

DR analysis tab

The user has the possibility of monitor the progress in the tab **DR Analysis** (fig: 8.8) during verification. The tab shows two chart controls; one for the applied force, and one for the velocity and acceleration. This feature can be practical to use for looking at what happens during clashing and if the validation object displays strange behaviour.

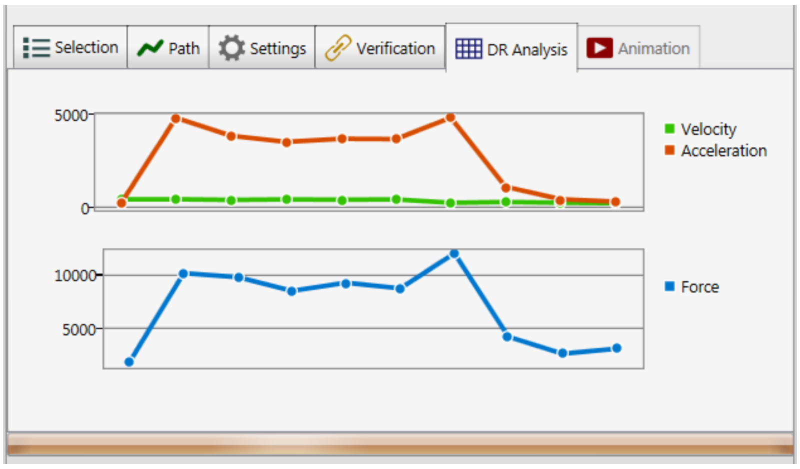
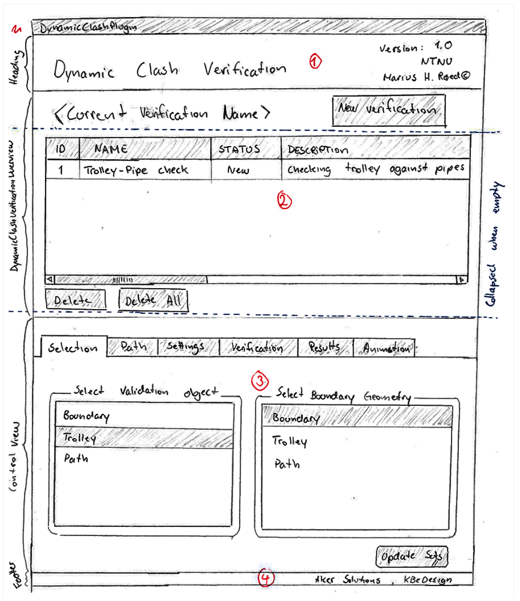
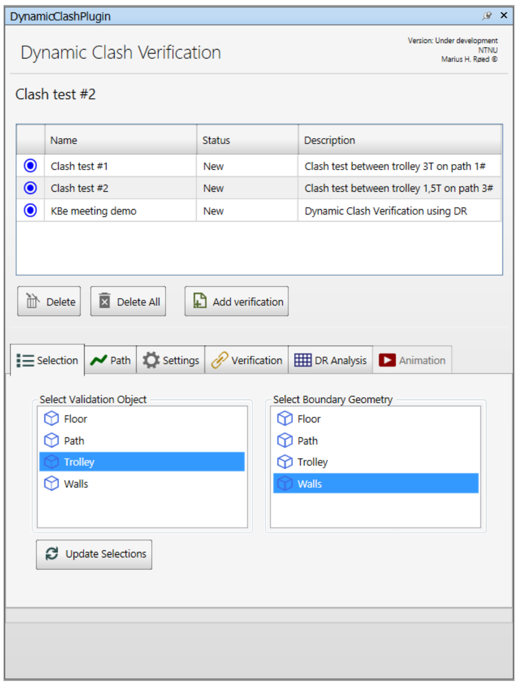


Figure 8.8: Screenshot of the DR Analysis tab view. The two chart controls shows the progress of the applied force, velocity and acceleration of the validation object.

8.2 User Interface - Usability



(a) Draft of user interface before implementation.



Dynamic Clash Verification

Version: Under development
NTNU
Marius H. Røed ©

Clash test #2

	Name	Status	Description
●	Clash test #1	New	Clash test between trolley 3T on path 1#
●	Clash test #2	New	Clash test between trolley 1,5T on path 3#
●	KBe meeting demo	New	Dynamic Clash Verification using DR

Select Validation Object
- Floor
- Path
- Trolley
- Walls

Select Boundary Geometry
- Floor
- Path
- Trolley
- Walls

(b) The final user interface.

The application's user interface uses techniques in human-computer-interaction for best possible design. There haven't been any usability tests performed with the future users of the application because of time issues and implementation problems. The user interface has therefore been developed based on experience and comments from the supervisors in Aker Solutions and techniques and principles from theory about human-computer-interaction.

Figure 8.9a shows the draft of the UI in a early phase of the development of the application. Figure 8.9b shows the final user interface developed, based on the draft. The two are very much a like, but with some deviations. The greatest changes in the UI during development have been in the *Settings*, *Verification* and *DR Analysis* (before called *Result*) tabs. There are unfortunately no draft figures of this.

Some of the principles tried followed during development is *Don Norman's Design Principles*. These principles are well known today, even though they are over 20 years old. Norman presents six principles for a good interactive design: ([21] and [22])

- Visibility
- Feedback
- Constraints
- Mapping

- Consistency
- Affordance

Visibility involves the idea that functions should be as visible as possible. This is because it would most likely result in the user having better ability to decide what to do next. Functions should not be hidden from the user. As the plug-in application doesn't have that many different functions, there has been a focus on emphasizing the different tabs and their main tasks, so that it is very clear what each tabs content does.

Feedback involves functions for sending feedback/ information back to the user about the current status and action which has been performed. This allows the user to have full control over the work flow in the application. The plug-in application uses the progress bar and status text and icons to give feedback to the user. It also uses short message boxes if something goes wrong.

Constraints involves including restrictions in the application to control which activities the user is supposed to do at different times. In the plug-in application the collapsed and enabled properties on different elements are used to control this. E.g. the control view will be disabled when no verification sessions have been created.

Mapping refers to a relationship which exists between different controls and the controls' effect in the real world. E.g. a round rotation button for controlling sound volume - the user must rotate clockwise to get more sound. There is not really any need for use of this in the plug-in application, since the amount of controls are at a minimum.

Consistency involves creating user interfaces, which have equal operations and elements do equal tasks. The user interface is consistent if it follows specific rules for similar elements/ tasks. E.g. the plug-in application uses the same method for selecting the validation object as the boundary geometry and path does.

Affordance involves giving elements a look which corresponds to their task. An element's task should be obvious to understand just by looking at it. In the plug-in application it is used a lot of symbols to achieve this. The symbols should relate to known features for the user. E.g. the path symbol, which looks like a path, or the settings symbol which is similar to all other known settings symbols, from other software.

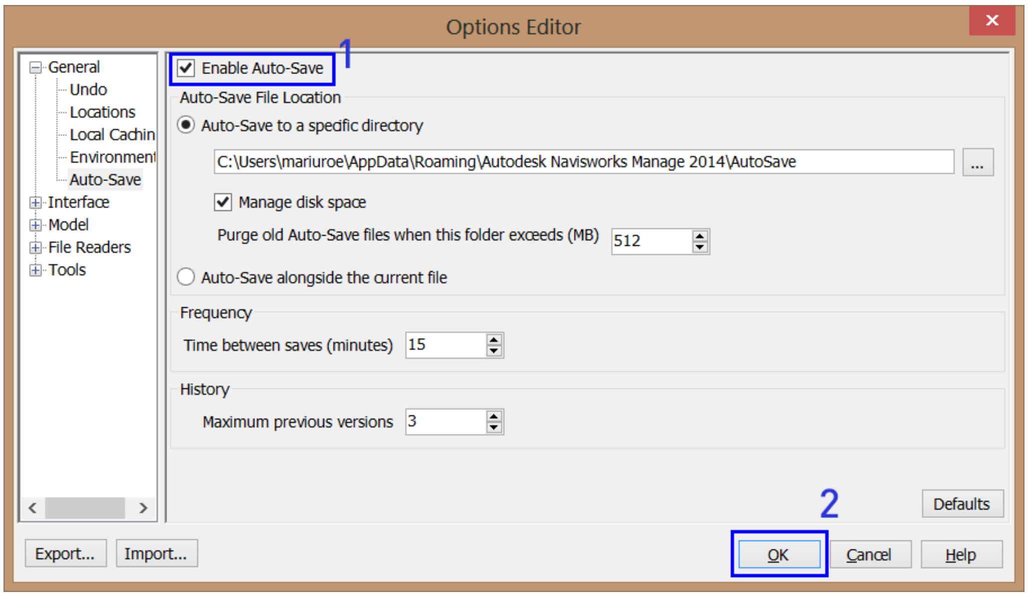
Since there haven't been any usability test for this plug-in application, there could potentially be a lot that could have been improved on the UI. This will be addressed more in further work (sec. 10.2).

8.3 Optimizing run-time

Autodesk Navisworks has a default global option that will decrease run-time on the plug-in application. Navisworks is set to auto-save the current document every time a clash test is done. As the plug-in application runs thousands of clash test during a verification, the auto-saving feature can consume a lot of time. The feature saves five temporary files/ copies of the document. This means that Navisworks needs to both create and delete a file for each clash. Tests done shows that the plug-in application uses five times more time with the auto-save function enabled, then when it's disabled. These tests are also done on a SSD, which means that it will take a lot of time on a ordinary HDD (as SSDs can have 100 times better performance than HDD). It is therefore important that the user turns the auto-save feature off during plug-in application use, and turn it back on when using Navisworks for other activities. [6]

To disable auto-save do:

1. **Select the main menu, pushing the Navisworks logo in the upper left corner.**
2. **Select Options.**
3. **General –> Auto-Save –> Enable Auto-Save (Fig: 8.9).**
4. **Unselect to disable.**
5. **Push OK.**

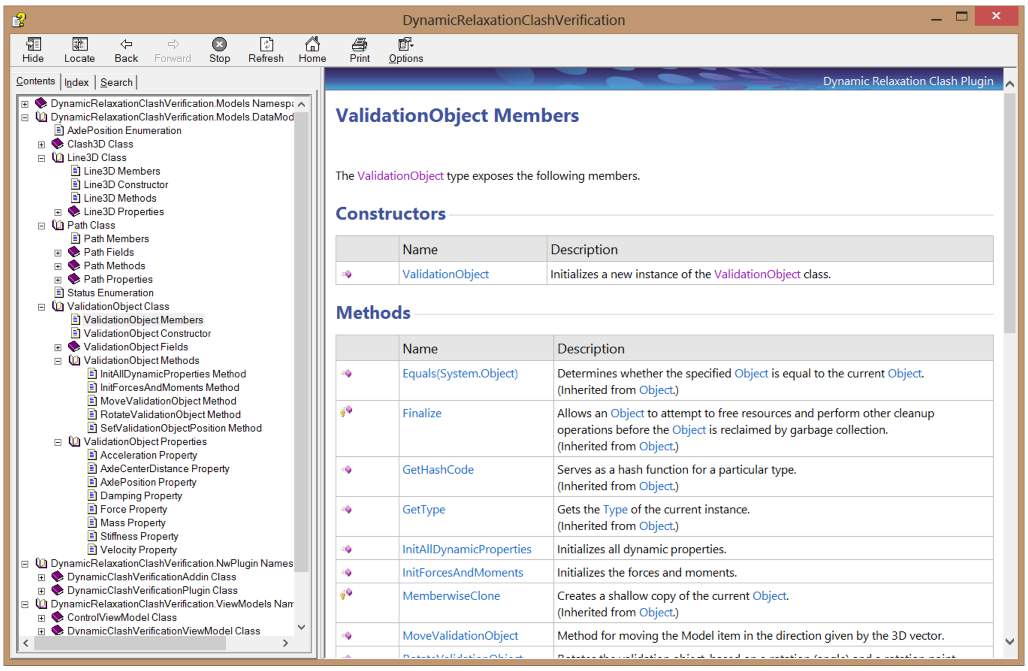


[✓] Enable Auto-Save
Auto-Save File Location
● Auto-Save to a specific directory
C:\Users\mariuroe\AppData\Roaming\Autodesk Navisworks Manage 2014\AutoSave
[✓] Manage disk space
Purge old Auto-Save files when this folder exceeds (MB) 512
○ Auto-Save alongside the current file
Frequency
Time between saves (minutes) 15
History
Maximum previous versions 3

Figure 8.9: Screenshot of the Option Editor, where the auto save function may be enabled.

8.4 Documentation

The plug-in application's source code folder includes full documentation as a .chm-file (Windows only), and as a a web page (needs to run on a php server. Works on all OS types.) The documentation is created using XML tags in the source code, and GhostDoc Pro to generate the .chm-file and the web page.



```
⊞ DynamicRelaxationClashVerification.Models Namespa
⊟ DynamicRelaxationClashVerification.Models.DataMod
    AxlePosition Enumeration
  ⊞ Clash3D Class
  ⊟ Line3D Class
      Line3D Members
      Line3D Constructor
      Line3D Methods
    ⊞ Line3D Properties
  ⊟ Path Class
      Path Members
    ⊞ Path Fields
    ⊞ Path Methods
    ⊞ Path Properties
    Status Enumeration
  ⊟ ValidationObject Class
      ValidationObject Members
      ValidationObject Constructor
    ⊞ ValidationObject Fields
    ⊟ ValidationObject Methods
        InitAllDynamicProperties Method
        InitForcesAndMoments Method
        MoveValidationObject Method
        RotateValidationObject Method
        SetValidationObjectPosition Method
    ⊟ ValidationObject Properties
        Acceleration Property
        AxleCenterDistance Property
        AxlePosition Property
        Damping Property
        Force Property
        Mass Property
        Stiffness Property
        Velocity Property
⊟ DynamicRelaxationClashVerification.NwPlugin Names
  ⊞ DynamicClashVerificationAddin Class
  ⊞ DynamicClashVerificationPlugin Class
⊟ DynamicRelaxationClashVerification.ViewModels Nam
  ⊞ ControlViewModel Class
  ⊞ DynamicClashVerificationViewModel Class
```

ValidationObject Members

The ValidationObject type exposes the following members.

Constructors

	Name	Description
	ValidationObject	Initializes a new instance of the ValidationObject class.

Methods

	Name	Description
	Equals(System.Object)	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
	Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
	GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
	GetType	Gets the Type of the current instance. (Inherited from Object.)
	InitAllDynamicProperties	Initializes all dynamic properties.
	InitForcesAndMoments	Initializes the forces and moments.
	MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
	MoveValidationObject	Method for moving the Model item in the direction given by the 3D vector.

Figure 8.10: Screen shot of the .chm-file with documentation.

An XML example for generating documentation for a method is shown below:

```
/// <summary>
/// Calculates the vector angle.
/// </summary>
/// <param name="previousVc">The previous vc.</param>
/// <param name="vc">The vc.</param>
/// <returns>System.Double.</returns>
public static double CalculateVectorAngle(Vector3D previousVc,
    Vector3D vc)
{
    UnitVector3D u = new UnitVector3D(previousVc);
    UnitVector3D v = new UnitVector3D(vc);

    return u.AngleWithReference(v, new UnitVector3D(0, 0, 1));
}
```

Chapter 9

Discussion - The new work flow v.s. the old

This chapter will give a short discussion based on the last sub-task from the assignment text:

- Validate the results of the new work flow (3) and the new application (4) against the current process (1).

where (3), (4) and (1) refers to the other subtasks. (See section 1.1.)

As addressed in section 2.2, the current process and work flow is very comprehensive and time consuming. The Material Handling Group has no record of how much time which is being used on transport verification today. They are only giving the impression that the current process is very time consuming and that it takes especially long time when large design changes occur.

One of the main problems with the current process is that the engineers have to do everything manually. By doing the verification process only partially automated, the engineers can perform other important tasks, regardless of how much time the automatic verification uses and thus save time. The plug-in application developed in this thesis is a step in this direction. The engineers still have to do some tasks manually, but the most time consuming part of the verification is done automatically. Even though the verification with using the plug-in can use some time, the engineers don't need to focus on it during the verification sequence. The engineers can simply start the verification with the plug-in and turn their focus on something else.

Even though the plug-in application has some run-time speed issues, the engineers can save a lot of time using it. If this concept may be further developed, with better clash detection, or even with a better software API than Navisworks, the engineers can save a substantial amount of time. This kind of engineering activity is a typical problem which should be done using automation that can save the offshore projects a lot of cost.

The plug-in application can also provide a much more accurate and realistic clash testing than what an engineer can do manually. Small clashes or movement solutions which the engineers are overlooking, can be detected by the application and in some cases be a solution to a problem. I.e. can the engineer in some cases give failed verification, while the application gives an approved verification. This can of course give a opposite result if the dynamic system in the application isn't implemented well and realistic enough. This is maybe one of the main challenges with this kind of application. One can may discuss if there are some better algorithms/ methods than dynamic relaxation, like methods used in the gaming industry. With a well defined path and with some modifications, will the application therefore most likely be able to illustrate a much more realistic movement of the validation object, than an engineer which uses the *"rotate"* and *"move"* feature in the 3D-tool software.

Chapter 10

Conclusions and further work

10.1 Conclusion

During the Master's thesis, a plug-in application for Autodesk Navisworks with C# in .NET and the .NET and COM API to Navisworks has been developed, in collaboration with KBeDesign, for use in the Material Handling Group in Aker Solutions. The assignment was do develop an application which could verify and visualize transport of equipment on offshore platforms, based on input parameters. Based on the current work flow and process for access way modelling and transport verification, a specific set of input parameters were defined; the verification object, the boundary geometry and the path, where each of these should be defined as a selection set in Navisworks. The path had to be created in PDMS, AutoCAD or another 3D modelling tool, since Navisworks can't generate geometry.

The plug-in application uses the dynamic relaxation method introduced and developed in the project work done fall 2013. Dynamic relaxation tries to find a geometry where all forces are in equilibrium. The plug-in application is used to check if an outer boundary geometry is good enough, so that an validation object can go through path by checking whether the object can be in equilibrium at all nodes defining the path.

The final plug-in application has two main phases; The path optimizing phase and the DR verification phase. Based on the input parameters defined as selection sets, the user can generate a path, alter start position on path, setting movement precision and axle settings and running optimizing and verification. The optimized path will most likely give a better basis for the verification, and therefore improve run-time speed. During the verification, the user can monitor the validation object's current velocity, acceleration and applied force with two chart controls.

The application has a MVVM architecture, and WPF is used as a UI presentation system. The UI has been developed based on experience, feedback from Aker and human-computer interaction theory. The UI has been given a custom styling, to ensure continuity between Windows 7 and Windows 8, as WPF gives different styling based on the OS. The custom styling is also based on the current UI in Navisworks.

10.2 Further work

The plug-in application has several improvement elements with the implementation. The `ControlView` should be divided into more view, for a more clean code. The `ControlViewModel` consists of very many properties and method, and it would be a more clean code if it was separated in more classes. F.ex. could each tab item in the tab control be divided into their own views and viewModels. There is also potentially some code which could be more reused in the application. The use of f.ex. value converters, would also give a more cleaner code. The converter was implemented in the end of the thesis, and was therefore not used with all wanted elements.

As discussed in the previous chapter, there are many possibilities to optimize the application, both considering run-time speed, usability etc., but also to look at other opportunities to develop a better clash detection or using an other dynamic system. Maybe is it possible to combine different dynamic system, or clash testing methods to get a better and more realistic result. It is also a possibility to use an other software tool, instead of Navisworks, f.ex. AML, which KBeDesign already use to run analysis, clash testing etc. on 3D models.

As the clash detective feature in Navisworks is very time consuming, one can look at the possibilities to develop a new clash detective feature more suited for this problem outline. It could also be possible to create an own data model and a model factory to fill the data model with geometry data through the COM API and then create suitable clash detection and verification based on the new data model. The DR algorithm could then be used as a kernel for the application, which could use both the data model, and the Navisworks Clash API. The Normal Intersection method, used in Navisworks is also not flawless and better methods more suited for the verification problem may exists.

Another issue that may should be more discussed, is the standalone v.s. plug-in application issue. The plug-in was created so that the engineers could use the verification application together with a well known software, which they already use on a daily basis. It is possible that a standalone application could be better, which then could be connected to many different 3D software applications. One could have a standalone application with it's own clash detective feature (No API in use), which could run analysis on a standard 3D model format file. It could also have its own API, with a interface making it possible to create custom plug-ins/ applications for difference software and visualizations, where the application could be an analysis kernel. This would be a great deal of work and is not suitable for an one person thesis, but could potentially be a good task for a smaller team, or individual thesis written in collaboration.

Bibliography

[1] Hans Petter Hildre, Ola Jon Mork, Vilmar Æsøy, *The Maritime Innovation Factory.* Aalesund University Collage, Aalesund, July 2010.

[2] Espen Messel, Cand. Scient, Solid Mechanics, *Explicit Dynamic Relaxation.* `http://folk.uio.no/esjen/kilder/NFEM.Ch22.pdf`,
University of Oslo

[3] Wanda J. Lewis, *Tension Structures: Form and Behaviour*. Thomas Telford Publishing, London, 2003.

[4] Aker Solutions KBeDesign webpage
`http://www.akersolutions.com/en/Global-menu/Media/Feature-stories/Engineering/KBeDesign/`
Date: 07.12.2013

[5] Norsk Stårforbund - NOROSK Standard (C-002 + S-002N)
`http://www.stalforbund.com/Standarder/norsok.htm`
Date: 20.08.2013

[6] Microsoft Developer Network - Windows Presentation Foundation
`http://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx`
Date: 14.12.2013

[7] Microsoft Developer Network - Data Bindings
`http://msdn.microsoft.com/en-us/library/cc278072(v=vs.95).aspx`
Date: 15.12.2013

[8] MVVM - Model View ViewModel
`http://en.wikipedia.org/wiki/Model\_View\_ViewModel`
Date: 14.12.2013

[9] Autodesk Navisworks
`http://www.autodesk.com/products/autodesk-navisworks-family/overview`
Date: 01.05.2014

[10] AEC DevBlog for Navisworks .NET API
Article: Navisworks 2014 API new feature - Override Transformation
`http://adndevblog.typepad.com/aec/2013/05/navisworks-2014-api-new-feature-override-t`

html
Date: 13.02.2014

[11] AEC DevBlog for Navisworks .NET API
Article: Navisworks .NET API 2013 new feature - Clash 1 & 2
`http://adndevblog.typepad.com/aec/2012/05/navisworks-net-api-2013-new-feature-cl`
`html`
`http://adndevblog.typepad.com/aec/2012/05/navisworks-net-api-2013-new-feature-cl`
`html`
Date: 13.02.2014

[12] Autodesk Navisworks 2015 HELP, Glossary, Clash Detective Terminology
Autodesk Knowledge Network
`http://help.autodesk.com/view/NAV/2015/ENU/?guid=`
`GUID-27EA59E6-1A15-4372-9D7D-90508936B512`
Date: 08.05.2014

[13] FIG Technologies Ltd.
Navisworks .NET API presentation
`http://fig-tech.com/downloads/3.\%20DotNETAPIIntroduction.pdf`
Date: 09.05.2014

[14] Autodesk Developer Network
About Navisworks .NET API
`http://usa.autodesk.com/adsk/servlet/index?id=15024694\&siteID=`
`123112`
Date: 09.05.2014

[15] CodePlex - Project Hosting for Open Source Software
Welcome to Windows Presentation Foundation
`http://wpf.codeplex.com/`
Date: 09.05.2014

[16] CodePlex - Project Hosting for Open Source Software
MVVM Light Toolkit
`https://mvvmlight.codeplex.com/`
Date: 09.05.2014

[17] FIG Technologies Ltd.
Navisworks Plugin API Overview
`http://fig-tech.com/downloads/4.\%20PluginAPIOverview.pdf`
Date: 11.05.2014

[18] Microsoft Developer Network
Libaries - IValueConverter Interface
`http://msdn.microsoft.com/en-us/library/system.windows.data.`
`ivalueconverter.aspx`
Date: 11.05.2014

[19] AEC DevBlog
Article: "Get primitives from solid of Navisworks"

Written by: Xiaodong Liang
http://adndevblog.typepad.com/aec/2012/05/get-primitive-from-solid-of-navisworks.html Date: 13.05.2014

[20] Virtual Testing of Mechanical Systems, Theories and Techniques
Chapter 2.2 - Transformations and Rotations
Presentation foilers from subject TMM2 Product Simulations, NTNU

[21] Summery of Don Norman's Design Principles
Source: Preece, J., Rogers, Y., Sharp, H. (2002), *Interaction Design: Beyond Human-Computer Interaction*, New York: Wiley, p.21
http://www.csun.edu/science/courses/671/bibliography/preece.html
Date: 19.05.2014

[22] Summery presentation TDT4180, Human-Computer-Interaction
By Dag Svanæs
NTNU, Trondheim 2012

[23] Hilber-Hughes-Taylor Method,
OpenSee, Berkely edu
http://opensees.berkeley.edu/wiki/index.php/Hilber-Hughes-Taylor_Method
Date: 20.05.2014

[24] Microsoft Developer Network,
Observer Design Pattern,
http://msdn.microsoft.com/en-us/library/ee850490(v=vs.110).aspx
Date: 23.05.2014

[25] Equation of motion
http://en.wikipedia.org/wiki/Equations_of_motion
Date: 09.06.2014

Appendices

Appendix A

User manual for plug-in application

This document will give a detailed introduction to the user interface and functionalities of the Dynamic Clash Verification Plug-in to Autodesk Navisworks Manage 2014, developed in the Master thesis spring 2014. The document will give a step-by-step presentation on how to use the application, as the needed pre-steps done with Navisworks functionality. The user manual is based on Windows 8 OS.

A.1 Setting up plug-in with Navisworks

In order to use the plug-in application with Navisworks Manage[1] 2014, the plug-in class library has to be imported to the \Plugins folder in Navisworks.

1. Locate the \Plugins folder in one of the following location. If one uses the first \Plugins folder, one needs to restart the system, to let Navisworks find the new plug-in. If one uses the second folder, from the \AppData location, the plug-in will work at once.
    - C:\Program Files\Autodesk\Navisworks Manage 2014\Plugins\
    - C:\Users\your user name\AppData\Roaming\Autodesk Navisworks Manage 2014\Plugins\ *
2. Copy the \DynamicClash.MHRK folder with the plug-in class library files from the Master's thesis attachments into the \Plugins folder. The class library folder should include a lot of `.dll`-files.

* The \AppData folder is by default hidden in Windows. To reveal the folder do the following:

[1]**Note** that the plug-in application only works with Autodesk Navisworks Manage 2014 and newer, and <u>not</u> Freedom and Simulate. This because of Navisworks Manage's clash detection feature, which is used by the plug-in.

- In the Ribbon toolbar select *View*.
- Check *Hidden Items*, under *Show/hide*

The plug-in application is know ready for use with Autodesk Navisworks Manage.

A.2 Setting input parametre in Navisworks

This sections expects that the user already has experience with Autodesk Navisworks Manage software.

1. Open model document, in which the model items to be verified is.
2. Use the *Save Selection* feature to create selection sets for the validation object, boundary geometry and path. The floor which the validation object is moving on, should not be a part of the boundary geometry.
3. The user should now have e.g. the following selection sets in the list of selection set in Navisworks: Trolley, Walls, Path.
4. Start the plug-in application "DynamicClashAddin" from the *Add-ins* tab.

A.3 The plug-in application

This sections goes through the different steps in the Dynamic Clash Verification Plug-in. The different elements of the plug-in UI is shown in fig: A.1.

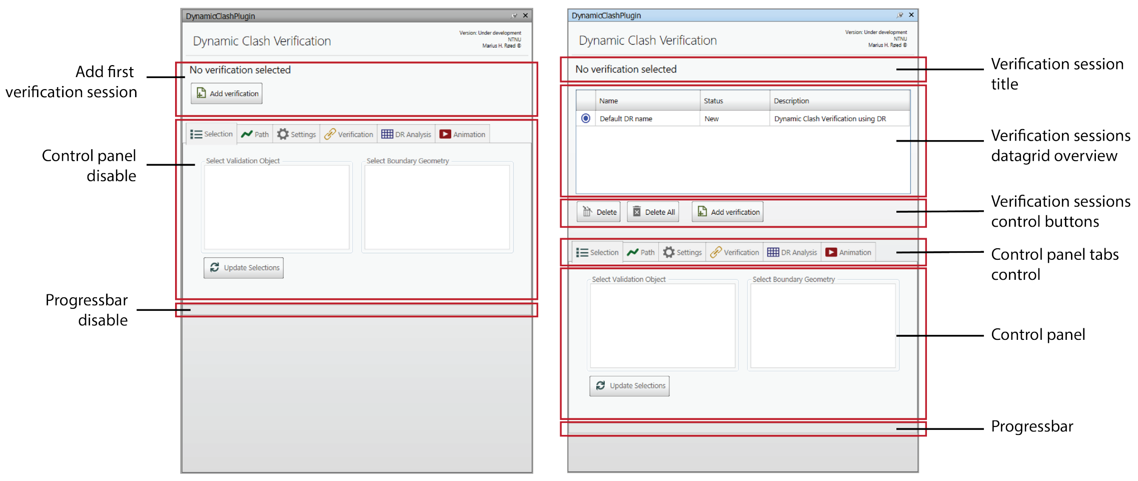


Figure A.1: Overview of the UI, and its different elements.

A.3.1 Creating a new dynamic validation session

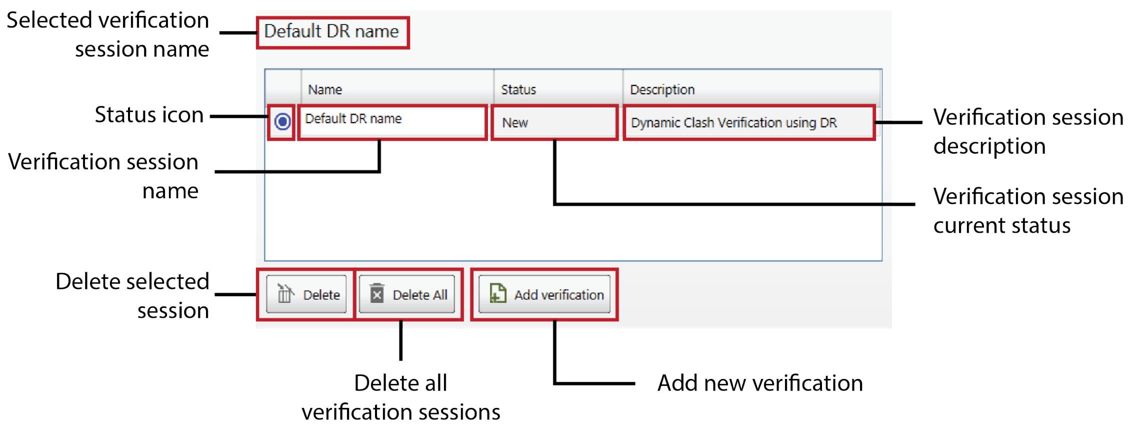

Selected verification session name
Default DR name
Name
Status
Description
Status icon
Default DR name
New
Dynamic Clash Verification using DR
Verification session description
Verification session name
Verification session current status
Delete selected session
Delete
Delete All
Add verification
Delete all verification sessions
Add new verification

Figure A.2: Overview of the verification sessions, and its elements.

The user can switch between, add and delete verification sessions in the verification session view. To create new verification session do:

1. Add new verification session using the *Add verification* button.
2. Create/ change name of the verification session.
3. Create/ change the description of the verification session.

A verification session has a status describing the current status of how long the verification has come in the verification process. The different statuses are:

Status icon	Status
◉	New
↻	Working
💡	Optimized
✓	Finished
⊘	Cancelled
⚠	Failed

Table A.1: Overview over the different status icons.

A.3.2 Setting selections

After the a verification session has been created, the control view enables. The user must now go through the tabs from left to right sequentially.

The first tab is the **Selection** tab. This tab is used to set the validation object and boundary geometry for the verification. (Fig: A.3)

1. Push the *Update Selections* button the update the list box with the newest created selection sets.
2. Select the selection set, which should be the validation object in the left list box.
3. Select the selection set, which should be the boundary geometry in the right list box.
4. Go to next tab (the Path tab).

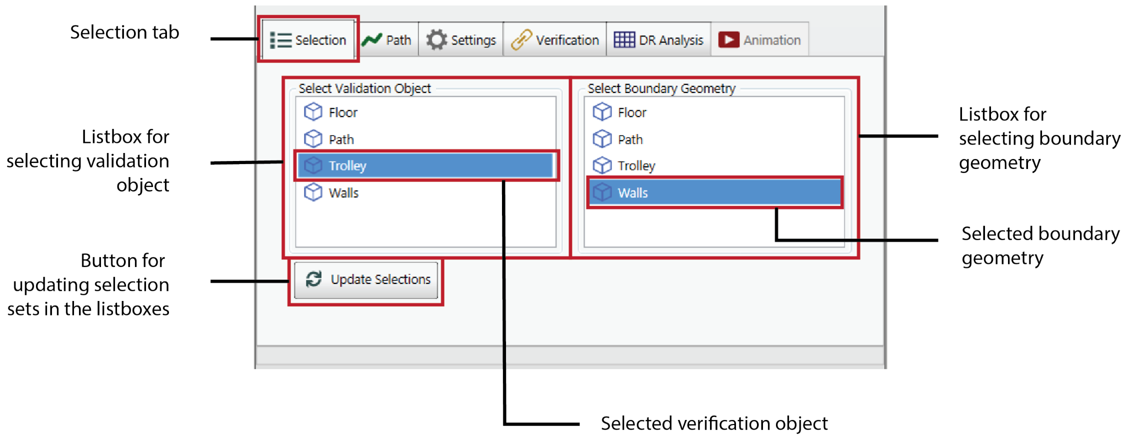

Selection tab
Listbox for selecting validation object
Button for updating selection sets in the listboxes
Listbox for selecting boundary geometry
Selected boundary geometry
Selected verification object
Selection
Path
Settings
Verification
DR Analysis
Animation
Select Validation Object
Floor
Path
Trolley
Walls
Select Boundary Geometry
Floor
Path
Trolley
Walls
Update Selections

Figure A.3: Overview of the selection tab, and it's different elements.

A.3.3 Setting path

The user should generate a path for use in the verification, from a selected selection set in the **Path** tab. (Fig: A.4)

1. Select the selection set, which should be the path from the list box.
2. Push the *Run Path Factory* button, to run path generation. When the progress bar is finished, go to (3).
3. Push the *Show/Hide Paths* button.
4. Control that the path looks correct in the path grid.
5. Revert the start point if needed using the *Revert Start Point* button. (See fig: A.5)
6. Use if necessary the check boxes to hide/unhide paths and points.
7. Unhide the popup view using the *Show/Hide Paths*.
8. Go to next tab (the settings tab).

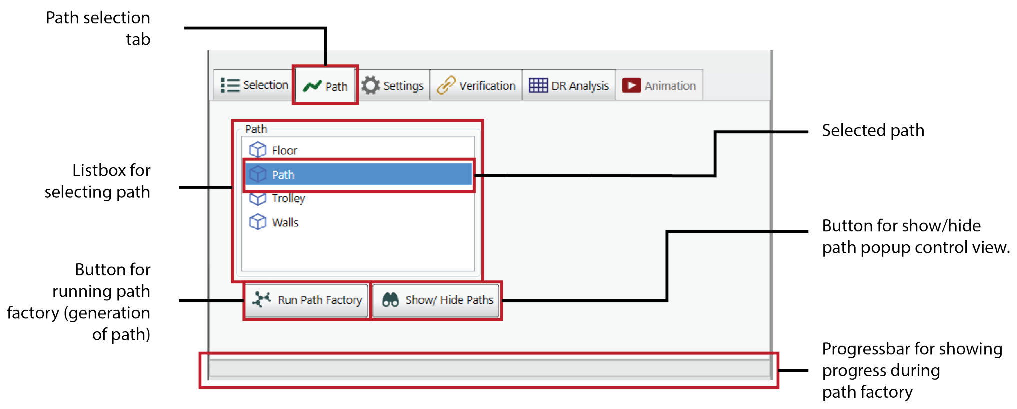

Path selection tab
Listbox for selecting path
Button for running path factory (generation of path)
Selected path
Button for show/hide path popup control view.
Progressbar for showing progress during path factory
Selection
Path
Settings
Verification
DR Analysis
Animation
Path
Floor
Path
Trolley
Walls
Run Path Factory
Show/ Hide Paths

Figure A.4: Overview of the path tab, and its different elements.

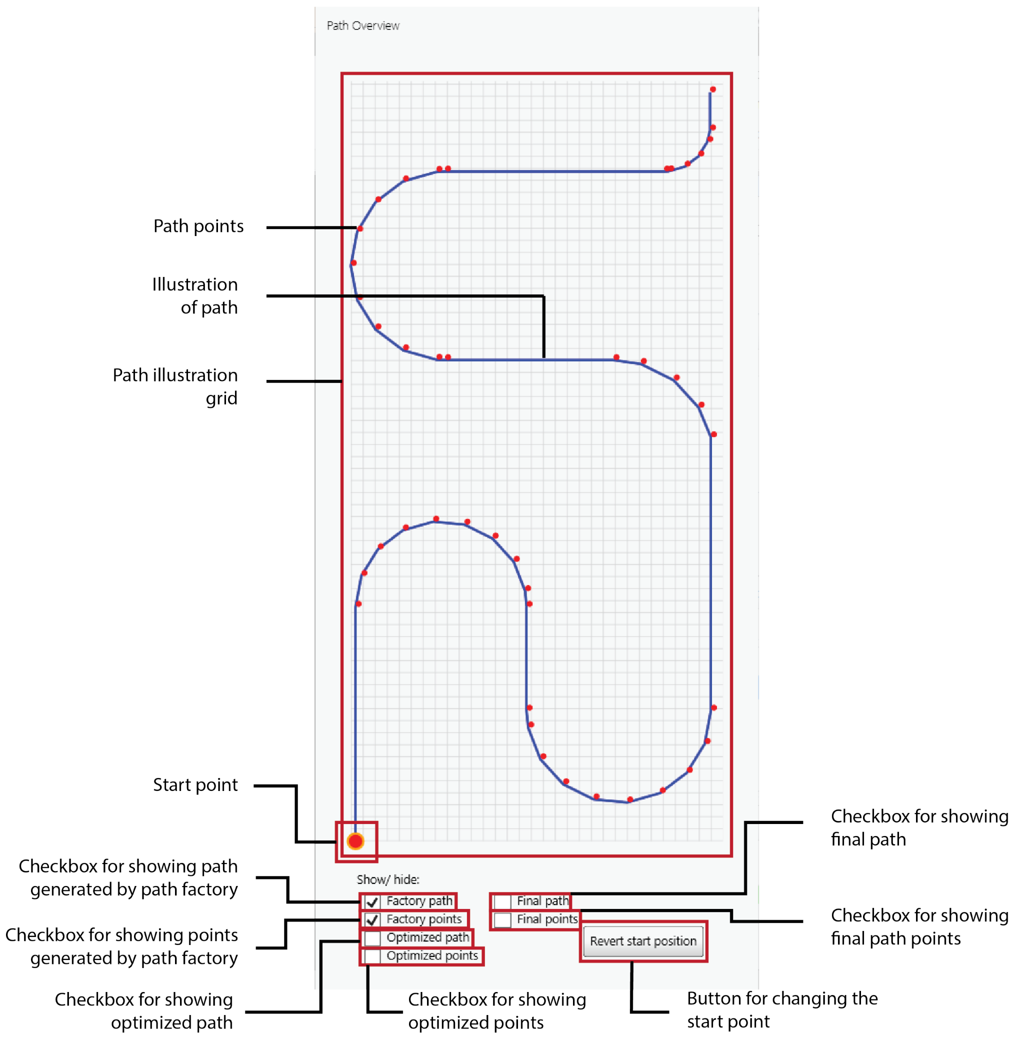

Path Overview
Path points
Illustration of path
Path illustration grid
Start point
Checkbox for showing path generated by path factory
Checkbox for showing points generated by path factory
Checkbox for showing optimized path
Checkbox for showing optimized points
Button for changing the start point
Checkbox for showing final path
Checkbox for showing final path points
Show/ hide:
Factory path
Factory points
Optimized path
Optimized points
Final path
Final points
Revert start position

Figure A.5: Overview of the path popup view, and its different elements.

The user can at any time go back to this tab to review the different paths generated after path factory, path optimizing and DR verification.

A.3.4 Setting the verification settings

The user should set needed settings for movement precision and axle settings in the **Settings** tab . (Fig: A.6)

1. Use the slider in Movement precision to set the step precision[2]. A high precision causes a low step size, and a low precision causes a high step size. The tool-tip of the slider gives the step size in [*mm*].
2. Set the Axle-centre length in [mm]. Default is 0. If no axle exists, the value should be 0.
3. Set axle position; Front or Rear. NOTE: The validation objects front has to be facing in the path direction. Default is centre. If no axle the axle position should not be selected. (It is not dangerous if it is, as long the axle-centre distance is 0.)
4. Go to next tab (the verification tab).

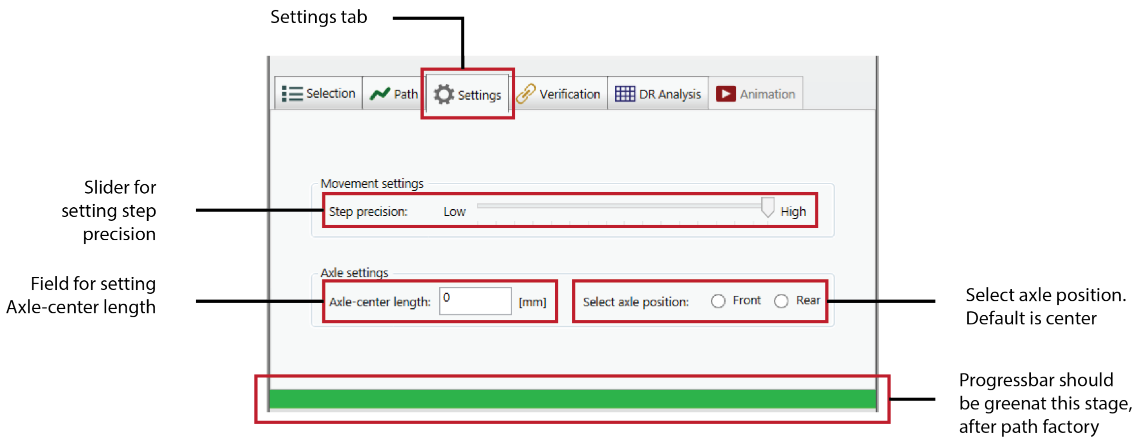
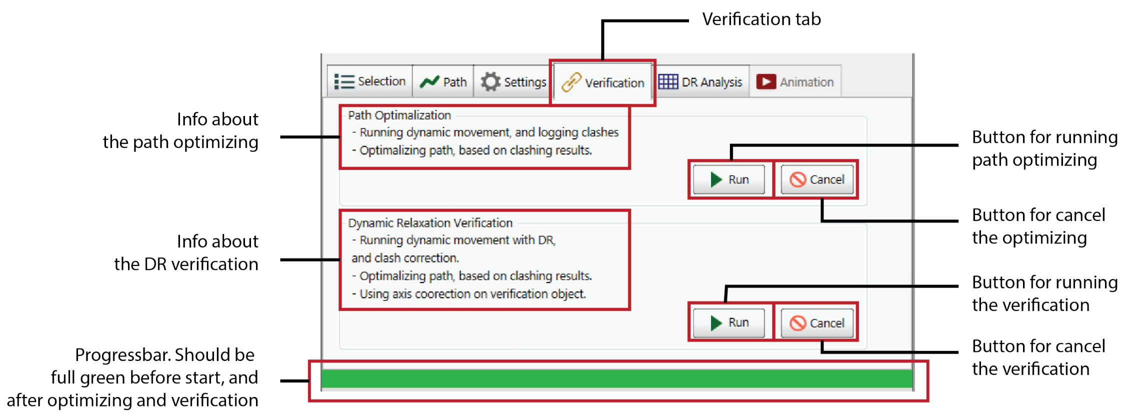


Figure A.6: Overview of the settings tab, and its different elements.

A.3.5 Running optimizing and verification

The user can run both the path optimizing and the DR verification in the **Verification** tab . The optimizing should be run before the DR analysis, but it is not demanded. The progress bar will show the progress during both optimizing and verification. The status in the verification sessions view will show the results.

[2]The step precision should be high (around 100-200 *mm*) during path optimizing, an lower during verification due to run-time speed (around 300-400 *mm*).


Verification tab
Selection
Path
Settings
Verification
DR Analysis
Animation
Info about the path optimizing
Info about the DR verification
Progressbar. Should be full green before start, and after optimizing and verification
Button for running path optimizing
Button for cancel the optimizing
Button for running the verification
Button for cancel the verification
Run
Cancel
Run
Cancel

Path Optimalization
- Running dynamic movement, and logging clashes
- Optimalizing path, based on clashing results.

Dynamic Relaxation Verification
- Running dynamic movement with DR, and clash correction.
- Optimalizing path, based on clashing results.
- Using axis coorection on verification object.

Figure A.7: Overview of the verification tab, and its different elements.

A.3.6 Monitor verification

The user can in the **DR Analysis** tab monitor the velocity, acceleration and applied force during clash of the validation object during the DR verification.

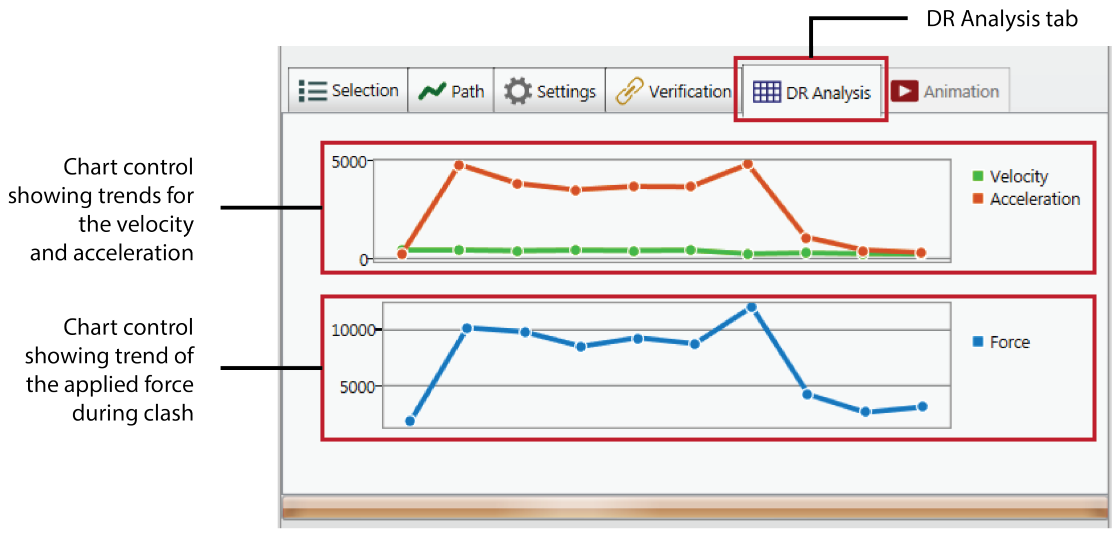
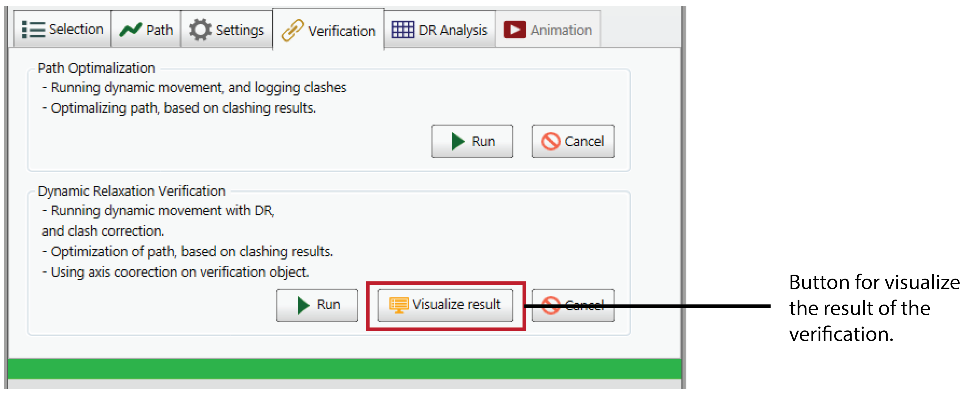

DR Analysis tab
Selection
Path
Settings
Verification
DR Analysis
Animation
Chart control showing trends for the velocity and acceleration
Chart control showing trend of the applied force during clash
5000
0
Velocity
Acceleration
10000
5000
Force

Figure A.8: Overview of the DR Analysis tab, and its different elements.

NOTE: The **DR Analysis** tab only works in some versions of Windows 8. It may occur some errors using it in Windows 8, and in Windows 7 with specific updates.

A.3.7 Visualize result

The user can in the **Verification** tab, also visualize the result after DR verification by using the "Visualize result" button (fig: A.9).



Path Optimalization
- Running dynamic movement, and logging clashes
- Optimalizing path, based on clashing results.

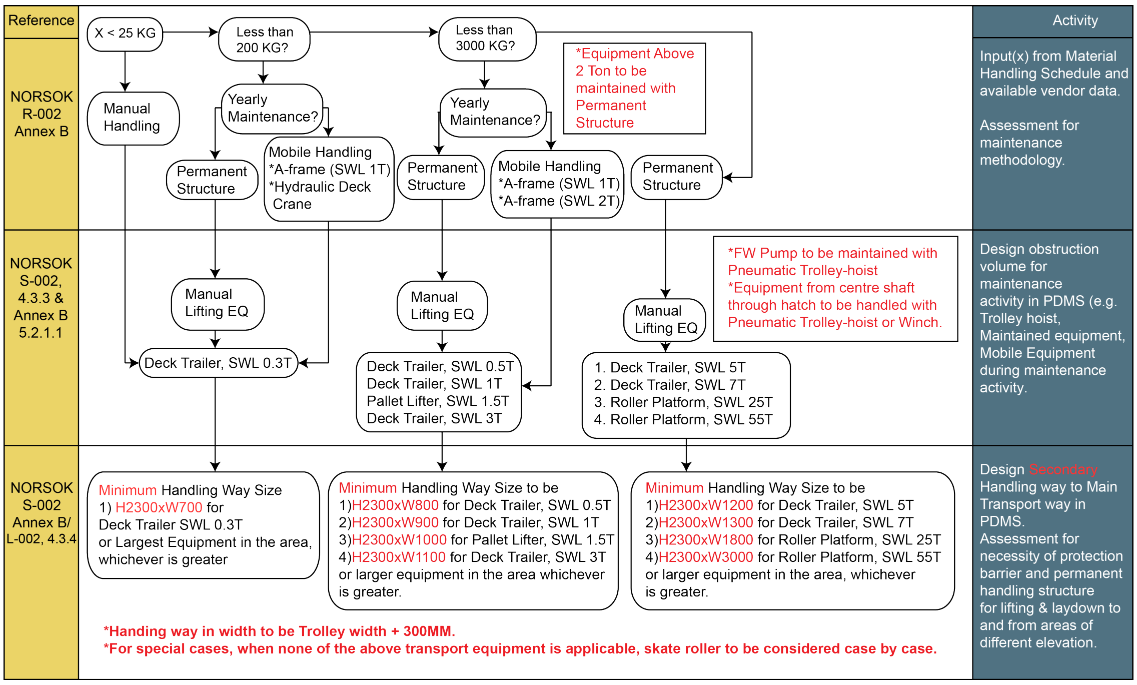
Dynamic Relaxation Verification
- Running dynamic movement with DR,
and clash correction.
- Optimization of path, based on clashing results.
- Using axis coorection on verification object.

Figure A.9: Overview of the verification tab after verification. The user can use the "Visualize result" button to visualize the result.

Appendix B

Documentation

B.1 NORSOK schema



Reference
NORSOK R-002 Annex B
NORSOK S-002, 4.3.3 & Annex B 5.2.1.1
NORSOK S-002 Annex B/ L-002, 4.3.4
X < 25 KG
Less than 200 KG?
Less than 3000 KG?
*Equipment Above 2 Ton to be maintained with Permanent Structure
Manual Handling
Yearly Maintenance?
Yearly Maintenance?
Permanent Structure
Mobile Handling *A-frame (SWL 1T) *Hydraulic Deck Crane
Permanent Structure
Mobile Handling *A-frame (SWL 1T) *A-frame (SWL 2T)
Permanent Structure
Manual Lifting EQ
Manual Lifting EQ
Manual Lifting EQ
*FW Pump to be maintained with Pneumatic Trolley-hoist *Equipment from centre shaft through hatch to be handled with Pneumatic Trolley-hoist or Winch.
Deck Trailer, SWL 0.3T
Deck Trailer, SWL 0.5T Deck Trailer, SWL 1T Pallet Lifter, SWL 1.5T Deck Trailer, SWL 3T
1. Deck Trailer, SWL 5T 2. Deck Trailer, SWL 7T 3. Roller Platform, SWL 25T 4. Roller Platform, SWL 55T
Minimum Handling Way Size 1) H2300xW700 for Deck Trailer SWL 0.3T or Largest Equipment in the area, whichever is greater
Minimum Handling Way Size to be 1)H2300xW800 for Deck Trailer, SWL 0.5T 2)H2300xW900 for Deck Trailer, SWL 1T 3)H2300xW1000 for Pallet Lifter, SWL 1.5T 4)H2300xW1100 for Deck Trailer, SWL 3T or larger equipment in the area whichever is greater.
Minimum Handling Way Size to be 1)H2300xW1200 for Deck Trailer, SWL 5T 2)H2300xW1300 for Deck Trailer, SWL 7T 3)H2300xW1800 for Roller Platform, SWL 25T 4)H2300xW3000 for Roller Platform, SWL 55T or larger equipment in the area, whichever is greater.
*Handing way in width to be Trolley width + 300MM.
*For special cases, when none of the above transport equipment is applicable, skate roller to be considered case by case.
Activity
Input(x) from Material Handling Schedule and available vendor data.
Assessment for maintenance methodology.
Design obstruction volume for maintenance activity in PDMS (e.g. Trolley hoist, Maintained equipment, Mobile Equipment during maintenance activity.
Design Secondary Handling way to Main Transport way in PDMS.
Assessment for necessity of protection barrier and permanent handling structure for lifting & laydown to and from areas of different elevation.

Figure B.1

Appendix C

Risk Assessment

NTNU		Utarbeidet av	Nummer	Dato
HMS	Kartlegging av risikofylt aktivitet	HMS-avd.	HMSRV2601	22.03.2011
		Godkjent av	Side	Erstatter
		Rektor		01.12.2006

Enhet: **Masteroppgave ved IPM** **Dato:** 29.01.2014

Linjeleder: **Marius Røed**

Deltakere ved kartleggingen (m/ funksjon): Marius Hansen Røed (student), Bjørn Haugen (veileder)
(Ansv. veileder, student, evt. medveiledere, evt. andre m. kompetanse)

Kort beskrivelse av hovedaktivitet/hovedprosess: Masteroppgave Marius H. Røed. Verifikasjon og visualisering av tilkomst av utstyr på offshore-plattformer

Er oppgaven er rent teoretisk? *(JA/NEI)* JA *"JA" betyr at veileder innestår for at oppgaven ikke inneholder noen aktiviteter som krever risikovurdering*

Dersom "JA": Beskriv kort aktiviteten i kartleggingskjemaet under. Risikovurdering trenger ikke å fylles ut.

Signaturer: *Ansvarlig veileder:* Bjørn Haugen *Student:* [illegible]

ID nr.	Aktivitet/prosess	Ansvarlig	Eksisterende dokumentasjon	Eksisterende sikringstiltak	Lov, forskrift o.l.	Kommentar
1	Sitte å skrive oppgave/ programmere på kontor.	MHR	http://innsida.ntnu.no/wiki/-/wiki/Norsk/Kontorarbeidsplass+-+ergonomi	http://www.ntnu.no/hms/retningslinjer/Musearm.pdf		
2						
3						
4						
5						
6						