TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIKKO SOUKKA

AREA LAYOUT DATA HANDLING SOLUTION FOR CONTAINER TERMINALS

Master of Science thesis

Examiner: prof. Hannu Koivisto
Examiner and topic approved by the Faculty of Engineering Sciences Council Meeting on 16th December 2015

ABSTRACT

MIKKO SOUKKA: Area Layout Data Handling Solution for Container Terminals
Tampere University of technology
Master of Science Thesis, 54 pages, 8 Appendix pages
December 2015
Master's Degree Programme in Automation Technology
Major: Information Systems in Automation
Examiner: Professor Hannu Koivisto

Keywords: map visualization, information system, layout modeling

This thesis studies on different solutions available on representing locational information in container terminals. Focus is given to creation of this data by using available design models of the container terminal. Main addressed issue was presented in Cargotec Finland Oy, under Kalmar brand. In Kalmar, several different solutions have been used to create this location data to represent the terminal. Different implementations of graphical user interface have been made by separate branches of the company, and each implementation utilizes differently structured data to represent the container terminal. Similarly, different control solutions of the terminal equipment require locational information, which is also separately defined on each use case. This thesis looks into current solutions and provides a new method utilizing a more uniform solution and increasing the level of automation of the process.

The thesis is structured as a division of three parts. Firstly, the problem and container terminal are discussed in general level. This defines the background of the problem as well as variables to be taken into account. Secondly, current solutions from both Kalmar and other companies are described, and possible alternatives to these are also discussed. Lastly, the selected solution is presented, results are analyzed, and future aspects are discussed.

TIIVISTELMÄ

MIKKO SOUKKA: Konttiterminaalien aluesuunnitelmien datankäsittelyratkaisu
Tampereen teknillinen yliopisto
Diplomityö, 54 sivua, 8 liitesivua
Joulukuu 2015
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Automaation tietotekniikka
Tarkastaja: professori Hannu Koivisto

Avainsanat: Kartan visualisointi, informaatiojärjestelmä, karttainformaatio

Tämän työn tarkoitus on löytää automatisoidumpi, ja näin ollen tehokkaampi tapa luoda esitys karttainformaatiosta konttiterminaaleissa. Työ tarkastelee erilaisia metodeja esittää karttainformaatiota sekä tapoja luoda tällaista informaatiota käytettävissä olevia malleja hyödyntäen. Alkuperäinen ongelman asettelu on tullut esiin Cargotec Finland Oy:n Kalmar-tuotemerkin sovelluksista, joissa karttainformaation esittäminen ja erityisesti sen muodostaminen vaihtelee sovelluskohtaisesti. Informaation luominen on myös usein hyvin manuaalinen prosessi. Lisäksi, metodit ja tuloksena saatavat rakenteet ovat sovelluskohtaisia, eli eri sovelluksiin muodostettua informaatiorakennetta ei voida käyttää suoraan toisessa sovelluksessa. Tämän työn tarkoituksena on löytää ratkaisu, joka tehostaa informaation luomisprosessia sekä yhtenäistää karttainformaation rakennetta.

Työssä käsitellään kolme suurempaa osa-aluetta. Ensimmäinen osa-alue on ongelman ja konttiterminaalien toiminnan esittely. Tämän osion tarkoitus on pääasiallisesti esitellä itse ratkaistava ongelma sekä auttaa lukijaa ymmärtämään konttiterminaalin toimintaa. Osiossa käsitellään myös huomioon otettavat muuttujat tarvittavan informaation osalta. Seuraava osa-alue tarkastelee nykyisiä sovelluksia sekä Kalmarin että muiden yritysten osalta. Näiden lisäksi esitellään muita mahdollisia vaihtoehtoja teoreettiselta pohjalta – sekä pohditaan mahdollisia hyviä puolia, mitä kustakin ratkaisusta voitaisiin soveltaa. Viimeisenä osa-alueena on itse lopullisen ratkaisun esittely ja tuloksena saadun datan analysointi, sekä valitun metodin mahdollistamien tulevaisuuden näkymien pohdinta.

PREFACE

This thesis was written for the Kalmar brand of Cargotec Finland Oy, where I have had the pleasure of working in multiple occasions while going through my path of studies. I want to express my gratitude for the R&D Engineering Manager of Kalmar, Petteri Kylliäinen, who made all this possible from the company side. I also wish to thank all my colleagues in Yard Automation team for the help, and especially for the most fruitful conversations that kept the working environment light and joyful.

As for the other side of the page, I want to thank Hannu Koivisto, who supervised the academic side of the writing process. Other than that, I wish to thank my friends to whom I had a chance to relate to, my family for their support, and especially my girlfriend, for believing in me all the way.

Tampere, 16 of December 2015

Mikko Soukka

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASC	Automatic Stacking Crane
AGNSS	Assisted Global Navigation Satellite System
AGV	Automatic Guided Vehicle
ALV	Automated Lifting Vehicle
API	Application Programming Interface
CAD	Computer-Aided Design
CHE	Container Handling Equipment
DGPS	Differential Global Positioning System
FMS	Fleet Management System / Flexible Manufacturing System
GPS	Global Positioning System
GUI	Graphical User Interface
MMS	Magnetic Measurement System
RMG	Rail Mounted Gantry
RS	Reach Stacker
RTG	Rubber Tyred Gantry
RTK	Real-Time Kinematic
STS	Ship-to-Shore Crane
SVG	Scalable Vector Graphics
TBA	A consultancy and software company specialized in container and bulk terminals headquartered in The Netherlands. TBA is a part of TEREX group of companies (TBA, 2015)
TEAMS	Terminal Equipment Automated Management System
TOS	Terminal Operating System
TT	Terminal Tractor
USD	United States Dollar
VB	Visual Basic
WMN	Wireless Mesh Network
XML	Extensible Markup Language

1. INTRODUCTION

Harbor is a complex environment consisting of various static elements such as devices, buildings, fences, laser poles and lanes. In addition to static elements, in an operational terminal the amount of movement and changes is astonishing. Whether the terminal is automated or not has little effect on this, although with automated system less personnel is needed on the field, and container handling equipment units follow more strict paths. Also, in automated environment, all container movements done in the field need to be registered and saved. More collected data about the transfers leads to easier tracking of missing containers and other problematic situations. Currently, several terminal operation systems (TOS) are available in the market, allowing this sort of tracking and logging of actions.

When actions are followed, terminal operator needs to have a clear vision on what is happening inside the terminal area. This is most often achieved with a virtual representation of the container terminal – a map. To present this map to the operator, a great amount of work is required to represent all static obstacles and virtual data in a structured format that can be accessed and read by the software. In addition, unless statically routed, vehicles in automated container terminals require all the same information of their surroundings to be able to avoid colliding with the obstacles.

The process of data collection and finally saving the data into a previously defined structure often requires high amounts of manual work. Also, due to the diversity of the required data and project specific availability of models, automating this process can be very difficult. This thesis discusses the requirements that are set for this map data, methods that can be used to save this information and most importantly, how to create the final structure as efficiently as possible, using automated solutions.

The rest of this document is divided into 6 parts. First, an overview of container terminals is given, where software, geographical areas, and vehicles used on different areas are presented. The point of view is given to the process of map creation, and the functionality of the areas of interest that will be presented in the final solution. Chapter 3 focuses more deeply into the requirements that are set for the end structure of map data. Different areas in the map data are described and the components required are enlisted and verbally described. Possible formats and data types used in the final structure of the map data are analyzed and compared in Chapter 4.

In Chapter 5, focus is moved from data analysis to current solutions available. Chapter looks into the methods used in Kalmar when creating map data for different software.

The functionality of the software is presented shortly, and the methods used to reach the end structure of map data in each of them is analyzed. Later, in Chapter 6, the view is widened outside Kalmar's solutions. In this chapter different methods from other companies, as well as theoretical possibilities are discussed.

In Chapter 7, the selected solution for this thesis is presented. Instructions are given to reach the presented format utilizing given model, and both end result and the performance aspects of the used software are tested. Future development and the improvable aspects of the solution are discussed in the final, 8th Chapter.

2. CONTAINER TERMINAL OVERVIEW

The first modern automated container terminal in the world was the ECT Delta Terminal in Rotterdam. It was opened in 1993, being the first container terminal to handle the transportation between the quay and container stacks with automated guided vehicles (AGV), and utilizing automated stacking cranes in the handling of containers in storage yard. (HPH, 2015) After ECT Delta Terminal several automated terminals have been commissioned, but as they require higher capital costs, the decision of building an automated terminal is often made only in high labor cost areas and when a new terminal is constructed (Steenken, Voß, & Stahlbock, 2015, p. 13).

Rather than using AGVs, also automated lifting vehicles (ALV) can be used for horizontal transport. The name automated lifting vehicle comes from the vehicle's ability to lift containers independently. This feature is the most notable difference between AGVs and ALVs, as it eliminates the waiting times in both quay side and in the storage yard. By eliminating the waiting times, the amount of required horizontal transport vehicles in container terminal can be drastically reduced[1].

Other than different container handling equipment (CHE) types, map data gathering process naturally bases on the layout of the environment. There are often big differences on how the container terminal layout is planned. The layout also depends on the equipment used in the harbor. Small terminals often use flexible solutions, utilizing as cheap and versatile machinery as possible, whereas larger harbors rely on more rigid and sizable, but also more efficient solutions. Selected machinery and whether the terminal is automated or not, among other factors, define which kind of layout is best for the terminal.

This chapter aims to give a general overview of usual container terminal layouts, functionality and machinery, emphasizing the requirements each solution brings to map data gathering process. As many problems in map data gathering process are only valid in automated container terminals, manual terminal solutions will only be introduced shortly and emphasis will be on automated systems.

2.1 Harbor layout

As mentioned above, harbors can be very different from each other in the areas of machinery and design. However, as harbor can be thought as a middleware between land area and a vessel, a basic structure of functionality can be defined. Harbor can be divided in three different areas; seaside, storage yard and landside (see Figure 1 below) (Wiese,

[1] According to the study by Iris F. A. Vis and Ismael Harika (Vis & Harika, 2005) 38% more AGVs must be used than ALVs to achieve same unloading time of a vessel, mainly due to the inevitable waiting times that come from using AGVs.

Suhl, & Kliewer, 2011, p. 220). Seaside area is also known as apron, meaning the area between quay wall and container storage area. Storage area is the area where containers are piled in stacks when they wait to be transported either to land vehicles or vessels. Landside covers deliveries of containers to and from storage yard with trains or trucks. However, terminal cannot contribute much to the functionality in landside area. (Brinkmann, 2011, pp. 25-28; Rijsenbrij & Wieschemann, 2011)

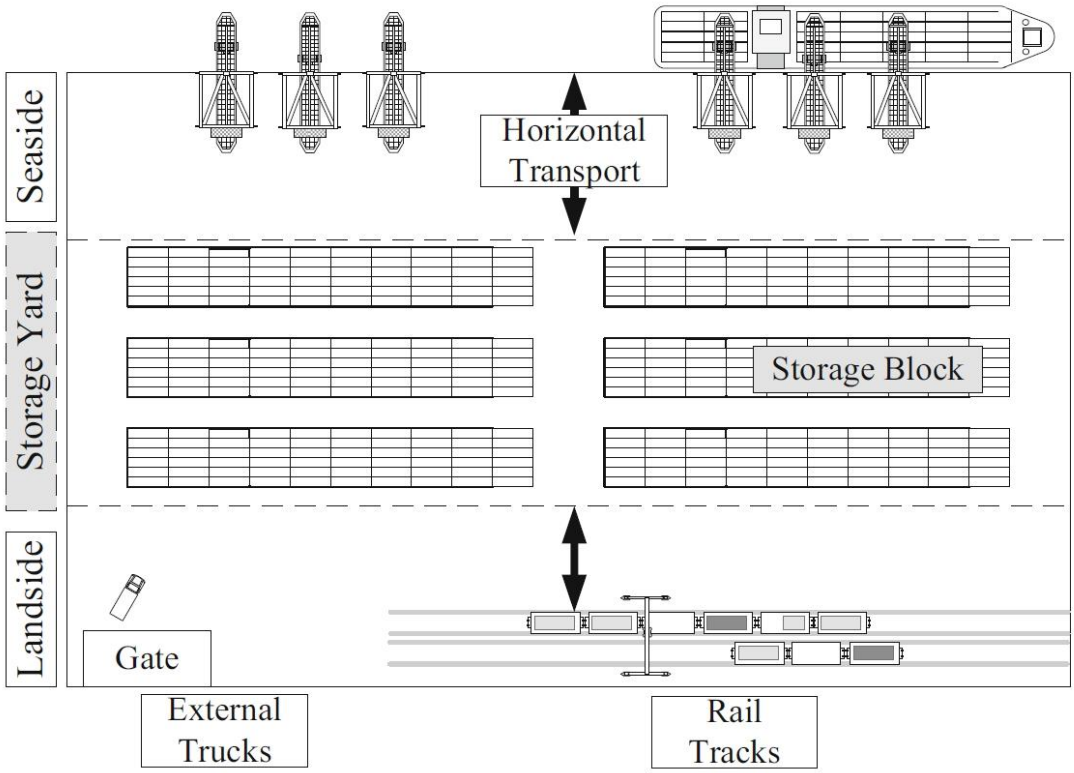


Figure 1: *General overview of a container terminal (Wiese, Suhl, & Kliewer, 2011, p. 220)*

Following sections 2.1.1-2.1.3 will divide container terminal area into above mentioned three sections (seaside, storage yard and landside) and give a closer look to each of them. Machinery used in container terminals will be discussed later, in Section 2.2.

2.1.1 Seaside

Term horizontal transport is used for the transportation between quay wall and container stack in storage yard (Wiese, Suhl, & Kliewer, 2011, p. 220). This can be handled either by manual or automatic system. Manual horizontal transport is still much more common, and on map data processing point of view doesn't offer much challenges for implemented software, as all vehicles are driven by drivers and obstacle handling is done mostly by the driver himself. Whereas manual system is easily modifiable, automatic horizontal transport makes the system relatively rigid, and hence any changes in the layout require

significant effort. One example of automatic horizontal transportation is to use automatic shuttle or straddle carriers, which will be inspected in more detail later.

The transfers between the vessel and apron area are usually handled with Ship-to-Shore cranes (STSs), which is the most efficient crane type for these transfers (Brinkmann, 2011, p. 30). STS works on a physical rail that follows the quay wall. Under STS, there are lanes where vehicles can drive next to each other. When speaking of control software of automated systems, this area needs special attention, as the software needs to recognize the area to be able to allow vehicles' movement in the area alternately.

In addition to the STS lane, other important information for map info gathering process in apron area are locations and sizes of any physical obstacles, like those of fences or light poles. Size information, in this case, means the height of the object, and external dimensions in the x-y plane. Also borders of the area, including the sea line, need to be specified.

2.1.2 Storage yard

There are three main types of storage yards: block stack, linear stack and high-bay racking. First two of types are more common, whereas high-bay racking is only applied when high throughput is needed, but available space is very limited. In this method containers can be stacked up to 12 container tiers (horizontal container layer) high piles (Brinkmann, 2011). Due to the very rare usage of this method it will not be discussed more in this document.

Block stacking is common on harbors where stacking area is limited, as it offers good density of containers relative to ground area used. Stacks are built with gantry cranes, so containers are brought to the end of the stack using horizontal transport. Block stack's height can be up to 8 container tiers. Linear Stacking refers to system where straddle carriers are used not only for horizontal transport, but also for storing containers. Using this method, stacks are not higher than four tiers, due to the physical limitations of SCs. This reduces the cost of the machinery, as the same machine can work on horizontal transport, but at the same time requires notably more space than block stacking. (Brinkmann, 2011)

If the system is automated, and block stack is used, interchange areas in both ends of the stack blocks include several details to be considered when creating a map representation of the area. In the layout map, places of lanes and exact container positions are required, alongside with the information of surrounding objects and light gate positions. Common practice is to have an interchange area on both ends of the block. Seaside interchange area is used to move containers to and from block stack to horizontal transportation vehicles. This transaction is done fully automatically if automated horizontal transportation system

is used, by calculating vehicle routes to exchange location synchronously. On the landside, truck drivers drive their vehicle in the defined lane where container can then be picked from or grounded to by the crane.

Also, maritime terminals have a growing need for refrigerated container ("reefer") handling solutions (Hughes, 2008). These containers have an integral refrigeration unit, but they rely on external power, and thus require modifications on standard container stacks (see Figure 2).

Figure 2: *Refrigerated container racks (MIP, 2013)*

The picture above represents racks between containers in block stacks. As can be seen from the picture, racks are high structures, and if operation is automated, it is crucial for software to know their height and location to be able to operate on top of them. Also, as refrigerated containers need to be plugged in manually, system needs a signal from whenever a person is in the operational area, to be able to either stop operation or handle transfers further away from the person.

2.1.3 Landside

Landside operations cover the transport between storage yard, empty container yard and different handling areas like railway station or barge terminal. Main concern when planning landside area is to create a clear and smoothly flowing structure for traffic. Most movements in this area are done manually, and the connection to road transport can easily turn into a bottleneck of the whole operation. Structure of the area should be efficient and preferably use automated methods in identifying trucks in both, in-gate and out-gate. Also, good planning can prevent congestion on public roads caused by container terminal. However, bad planning of infrastructure of public roads can also have a negative effect on terminal area efficiency. For this thesis' scope landside has little relevance, as the automated operation focuses in seaside and storage yard. However, a possibility for automated functionality exists for terminals where landside operations are structured between block stack and railways, and transports are handled with e.g. ALVs rather than terminal tractors.

2.2 Equipment

There are a few main groups of container handling equipment (CHE) in harbors. Equipment can be roughly divided into ship-to-shore cranes, rail mounted and rubber tired gantry cranes, shuttle and straddle carriers, terminal tractors, reachstackers and both loaded and unloaded container handlers (Kalmar, 2014; Konecranes, 2015). Rail mounted automated gantry cranes are better known as automatic stacking cranes (ASC). Also both shuttle and straddle carriers are available automated (Kalmar, 2014), and in the future automated rubber tired gantry (RTG) systems will be competing (Konecranes, 2014; Ylä-Himanka, 2014) with ASC systems. Offering a choice for automated straddle carrier systems, for example ports in Rotterdam, Netherlands and in Hamburg, Germany utilize AGVs efficiently as horizontal transportation devices (Terex, 2015).

The solutions described in this thesis are aimed to cover automatic devices, so the demands of automated vehicles considering environmental factors are in essential position when defining requirements for the system. From the layout point of view these equipment pose a very different requirements from those of manual operation, but also from each other. Following the logics of area division, equipment is divided into quay cranes, horizontal transportation vehicles and gantry vehicles. Equipment used in the landside area of terminal will not be taken into account as they do not require special focus on applications using layout information.

2.2.1 Positioning and coordinates

In automated container terminal, the terminal area is described in local coordinates (also known as yard coordinates), which allows the usage of coordinates relative to local origin instead of for example raw GPS coordinates. This improves the readability of the information drastically, and also reduces the risk of mistakes when modifying location data. GPS has been used in container terminals since 1990s, and it is still very often found in container terminals. Global positioning components are installed to most container transporting vehicles, after which position of the vehicle can be read, translated into local coordinates and sent to operating system. (Steenken, Voß, & Stahlbock, 2015). While GPS not being the most accurate choice, current terminals rarely qualify GPS “as is”, but use more advanced assisted global navigation satellite systems (AGNSS), such as differential GPS (DGPS) – optimally with real-time computation using real-time kinematic (RTK).

Due to large amount of high metallic structures, container terminal is a challenging environment for GPS measurements. There can be issues in the accuracy of received signals, especially due to signal multipath effect, which practically means signals reflecting from the surface of the Earth or nearby obstacles (Kuusniemi, 2005, pp. 45-48). In this case, obstacles are practically container stacks and tall cranes, such as STS cranes. This said, GPS measurements can be aided with other methods of localization and sensors that keep

track on vehicle's location and heading. Vehicles can also be tracked using optical systems, especially laser scanners (Steenken, Voß, & Stahlbock, 2015), that also offer a good method for preparing against unplanned environment changes.

As mentioned earlier, coordinate systems used are separate from GPS coordinates. Coordinate systems are not tied to anything and can be selected freely, as long as same coordinates are used consistently throughout the system. Most often origin is set in the corner of the container terminal area, and all points in the area are positive, representing common measurement units. These units are of course logically selected according the preferences of the country – e.g. meters in central Europe, feet in America. This is an important detail when working with data from different container terminals. If a transformation between file types is done, it is important to make sure that coordinate systems do not change in the process.

The same coordinate system should also be used when drawing vehicles on the virtual presentation. Vehicle positioning is done on-board in the vehicle, and location then sent to the information system. Similarly to GPS signals, communication problems on data transfers may occur due to the large quantity of structures and obstacles in the area. These communication issues may result in a difference between locations of the vehicle in real world and on the virtual screen, or in so-called rubber band effect, where vehicle jumps on the screen when it gets its position too late. Also monitoring the heading and moving direction of the vehicle can be a difficult task. Pictured vehicle will easily be shown pointing to wrong direction especially on situations where vehicle is staying still or making slight movements.

2.2.2 Quay transfers

The process of unloading and loading a ship was previously handled with on-board lifting gear of the ship or conventional quay cranes (QC). This equipment is still in use, but only in low container throughput terminals. Nowadays the most common and efficient crane for ship-to-shore operation is Ship-to-Shore crane (STS). STS is a gantry crane specified for the purpose of handling container transfers between the vessel and apron area. STS is used for ship-to-shore operations in medium and large terminals (Brinkmann, 2011), and thus it will be the only vehicle type of this category considered in this thesis. Although STS is often referred to as quay crane, Brinkmann describes it as gantry crane, so for clarification abbreviation STS will be used later in this document.

The beginning of container terminal planning is determining how many STS cranes the terminal will need to fill the throughput requirements planned for it. This information is formed by forecasting the amount of vessels arriving for example by executing simulation studies. Due to the size of STS and its large capacity, layout of the seaside area has to be planned to allow smooth horizontal transport. (Wiese;Suhl;& Kliewer, 2011) STS works on a lane next to the quay wall, handling transfers of containers between the vessel and

apron area. The vehicle used for horizontal transport is driven under STS to handle the container that STS unloads from the vessel as demonstrated in Figure 3 below. Depending on the equipment used in the terminal, container is either placed on the ground or on top of a horizontal transportation vehicle.

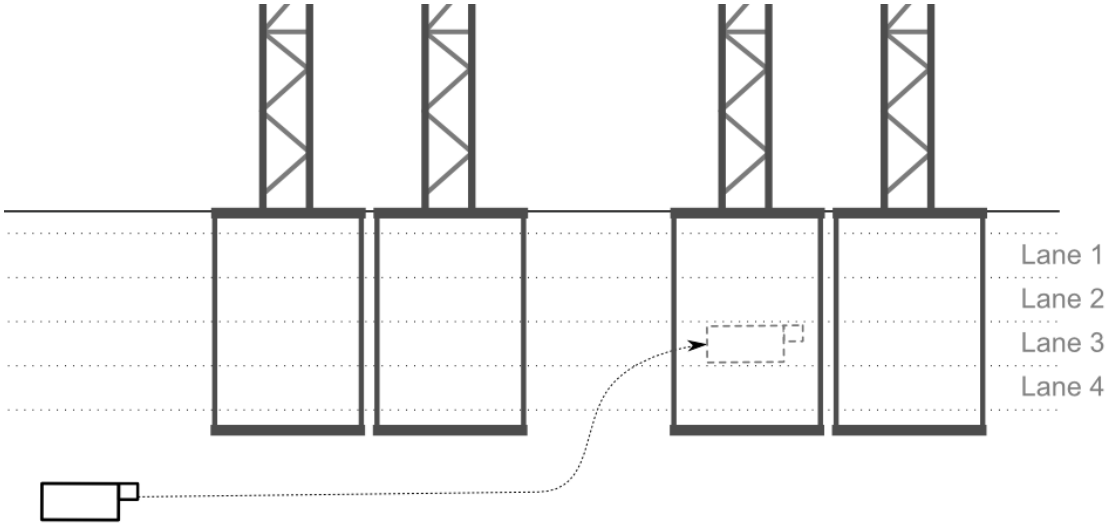


Figure 3: *STS lane usage example (Mäkelä, 2015)*

As vehicles are driving simultaneously in the same area, it is crucial to have positional information of each vehicle, and to be able to recognize the working area of STS (the physical lane) unambiguously. However, representing STS on a map is relatively easy due to the simple nature of its movement. As long as the vehicle type is known, vehicle can be represented with coordinates relative to rail beginning or to area origin, as it is always known that the position is within the rail.

2.2.3 Horizontal transportation equipment

Most common manual systems utilized for horizontal transport are either a combination of terminal tractors (TTs) and reachstackers or SCs (Brinkmann, 2011). If the system is automatic, either automated guided vehicles (AGV) or automated lifting vehicles can be utilized (Pirhonen, 2011; Duinkerken;Evers;& Ottjes, 2002). Out of the latter two, AGV requires containers to be grounded on top of it, or conversely picked from it. This means that cranes have to wait for AGV to drive under them in both ends of the horizontal transport, before container can be grounded or picked. This causes involuntary waiting times, as AGV is bound to stay still until containers are handled by someone else. Shuttle and straddle carriers are more independent, and can pick and ground containers by itself, thus allowing other vehicles move more freely.

When speaking of manual systems, TTUs can be compared to AGVs, as they also need to be driven under the container, which is then picked or landed on the chassis of the TTU. Manual operation can be achieved many ways. Large or medium size terminals

often implement SC system as they give high flexibility and accessibility while small and medium sized terminals usually prefer a combination of terminal tractors and reachstackers. This is because reachstackers can be used for many different operations, which is desirable especially in small and multi-purpose terminals. (Brinkmann, 2011) When speaking of manual terminals, a lot less information is required by fleet management system (FMS) than in automated terminal. In manual terminal collision avoidance can be aided by technology, but merging of the routes is handled by drivers, not control system. Map information can however be used to manually decipher which job is given to which driver, by comparing the visual distances between vehicles and the job location on the map. Nevertheless, map information is needed more for representing the information in the map, not for controlling the equipment.

Automated environment brings more requirements for software functionality, as the information is used to control vehicles alongside each other, avoiding collisions. All automated vehicles have certain routes they are allowed to move. These routes are dependent on each vehicle's turn radius, positions of lanes and obstacles in the area and allocated container positions. However, these allowed turns belong to scope of control software, so they will not be discussed further in this document.

Operating vehicles themselves are relatively easy to represent in layout map, as long as there is an accurate method available for locating the devices. Hence, only information needed for simple dotted representation, alongside with vehicle type and vehicle id, is coordinate information. If representation is more sophisticated and vehicle is pictured with more realistic virtual figure, also heading and moving direction information are needed.

2.2.4 Gantry cranes

Gantry cranes can be either rail mounted or utilize tires. Most common types of these are previously mentioned ASCs and RTGs. They both move on rails, ASC on physical and RTG on virtual, where container positions are accurately specified. Especially when speaking of automated systems, rails are most often surrounded by either fence or similar methods, such as concrete stoppers. In this case they also need to be taken into account when drawing the map. In addition, on ASC operation, both waterside and landside of the rail have an interchange area. Depending on the automation level of the system, accessing interchange area can be controlled with either physical gate, or virtual methods, like light gates. Also, lanes can be separated from each other with a physical obstacle, usually a fence (see Figure 4).

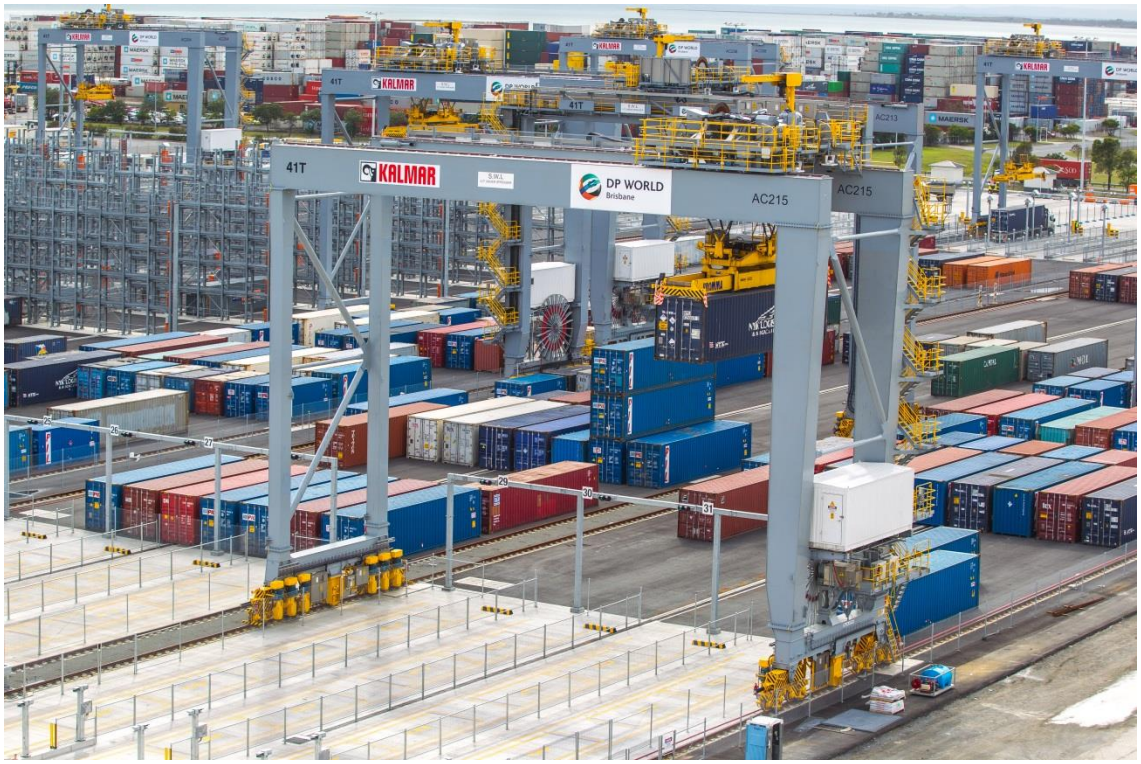


Figure 4: *Kalmar ASC Brisbane (Cargotec Corporation, 2015)*

In the situation of the above picture, different lanes are separated with fences and accessing them is controlled by light gates. Light gates are used to detect shuttle or terminal tractor accessing the lane by recognizing an interruption in the sent light beam. As can be seen from picture above, there are two sets of light gates. The reason for this is that gates need to recognize that the element in between them is actually the expected vehicle, as it is also possible that the light beam is interrupted by another obstacle, for example a bird flying past them.

Operation with RTG is slightly more complex, as tires allow more freedom in movements. Where ASC is strictly bound to its rail, RTG can be driven from block to another when all jobs on first one are done in the first one. This not only makes automated control very difficult, but also creates a problem for safety concept, as automated machine will move in the area where people also have an access to.

When controlling vehicles that work above trains, functionality is slightly different from normal interchange area work. Trains are usually filled and emptied with a rail mounted gantry (RMG) that has a cantilever on either or both sides. Operation can also be automatic, but rails are not necessarily straight under the RMG. Also, common practice is to load containers on trains with doors facing each other. This means, that the RMG needs to be notably more flexible than in regular block stack. These requirements are met by using a rotating spreader, which allows changing the orientation of container before grounding it on train, and also picking up containers that are not parallel to RMG rails.

2.2.5 Summary

Container terminal operation is a co-operation of many different CHEs, which on the software level often requires very different data handling procedures. For example, when comparing SC and a RTG, SC has a static position of spreader in relation to its center position. This means, that shuttle's spreader moves only vertically, so its position in x-y plane can be tracked by its spreader position. In RTG's case machine center point and spreader center do not have a static offset, as spreader is moved orthogonally to RTG's movement direction, and thus an offset is needed. Also, in addition to container handling equipment co-operating in the terminal area, the area layout consists of not only roads and containers, but also various different components and obstacles significant to any operation performed in area, and hence crucial to be taken into account when managing map data. When considering the data to be taken into account when creating the map representation of the area, big differences also come from the used automation level, as less human interaction is needed when actions are automated, but clear visual representation of events from the site is emphasized.

As an example, on a manual system driver can see whether other vehicles are coming to its way, and also be informed by the operator, but on automated system all routes are handled by terminal operating system (TOS). The responsibility for the safe operation is laid on software, and operator has to be able to see clear representation on what is happening in real time. In other words, the visualization of the terminal area should realistically represent the whole area limits, obstacles, routes and vehicles in it. The main focus of this thesis is on the visualization of the data, although compatibility with control applications is considered. If applications follow same data structures and naming conventions, data can be used across different applications.

2.3 Information system

The physical controlling and monitoring container terminal takes place in control room within the terminal area. Even if actions in the terminal were automated, functionality is administered and supervised by an operator, using either a control station or a computer on site. The interface of the control station consists of either ordinary computer, or special control station consisting of displays visualizing actions and several levers and buttons to perform actions. The presented view shows a visualization of the container terminal, status information of vehicles, possible alarms and allows operator to control the fleet if interventions are needed. Also live views from cameras can be included – especially when performing tasks demanding high accuracy.

Figure 5: *ASC control table*

The terminal operator's main task is to monitor the operation in the terminal area and to react to abnormal situations, such as receiving alarm messages from vehicles, or faults occurring in the system. When interference is required, or specific move required in manual terminal, operator can give tasks to vehicles. In automated terminals and large terminals container handling cycle is handled by terminal operating system (TOS). TOS is responsible for high level monitoring and management of container terminal operations. The main task of TOS considering terminal operations is to calculate the best schedule for container operations. Optimally configured TOS minimizes waiting times and travelled distance, whilst being as efficient as possible in unloading and loading vessels. TOS can also handle tasks such as data storing, automatic gate surveillance and messaging, billing, reporting services and more, depending on the configuration on current site. Some known TOS suppliers are Navis, Jade and CyberLogitec (Schuett, 2011).

When TOS is considered as one and only entity controlling the terminal, it uses two interfaces for communicating with container terminal equipment: Order interface and equipment control interface. The former is used to send orders to the equipment and the latter to send and receive status information of the equipment. (Schütt, 2011) Different way of looking into the process is that in between TOS and physical equipment there is software that works as a translator between TOS and equipment. This middle software will further on be referred to as fleet management system (FMS), although different names, such as TBA's Terminal Equipment Automated Management System (TEAMS) (TBA, 2015) have been given case-by-case. In this thesis, a clear separation between TOS and FMS functionality will be made further on. FMS is responsible for the actualization of the tasks defined by TOS, optimizing the routes and handling both automated collision and deadlock avoidance procedures. Also error and warning handling is done with FMS.

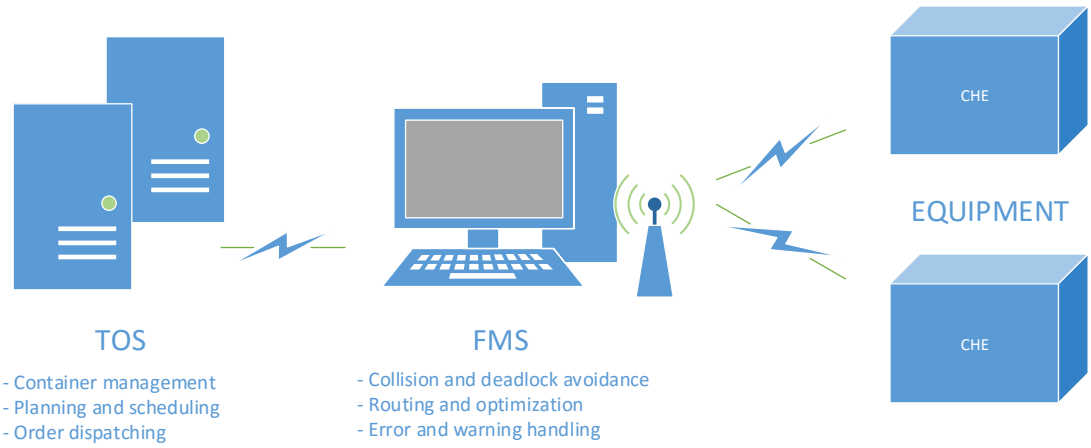


Figure 6: *Container terminal information system structure*

Communication between different systems can be handled in various ways. Between TOS and FMS the separation is simple, as they can in some cases even run on the same server. Equipment to FMS communication is handled utilizing wireless network, most often IEEE 802.11 specifications with specific protocol or protocols defined according to local preferences. Also other protocols have been tested, such as node based Mesh networking which allows each device in the network to work as an access point, and thus in theory method would provide more reliable connection. However, large amount of challenges remain with wireless mesh network (WMN) solutions, of which not least concerning the shortages in security. Messages can hop relatively freely between clients, which makes end-to-end security nearly impossible to guarantee (Mogre, Hollick, & Steinmetz, 2007; Gungor;Natalizio;Pace;& Avallone, 2008).

3. STRUCTURAL REQUIREMENTS

Container terminal is an environment with an emphasis on efficient operation. Especially when talking of automated and semi-automated container terminals, the responsibility of actualizing the logistics concept of the terminal is laid heavily on terminal operating system (TOS) (Saanen, 2011, p. 90). Logistics concept, in this case, refers basically to the way terminal is operated and controlled. For smaller and manual terminals, it can be difficult to find a TOS, which would fit to their needs, and which they could afford (Esoware B.V., 2011). TOS can still be worth considering because well configured TOS can bring clear cost reductions by improving the way terminal is operated (Saanen, 2011, pp. 90-94). This thesis focuses on implementations using TOS, and more accurately FMS.

The operator of the container terminal is given a graphical interface which consists of visualization of the terminal alongside with functionality to control operations in the field. The visual data represented for the operator and the data which is used by software itself to control are similar in many ways, although visual data heavily lacks in precision and informational capacity in comparison. Main requirements for visual data used in standard 2D representation are geometrical information in x-y plane and the type of the object, whereas controlling the equipment requires identifiers, safety limits, allowed routes, location information and heights of obstacles with high accuracy.

The aim of this chapter is to give reader a more specific view on what data is required from the container terminal when planning a map representation. Analysis is made from the software point of view, but also required container terminal elements are introduced with a layer-based division. The two implementation environments – graphical usage and control solutions – are more carefully inspected in the following sections, followed by more accurate analysis of the data structure that will be formed for container terminal elements.

3.1 Implementation environment

As described above, there are two different application types that require different levels of accuracy from the map information; applications providing user interface and control software. The actual controlling of CHEs is done with information which is collected with accurate measurements from the site. Visualizing the data to user interface is also an accurate process, but it doesn't have as hard requirements of precision as for control system data. Information used to visualize terminal is not used in controlling the vehicles, but merely to visualize the terminal area for user and allow user to send commands to actual control interface. However, all obstacles and vehicles must be taken into account, as operator has to be able to see events occurring in terminal area.

Using different data layers for different types of obstacles allows drawing them realistically on the map. Main point for graphical user interface is to offer distinct information from the site area for operator, preferably with as little distractions as possible, whereas control software handles movements using the information – or in other words trusts the data to offer accurate information of CHEs' positions in relation to obstacles.

Also virtual technologies have taken big steps forward, and it is a tempting possibility to utilize cloud based services either instead of or alongside local systems. Cloud based solutions are recognized widely in areas such as truck transportation, public transport and mining solutions (Fleetio, 2015; GISCloud, 2015; Takada, et al., 2014; Intel Corporation, 2015). Automated container terminal environment is, however, more complex in terms of required accuracy and real-time needs. Above referred analysis from year 2014 (Takada, et al., 2014) indicated, that the latency of cloud based services in comparison to local stand-alone system for real time control process is still considerably higher, but also proved that current methods are reaching better and better results in reducing them. Nevertheless, discussion about whether it is desirable to use cloud based services in the real time control of container terminal equipment instead of local system, is evidently out of this thesis' scope, as solutions presented and compared are all based on local solutions.

However, even though fleet management relies on real-time information, several TOS-level tasks, such as high-level fleet monitoring, can be handled with more delays. For such operation, cloud services can be applied. Web-based implementations of map can be achieved either by using local maps, or any of the several application programming interfaces (API), from Open Source (Mapbox, 2015; Wiki, 2015) or proprietary (Here Maps, 2015; Bing, 2015; Google, 2015; ArcGIS, 2015) sources. This would allow representing several terminals around the World in the same map, although with a cost of reduced accuracy of the map comparing to local map solutions[2]. Described web-based system would, however, be dependent on the provider of the API, which can become a problem. Also, it is to be noted, that the information can be also otherwise changed by the provider[3], although often only towards higher precision.

Following sections will discuss graphical user interfaces and control systems separately, looking deeper into their features and requirements from the locational data point of view.

[2] According to a relatively wide study in 2008 (Potere, 2008), root mean square errors (RMSE) of Google Earth satellite images in comparison to real locations were between 5.4 and 163.3 meters depending heavily on the country of the location under measurement. Another study, with measurements done in 10 separate countries between years 2009 and 2011 also compared different providers, reaching RMSEs of 8.2 meters for Google, and 7.9 meters for Bing maps (Ubukawa, 2013). Third study, that took place in 2012 in Khartoum, Sudan, (Mohammed;Ghazi;& Mustafa, 2013) reached RMSE of 1.8 meters for horizontal coordinates of Google Earth.

[3] According to the study (Mohammed;Ghazi;& Mustafa, 2013), the average RMSE calculated from same 16 locations changed from 3.63 meters to 1.8 meters between September and October 2012.

First, different implementation options of graphical user interfaces are handled, and further on in the latter chapter control systems are defined and their functionality is described.

3.1.1 Graphical user interface

Graphical user interface (GUI) accuracy requirements for the map data are less strict than that of control software's. GUI is an interface for the user to communicate with the control system, give commands to control the fleet and to monitor the status of CHEs on the field, amongst other tasks within the scope of FMS system. The visual representation of the area in the GUI can vary from a simple map with plain lines to beautiful 3D representation with realistic pictures of containers and machinery.

When designing GUI view, technical properties are often emphasized, which can lead to usability issues. The interface easily turns out to be too complicated and hard to use, whereas the best usability of the GUI is achieved by simplicity and clarity. (Luostarinen;Manner;Määttä;& Järvinen, 2010) As container terminals are often very large entities, planning the GUI to be clear and easy to use can be a difficult process. A realistic 3D representation can easily become very unclear, or aim user's focus away from important events in the area. On the other hand, even though simple line based 2D representation keeps the focus in the entity better, it cannot offer similar perception of the heights of obstacles, machines or container stacks.

Although it can be debated whether the visualization should be done in 3D or not, base information about the area remains similar. However, depending on the accuracy in customizing the visual elements used to represent real life ones, more identifiers might be needed, for example to separate different colored containers from each other. Other than that, locations of the elements remain the same, but 3D map also requires a lot more information in form of 3D designs of the elements. Generating the locations of obstacles in x-y plane from a ready design model of the container terminal would be beneficial, as it would drastically reduce manual labor from the map creation process. Nevertheless, depending on the nature of the conversion it can be difficult to preserve all crucial information throughout the conversion process.

Further than visualizing the map to the operator, methods used must allow the presenting of status information and attributes of vehicles, and when needed, alarms and diagnostic information from the vehicles. Often a separate view is solely for this purpose, but for clarity, some of these notifications can be implemented to be shown inside the map representation with different colors or icons. This allows the user to visually connect the information to the source, and can thus be much more informative than a verbal description.

3.1.2 Control systems

By definition control systems are “Interconnections of components forming system configurations which will provide a desired system response as time progresses.” (Farlex, 2015) Following the definition, TOS is a control system that provides outputs in operational level. FMS continues to lower abstraction level, and outputs in more concrete actions over terminal tasks. These lower level functions and control can still be distributed to one or more smaller entities. Distributing functionality offers a possibility to divide calculation amongst more than one computer, and makes it easier to handle separate control entities. This kind of separation is also done by Kalmar utilizing so called UniQ-platform[4], where services executing separate features are implemented. These services can for example monitor driver actions, offer positioning information or handle steering and controlling of a CHE unit (Kylliäinen, 2010).

In this thesis, focus is given on entities that focus on controlling vehicles, and thus the term control system will further on used to define a system that works on low abstraction level providing methods for controlling vehicles. In other words, control systems refer to software that utilizes the location information of the container terminal to determine where the vehicle can be steered without danger of collisions. As an example, a control system handling ASC movements in its block requires exact location information of everything physical that is in the area alongside with height information of each element, as well as virtual information of allowed areas, moves and waiting positions in order to move safely. Although safety distances are used to cover certain error limits, the positions must be known very accurately.

The forming of map information for control systems is a very manual process. The information received from the design of the container terminal is practically never accurate enough to be used *as is* for control systems. To get more accurate information, measurements need to be done physically on the terminal area. Also, there is no one and only method for saving the measurement information. One option is to upgrade older models according to new measurements, but this is rarely the case, unless specifically ordered. Because of the inaccuracies of the models and inconsistent ways of handling newly gathered data, the generation of the map information is very difficult. In the worst case it creates more manual work in inspecting and testing the generated data, than manual drawing with a fluently functioning layout drawing software would.

3.2 Information structure

In the final representation of the location information, there should be a clear difference between many types of data. In the visualization used in GUI, a drawn line between coordinates (x_1, y_1) and (x_2, y_2) can represent a wall of a physical building, an edge of a

[4] Formerly Cubic platform

container, a cable over the area or just a position of a road centerline. All of these elements, along with many more are important information when creating the map but should all be handled differently, if shown for the user at all. Some information is not important for control software, for example a building outside driving area, but can be important to be drawn in GUI as it gives a better view of the area to the end user. On the other hand some information does not necessarily need to be drawn in the map, but is important for control software, e.g. an allowed turn of a CHE.

These separated sets of information will further on in this document be referred as layers. In the software level, a layer representation can be achieved relatively easily, for example by using a markup language that gives a possibility to use named tags, or using software that has inbuilt support for layers. However, as different programs handle layers with alternate methods, it can be problematic to transfer data from one representation to another. Hence, to create an entity, where separate systems don't need customized data, it is profitable to use one method for all data.

The separation of data into different layers can be done in practically indefinite amount of ways, so no one best solution can be defined. Although details can be handled in different ways, automated container terminals have many requirements that have to be covered. Following chapters will go through different areas of a container terminal, defining common components and factors to be taken into account when representing them in map data.

3.2.1 Area limits and zones

For control software it is important to know the physical limits of the allowed driving areas. For GUI, defined area can be a bit larger, or even the whole container terminal area, depending on the selected solution. Area is often bordered by the sea on one end, and buildings, fences, or other constructions on the other sides. Areas without a clear border are a safety issue and should not exist. The apron area of large container terminals can further on be divided into cells (a grid dividing the area into smaller entities), which allows faster visual navigation within the area.

Other than cells, in apron area there can also be separate zones where one can be allowed to drive normally, slowly, or not be allowed to drive at all. Defining these areas can happen either statically when the maps are created, or dynamically, while vehicles are operated on the field, for example in case of accidents. Areas can either be defined only in the software, or there can be components in the field, that create the limits which software follows, such as light gates, lasers or fences.

3.2.2 Physical obstacles

Some physical obstacles, such as fences, buildings, lane dividers and laser poles are situated within the driving area. These constructions are important information for both, GUI and control software. Physical objects can be considered as a layer, where no-one is allowed to drive, as vehicles are rarely allowed to drive over physical elements except for other containers. A special obstacle can define other borders for obstacles, it's size can be changed, or it can have other functionality, depending on the nature of the obstacle. In this case, object should be identified from the mass of obstacles. For a normal static obstacle, it is enough to define the locations of its horizontal borders with added safety limits, according to safety protocols of the system. In addition to static obstacles it is possible to see obstacles that are not defined anywhere, such as misplaced containers or wildlife that has passed through fence. These are important to be considered when planning the safety systems for control logics, but do not affect map creation process.

3.2.3 Gates

Gates are all over container terminal. There are gates for both vehicles and personnel – for example reefer racks can be accessed by personnel when areas are isolated from automated actions, and vehicles can be driven to a separate service area when maintenance is required. Due to their changing status, gates cannot be considered as a normal obstacle. There can also be differences in the way gates are opened; by sliding them, more traditionally with a certain opening radius, or completely in software level, in case of light gates. In software level, when movement is allowed through (gate is opened), area can be considered as a normal road, and respectively closed or closing gates should be considered as a solid obstacle.

When talking of light gates, functionality is different than with physical gates. Where human access is controlled with for example traffic lights and physical gates, automated systems can use light gates, which only recognize access, as software handles movements to area already. Light gates also can be positioned in the terminal to define areas instead of just accessibility. For example in completely automated terminal there is an automatic transfer area in the waterside of block, which can be enclosed with light gates. However, the functionality on how gates are handled in software level is not in this thesis' scope, but gate types that act differently comparing to each other are to be recognized in the container terminal area data.

3.2.4 Blocks

Block defines an area which encompasses the functionality of one container block and the vehicles working in that block. In other words, block is an outline for all functionality; rails, container positions, vehicle positions, obstacles, allowed movements, truck kiosks

and other information that is tied within that certain block. Each block has own configuration of information, some of them have reefer positions, some are longer than others and so on. Due to their complexity, large amount of information is saved on each block, and it is often required that blocks are handled separately when saving their information.

For GUI, however, a lot of mentioned information can be eliminated, as presenting all information would result in very unclear set of overlapping lines. Most important pieces of information are ASC lanes, vehicle position and clear obstacles in the area, such as reefer racks. Adding even container positions can result in unclear representation.

3.2.5 Rails and lanes

Rails can be of two types, physical or virtual. Physical rails are used by ASCs, RMGs or STS cranes. Virtual rail is, instead of a trail on the ground, a line between two coordinates used by the control system of RTG to recognize if RTG is moving straight. Functionality of physical and virtual rails is often same or similar – meaning that RTG is driven as if it was on physical rails, with the exception of added perpendicular position error handling. ASC and RTG rails are also a part of a container block.

Rail information can also be used to specify certain vehicle to its lane. This makes it possible to use offsets between the static rail and spreader, to control spreader location in respect to the container positions. Container positions themselves are discussed in their own chapter below. Also, lanes exist not only in RTGs case, but also each automated CHE that moves in apron area has to have allowed movement routes. Practically this means selected amount of horizontal lanes and allowed turns between them.

3.2.6 Container positions

Container positions are a complex variable to take into account. There are many different lengths of possible containers, as well as some variation in height. External dimensions of containers are defined in ISO 668 standard, albeit some variations to these dimensions exist in area specific customary systems (GDV, 2015). In 2012, most common container sizes were 20’ (6.058 m) and 40’ (12.192 m) in length and either 8’6’’ (2.591 m) or 9’6’’ (2.896 m) in height (CST, 2012). Less common lengths of containers defined in ISO 668 are 10’, 30’ and 45’. In addition to these, in some areas sizes of 24’, 41’, 43’, 49’, 53’ and 57’ are allowed. (GDV, 2015)

Depending on the sizes of allowed containers, positioning of the containers should be planned carefully, and all possible positions saved. However, when designing GUI, positions are not necessarily wanted to be shown, as they easily make the screen overly full and unclear.

3.2.7 Miscellaneous

When talking of a map that is shown to user via GUI, showed information is rarely strictly limited to the physical elements in the area. In addition to these elements and vehicles, also data such as logos, labels, descriptions or raster images can be added in the representation. In the area there are also several sensors and devices that can for example indicate maintenance positions or limits of movement areas. Depending on the functionality of the pieces, they can be wanted to be included as separate component in the representation. Moreover, status information of each automated device is crucial for safe operation, as well as information on each occurring fault or error of the vehicle.

4. DATA FORMAT

There is no one correct solution for saving location data from container terminals – or anywhere else. Data of this kind can be saved in practically any format as long as the required tools are available for parsing and/or using the data. This chapter will discuss and compare different solutions in respect of map visualizing solutions. Firstly, binary and text based file saving used with layout editors are compared shortly. In Section 3.2.1.2 images are shortly introduced in general level, and their usability in the previously defined software solutions is evaluated respectively. Finally, different file formats and their properties are compared.

4.1 Binary and ASCII

Files in general can be classified in two main categories, binary and ASCII files. ASCII file is a binary file, which consists of ASCII characters, or in other words 7-bit encodings stored in a byte. A binary file uses all 8 bits of a byte, thus allowing a full 256 bitstring patterns (whereas 7 bits of ASCII files only allow 128). What this means, is that ASCII file loses one bit on every byte, and generally spends more space when saving the information to retain the readability of it, whereas binary format strives for minimal space usage. (UMD, 2003) However, there are good methods for compressing ASCII files, and in a study made by Isenburg and Snoeyink, the difference between binary and ASCII sizes was reduced on average to 1.7% using gzip compression (Isenburg & Snoeyink, 2003). Savings of disc space in this magnitude have little importance on the process, as maps are mainly used locally.

The first version of the terminal area information should be gathered in the beginning of the project, as all implemented solutions should be tested comprehensively before use. However, slight modifications are made to the coordinates of the container terminal as project develops and more and more accurate measurements are made. Hence, information should be saved in a format that is readable and easy to process, and that is not tied into certain environment. Subsequently, even though binary formats offer better efficiency than ASCII files in loading, instead of having to use an editor to make changes to files without exception, being able to modify the files also with a simple text editor can be argued to be more profitable for the map data storing solution.

4.2 Images

Images are a tool to represent large amount of information where text is not sufficient, which is why they play an important role when planning a GUI. A simple two-dimen-

sional map itself is already one type of an image, although in nowadays container terminals it is possible to see even 3D-representations of the terminal. Container terminal GUI represents information from a large area, but also should be able to show smaller details when needed. Hence, scalability is a major factor on image format selection. Other affecting factors are, as described already in previous section, file size, data readability, modifiability and compatibility between different components.

Images can be divided into raster and vector graphics. They have a color space to represent their coloring in numeric form, and a defined structure to describe contents. Raster graphics consist of dots (pixels), which are saved in a matrix called bitmap, whereas vector graphics describe geometrical primitives, such as points, lines and polygons. (Ferilli, 2011, pp. 28-43) The difference of the mentioned methods is very well seen when picture is up-scaled. Where raster graphics become pixelated, vector graphics retain their shape, as seen in Figure 7 below.

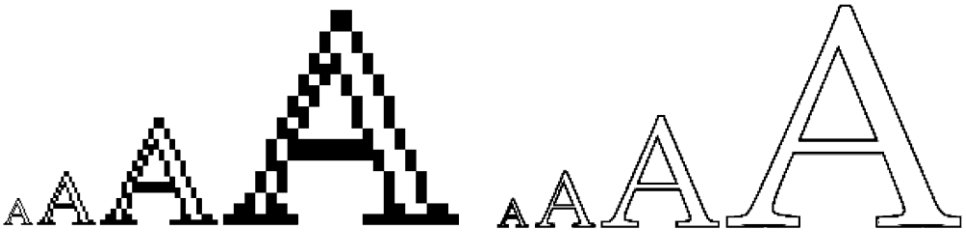


Figure 7: *Upscaling of raster (left) and vector graphics*

Raster images are usually used with photographs, as they would be very difficult to be shown in mathematical primitives. When talking of geometrical shapes and directions, as in zone, lane or building modeling in container terminals, using vector data is a natural choice. Raster data can, however, be used alongside of vector data, if more complex photos or pictures are wanted to be shown to user.

4.3 File format comparison

Based on above mentioned analysis, two advantageous qualities of the file format for projects handling map data are its text based nature and capability of representing vector data. This chapter focuses strongly on formats filling these two qualities. Example code used on each section defines fictional values for a terminal, defining an identification number, name, origin coordinates, two obstacles and one lane. The purpose of presented examples is to demonstrate the properties of each format.

4.3.1 XML

Extensible Markup Language (XML) derives from Standard Generalized Markup Language (SGML). It was developed by XML Working Group, organized by World Wide

Web Consortium (W3C) in 1996, to fill the following ten design goals defined in the W3C Recommendation of XML 1.0:

1. *XML shall be straightforwardly usable over the Internet.*
2. *XML shall support a wide variety of applications.*
3. *XML shall be compatible with SGML.*
4. *It shall be easy to write programs which process XML documents.*
5. *The number of optional features in XML is to be kept to the absolute minimum, ideally zero.*
6. *XML documents should be human-legible and reasonably clear.*
7. *The XML design should be prepared quickly.*
8. *The design of XML shall be formal and concise.*
9. *XML documents shall be easy to create.*
10. *Terseness in XML markup is of minimal importance*. (W3C, 2008)

In other words, XML was developed as a markup language that serves general purpose. Hypertext Markup Language (HTML) has limitations concerning separation of content and presentation, which was also a motivation for the development of XML. Opposite to HTML, XML offers clear separation of data descriptions, data and their representation, and allows the definition of user-defined data tags, data types and structures. (Wilamowski & Irwin, 2011, pp. 56-6 - 56-10) XML can be validated with either XML Schema Definition Language (XSD) file or Document Type Definition file (DTD).

There are few reasons why you should select DTD over XML Schema, such as the need for overriding definitions easily, or if backwards compatibility is an issue (Gulbransen, 2002, pp. 50-53). However, DTD does not allow inheritance, scoping or datatyping, and unlike XML Schema, DTD documents do not follow XML syntax. Due to these limitations DTD documents are not discussed further in this document.

As mentioned previously, XML Schemas are themselves well-formed[5] XML documents. This is a considerable advantage, as same XML aware applications that are used for XML documents can be used to parse and process XML Schemas. In addition to simplicity of

[5] *Well-formed* XML-document is a document that follows correct XML syntax specified in (W3C, 2008). *Valid* XML document is *well-formed* and conforms to a document type definition. (W3Schools, 2015)

document handling, datatyping allows XML Schemas to create constraints of elements in the XML document to be in a specific data format. (Gulbransen, 2002, pp. 15-16)

XML syntax is defined by World Wide Web Consortium (W3C, 2008). XML bases on named tags that allow attribute definitions. Tag names are encompassed between angle brackets (‘<’ and ‘>’). Attributes are defined in starting tag, separated from the name and each other by space character (one or more). Ending tag is defined with a slash character before the tag name. An example of the syntax simulating a definition of a container terminal with an origin and two separate zones is shown below:

Program 1: XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <name id=1>Container Terminal</name>
   <origin>
      <x>0.0</x>
      <y>0.0</y>
   </origin>
   <zones>
      <zone>
         <xMax>1.12</xMax>
         <xMin>1.11</xMin>
         <yMax>1.22</yMax>
         <yMin>1.21</yMin>
      </zone>
      <zone>
         <xMax>2.21</xMax>
         <xMin>2.11</xMin>
         <yMax>2.22</yMax>
         <yMin>2.12</yMin>
      </zone>
   </zones>
</root>
```

XML is also used as a base for further development of file types. The current standard for vector graphics defined by W3C is Scalable Vector Graphics (SVG). SVG standard was created in 1999 and it is currently widely supported by major web browsers, excluding Internet Explorer. SVG defines basic mathematical shapes (rectangle, circle, ellipse, line, polyline and polygon), text and graphical elements. SVG also supports animation definitions and added metadata. (Ferilli, 2011, pp. 43-45) All SVG files are pure XML, meaning that SVG file is also a well-formed XML-file. Subsequently, same XML parsers that are used for self-defined XML files can be used to process SVG files.

Also other XML based data formats have been developed, for example geography markup language (GML). GML is defined in the Open Geospatial Consortium (OGC) to express geographical features. (OGC, 2015) Later on Google developed keyhole markup language (KML), which derives elements from GML 2.1.2. KML was originally created to represent geographical data in Google Earth, but is currently used more widely in geographical information system (GIS) applications. (OGC, 2015) Geographical information

systems, as well as data types and structures used in them are more accurately described in following section.

4.3.2 GIS

ESRI (ESRI, 2015) describes geographical information systems as follows: *"GIS software is designed to capture, manage, analyze, and display all forms of geographically referenced information. GIS allows us to view, understand, question, interpret, and visualize our world in ways that reveal relationships, patterns, and trends in the form of maps, globes, reports, and charts."* GIS is a very wide description of systems that encompass information about locations on the earth and attribute data that relates to defined locations.

GIS data consists of both vector and raster data. Vector data is divided into three types: polygons, lines/arcs and points. Points are most commonly used to represent discrete data, points of interest. Linear features, such as roads, are represented with lines and arcs. Polygons are commonly used to represent areas and geographic features. Raster data (grid data) can be used to represent surfaces. Raster data can represent for example temperature and elevation measurements (Morais, 2000)

4.3.3 JSON

JSON, or JavaScript Object Notation is a data-interchange format basing on JavaScript programming language, although not being dependent on it. JSON was developed by Douglas Crockford and its structure is defined in the standard ECMA-404 (ECMA, 2013). JSON was formed basing on the object literals of JavaScript programming language – which is also known as ECMAScript. The definition of JSON and its core functions are defined in ECMA-262 standard (ECMA, 2015, pp. 469-475). Most notable for basic usage of JSON are functions *parse()* and *stringify(),* that allow altering between ASCII-form and JSON data format.

In the standard ECMA-404 JSON is defined as *"a text format that facilitates structured data interchange between all programming languages"* (ECMA, 2013, p. ii). In the same document it is also described that due to the simplicity of JSON notation, it is unlikely that it will ever change, making JSON a very stable notation by nature. It is to be noted, though, that JSON does not have a direct support for cyclic graphs and it is not intended to be used with applications that require binary data. (ECMA, 2013)

JSON object consists of unordered name-value pairs encompassed between curly brackets (‘{‘ and ‘}’). Name and value are separated from each other with colon (‘:’), and name-value pairs from each other with comma (‘,’) (see program2 below). Name-field is a string of text and value can be string, number, object, Boolean value, null or an array of previously mentioned values. (ECMA, 2013)

Program 2: JSON example

```
{
	"id": 1,
	"name": "Container Terminal",
	"origin":{
		"x": 0.0,
		"y": 0.0
	},
	"zones": [
		{
			"xMin": 1.11,
			"yMin": 1.21,
			"xMax": 1.12,
			"yMax": 1.22
		},
		{
			"xMin": 2.11,
			"yMin": 2.12,
			"xMax": 2.21,
			"yMax": 2.22
		}
	]
}
```

JSON bases on universal data structures, and as mentioned before, is supported by all major programming language. JSON is often compared to XML due to their similarities, although when compared *as is*, JSON is more lightweight solution. Similarly to XML, it is also possible to define a JSON Schema to create constraints to the structure of the JSON document. Similarly to XML Schema, also JSON Schema follows the structure of JSON data. (Internet Engineering Task Force, 2013)

Similarly to XML, JSON has been used to create geographical markup languages, such as GeoJSON. GeoJSON is described to be a format for encoding a variety of geographic data structures. Loyal to the GIS information structure, GeoJSON describes Point, LineString, Polygon, MultiPoint, MultiLineString, and MultiPolygon types, but allows additional properties be added into geometries using Feature-objects.

4.3.4 AutoCAD files

Computer-aided design (CAD) files are generally used in plan designing or drafting. Autodesk, Inc. has created a proprietary AutoCAD Drawing (DWG) format to represent geometric information for technical tasks, such as architecture, industrial production and prototyping. Specification of DWG format was never made public, so Autodesk published another format, AutoCAD Drawing Interchange File (DXF), which serves the same purpose. The latter is often preferred, as there are many open-source programs that can operate with DXF files, and thus purchasing a license for AutoCAD software is not necessarily required. (Fernández Caramés, 2012)

When a container terminal is being planned and implemented, CAD files are a *de facto* standard on the terminal area designs. These files can be used in one form or another to aid in the forming of locational data for either GUI representation or control systems for vehicle control. The structure of DWG-files, as mentioned above, is not public information, so only DXF-files can be analyzed further. DXF-files following the newest standard of 2014 consist of following 6 sections:

1. *HEADER*
2. *CLASSES*
3. *TABLES*
4. *BLOCKS*
5. *ENTITIES*
6. *OBJECTS* (AutoDesk, n.d.)

HEADER section defines general information of the drawing, such as system variables. CLASSES section saves application-defined classes that are used later in BLOCKS, ENTITIES and OBJECTS. TABLES section encompasses several different symbol tables that are used to describe the information saved in the document. BLOCKS section contains block definition and drawing entities, and ENTITIES section again refers to blocks, itself containing graphical objects. OBJECTS section contains all nongraphical information of the drawing. (AutoDesk, n.d.) Due to the complexity of the DXF file structure[6], it is not practical to present an example of this file type in this document. However, whilst DXF files themselves are relatively unreadable as text, due to the popularity of them, there are several tools to process and/or to convert them to other formats. This makes it possible to use AutoCAD or similar program to create the plan of the terminal, and later process the information into another format for further use.

4.3.5 Summary

Container terminals often rely heavily on AutoCAD files when they are planned, but AutoCAD files (DWG/DXF) themselves are very unclear representation of data. If AutoCAD files are to be used, any modifications to the files require an editor, unless file is transformed into another, more readable format. However, efficient use of AutoCAD files can reduce the iterativeness of the map creation process, as most of the required locational data is often already available in these models.

GIS information in general is more suitable for more wide area map information handling than describing a specific container terminal area. This is due to the fact that GIS data does not support many basic geometric formats, such as circles, ellipses or rectangles.

[6] Drawing of two zones as described in examples of both JSON and XML representation renders to over 10 000 lines of text when it is saved into DXF (2013) format.

When comparing a representation of circle translated into SVG or GIS formats, this difference is seen clearly. SVG allows to describe the form as a circle or as an ellipse, using geometric information of its location and radiuses:

```
<ellipse stroke="rgb(0,127,0)" rx="1.15885" ry="1.15885" transform="translate(1.15885,1.15885) rotate(0.00)"/>
```

On the contrary, GIS formats like GML or GeoJSON break the perimeter of the ellipse into points, and represent it as a group of lines (see Appendix 1 and Appendix 2). The transformations to each representation are made from the same exact model, using software called FME by Safe Software Inc (Safe, 2015).

The most usable markup languages for saving locational information for described usage are XML and JSON. If data structure is created by hand, the selection between XML and JSON can be argued to favor either side. If structure is generated by another method, the availability of tools suited for the application make the most difference in the proficiency of development.

5. SOLUTIONS OF KALMAR

Kalmar both offers and uses several solutions utilizing data to represent locations in the terminal area. Often this information is also used in applications visualizing the data for user in some form of GUI. The GUI can be used to just edit data, especially when data is created for control system, or as a part of a monitoring or control system. Currently these solutions rely heavily on the use of XML. Format of the documents is defined according to the needs of the application. In this chapter, some solutions of Kalmar are inspected as an example of currently utilized solutions. An abstract level description of these solutions will be given, and software usage and functionality are described in general level.

5.1 Kalmap

Kalmar offers a solution for steering Kalmar RTG's, called SmartRail. SmartRail saves the data of RTG block in XML form. The older software that handled RTG-related map data creation, Kalmap, is now a legacy software. It was officially used in 2000-2013, but is still used for many terminals, as the terminal architecture utilizes older components. Kalmap saves data in ASCII-form, practically as lists of coordinates. Each coordinate is separated from each other with a space-character, and each list represents either a single block or a group of blocks. This means that the data is readable, although relatively difficult to fathom. Kalmap saves coordinates of container blocks in .map format. To create the files, coordinates of the containers in two adjacent corners of each block and the spacing between them are given to software, and the center point of each container in the block and its rotation are calculated by the software. Rail positions are then added according to measurements done in the terminal. (Kalmar, 2015)

However, data collected with this method was never used in system level user interface implementations including visualization of the area, but only in the graphical user interface of the planning software itself (see Figure 8). As seen from the picture, visualization is done very simply by adding squares to represent containers.

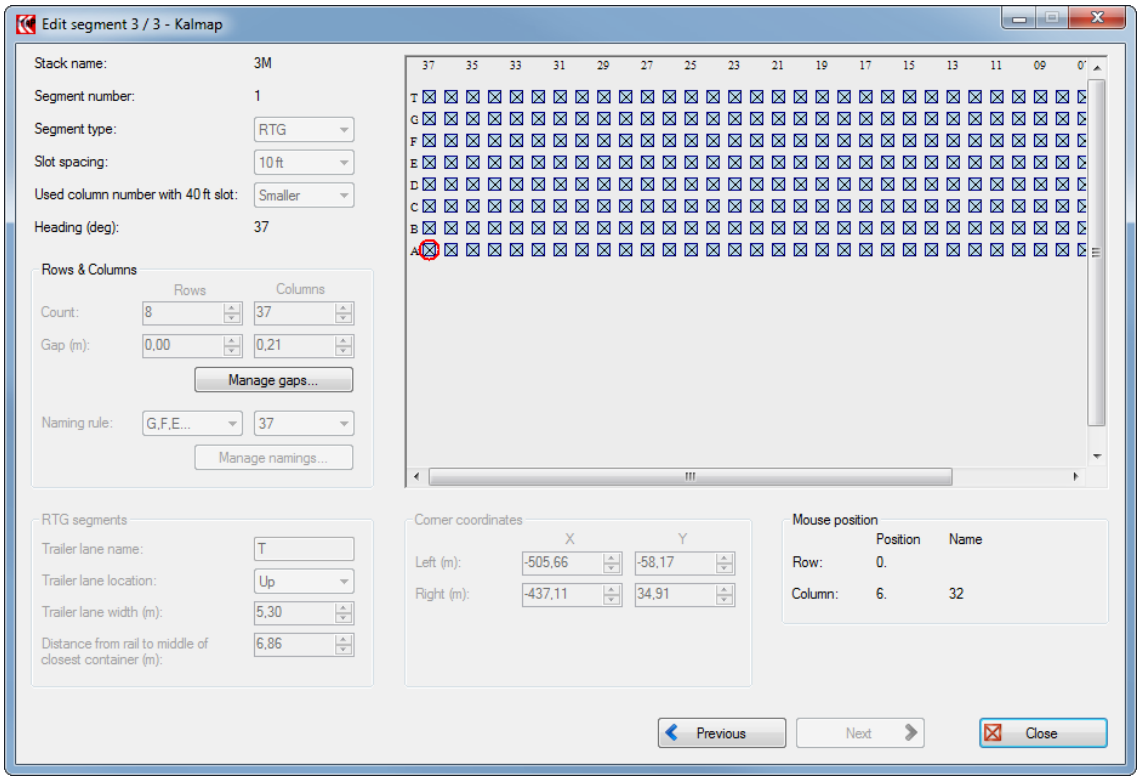


Figure 8: *Kalmap block visualization*

Because the control computer used in these systems is no longer produced, an update for these systems was required – and thus there is no longer a need for such implementation. In the updated system a different computer is used and data is converted to XML form, which makes it a lot more readable. However, even though data type has changed, control logic has been left unchanged, and original data is mostly still produced with Kalmap software.

5.2 MTS

MTS (Marine Telematics) is an application created to present real-time data from equipment and other sources to deduce activity at a marine terminals. Application offers various surveillance options for manual equipment. Software does not function as TOS or FMS, but offers user a graphical user interface with tools for collecting data from various operations on the field (see Figure 9 below). The application was created with C++, using an environment called Embarcadero C++ Builder, by the team in San Jose.

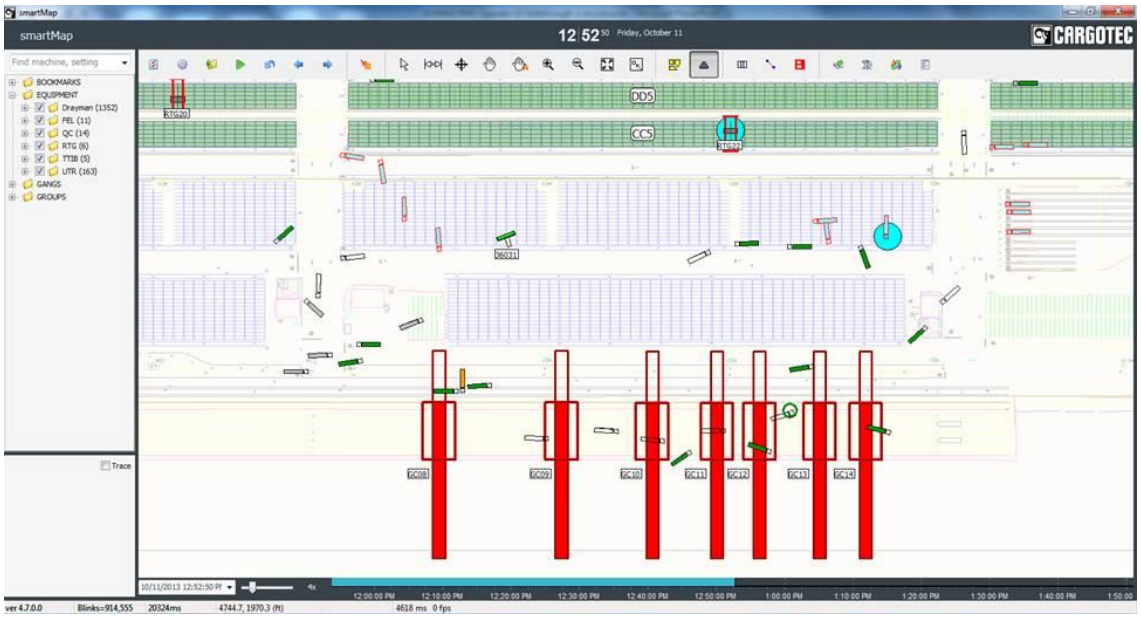


Figure 9: *MTS SmartMap, zoomed view*

Software uses Cartesian coordinate system in feet, converting all external information into internal format. Positions in the field are represented using a WMF (Windows meta-file format), which is generated from an Autocad-model. However, this model is not used for functional purposes, but merely as a background image. In the process of exporting DXF-file from Autocad into WMF, some accuracy is reported to be lost. Any business logic that requires locational information are defined as polygons and overlaid on the map. These polygons and their attributes are defined using a proprietary tool. Vehicles are shown with rather simple representations according to the vehicle type, as seen in the picture above.

5.3 UniQ GUI, FV

Fleetview is a graphical user interface mainly intended for monitoring purposes. Software is offered as a part of Kalmar's SmartFleet package. Fleetview offers several views for the operator, from machine specific views to real-time map representation of the container terminal. When required, software can also be used to send commands to fleet, which can be compared to FMS functionality. Software was created by Kalmar team in Tampere, Finland, using a relatively old version of QT Framework as a development environment. Development was done mainly with C++.

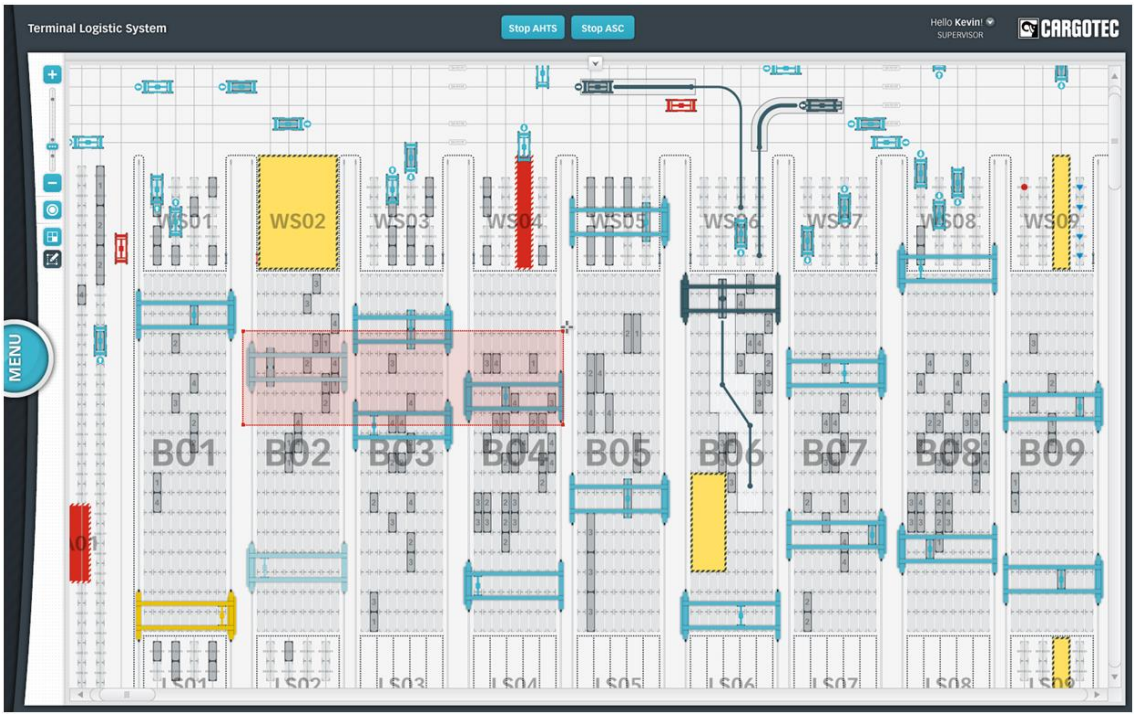


Figure 10: *Fleetview, Fleet overview*

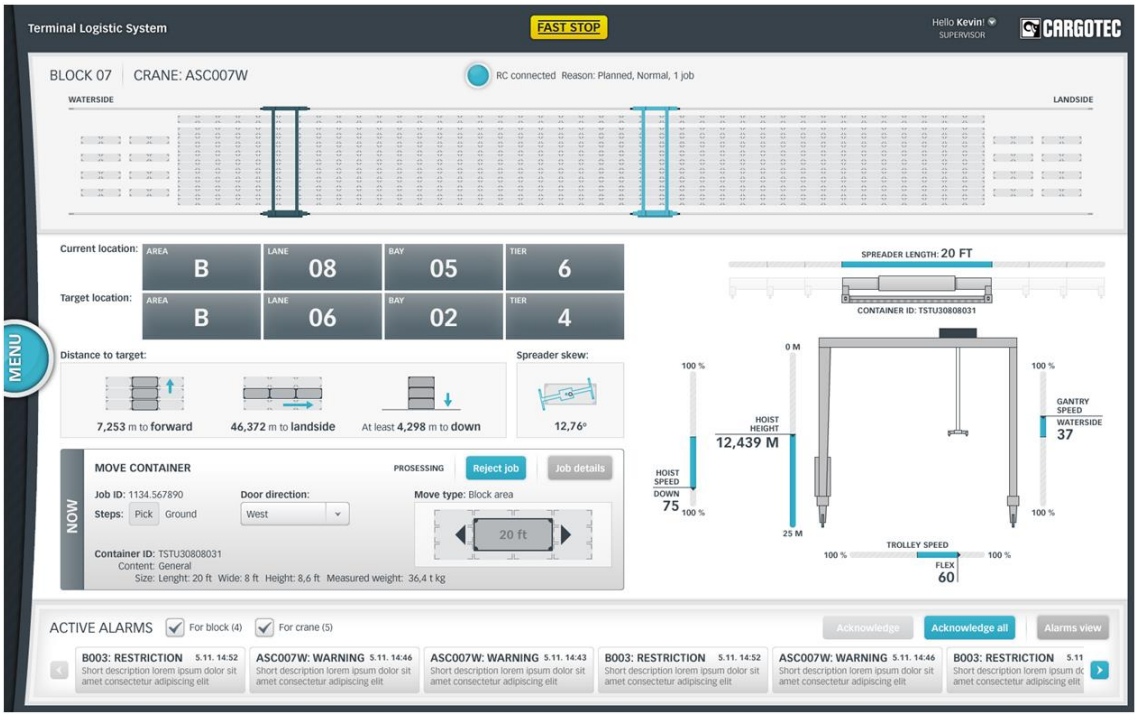


Figure 11: *Fleetview, Operations overview*

The map that is shown in Fleet overview (see Figure 10) is built together in several phases. An AutoCAD file of the terminal is used as a base, from where points of interests are collected using a special AutoCAD plugin. This plugin saves a selected point in separate Microsoft Excel file. When all of the required points are collected in Excel file, they are

transformed into XML file format, either by hand or using a separate, self-written script. This file is later parsed into the image shown in Fleetview.

5.4 HTCS Layout Editor

HTCS Layout Editor is a program that works as a tool that allows user to form lanes and allowed turns for shuttle and straddle carriers. Software itself was written in Java by ACT (Advanced Cargo Shipment) branch of Kalmar in Netherlands. Software gives a tool to plan the apron area of an automated container terminal. As told above, software is used to plan allowed routes for shuttle or straddle carriers that handle movements in the apron area. Correct locations of the lanes are physically measured in the container terminal, after which the places are manually recorded to the software. Later, user has to define possible curves that the vehicle can drive. Software is after that able to calculate fitting curves between each of the lanes, and also flatten the curves, if necessary. Steepening curves is not allowed, as it would make it possible to cross the physical limits of vehicle's turning radius. Software would also allow planning complete routes instead of curves and straights, but in case of automated shuttle or straddle carriers, it is desirable to have freedom of movements.

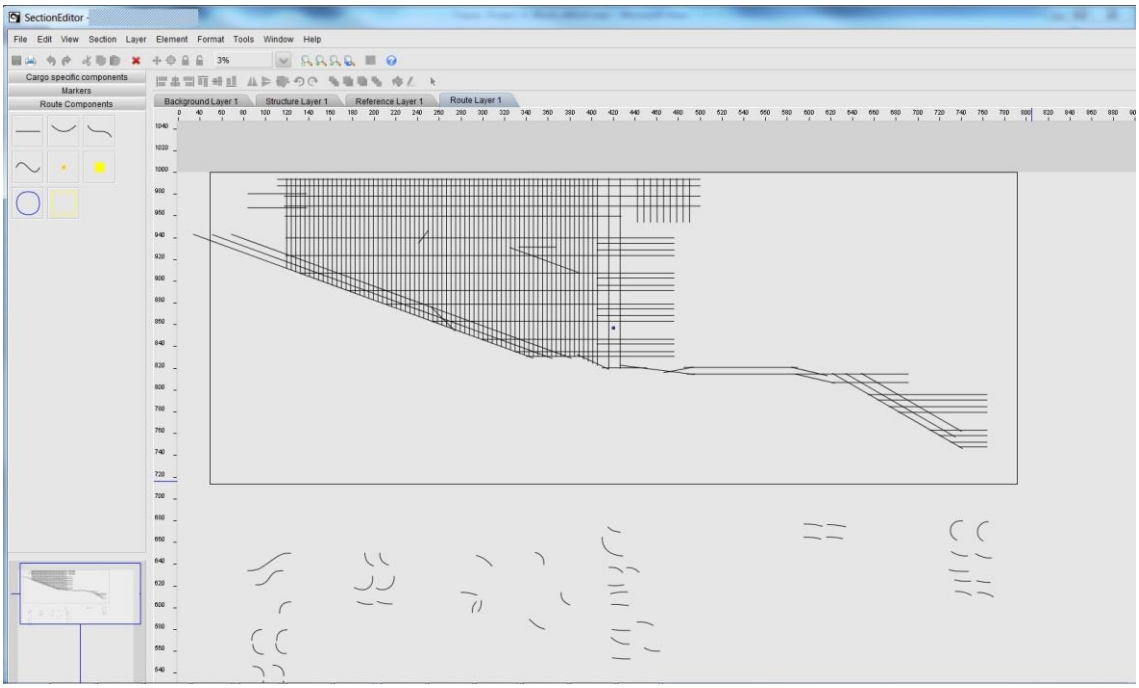


Figure 12: *HTCS Layout Editor*

As all the measurements are taken from actual world, automating the drawing process is not really possible. Software allows, though, using an AutoCAD model file (interchange format, DXF) as a background while drawing. This gives a visual reference for the locations of the lanes. Once lanes and curves are configured, they can be exported into a file called layout.export. The generated layout file is written in XML, and usable as is by other software.

5.5 ASCCS Layout Editor

ASCCS stands for Automated Stacking Crane Control System. ASCCS is a control system that handles all ASC functionality and information regarded to it. In other words, control system implements a specific database for each block stack. This database holds information about each possible container position of each container size, all obstacles in the area, maintenance positions, waiting positions, other defined areas, reefer position and allowed movements that belong to the scope of the concerned block stack. Information is effectively used to handle ASC movements within the block.

ASCCS Layout Editor is software that is used to save all the information mentioned above. The tool offers simple but relatively efficient interface for making modifications to the block data, as seen in Figure 13 below. For example, program offers tools specified for repetitive adding of container position with defined spacing, and possibility to create gaps between them without breaking the layout, which is especially handy when adding reefer positions in the block.

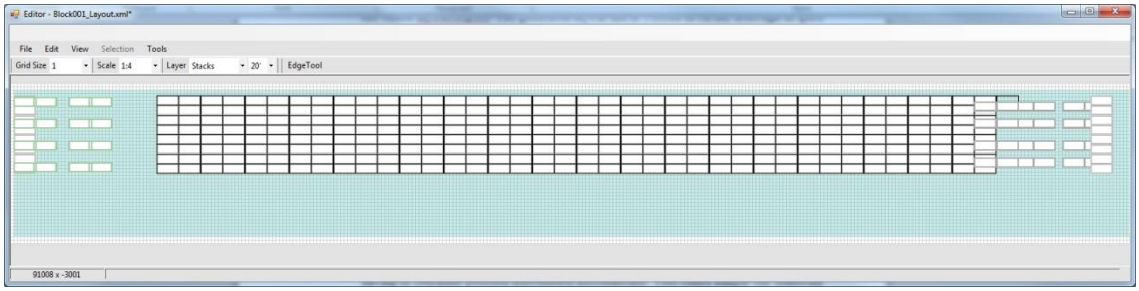


Figure 13: *ASCCS Layout Editor*

When block layout has been created, it is generated into several XML files, covering all above-mentioned data. Data can then be utilized in ASCCS to perform automatic movements of ASCs on site. Current data format consists of information about areas, cranes, layout and predefined jobs, where cranes and areas files define the types of cranes used on the stack, and the areas defined within the block. Layout file contains most of the information of the block, including container positions and allowed movements between positions.

6. ALTERNATIVE SOLUTIONS

Solutions regarding forming a map from an area of interest are not limited in harbors. Different applications naturally have different requirements, and thus a solution from another environment is rarely fully applicable to harbor environment. Nevertheless, similarities in layout creation methods can be recognized in many areas. Some Flexible Manufacturing Systems (FMS) in industrial environment, that use Automated Guided Vehicles (AGV) to move pallets or materials, are for example very comparable to harbor environment by their functionality (Jawahar;Aravindan;Ponnambalam;& Suresh, 1998). Freight terminal applications on airports or on rail stations are good examples of similar processes as well (Günther & Kim, 2005, s. XI). Some differences occur in both, as harbor systems spread out in large areas outdoors, while FMSs utilizing AGVs strictly work indoors. Also the used AGV type is very different, as harbors utilize flat vehicles designed for container carrying, whereas AGVs used in FMS can have a multitude of different designs. The most traditional industrial AGV has a close resemblance to forklifts.

AGV scheduling in FMS is similar to AGV system used in harbors. There is a positioning system to create a network of allowed movements for AGVs. If an object needs to be moved, closest free AGV will follow the route to reach for it, and handle required operations. In FMS systems, as in harbor systems, there is often an operator who supervises the actions via a remote interface. This interface can show in real time where AGVs move, where they are needed and so forth. The creation of this interface is a very similar process to harbor's corresponding system. In the following section, few selected different approaches from companies are presented. The latter section focuses on deliberation of other solutions that can be derived from the ideas in and outside of Kalmar.

6.1 Other companies' solutions

To create described user interface map, it is not often seen that the whole map would be created from scratch without using separate models. However, as described earlier, when creating more accurate models, as for the controlling of the vehicles, it is often required. Also, TBA has developed an own, proprietary tool to create the map used in their fleet management system TEAMS instead of using models. TBA is a known provider for harbor equipment and software. The software allows replaying situations on the field in 3D, which is a very handy functionality in situations, where 2D representation is not sufficient – but also makes the data structure requirements more complicated. Unfortunately, the information structure, format and methods used in the process are not revealed by the company, so solution cannot be efficiently compared to Kalmar solutions.

Several FMSs utilizing AGVs work using CAD models as a base. Examples of these are a solution called Q-CAN from Savant and E'zmap by Egemin. Q-CAN AGV System solution name comes from “Quick Configurable AutomatioN”. The solution bases on Savant’s software that allows user to import a CAD model of the environment, after which allowed routes can be added directly into it. On hardware level the system utilizes Savant’s magnetic sticker based navigation for the AGVs. (Savant, 2012) Egemin E'zmap functions on a basis of an “AutoCAD-based Guidepath Design” concept. What the concept means, is that the virtual path given for AGVs can be altered and expanded using AutoCAD. E’zmap provides the tools to verify the consistency of the Guidepath and visualize potential problem areas. It also allows user to modify their layout if needed. However, to fully modify the map file, AutoCAD software is required. (Verhoeven, 2015)

A very different approach has been taken by Tideworks. Tideworks offers solutions for container terminals. Their user interface is designed to visualize the movements tracked by any other third-party system such as fleet management system. Visualization is done completely in 3D, using a separate graphics engine from gaming industry. Implementation of the map itself is made utilizing Adobe Flex framework, to allow cross-platform functionality (Adobe, 2015). Tideworks offer their visibility tool, going by name TerminalView 2.0, as a separate component that offers standard application programming interface (API) for connectivity between applications. (WorldCargo, 2013)

6.2 Other methods

Some implementations of FMSs, as well as for example different mining solutions (Carlson, 2014; Huber & Vandapel, 2003; Xing, Peihuang, Jun, Xiaoming, & Dunbing, 2013) apply so called simultaneous localization and mapping (SLAM) method, either alone or in combination with other solutions. SLAM is a widely known Kalman filter approach for localization. The method bases on attached scanners on the vehicle, which scan the surrounding area, and while the vehicle moves, create a virtual representation of the area. However, if the environment changes repeatedly, this method is prone to errors, in addition to which, the process of mapping itself is relatively slow. Also, method poorly recognizes different elements during mapping – and due to these reasons, SLAM method is not applicable for data collection in container terminals. However, similar environment surveillance is needed to recognize unexpected objects around the vehicle in real-time while operating with the vehicles.

Model based solutions are more adequate in container terminal areas, as the basic structure of the area is static, and most often only containers are moved. As mentioned before, AutoCAD files are very often used to create a model of the surroundings. These files can further on be used in various ways. In some occasions, AutoCAD files are not available, and then data collection method is prone to be more manual. However, when models of the area are available, information can be collected in various ways. Kalmar solutions already described a possibility of using the model as a background image for a layout

editor. This is highly visual method that helps user to keep track on where in the area the changes happen. Other method described on Fleetview's case is where points of interest are selected and then transferred into the required structure. These both methods, though, require a lot of iterative work with an already existing model.

Another possibility would be to parse information straight from AutoCAD file. This method is not currently used by Kalmar and would require additional efforts in keeping AutoCAD models of the areas updated. Also, the accuracy of the models would need to be refined to match the requirements set for the end solution. AutoCAD offers instructions for creating an own reader and writer for the interchange format files (DXF) (Autodesk, n.d.). Another option is to use an already made, open source parser for parsing AutoCAD files. If an open source parser is used, testing is required to make sure parser supports all required functionality and does not distort the accuracy of the model. Using either self-made or open source parser would allow reading the data and saving it into database – or practically any other form. Saving data to another file type than DXF can be beneficial if small changes are wanted to be done, as DXF file requires a specific program to edit files, or alternatively parsing of the whole file back and forth to another form.

Instead of handling the parsing of an AutoCAD file in a separate program, this step can be skipped by using a converter to save data into different form. Many programs exist with wide variety of possible formats. Using an already written and tested program can reduce the cost of the process, as developing is already done, as well as testing to some extent, although suitability for the current problem is to be defined. When selecting the end format, as described in Chapter 4, there are several formats to consider. Out of these, this thesis focuses mostly on XML and JSON. Using a GIS models is also excluded, as it would become difficult, if round figures are needed to be modeled, as summarized in Section 4.3.5.

Autodesk has also published a developer's guide for their .NET API (Autodesk, 2010). The AutoCAD .NET API covers the usage of Microsoft Visual Studio 2008 as an environment, and the languages Microsoft Visual Basic and Microsoft Visual C#. This allows user to use .NET framework to extract, create or modify objects in drawing files. In other words, mentioned languages can be used to automate process of collecting data from AutoCAD file to database.

7. SOLUTION

As divided previously, control systems and systems that are used in end solutions as graphical user interfaces require different levels of accuracy. The structures required for control system are stricter and usually encompass a smaller entity than the ones used for monitoring purposes. This is seen in Kalmar's solutions, where for example ASCCS layout editor is used to handle ASC specific location data and HTCS layout editor creates models for Shuttle or Straddle carriers. Control system information is also possibly only shown on the editor software where data is modified, and thus serves a very different purpose than that of GUIs of end products, which are shown to user while actual operation. When inspected, current control system solutions of Kalmar are well developed in their fields of focus. Other than the improvement of the software themselves, a focus could be set towards unifying solution that would serve as a middleware between the different software. What comes to GUI creation process, current methods used by Kalmar are hardly automated. AutoCAD is used to some extent, but the actual work in data model creation is done by people.

The final solution presented in this thesis is to reduce data in AutoCAD model to represent only the information needed in the GUI solution, divide it into accordingly named layers, and transfer it into SVG format preserving the coordinate system that is used in the original model. The selected software for this is called FME from Safe Inc. The reason FME was selected is that it was the only software of the many tested that could process big enough amount of data into SVG format with a reasonable amount of processing capacity while still preserving the layer structure.

Several other programs were tested, which are all listed in following section. In the next section, also the selection process and testing of the programs is described more closely. Later on, Section 7.2 will describe the final selected method, and latter sections will give a closer look into its testing, performance and price.

7.1 Selection

There are many programs that are capable of transforming data between DXF and SVG formats. In this thesis, programs listed below were tested:

- *FME* from Safe Software Inc.
- *Inkscape,* free and open source software
- *QCAD,* open source software
- *Illustrator CC* from Adobe Systems
- *Easy CAD to SVG Converter* from Benzsoft Corporation
- *DWG to SVG Converter MX* from DWG TOOL studio

- *Any DWG to SVG Converter* from AnyDWG Software, Inc
- Command line tools named *dxf2svg-inkscape* from Matthew Squires

One additional software was tested, but it is excluded from this thesis without further mentioning, as its installation process entailed advertising malware. Selection of the final solution from all programs was done with ad hoc testing, originally with relatively detailed AutoCAD file, to test both, the performance considerations of the software, and its layer handling methods. Selection of the software was eventually very straightforward, as most of the programs were not able to handle large files. Capability of handling large files is required as container terminals can have tens of ASC blocks surrounded by details.

Second important area in the testing was layer handling. SVG offers an internal structure for layers, with a tag "<g>". The abbreviation comes from the word "group", and like layers, it means a group of elements. Unfortunately, only few of the listed software make use of layers defined in AutoCAD, but destroy the structure and group objects according to own strategy. However, instead of grouping, another possibility would be to use colors as an identifier. Each drawn line has a HEX code to save the color, which is preserved from the original picture. Using predefined HEX codes for layer elements, same structure would apply. This method is, though, prone to errors and leads to more work adjusting the AutoCAD model.

7.2 Conversion process

This chapter describes the process of converting AutoCAD model to SVG model using FME by Safe Software Inc. The conversion process is described verbally and pictures are added for clarity. Conversion process happens in two phases; modifying the original AutoCAD model of the container terminal, and later using FME software to complete the transformation to SVG file format. Step-by-step actions required to be performed with FME are separated in user manual provided as an appendix (Appendix 3).

The original AutoCAD model is expected to implement much more details than is required in the end result. Often also unnecessary components are included in the model. Images below (Figure 14 and Figure 15) show examples of these two cases.

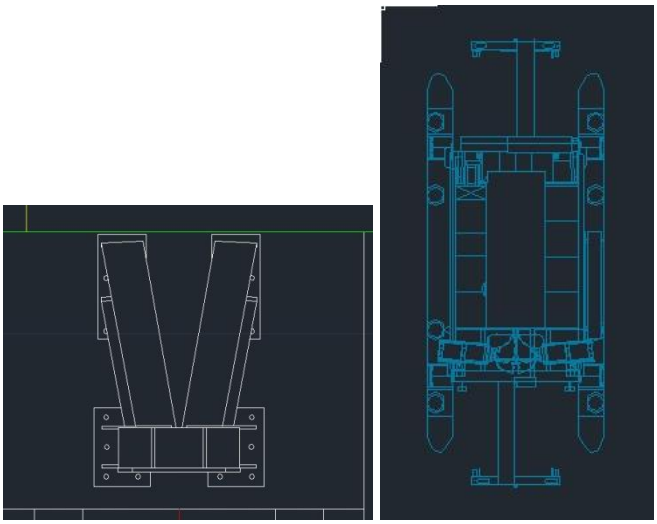


Figure 14: *Modification example, removable complex elements*

The two examples above represent objects that are not needed in the final presentation. The picture on the left presents a stopper in the end of ASC lane. As the stopper is inside a wider area of obstacle, it can be removed from the model. The main information from stopper is the outline of the surrounding obstacle, and possibly its height. The picture on the right is drawn as a figurative design of the vehicle. Components such as vehicles are not static elements in the container terminal area, and are thus not required in the end solution. If these unnecessary elements are desired to be kept in the original model for clarity reasons, another option is to add them in a new layer, which is later excluded from the transformation process in FME.

Similarly, as with the above examples, in Figure 15 below, an image of a laser pole is presented. For the GUI, main information needed of the laser pole is its outline, and all details inside it are irrelevant, and can thus be removed from the model. The original picture is on the left, and modified on the right.

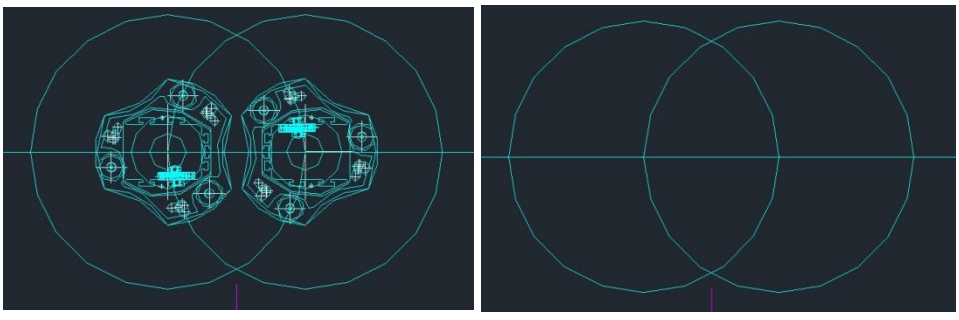


Figure 15: *Modification example, laser poles*

Depending on the model, a large amount of either unnecessarily accurate information or information outside the limits of the area of interest can be deleted from the model. The required area is also often smaller than the original model of the whole terminal. For

example the landside of the terminal is less used in terminal management solutions, as described in 2.1.3.

After the data is reduced to cover only the required information, layer naming should be done to separate different types of components from each other, as explained in 3.2. If more information is later on required on some layer, in layer properties more variables can be added. Also, FME software allows the naming of the layers in the conversion phase, but it is possible to import the layer name of AutoCAD model, and thus either of the methods can be used effectively.

Next, and the last step to do with modifying AutoCAD file, is to add the origin point (x=0, y=0, z=0) to the model. It is assumed that the origin point is set so that all the locations in the area are in the positive side of the coordinate axes. This step, although slightly unusual, prevents software from scaling the coordinates down to only the image area, and preserves the exact coordinates that are used in AutoCAD model. Point can be added in extra layer and deleted from the model after the conversion is done with FME.

In FME software, to achieve wanted transformation, reader and writer must be selected accordingly. Also, precision setting of writer will need to be defined separately, if the container terminal is large, as more accurately described below in Section 7.3.2. Layers for the end result can then be manually modified, or added straight from the AutoCAD model. Layers between AutoCAD and SVG representations are then connected to each other with lines in the main window of the software. These actions are instructed in detail in appendix 3. After this, conversion can be done and SVG file will be created in the location specified by the user.

7.3 Testing and evaluation

Testing of the transformation was divided into three sections; functionality, accuracy and performance. Firstly, a layer was added into the used model, where lines were drawn inside the container terminal area, to validate the consistency of original and resulting coordinates. Secondly, when functional configuration was found, accuracy was tested by selecting an actual object from the original model and comparing resulting coordinates with different precision settings. Finally, transformation with previously defined settings was done using a big model, to test if same method can be used for large terminals.

Used model is from a real, although relatively small container terminal. Functionality and accuracy testing were done with a greatly reduced file, in which only few key areas were left, all complex figures were deleted and layers reduced to very minimum. This was done to allow fast transformation – and to assure the efficiency of the testing. For performance testing, the main idea was to find out if software is capable of handling large amounts of data. For this test, the original model was left as is. This means that the model has detailed

components and areas that would in real case be left out, but for simulating the load of much larger, although simpler file, the model is adequate.

7.3.1 Functionality

Testing of the end results of the transformation was done firstly by adding separate control figures in the original model in a layer called “CONTROL_POINT”, and comparing resulting coordinates with the original. This method was used until a functional configuration was found. Added control points were three lines, from point (50, 950) to (50, 970), from (700, 1050) to (700, 1030) and from (600, 580) to (620, 580). Without adding the origin point to the model, coordinates were distorted rather much, and these three comparison lines were transformed into following:

```
<g id="CONTROL_POINT" >
<path    fme:autocad_layer="CONTROL_POINT"    stroke="rgb(255,127,0)"    d="M
688,485.496 l 0,-20 "/>
<path    fme:autocad_layer="CONTROL_POINT"    stroke="rgb(255,127,0)"    d="M
588,15.4964 l 20,0 "/>
<path    fme:autocad_layer="CONTROL_POINT"    stroke="rgb(255,127,0)"    d="M
38,385.496 l -7.10543e-015,20 "/>
</g>
```

When the origin point was added in the model, coordinates were not distorted, and all of the lines were as they were originally positioned in the model.

```
<g id="CONTROL_POINT" >
<path fme:autocad_layer="CONTROL_POINT" stroke="rgb(255,127,0)" d="M 700,1050
l 0,-20 "/>
<path fme:autocad_layer="CONTROL_POINT" stroke="rgb(255,127,0)" d="M 600,580 l
20,0 "/>
<path fme:autocad_layer="CONTROL_POINT" stroke="rgb(255,127,0)" d="M 50,950 l
-7.10543e-015,20 "/>
<path fme:autocad_layer="CONTROL_POINT" stroke="rgb(255,127,0)" d="M 0,0 m -
0.47599,-0.47599 l 0,0.95198 0.95198,0 0,-0.95198 z "/>
</g>
```

Only noticeable detail in this result is the transformation of -7.10543e-015 of the third line, where transformation should have been 0. This can be explained with imperfect setting of the line in AutoCAD which lead to rounding error. However, as the transformation in real life is some femtometers, the number is insignificant.

7.3.2 Accuracy

Accuracy was tested with a real rectangular object from the original model, chosen far from control lines. The corner points of the building in AutoCAD model were following; (644.5875, 868.9831), (644.5938, 791.3831), (634.5938, 791.3823), (634.5875, 868.9823). Same building in SVG format was modeled as follows:

```
<path fme:autocad_layer="BUILDING" stroke="rgb(255,255,0)" d="M
644.588,868.983 l 0.00627539,-77.6 M 634.588,868.982 l 10,0.000808684 M
634.594,791.382 l -0.00627539,77.6 M 644.594,791.383 l -10,-0.000808684 "/>
```

In the format, lines are modeled as line starting points marked with the letter “M” and transformation from that point, marked with letter “l”. By default, FME applies 6 significant digit precision. If terminal area is very large, rounding errors in the scale of millimeters can occur. Above shown building measurements use default parameters, and the building outline can be shown in four lines named as A, B, C and D below.

```
A = (644.588, 868.983), (644.59427539, 791.383)
B = (634.588, 868.982), (644.588, 868.982808684)
C = (634.594, 791.382), (634.58772461, 868.982)
D = (644.594, 791.383), (634.594, 791.281191316)
```

When these four lines are rounded in 6 digit precision, errors of one digit occur in third decimal:

```
A to D = (644.588, 868.983), (644.594, 791.383)
B to A = (634.588, 868.982), (644.588, 868.983)
C to B = (634.594, 791.382), (634.588, 868.982)
D to C = (644.594, 791.383), (634.594, 791.281)
```

In this case, as coordinate unit is meters, errors of one millimeter occur. Bigger areas than one kilometer can result errors in precision of centimeters. Ergo, if container area is very large, default precision may not be sufficient, and it has to be altered. Safe precision to use is 8, as it allows transformation with rounding errors of millimeter precision on terminals less than 100 kilometers in length. With precision setting 8, results of the same building are following.

```
<path fme:autocad_layer="BUILDING" stroke="rgb(255,255,0)" d="M
644.58753,868.98306 l 0.0062753884,-77.6 M 634.58753,868.98226 l
10,0.00080868369 M 634.5938,791.38226 l -0.0062753903,77.6 M
644.5938,791.38306 l -10,-0.00080868416 "/>
```

Rounded to 8 digits precision: Similarly, building can be defined in four lines, from A to D.

```
A to D = (644.58753, 868.98306), (644.5938053884, 791.38306)
B to A = (634.58753, 868.98226), (644.58753, 868.98306868369)
C to B = (634.5938, 791.38226), (634.5875246097, 868.98226)
D to C = (644.5938, 791.38306), (634.5938, 791.38225131584)
```

And when results are rounded in 8 significant digits, it can be seen that errors may occur in fifth decimal, which in this case means 0.01 millimeters.

```
A to D = (644.58753, 868.98306), (644.59381, 791.38306)
B to A = (634.58753, 868.98226), (644.58753, 868.98307)
C to B = (634.5938, 791.38226), (634.58752, 868.98226)
D to C = (644.5938, 791.38306), (634.5938, 791.38225)
```

7.3.3 Performance

Performance testing was done on a laptop computer with Intel® Core™ i7 processor i7-2620M and 8 GB of memory in Windows 7 Professional (64-bit) environment. The container terminal model was an original model which was used without reducing data from it. In comparison, with the greatly reduced model, which was used in functional and accuracy testing, FME created 19 987 features as an output, whereas with the original file without modifications the same amount was 829 081. FME was able to transform the file, and create an output in 8 minutes and 35.4 seconds without any issues. Maximum memory usage was 4.03 GB out of the available 7.72 GB, which is well in reasonable limits. With other tested solutions, memory usage proved to be an increasing problem when file size increased.

7.4 Pricing

FME Desktop software is sold by Safe Software Inc. For the usage defined in this thesis, the cheapest license called FME Professional is sufficient. After 30-day testing period, the price of the software, as requested from the company, is either fixed upon each license, or so called floating license, where cost is defined by the amount of concurrent users. One fixed license costs 2 250 USD. Floating license costs 6 400 USD for the first user, and 2 100 USD for each additional user. Included in the price are software updates and maintenance for the first year, after which they will cost an extra 20% of the standard list prices.

7.5 Summary

In actual projects, a project engineer's responsibility is to go through the original AutoCAD model of the container terminal, delete unneeded information, restructure the layers to match the wanted result, add origin point and go through the transformation in FME software. After these steps map data is available in SVG format and can be used in further solutions. FME software itself is relatively expensive, but also offers good licensing options.

In short, the only adequate solution of the tested programs to transform AutoCAD file into SVG format was FME. Transformation was proved to be functional and accurate. Also, a very large file was transformed to test the performance. Even when talking of large container terminals, realistic file size of the modified model will unlikely get as high as the mentioned file. If the container terminal is indeed this massive, more efficient data handling procedures are to be considered in general. Thus it can be concluded that the file transformation will not become an issue in this project.

8. CONCLUSION

In this document, an analysis was made to find a method to make the process of map data creation and handling more efficient. Solution was made considering the current and future direction of actions in Kalmar, a brand of Cargotec. Found and tested solution successfully utilizes a commercial software to transform an AutoCAD model into SVG format. The decision of the actual commissioning of this method will be made by company representatives. If this decision is made, plans will be required for future development.

When talking of control system related design, current solutions of the company rely on separate software handling separate areas of container terminal. In designer point of view this is quite unwanted, as the container terminal in its entirety is not perceived well. On the contrary, when making detailed designs, it is an advantage to focus on specific area at the time. These thoughts would support the idea of creating a combined view where specific areas can be selected and designed further on. This would allow user to have a clear idea on where the design is taking place without any background work.

When talking about the solution presented in this thesis, a positive addition is achieved from the fact that there is no middleware needed between AutoCAD model and end product. The transformations in between are often made for only singular use, and are thus less efficient methods than modifying an already existing model to match the need. Yet hypothetically, if the AutoCAD file is made accurate enough, extending the usage to creating map data also for control purposes could be possible. This, however, also requires adding data to the model, such as safety distances to obstacles, and also much more accurate set of data on the areas which are now handled with separate software.

What comes to the selected file type, SVG itself is an excellent format for future development, as the world is increasingly directed towards internet solutions. HTML5 introduced a new element specifically for SVG file format. Of the common web browsers, all newer versions[7] support SVG format with no modifications, and data can be added directly inside an <svg>-tag. This makes it possible to use the map in web solutions without the need of transforming the data again.

On the contrary to the positive effects, utilizing a proprietary software also brings some negative effects. FME is relatively expensive software, which in this case would be used relatively rarely, and only on single transformation. However, software license is paid only once – and thus, in the scale of Cargotec's turnover, the price of the software is insignificant.

[7] First versions to fully support <svg> tag are Chrome 4.0, Internet Explorer 9.0, Firefox 3.0, Safari 3.2 and Opera 10.1 (W3Schools, 2015)

WORKS CITED

Adobe. (2015). *Flex*. Retrieved October 6, 2015, from Adobe Systems Software Ireland Ltd. Web site: http://www.adobe.com/fi/products/flex.html

ArcGIS. (2015). *ArcGIS API for JavaScript*. Retrieved May 22, 2015, from ArcGIS for Developers: https://developers.arcgis.com/javascript/

Autodesk. (2010). *AutoCAD .NET Developer's Guide*. Retrieved from Autodesk web site: http://docs.autodesk.com/ACD/2010/ENU/AutoCAD%20.NET%20Developer's%20Guide/index.html?url=WS1a9193826455f5ff2566ffd511ff6f8c7ca-4875.htm

AutoDesk. (n.d.). *DXF File Structure*. Retrieved September 1, 2015, from Autodesk Web site: http://www.autodesk.com/techpubs/autocad/acadr14/dxf/dxf_file_structure_al_u05_b.htm

Autodesk. (n.d.). *Writing a DXF Interface Program*. Retrieved from Drawing Interchange File Formats: http://www.autodesk.com/techpubs/autocad/acad2000/dxf/writing_a_dxf_interface_program_dxf_aa.htm

Bing. (2015). *Bing Maps API*. Retrieved May 22, 2015, from Microsoft: http://www.microsoft.com/maps/choose-your-bing-maps-API.aspx

Brinkmann, B. (2011). Operations Systems of Container Terminals: A Compendious Overview. In J. W. Böse, *Handbook of Terminal Planning* (pp. 25-39). Oklahoma, USA: Springer.

Cargotec Corporation. (2015, May 22). *Kalmar to deliver two additional automatic stacking cranes for DP World Brisbane.* Retrieved June 2, 2015, from Kalmarglobal: http://www.kalmarglobal.co.uk/newsroom/press_releases/2015/kalmar-to-deliver-two-additional-automatic-stacking-cranes-for-dp-world-brisbane-/

Carlson. (2014). *Site Management & 3D Positioning Solutions for Mining, Landflls, Dredging & Earth Moving.* Retrieved October 6, 2015, from Carlson Software Web site: http://www.carlsonsw.com/wordpress/wp-content/uploads/2014/02/MC-Catalog-Feb2014-Final-web.pdf

CST. (2012). *World Container Fleet Overview*. Retrieved August 7, 2015, from Container Services International Web site: https://web.archive.org/web/20150718151432/http:/www.csiu.co/resources-and-links/world-container-fleet

Duinkerken, M. B., Evers, J. J., & Ottjes, J. A. (2002). Improving Quay Transport on Automated Container Terminals. *IASTED International Conference Applied Simulation and Modelling (ASM 2002).* Delft, Netherlands: Delft University of Technology.

ECMA. (2013). *Standard ECMA-404, The JSON Interchange Format.* Geneve: Ecma International 2013.

ECMA. (2015). *Standard ECMA-262, 6th Edition, ECMAScript® 2015 Language Specification.* Geneva, Switzerland: ECMA International 2015.

Esoware B.V. (2011). Flexible TOS Software for Any Budget and a Wide Range of Applications. *Port Technology International, 50th edition,* p. 88. Retrieved June 17, 2015, from Port Technology website: http://www.porttechnology.org/technical_papers/flexible_tos_software_for_any_budget_and_a_wide_range_of_applications/

ESRI. (2015). *What is GIS?* Retrieved August 31, 2015, from Esri Web site: http://www.esri.com/what-is-gis/howgisworks

Farlex. (2015). *Control Systems*. Retrieved July 16, 2015, from The Free Dictionary by Farlex Inc: http://encyclopedia2.thefreedictionary.com/Control+systems

Ferilli, S. (2011). *Automatic Digital Document Processing and Management - Problems, Algorithms and Techniques.* New York, USA: Springer.

Fernández Caramés, C. (2012). *Técnicas de navegación para un robot móvil utilizando sistemas de razonamiento espacial.* Salamanca, Spain: Universidad de Salamanca.

Fleetio. (2015). Retrieved July 24, 2015, from Fleetio Web site: https://www.fleetio.com/

GDV. (2015, August 7). Container Handbook, Cargo loss prevention information from German marine insurers. Berlin, Germany.

GISCloud. (2015). *Fleet Management*. Retrieved July 24, 2015, from GIS Cloud Web site: http://www.giscloud.com/apps/fleet-management

Google. (2015). *Google Maps API.* Retrieved May 22, 2015, from Google: https://developers.google.com/maps/

Gottwald. (n.d.). *Automated Container Transport - Proven Technology from Gottwald.* Düsseldorf: Gottwald Port Technology GmbH.

Gulbransen, D. (2002). *Special Edition Using XML Schema.* USA: Que Publishing.

Gungor, V. C., Natalizio, E., Pace, P., & Avallone, S. (2008). *Challences and Issues in Designing Architectures and Protocols for Wireless Mesh Networks.* Georgia, USA: Springer.

Günther, H.-O., & Kim, K. H. (2005). *Container Terminals and Automated Transport Systems.* New York, USA: Springer.

Here Maps. (2015). *Consumer Mapping*. Retrieved May 22, 2015, from HERE: https://developer.here.com/web-experiences

HPH. (2015). *History*. Retrieved June 15, 2015, from ECT website: http://www.ect.nl/nl/node/484

Huber, D. F., & Vandapel, N. (2003). Automatic 3D underground mine mapping. *The 4th International Conference on Field and Service Robotics* (pp. 14-16). Pittsburgh, USA: The Robotics Institute, Carnegie Mellon University.

Hughes, A. (2008, August 13). *Plugging in*. Retrieved May 27, 2015, from Port Strategy - insight for senior port executives: http://www.portstrategy.com/news101/port-operations/port-services/reefer_article

Intel Corporation. (2015). *Intelligent Fleet Management.* USA: Intel Corporation.

Internet Engineering Task Force. (2013, August 3). *JSON Schema: core definitions and terminology*. Retrieved September 1, 2015, from JSON Schema Web site: http://json-schema.org/latest/json-schema-core.html#anchor23

Isenburg, M., & Snoeyink, J. (2003). *Binary Compression Rates for ASCII Formats.* North Carolina: University of North Carolina. Retrieved from https://www.cs.unc.edu/~isenburg/papers/is-bcraf-03.pdf

Jawahar, N., Aravindan, P., Ponnambalam, S. G., & Suresh, R. K. (1998). AGV Schedule Integrated with Production in Flexible Manufacturing Systems . *Advanced Manufacturing Technology*, 428-440.

Kalmar. (2014). *Equipment*. Retrieved May 7, 2015, from Kalmar: https://www.kalmarglobal.com/equipment/

Kalmar. (2015). *Kalmar TLS*. Retrieved June 18, 2015, from Kalmar website: https://www.kalmarglobal.com/automation/TLS/

Kalmar. (2015). *Legacy SmartRail upgrade guide.* Tampere, Finland: Kalmar. Retrieved May 27, 2015

Kalmar. (n.d.). Kalmar Automatic Horizontal Transportation System, brochure. Tampere, Finland.

Konecranes. (2014). *Konecranes wins historic order for automated RTG system from Indonesian container terminal operator.* Konecranes. Retrieved May 7, 2015, from Konecranes: http://www.konecranes.com/resources/media/releases/2014/konecranes-wins-historic-order-for-automated-rtg-system-from-indonesian-container-terminal

Konecranes. (2015). *Container Handling Cranes*. Retrieved May 7, 2015, from Konecranes Web site: http://www.konecranes.com/equipment/container-handling-cranes

Kuusniemi, H. (2005). *User-Level Reliability and Quality Monitoring in Satellite-Based Personal Navigation.* Tampere, Finland: Tampere University of Technology, publication 544, doctoral thesis.

Kylliäinen, P. (2010). *A Graphical User Interface Framework for Container Handling.* Tampere, Finland: Tampere University of Technology, thesis.

Luostarinen, R., Manner, J., Määttä, J., & Järvinen, R. (2010). User-centered design of graphical user interfaces. *The 2010 Military Communications Conference - Unclassified Program - Cyber Security and Network Management* (pp. 50-55). Espoo, Finland: IEEE.

Mapbox. (2015). *Mapbox for Developers*. Retrieved May 22, 2015, from Mapbox: https://www.mapbox.com/developers/

MIP. (2013). *Container Services*. Retrieved May 27, 2015, from Mersin International Port: http://en.mersinport.com.tr/port-services/detail/PORT-SERVICES/404/632/0

Mogre, P. S., Hollick, M., & Steinmetz, R. (2007). *Qos in Wireless Mesh Networks: Challenges, Pitfalls, and Roadmap to its Realization.* Darmstadt, Germany: Technische Universitaet Darmstadt.

Mohammed, N. Z., Ghazi, A., & Mustafa, H. E. (2013). Positional Accuracy Testing of Google Earth. *International Journal of Multidisciplinary Science and Engineering, Vol. 4, No. 6,* 6-9.

Morais, C. D. (2000, (original)). *GIS Data Explored - Vector and Raster Data.* Retrieved August 31, 2015, from GIS Lounge: http://www.gislounge.com/geodatabases-explored-vector-and-raster-data/

Mäkelä, J. (2015). *Lane detection system specification.* Tampere, Finland: Cargotec Finland Oy.

OGC. (2015). *Geography Markup Language*. Retrieved September 8, 2015, from Open Geospatial Consortium Web site: http://www.opengeospatial.org/standards/gml#schemas

OGC. (2015). *KML*. Retrieved September 8, 2015, from Open Geospatial Consortium: http://www.opengeospatial.org/standards/kml/

Pirhonen, J. (2011). Automated Shuttle Carrier Concept - Comparison to Conventional RTG Crane and Yard Tractor Concept. In B. J. W., *Handbook of Terminal Planning* (pp. 40-59). Tampere, Finland: Springer Science.

Potere, D. (2008). Horizontal Positional Accuracy of Google Earth's High-Resolution Imagery Archive. *Sensors*, 7973-7981.

Rijsenbrij, J. C., & Wieschemann, A. (2011). Sustainable Container Terminals: A Design Approach. In J. W. Böse, *Handbook of Terminal Planning* (pp. 61-82). New York, USA: Springer Science.

Saanen, Y. A. (2011). Modeling Techniques in Planning of Terminals: The Quantitative Approach. In J. W. Böse, *Handbook of Terminal Planning* (pp. 83-102). Hamburg, Germany: Springer Science.

Safe. (2015). *Home page*. Retrieved September 8, 2015, from Safe Software Inc Web site: http://www.safe.com/

Savant. (2012). *Q-CAN AGV Systems: Fast, Easy-to-Use System Design PC Program.* Retrieved October 6, 2015, from Savant Automation web site: http://www.agvsystems.com/wp-content/uploads/2012/08/Q-CAN_AGV_Savant5.pdf

Schuett, H. (2011). How to avoid checkmate. *Port Technology International*, 82-84.

Schütt, H. (2011). Simulation Technology in Planning, Implementation and Operation of Container Terminals. In J. W. Böse, *Handbook of Terminal Planning* (pp. 103-116). Bremerhaven, Germany: Springer Science.

Steenken, D., Voß, S., & Stahlbock, R. (2015). Container terminal operation and operations research – a classification and literature review. In H.-O. Günther, &

K. H. Kim, *Container Terminals and Automated Transport Systems* (pp. 3-49). Hamburg, Germany: University of Hamburg.

Sun, Z., Ma, P., & Su, X. (2009). *A Perspective on Trusted Hardware and Operating System of Cloud Terminal.* Harbin, China: IEEE.

Szpytko, J., & Hyla, P. (2011). Automated Guided Vehicles Navigating Problem In Container Terminal. *Logistics and Transport*, article, pp. 107-116.

Takada, M., Yamada, M., Kakutani, Y., Sodeyama, K., Hane, S., Fujishiro, T., . . . Watanabe, H. (2014). An Experimental Analysis on Latency Improvement of Cloud-based Fleet Management System. *ACM 7th International Conference on Utility and Cloud Computing* (pp. 505-506). London, GB: ACM.

TBA. (2015). *TEAMS*. Retrieved June 18, 2015, from TBA website: https://www.tba.nl/en/software/teams

Terex. (2015). *AGV Automated Guided Vehicles Automated Container Transport for Performance-Orientated Terminals*. Retrieved June 12, 2015, from Terex website: http://www.terex.com/port-solutions/en/products/automated-guided-vehicles/agv/index.htm

Terex. (2015). *Fully-automated container terminals*. Retrieved May 25, 2015, from Terex web site: http://www.terex.com/port-solutions/en/solutions/solutions-for-terminals/fully-automated-container-terminals/index.htm

Ubukawa, T. (2013). *An Evaluation of the Horizontal Positional Accuracy of Google and Bing Satellite Imagery and Three Roads Data Sets Based on High Resolution Satellite Imagery.* Columbia: Center for International Earth Science Information Network (CIESIN).

UMD. (2003). *Ascii vs. Binary Files*. Retrieved from The Computer Science Department of the University of Maryland: http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/BitOp/asciiBin.html

W3C. (2008, November 26). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Retrieved August 18, 2015, from The World Wide Web Consortium (W3C) Web site: http://www.w3.org/TR/REC-xml/

W3Schools. (2015). *HTML5 SVG*. Retrieved November 27, 2015, from W3Schools Web site: http://www.w3schools.com/html/html5_svg.asp

W3Schools. (2015). *XML Document Types*. Retrieved August 31, 2015, from W3Schools Web site: http://www.w3schools.com/xml/xml_doctypes.asp

Verhoeven, W. (2015, September 30). Email interview.

Wiese, J., Suhl, L., & Kliewer, N. (2011). Planning Container Terminal Layouts Considering Equipment Types and Storage Block Design. In J. W. Böse, *Handbook of Terminal Planning* (pp. 219-245). Hamburg, Germany: Springer Science.

Wiki. (2015). *Openstreetmap API v0.6*. Retrieved May 22, 2015, from Wiki Libraries, Openstreetmap API: http://wiki.openstreetmap.org/wiki/API_v0.6

Wilamowski, B. M., & Irwin, J. D. (2011). *Industrial Communication Systems.* Auburn, USA: CRC Press.

Vis, I. F., & Harika, I. (2005). Comparison of vehicle types at an automated container terminal. In H.-O. (. Günther, & K. H. Kim, *Container Terminals and Automated Transport Systems Logistics Control Issuesand Quantitative Decision Support* (pp. 51-78). Berlin, Germany: Springer Science.

WorldCargo. (2013, November). Tideworks charts its path. *WorldCargo News*, pp. 30-31.

Xing, W., Peihuang, L., Jun, Y., Xiaoming, Q., & Dunbing, T. (2013). Intersection Recognition and Guide-path Selection for a Vision-based AGV in a Bidirectional Flow Network. *International Journal of Advanced Robotic Systems*.

Ylä-Himanka, V. (2014). *Development of Positioning Concept for Automated Rubber Tired Gantry Crane.* Tampere, Finland: Tampere University of Technology, thesis.

APPENDIX 1: GML2 ELLIPSE EXAMPLE

```
<gml:polygonProperty>
      <gml:Polygon srsName="">
            <gml:outerBoundaryIs><gml:LinearRing><gml:coordi-
            nates>1.76455291044872,1.97306904106898,0
            1.76014313067873,2.07406960304115,0
            1.74694735245549,2.17430148975091,0
            1.72506600361912,2.27300187601873,0
            1.69466561444658,2.36941959230818,0
            1.65597755025572,2.46282084157969,0
            1.60929625057555,2.5524947839291,0
            1.55497698828355,2.63775894650851,0
            1.49343316576464,2.71796441755678,0
            1.42513316866937,2.79250078500986,0
            1.35059680121629,2.86080078210514,0
            1.27039133016801,2.92234460462405,0
            1.1851271675886,2.97666386691604,0
            1.09545322523919,3.02334516659622,0
            1.00205197596768,3.06203323078707,0
            0.90563425967823,3.09243361995961,0
            0.806933873410415,3.11431496879598,0
            0.706701986700655,3.12751074701922,0
            0.60570142472848,3.13192052678922,0
            0.504700862756306,3.12751074701922,0
            0.404468976046546,3.11431496879599,0
            0.305768589778731,3.09243361995961,0
            0.20935087348928,3.06203323078707,0
            0.115949624217766,3.02334516659622,0
            0.026275681868361,2.97666386691604,0 -0.05898848071105,2.92
            234460462405,0 -0.139193951759327,2.86080078210514,0 -0.213
            730319212406,2.79250078500986,0 -0.282030316307681,2.717964
            41755679,0 -0.343574138826589,2.63775894650851,0 -0.3978934
            01118587,2.5524947839291,0 -0.444574700798764,2.46282084157
            969,0 -0.483262764989616,2.36941959230818,0 -0.513663154162
            157,2.27300187601873,0 -0.535544502998528,2.17430148975091,
            0 -0.548740281221765,2.07406960304115,0 -0.55315006099176,1
            .97306904106898,0 -0.548740281221766,1.8720684790968,0 -0.5
            35544502998529,1.77183659238704,0 -0.513663154162158,1.6731
            3620611923,0 -0.483262764989618,1.57671848982978,0 -0.44457
            4700798766,1.48331724055826,0 -0.39789340111859,1.393643298
            20886,0 -0.343574138826592,1.30837913562945,0 -0.2820303163
            07684,1.22817366458117,0 -0.21373031921241,1.15363729712809
            ,0          -0.139193951759331,1.08533730003282,0          -
            0.058988480711054,1.02379347751391,0
            0.026275681868357,0.96947421522191,0
            0.115949624217762,0.922792915541733,0
            0.209350873489275,0.884104851350881,0
            0.305768589778726,0.85370446217834,0
            0.404468976046541,0.831823113341968,0
            0.504700862756301,0.818627335118731,0
            0.60570142472848,0.814217555348736,0
            0.70670198670065,0.81862733511873,0
            0.80693387341041,0.831823113341966,0
            0.905634259678225,0.853704462178337,0
            1.00205197596768,0.884104851350878,0
            1.09545322523919,0.922792915541729,0
            1.1851271675886,0.969474215221906,0
```

```
            1.27039133016801,1.0237934775139,0
            1.35059680121628,1.08533730003281,0
            1.42513316866936,1.15363729712808,0
            1.49343316576464,1.22817366458116,0
            1.55497698828355,1.30837913562944,0
            1.60929625057554,1.39364329820885,0
            1.65597755025572,1.48331724055826,0
            1.69466561444657,1.57671848982977,0
            1.72506600361912,1.67313620611922,0
            1.74694735245549,1.77183659238704,0
            1.76014313067872,1.8720684790968,0
            1.76455291044872,1.97306904106898,0</gml:coordi-
            nates></gml:LinearRing></gml:outerBoundaryIs>
      </gml:Polygon>
</gml:polygonProperty>
```

APPENDIX 2: GEOJSON ELLIPSE EXAMPLE

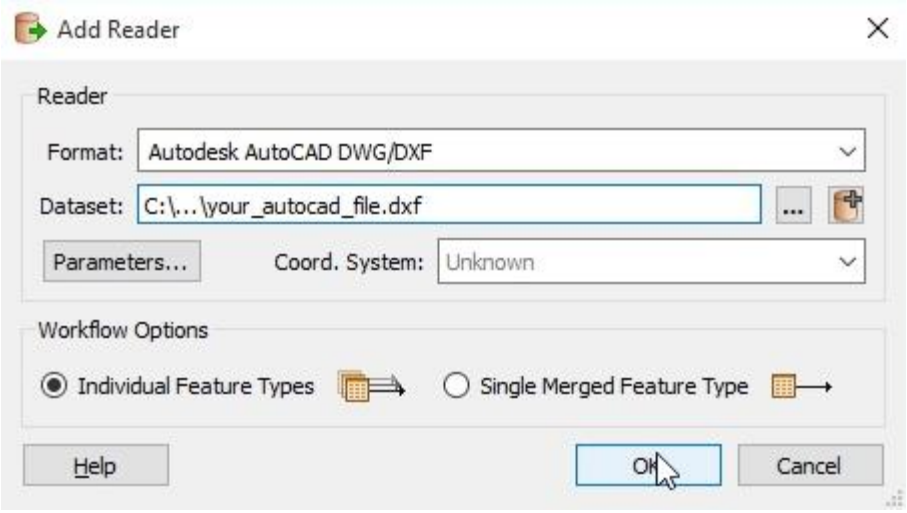

```
{"type": "Feature", "geometry":
      {
            "type":"Polygon",
            "coordinates":
            [
                  [
                        [1.76455291044872,1.97306904106898,0],
                        [1.76014313067872,2.07406960304115,0],
                        [1.74694735245549,2.17430148975091,0],
                        [1.72506600361911,2.27300187601872,0],
                        [1.69466561444657,2.36941959230818,0],
                        [1.65597755025572,2.46282084157969,0],
                        [1.60929625057554,2.55249478392909,0],
                        [1.55497698828355,2.6377589465085,0],
                        [1.49343316576464,2.71796441755678,0],
                        [1.42513316866936,2.79250078500986,0],
                        [1.35059680121629,2.86080078210514,0],
                        [1.27039133016801,2.92234460462404,0],
                        [1.1851271675886,2.97666386691604,0],
                        [1.09545322523919,3.02334516659622,0],
                        [1.00205197596768,3.06203323078707,0],
                        [0.905634259678229,3.09243361995961,0],
                        [0.806933873410414,3.11431496879598,0],
                        [0.706701986700655,3.12751074701922,0],
                        [0.60570142472848,3.13192052678921,0],
                        [0.504700862756306,3.12751074701922,0],
                        [0.404468976046547,3.11431496879598,0],
                        [0.305768589778732,3.09243361995961,0],
                        [0.209350873489281,3.06203323078707,0],
                        [0.115949624217767,3.02334516659622,0],
                        [0.0262756818683624,2.97666386691604,0],
                        [-0.0589884807110478,2.92234460462404,0],
                        [-0.139193951759325,2.86080078210514,0],
                        [-0.213730319212404,2.79250078500986,0],
                        [-0.282030316307678,2.71796441755678,0],
                        [-0.343574138826587,2.63775894650851,0],
                        [-0.397893401118584,2.5524947839291,0],
                        [-0.444574700798761,2.46282084157969,0],
                        [-0.483262764989613,2.36941959230818,0],
                        [-0.513663154162154,2.27300187601873,0],
                        [-0.535544502998525,2.17430148975091,0],
                        [-0.548740281221762,2.07406960304115,0],
                        [-0.553150060991757,1.97306904106898,0],
                        [-0.548740281221763,1.8720684790968,0],
                        [-0.535544502998526,1.77183659238704,0],
                        [-0.513663154162155,1.67313620611923,0],
                        [-0.483262764989614,1.57671848982978,0],
                        [-0.444574700798763,1.48331724055827,0],
                        [-0.397893401118587,1.39364329820886,0],
                        [-0.34357413882659,1.30837913562945,0],
                        [-0.282030316307682,1.22817366458117,0],
                        [-0.213730319212408,1.15363729712809,0],
                        [-0.139193951759329,1.08533730003282,0],
                        [-0.0589884807110519,1.02379347751391,0],
                        [0.0262756818683582,0.969474215221913,0],
                        [0.115949624217763,0.922792915541736,0],
```

```
                        [0.209350873489277,0.884104851350884,0],
                        [0.305768589778727,0.853704462178343,0],
                        [0.404468976046542,0.831823113341971,0],
                        [0.504700862756301,0.818627335118734,0],
                        [0.60570142472848,0.814217555348739,0],
                        [0.70670198670065,0.818627335118733,0],
                        [0.806933873410409,0.83182311334197,0],
                        [0.905634259678225,0.853704462178341,0],
                        [1.00205197596767,0.884104851350881,0],
                        [1.09545322523919,0.922792915541732,0],
                        [1.18512716758859,0.969474215221908,0],
                        [1.270391330168,1.02379347751391,0],
                        [1.35059680121628,1.08533730003281,0],
                        [1.42513316866936,1.15363729712809,0],
                        [1.49343316576464,1.22817366458117,0],
                        [1.55497698828354,1.30837913562944,0],
                        [1.60929625057554,1.39364329820885,0],
                        [1.65597755025572,1.48331724055826,0],
                        [1.69466561444657,1.57671848982977,0],
                        [1.72506600361911,1.67313620611922,0],
                        [1.74694735245548,1.77183659238704,0],
                        [1.76014313067872,1.8720684790968,0],
                        [1.76455291044872,1.97306904106898,0]
                    ]
                ]
            },
            "properties":null
        }
```

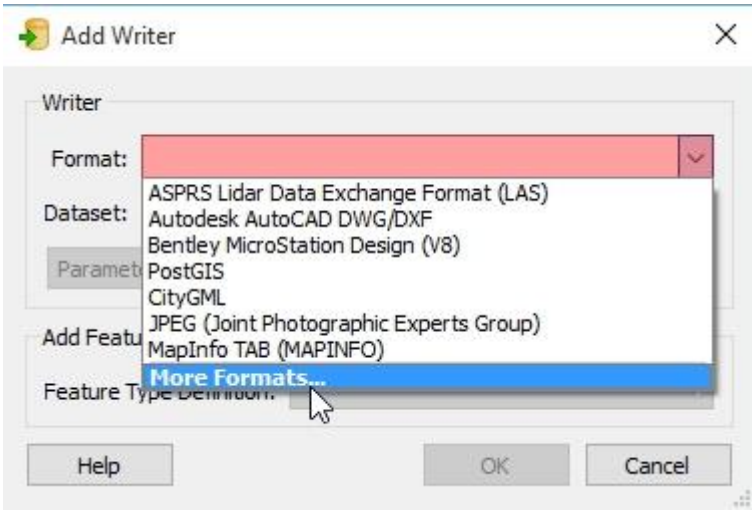
APPENDIX 3: FME INSTRUCTIONS FOR AUTOCAD-TO-SVG TRANSLATION FOR MAP DATA CREATION PROCESS

Following instructions are meant for transforming an AutoCAD model to SVG format using FME Desktop program from Safe Software Inc. Required steps before transformation are following: creating workspace, selecting reader to process the AutoCAD model, selecting writer to create resulting file type and creating links between these two. *Italic* text is used to describe text you see in the program. Screenshots are added to clarify the required actions.

1. Open FME and select "*File" – "New…"* and select "*Blank workspace"* to begin with
2. Select "*Readers" – "Add Reader…"*
3. Browse for the original AutoCAD-model to the section under "*Reader"* - "*Dataset*", and the format is automatically selected to *"Autodesk AutoCAD DWG/DXF":*

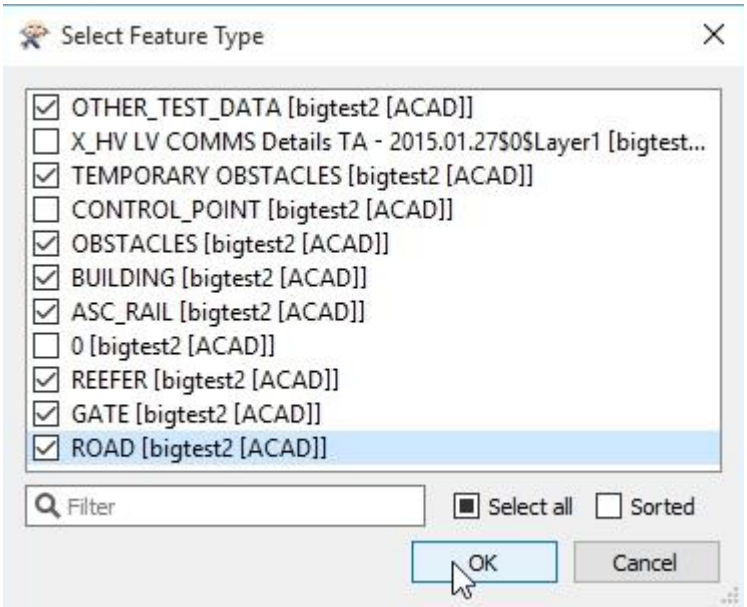


4. Select all layers on the opening windows and press "*OK".* Layers will be shown as blocks in the main window.
5. Select *"Writers" – "Add Writer…"*
6. Open the drop-down menu in *"Format"* and press *"More Formats…":*

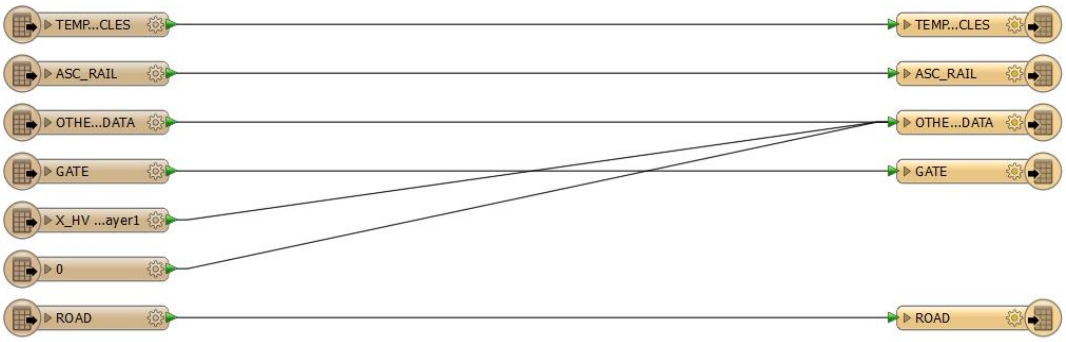


7. Browse for "SVG (Scalable Vector Graphics)" and select it:

8. If the original precision (6 significant digits) is wanted to be changed, select "*Parameters"* and change the precision
9. In "*Dataset"* browse for a folder in which you want the data to be selected, and name your result file. After this, select "*OK*"
10. In the opening layer list, select layer names which you will want to reutilize in the resulting file:

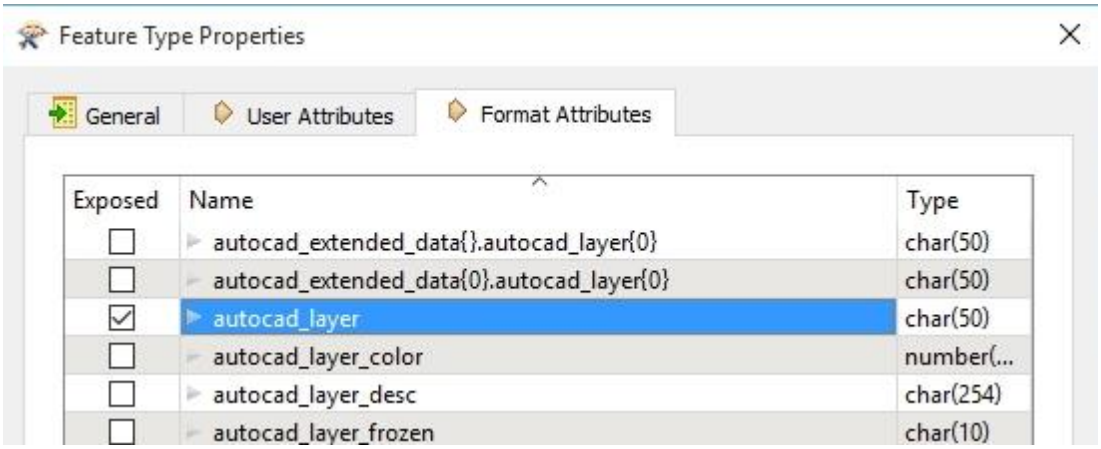


After this, selected reader layers are listed on the left and writer layers are on the right. In order to save layers from AutoCAD into the resulting SVG, they can be simply connected with lines, as shown below.

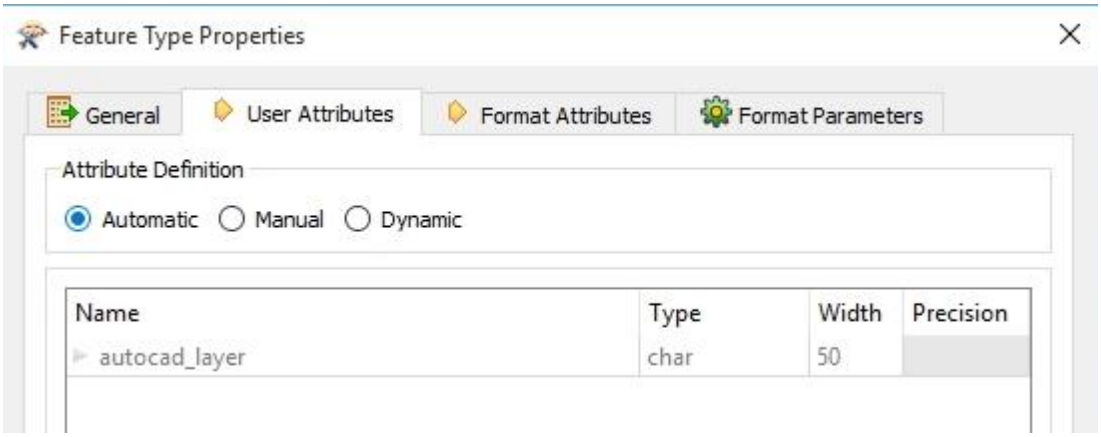


Following parts 11-13 describe actions that can be done to modify the transferred data:

11. Features, such as the original AutoCAD layer name can be extracted from any of the reader's layers by double-clicking a layer, selecting "*Format Attributes"*, browsing and selecting the wanted property:



12. Accepting these features in a layer in writer can be done by double-clicking a layer, selecting "*User Attributes*" and defining "*Attribute Definition*" to "*Automatic*":



13. In the same window, variables for this specific layer can be added to the main view by selecting "*Manual*" attribute definition.

When all data is selected according to the instructions given in this manual, transformation can be started by pressing  -button in the toolbar. When transformation is finished, a message of a successful translation should be seen in the "*Translation Log*" window: