Lauri Lammi

A tool for defining models of generic mobile machines

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 13 May 2014

Thesis supervisor:

Prof. Ville Kyrki

Thesis advisor:

M.Sc.(Tech.) Matthieu Myrsky

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Lauri Lammi

Title: A tool for defining models of generic mobile machines

Date: 13 May 2014 Language: English Number of pages: 6+73

Department of Electrical Engineering and Automation

Professorship: Automation technology Code: AS-84

Supervisor: Prof. Ville Kyrki

Advisor: M.Sc.(Tech.) Matthieu Myrsky

In this thesis work, a software tool for quickly and easily defining a mobile vehicle is presented. Vehicles are defined through a simple, intuitive 3D graphical user interface. Based on the vehicle definition, a kinematic model is generated for the vehicle. The kinematics calculations use state-of-the-art knowledge on the kinematics of different vehicle types.

The generated kinematic models can be used in a separate simulator module, also created for this thesis work, to simulate arbitrary vehicle configurations. Supported vehicle types include the most common mobile industrial vehicles, such as car-like, tracked, center-articulated or passively linked vehicles. Simulated vehicles can also be combinations of these types.

Kinematic models generated with this software are tested against data sets gained from different real-world vehicle configurations. The models are found to be accurate and suitable for various purposes requiring a kinematic model of a vehicle.

Keywords: mobile machines, kinematics, simulation, robotics, model generation

AALTO-YLIOPISTO
SÄHKOTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Lauri Lammi

Työn nimi: Työkalu geneeristen liikkuvien koneiden mallintamiseksi

Päivämäärä: 13/05/2014 Kieli: Englanti Sivumäärä: 6+73

Sähkötekniikan ja automaation laitos

Professuuri: Automaatiotekniikka Koodi: AS-84

Valvoja: Prof. Ville Kyrki

Ohjaaja: DI Matthieu Myrsky

Tässä työssä esitellään mielivaltaisten liikkuvien työkoneiden nopeaan ja helppoon määrittelyyn tarkoitettu ohjelmisto. Käyttäjä määrittelee työkoneen yksinkertaisen, intuitiivisen 3D-käyttöliittymän avulla. Työkoneen määritelmän perusteella generoidaan sen kinemaattinen malli. Kinemaattisissa laskuissa käytetään uusimpia tutkimustuloksia erilaisten ajoneuvojen kinematiikasta.
Generoiduilla kinemaattisilla malleilla voidaan simuloida mielivaltaisia konekonfiguraatioita erillisessä simulaattorimoduulissa, joka luotiin osana tätä työtä. Simulaattori tukee yleisimpiä teollisuudessa käytettyjä liikkuvia konetyyppejä, kuten automaisia, telaketjullisia, runkonivellettyjä tai passiivisesti yhdistettyjä koneita. Simuloidut ajoneuvot voivat myös yhdistellä eri konetyyppejä.
Ohjelmistolla generoiduilla kinemaattisilla malleilla simuloidaan erilaisia tosimaailman ajoneuvokonfiguraatioita. Simulaatiotuloksia verrataan oikeista koneista saatuun dataan. Mallien havaitaan olevan tarkkoja ja sopivia erilaisiin tarkoituksiin, jotka vaativat ajoneuvon kinemaattisen mallin.

Avainsanat: liikkuvat työkoneet, kinematiikka, simulaatio, robotiikka, mallintaminen

Preface

This work is part of the MOTTI project which was conducted in the Energy and Life Cycle Cost Efficient Machines (EFFIMA) research program, managed by the Finnish Metals and Engineering Competence Cluster (FIMECC), and funded by the Finnish Funding Agency for Technology and Innovation (TEKES), research institutes and companies. Their support is gratefully acknowledged.

Contents

1 Introduction

Automation has emerged as the primary method of increasing productivity and efficiency in industrial processes. Different manufacturing processes have already been automated to various degrees for decades. In contrast, the automation of mobile industrial vehicles is a relatively young field. Industrial processes involving the use of automated vehicles have begun to surface only in recent years though more and more vehicle types are being automated. The level of automation in mobile vehicles ranges from safety systems that help human drivers to fully autonomous fleets of vehicles.

The rate at which new applications for autonomous vehicles are developed is increasing. Already a wide variety of different autonomous mobile machines are utilized in a number of automated engineering tasks. Autonomous vehicles have many potential benefits e.g. in mining (Fig. 1a), cargo hauling (Fig. 1b), construction (Fig. 1c), warehousing and forest work, among others. Autonomous vehicles offer numerous benefits for these tasks, such as improved safety and working conditions for vehicle operators. Examples of mobile industrial machines that have been automated include cargo transporters with four-wheel steering [3] and load-haul-dumpers with an articulated body [4].

Automating a heterogeneous fleet of mobile vehicles is not a trivial task. It consists of multiple phases from system specification to deployment, each of which is complex in its own right. The end result should be both efficient and safe while also minimizing system costs by e.g. using pre-existing machinery or by minimizing the size of the work site. At the same time, the automated vehicles need to co-operate both with other machines and, in the future, human operators. It can be difficult and time consuming to experiment with different vehicle types, fleet configurations, traffic routes and rules to create an automated system suitable for any specific task.

Often the processes of system specification and simulation require working with many different, complex tools that at times may work poorly together. It may be necessary to simulate multiple different vehicles to test their suitability for a given task or to better match those tasks to the vehicles, or the work site may involve multiple vehicle types working together, each requiring their own simulator models for development purposes. Simulator models are not readily available for most vehicles and their creation is often slow, difficult and needlessly complex considering the relatively simple nature of their use. Thus the creation of a quick and easy to use, abstract vehicle definition and kinematic model export tool would speed up the system design, task planning and simulation phases of the automation process.

The goal for this thesis work was the creation of a software tool that can be used to quickly and easily define kinematic models for both common and arbitrary vehicle configurations. A kinematic model describes the motion of a vehicle without considering its causes and can be used to estimate said motion over time when given certain control inputs. The tool would thus define certain key parameters for a vehicle, which can then be used to calculate its kinematics. Supported vehicle types are limited to ground-based ones, as most industrial vehicles operate solely on the ground.

(a) AutoMine. Photo: Sandvik [1]

(c) Avant 320 loader

(b) Straddle carrier. Photo: Konecranes [2]

Figure 1: Mobile machines that have been automated in the past.

As a proof of the suitability of the tool and the vehicle models defined with it for the purpose of kinematics calculations, the secondary goal for this work was the creation of a kinematic simulator model for generic mobile vehicles. The model would read the parameters generated with the definition tool to calculate the kinematics of a defined vehicle, which would then be simulated in a two-dimensional multi-machine environment. The simulator would need to support multiple different vehicle types running at the same time in the same simulation. Simulated vehicles would need to be controllable using specific control interfaces and communications infrastructure. The simulator component together with the definition tool would allow a user to rapidly define and then experiment with different vehicle configurations.

To meet these goals, an abstract vehicle definition tool was created. With it, vehicles are defined using an intuitive graphical user interface that allows models of the vehicles to be exported for use in other software. Vehicle models are built using a simple, building block type design system that allows the user to freely place components representing common vehicle parts such as the chassis, axles or sensors of the vehicle. An existing kinematic simulator environment was expanded with a generic kinematic model for the arbitrary vehicle configurations defined with the tool.

The software created for this thesis work will be used to e.g. support research on path planning and controlling heterogeneous groups of vehicles as well as the creation of traffic rules for such groups. Testing how individual vehicles behave when given certain control inputs is not one of the intended uses of the software.

As such, generalizability and quick deployment are more important than the overall accuracy of the simulation model of a vehicle. Kinematic models will thus provide sufficiently accurate motion estimates for any vehicles given the intended uses of the software.

The following chapters will cover the main features and algorithms of the vehicle definition tool and its accompanying simulator component. First, their requirements are discussed in detail in Chapter 2. Chapter 3 compares different existing software solutions for defining a vehicle and explores different file formats that could be used for storing or exporting a vehicle model. A literature study on the state-of-the-art research of the kinematic models of different vehicle configurations is presented in Chapter 4. These models will then be used in the actual simulator software. The structure and features of the created machine definition tool are presented in Chapter 5. Chapter 6 covers the implemented generic mobile machine simulator model and its features. It also presents the main features of the machine interface and communications infrastructure used to control the simulated vehicles. In Chapter 7, several models of mobile machines are created with the new definition tool and then simulated using the model for generic vehicles. The results of these simulations are compared with data gathered from actual real-world machines to analyse the accuracy of the generated kinematic models and the viability of the system as a whole. Finally, the thesis work and its results are summarized in Chapter 8.

2 Background and system requirements

This work is a part of the Motti multi-machine information control research project [5]. Motti aims to develop a machine-system interface that enables fleets of heterogeneous vehicles to operate together. This involves the development of tools e.g. for machine task management and execution, traffic control and routing as well as generic components for multi-machine control. The developed tools are intended as a part of the Motti's tool chain as shown in Figure 2. The tool chain aims to minimize the amount of time and work needed to deploy a new autonomous multi-vehicle system and to automate as much as possible of the work involved.

Currently, the majority of autonomous vehicle fleets operate based on pre-defined, hand-crafted rule sets. Rule definition is time consuming and the resulting rule sets are difficult to test comprehensively. The current Motti tool chain in Figure 2 shows the different stages involved as well as their interdependencies. The process starts with models of both the operating vehicles as well as the work site. These are used to generate a network of routes the vehicles are capable of traversing, which in turn form the basis for a set of traffic rules the autonomous vehicles are to follow to ensure safe and efficient operations. Finally, tasks and missions are specified for the work site and distributed to vehicles either in the real world or in a simulated environment. This thesis work contributes to the vehicle definition and simulation phases of the Motti tool chain.

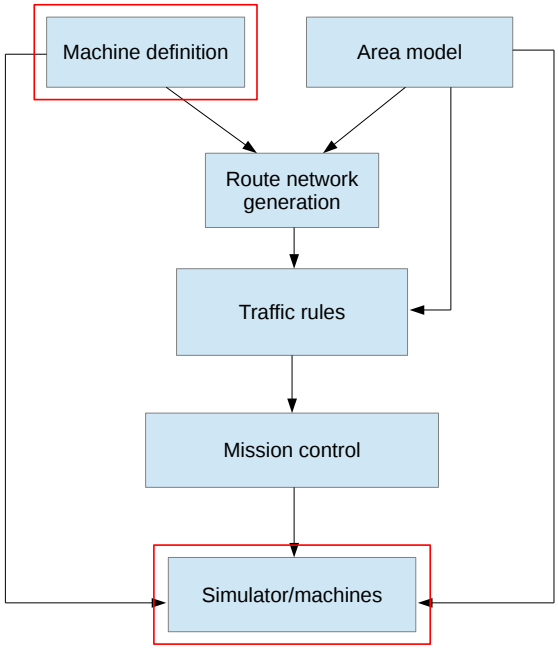


Figure 2: The Motti tool chain with the areas this thesis work contributes to highlighted.

The main goal for this thesis work is the creation of a software tool that can define the dimensions and other principal characteristic of an arbitrary mobile vehicle. With the tool, an abstract graphical representation of the vehicle can be designed quickly and easily. The kinematics of the vehicle can then be deduced from the abstract model and exported for use in other software. To demonstrate the suitability of the models created with the definition tool for the purpose of kinematics calculations, a simulator component capable of reading models exported from the tool and generating kinematic models based on them will also be created. This will require the creation of a functioning kinematic model for generic mobile machines. All of the software created for this thesis work are to be parts of the tool chain of the Motti project, shown in Figure 2.

2.1 Requirements for the definition tool

This section covers in detail the requirements for the definition tool. The requirements are divided into functional, user interface and system interface requirements.

Functional requirements

R1. The definition tool will be used to define the structure of arbitrary vehicles. It must therefore support creating and modifying objects representing at least the vehicle body as well as its axles and joints.

- R1.1. The tool must be able to distinguish between each object type.
- R1.2. It should have the functionality to add range scanners to vehicle models.

R2. The software shall have the functionality to define

- R2.1. a top speed, both forwards and backwards, and rates of acceleration and deceleration for vehicles.
- R2.2. maximum angles and turn rates for the joints and axles.
- R2.3. whether joints and angles are actuated, i.e. steering.
- R2.4. fields of vision and ranges for the range scanners.

R3. The dimensions and location of each object must be modifiable after creation.

R4. The software should support copying and pasting objects.

R5. It should also support undoing and redoing any actions related to objects.

R6. The tool must support saving vehicle models in a commonly used, portable format.

R7. It must also support loading previously saved models afterwards with no loss of data.

R8. As a critical requirement, vehicle models must be exportable in a form that can be used by the chosen simulator framework to generate a kinematic model for the vehicle.

R8.1. Generating a kinematic model shall require no additional input from the user after the vehicle definition is finished.

User interface requirements

R9. The tool shall have a three-dimensional graphical user interface (GUI).

R10. The GUI shall show

R10.1. a design view with a visual representation of the current vehicle model.

R10.2. the available design tools.

R10.3. the properties of any objects the user has selected.

R11. Vehicle parts shall be defined using the mouse.

R11.1. The user must be able to create at least three-dimensional geometric primitives such as boxes and cylinders, which shall represent the vehicle body and axles respectively.

R11.2. Creating a single object must not require more clicks of the mouse than the object has vertices.

R12. All of the tools and functions of the software shall be accessible from the GUI.

System interface requirements

R13. The Motti tool chain is primarily Linux-based and as such, all of its tools must be Linux-compatible. Thus, all tools and libraries used and created in this work must function on Linux.

R13.1. Cross-platform compatibility is not required.

2.2 Requirements for the simulator component

This section covers in detail the requirements for the simulator component that is used to create kinematic models for the vehicles defined with the definition tool. The requirements are divided into functional and interface requirements.

Functional requirements

R14. The multi-machine simulator called StageMaCI or Motti Simulator is already a part of the Motti tool chain. It shall be used as the basis for the simulator component.

R15. The simulator component shall consist of a functioning kinematic model for generic mobile machines. In other words, it must be able to generate a kinematic model for and simulate any vehicle exported from the definition tool. This will mean calculating the kinematics for at least the most common industrial mobile vehicle types.

R16. The kinematic calculations used by the generic model must be based on established research.

R17. The generic kinematic model used by the simulator component shall support vehicles with any number axles or joints along their body. Both the axles and the joints can be actuated, i.e. steering.

R18. The simulator must be able to simulate at least ten generic vehicles at the same time. The vehicles can be of different types.

R19. The simulator component and the simulated vehicles should support an arbitrary number of range scanners.

Interface requirements

R20. Simulated vehicles must be controllable both directly by giving them speed and steering commands, and by giving them lists of coordinates to drive to.

R20.1. Coordinates on the list will consist of a target velocity as well as a location; If the target velocity is negative, the vehicle shall drive in reverse to the location.

R21. The simulator component shall make the current position of its origin and axles in world coordinates available for use in other software.

R22. Data from any range scanners shall be accessible for other software as well.

3 Existing software infrastructure

3.1 Generic design software

This chapter takes a look at different tools capable of creating a suitable model of a vehicle. The main focus is on design tools that can easily draw a model of a vehicle and be modified to export suitable simulator models. The purpose is to determine whether an entirely new program will need to be created for this work or not.

For a program to be considered for use in this task, it must either fulfil the requirements for the definition tool listed in Chapter 2 as is, or support plug-ins to add any missing features. Thus, the different software tools that will be covered in this chapter fall into several categories. These include CAD software, graphical design tools and computational software.

Traditionally engineering-oriented graphic design has been done using CAD software such as AutoCAD [6]. They are powerful tools for designing components or even entire machines, sometimes offering support for kinematics calculations out of the box. CAD software are in such wide-spread use that for many real-world vehicles, a CAD model has likely already been made. However, the availability of such models may be limited. If a CAD model for a vehicle is available, it can be a major benefit, eliminating any need for any further vehicle definition. CAD programs have the downside of being complex, difficult and time consuming to use, as CAD models are normally created with a high level of detail before they are used elsewhere. Commercial CAD software can also be highly expensive, while the few free CAD programs that are available are often limited in their features. CAD software also have very limited support for Linux, breaking requirement R13.

Graphical design software comes in many different variations. Most of them meet at least the stated requirements for the user interface of the definition tool (R9–R12) and many of the functional ones well. For instance, the open source 3D computer graphics tool Blender [7] has many functionalities that make it attractive for a task similar to this one. Aside from being intended for the creation of 3D models right from the start, it supports many different XML-based file formats such as Collada or X3D (R6). It can also be extended with Python plug-ins to add any required features that are not originally supported. In addition, Blender provides its own physics engine which could conceivably be used to simulate any vehicles defined with it.

While the goal is to create 3D models of vehicles, this does not necessarily mean that only 3D design tools should be considered. 2D vector graphics editors such as Inkscape [8] can also be used to quickly create the sorts of abstract, geometric drawings of vehicles that will be used to generate simulation models. While the 2D nature of such editors is a limitation, it can be circumvented by e.g. drawing an object from two viewpoints, top and side, and then linking these different views of the same object together. Inkscape, for example, supports extensions through plug-ins and XML-based, extendible file formats, making the creation of custom tools for these purposes possible.

Many commercially available software tools provide graphical design capabilities,

some even supporting simulations on created models by themselves. On the other hand, computational software such as Matlab [9] and its Simulink toolbox [10] can be used to create simulations of virtually any conceivable vehicle configuration, Simulink also allowing block-based graphical simulation model creation. However, they lack a dedicated graphical design functionality, violating one of the stated user interface requirements (R9).

A comparison of the covered different generic design and simulation tools relative to the requirements in Chapter 2 is shown in Table 1. For existing software, the most important requirements are a design view (R10.1) and support for 3D primitives (R11.1). Another important requirement is support for commonly used, XML-based formats (R6). As the use of StageMaCI for simulations had already been decided, out of the box support for the creation of Stage models (R8) would also have been a major benefit. Each of the considered software could be extended to support Stage model creation, though, making this a less important feature. It is unlikely that any one existing software tool fulfils every requirement in Chapter 2, making it important for the tools to be extensible. Any existing software should also be openly available for further development.

Although all of the discussed software tools have a number of good qualities and match certain of the requirements such as R10.1 and R6 well, they each also have significant drawbacks. The high price of commercial CAD software makes them unappealing, while their complexity makes them unsuited for rapid vehicle defining. Free CAD software also lack important features. While many vehicles have existing CAD models, one cannot assume that such a model is available.

Blender and other programs like it require a lot of practice before they can be used efficiently. They also have many features that will be entirely unnecessary for this task, resulting in a user interface that is cluttered with data and functionalities that serve no purpose in terms of vehicle model creation. Thus, the 3D graphical design tools currently available are unsuitable for the purposes of this thesis work as well.

Vector graphics editors require the creation of so many new extensions, tools and plug-ins, while still being less suitable for this task than a purpose-built tool, that it is simply impractical to use one. Alternatively, a lengthy tutorial on specifically how to draw vehicles would be required. This would make the software more unintuitive and more difficult to use.

	AutoCAD	Blender	Inkscape	Matlab / Simulink
Design view (R10.1)	Yes	Yes	Yes	No
3D primitives (R11.1)	Yes	Yes	No	No
XML support (R6)	No	Yes	Yes	Yes
Stage support (R8)	No	No	No	No
Extensible	Yes	Yes	Yes	Yes
Open source	No	Yes	Yes	No

Table 1: Comparison of different generic design and simulation tools

While Matlab and Simulink are powerful simulation tools, neither of them are particularly easy or quick to use, requiring programming, in-depth understanding about the mathematics behind any vehicle model and possibly separate hand-crafted simulation models for every vehicle. Their lack of a design view is also a significant drawback, making them the only considered software to break requirement R10.1. Thus, computational software is not suitable in this case.

It is clear that a design tool suitable for the quick and easy creation of abstract 3D models of mobile vehicles, capable of creating simulator models based on them, does not exist previously. As such, it is necessary to create an entirely new, purpose-built vehicle definition tool with all of the required features. It will be supported by extending StageMaCI with a generic vehicle class for required the simulations and kinematics calculations.

3.2 File formats

Another important requirement for the tools to be designed as part of this thesis was the possibility to store and export vehicle models in some widely used file format (See R6 and R8). To this end, different commonly used file formats suitable for the task were studied. In this chapter, different formats suitable for this work are discussed.

If the intention is to generate a kinematic model based on a stored file of a vehicle, the format itself does not need to be complex. Usually, kinematic models depend on fairly few parameters, most of which are related to the physical locations and dimensions of specific vehicle parts. This means that as long as the position of the parts in relation to one another can be deduced from the file, it is possible to generate a kinematic model for the described vehicle. While such information could be stored in virtually any format, there are benefits to using pre-existing formats. Portability, one of the stated functional requirements for the definition software (R6), is one such benefit.

XML-based file formats are a flexible way of storing vehicle models and are well suited for the needs of this thesis work. XML has the enormous benefits of being text based, highly expandable and easily understandable and modifiable even when using just a text editor [11]. Many XML-based formats are in wide-spread use, making them highly portable. These qualities make XML-based formats well suited in terms of requirement R6. A number of binary file formats for storing 3D models exist, such as the CAD format DWG, but they tend to be proprietary or otherwise commercial. This limits support for them especially in free software.

Several XML-based file formats created specifically for storing graphic data exist. Of these, formats such as Collada [12] and X3D [13] are meant for 3D data storage. Both offer diverse features ranging from object materials to scene lighting. Collada also supports defining physical attributes for objects, making it suitable for storing information on more than just graphics. For example, Collada can be used to define kinematic parameters for objects.

Software support for Collada and X3D varies, with Collada being supported by numerous software tools such as Blender or the robotics development environment

OpenRAVE [14] while X3D use is currently mainly limited to open source tools. The downside to Collada is its complex structure. X3D, on the other hand, is structurally very simple and easy to understand. It is especially well-suited for representing simple geometric primitives. Although its use is currently somewhat limited, it aims to become integrated in HTML5 [13].

While the models needing to be stored will be three-dimensional, the file format used to store them does not have to be. SVG [15] is a 2D vector image format supported by a huge number of different programs. While potentially limiting its usefulness, the two-dimensionality can be overcome by e.g. using two different viewpoints per object to be stored as mentioned previously. Different views of the same object can be linked to each other with for instance object attributes. Most graphic design tools support SVG to some extent, at the very least being able to export it, and SVG images can be viewed even in web browsers. Its XML representation is simple and can be easily understood even by a human. There are some binary alternatives to SVG, such as PDF and the Flash format SWF, but their creation requires additional resources and functionalities.

There is of course a huge amount of different file formats related to graphic design currently available. Aside from the ones discussed here, however, they tend to be either intended for a more specialized by nature, obsolete or in little use. As such, the graphical file formats discussed here represent the most important ones with regard to the thesis work. However, file formats need not be intended for use in graphic design to be viable. Essentially, any format that can store the physical structure of a vehicle can also be used both by the tool. Several more specialized XML-based file formats intended for storing information on complex physical scenes and machines also exist.

An example of formats intended for specialist use is the Robot Modeling Language (SDF) that the dynamic robot simulator Gazebo uses [16]. The SDF format is particularly geared towards describing complex physical objects with support for a number of physical parameters ranging from inertial parameters (e.g. mass) to collision detection using three-dimensional geometry data. However, SDF has the same downsides as Collada, namely its complexity.

A comparison of the features of different file formats is shown in Table 2. As portability between different software was considered one of the requirements for the definition tool (See R6) and proprietary formats are in limited use, the most important feature in this case was the openness of the format. Similarly, being text based would increase the number of tools that could be used to at least view stored vehicle models. As the formats will be used to store three-dimensional vehicle models, support for three-dimensional objects would be a benefit though not a requirement. While being able to define kinematic parameters for objects on the file format level is not a requirement, it may be a beneficial feature.

The choice to limit the considered file formats mainly to XML-based ones is well founded. Most binary formats suitable for storing a vehicle model are in relatively little use when compared with XML-based ones, because binary formats tend to be proprietary. Using more commonly available text-based formats ensures compatibility with other software as well as making file handling easier to program and debug.

	DWG	Collada	X3D	SVG	SDF
Open source	No	Yes	Yes	Yes	Yes
Text based	No	Yes	Yes	Yes	Yes
Three dimensional	Yes	Yes	Yes	No	Yes
Kinematics support	No	Yes	No	No	Yes

Table 2: Comparison of different file formats

Based on Table 2, the most obvious choices for formats would seem to be Collada and SDF. However, there are complicating factors that need to be considered.

Many of the XML-based formats come packed with a multitude of features. Some features such as scene lighting settings in X3D and Collada are unnecessary for the purposes of this thesis task and needlessly complicate created files. However, thanks to the flexibility inherent to all XML-based formats, any unnecessary features can be safely left out of any such files. Despite this feature, Collada and other more specialized formats such as SDF remain very complex. For example, in Collada describing even simple geometric shapes requires a complicated mesh notation. This complexity makes Collada more difficult to use than for example X3D.

The two-dimensional nature of SVG can be a factor limiting its usefulness. Although having to store two SVG elements for each object leads to somewhat more complex SVG files, this would allow a 3D model to be viewed and edited even in a 2D vector graphics program such as Inkscape. The clear, simple structure of the SVG format makes it a very attractive choice.

In the end, it was decided to focus on the relatively simple X3D and SVG formats. Both of them are very light weight and easy to understand, as well as being supported by a number of other software tools. As the primary intention is to use them to store vehicle models consisting of simple geometric primitives, support for more complex features was not considered necessary.

4 Kinematic models

Kinematics is a very fundamental part of the design, analysis, control, simulation and path planning of a vehicle. Kinematics is the study of the motion of a machine without taking into account the forces that cause that motion [17]. The kinematic model of a mobile machine usually describes how its pose in two dimensions, i.e. its XY-location in the world coordinate system and its heading θ, changes over time. By leaving out the forces and torques causing the motion, the result is a relatively simple way of estimating the movement of a machine while ignoring any complex, hard to model dynamics involved in its environment. The downside is that it is difficult to create an accurate kinematic model for machines that interact dynamically with their environments to any significant degree. For example, uneven or slippery terrain is a dynamic environment and thus any machine operating in it is subjected to a high degree of dynamic behaviour. Thus, such environments are difficult to take into account in the kinematic model of a vehicle.

A kinematic model is dependent on the machine it is created for and its features, e.g. what it uses for locomotion. As such, different machine types have very different kinematic models. Once a kinematic model for a machine has been formed, it can be used for example to design a machine capable of navigating in a certain area, to optimize the trajectory of a robot, to analyse the capabilities of a machine or to control its pose.

Kinematic models for mobile vehicles most often calculate the current velocity of a vehicle in the X and Y directions and angular velocity about the current center of rotation of the vehicle. The models use certain variables as control input to produce pose estimates. The exact variables used depend on the configuration of the vehicle being modelled, but there are certain common ones used in many of the kinematic models. Commonly used variables include the current forward velocity and heading of the vehicle measured at a known point on its body, the current angles of steering vehicle parts such as axles and joints, as well as the distances between them. In general, any variables that affect the trajectory of a vehicle are also used in its kinematic model.

The alternative to kinematics is dynamics, which focuses on the forces and torques causing the motion of a machine to estimate acceleration and trajectories, as opposed to kinematics [18]. Dynamics tends to be much more computationally intensive than kinematics, as there are usually a number of forces and torques acting on a machine at any given time. On the other hand, dynamics is better suited than kinematics for estimating certain types of motion, such as lateral slipping or collisions between objects. On the whole, dynamic models can be more accurate than kinematic models in situations where accurate information is available on the interactions of a specific vehicle with its environment. However, in the generic case such information is not available, making kinematics more suitable. Dynamic models will not be covered in this work.

In this chapter, the kinematic models of the most common ground-based vehicle types used today are presented. Different mobility configurations are first introduced briefly to give a general look at the technology available. State-of-the-art research

on the kinematic models of the most common vehicle types is then presented. These kinematic models include common car-like vehicles with a given number of (steering) axles, center-articulated vehicles such as the ones used in mining operations by Sandvik (Fig. 1a), differential drive and skid-steered (Fig. 1c) or tracked vehicles. While the kinematic models of more specialized vehicles such as the straddle carrier with eight steerable wheels in Figure 1b will not be covered, their kinematics can be approximated e.g. using a model for car-like vehicles.

4.1 Mobility configurations

For ground-based industrial vehicles, there are three basic types of locomotion available: Wheels, tracks or limbs. These are in effect what define how the vehicle moves. They can be placed on the vehicle in different configurations that affect what trajectories the vehicle is capable of, how hard it is to control, and how well-suited it is for certain environments. In this chapter, several possible configurations are introduced.

Wheels are by far the most common method of locomotion in industrial vehicles. They are simple, reasonably robust and easy to use. Wheeled vehicles are often also more energy efficient and faster than tracked or limbed ones [19]. Wheels can be placed on a machine in a vast number of different configurations, ranging from the classic two-axle car-like configuration with one steering axle to different omnidirectional drive systems.

In industrial vehicles, the most common wheel configuration is to have the wheels placed on a number of axles, some of which are steering, that are symmetrical along the central axis of the vehicle. These car-like vehicles are often controlled using a combination of the Ackerman steering geometry and the bicycle model of kinematics. Ackerman steering ensures a single center of rotation for the vehicle and eliminates lateral slipping from its motion, thus improving control accuracy [20, p. 21]. For example, the straddle carrier in Figure 1b is in practice controlled using Ackerman steering even though it is not strictly car-like, as this reduces wear on its wheels. The bicycle model, on the other hand, simplifies the geometry of the vehicle and thus the resulting kinematic model [21, p. 67-69].

Some vehicles, such as the load-haul-dump truck in Figure 1a, use an articulated joint somewhere on their structure for steering instead of the wheeled axles. Steering in this manner gives the vehicle a tighter turn radius [22]. Other times, the joint may be passive, as with a vehicle pulling a trailer behind it. While the kinematic models for vehicles with actuated joints are relatively similar to those for car-like vehicles, passive joints are relatively dynamic in nature and vehicles that have them require more complex models.

Differential drive is another very common wheel configuration, often used by smaller robots and older AGVs. A differential drive machine has two wheels placed on a single axis. The wheels have independent motors, allowing them to be controlled separately. The trajectory of the machine is determined by the velocity differential of its two wheels [20, p. 20]. For example, differential drive vehicles turn by rotating one wheel faster than the other. Additional castor wheels may be used to improve

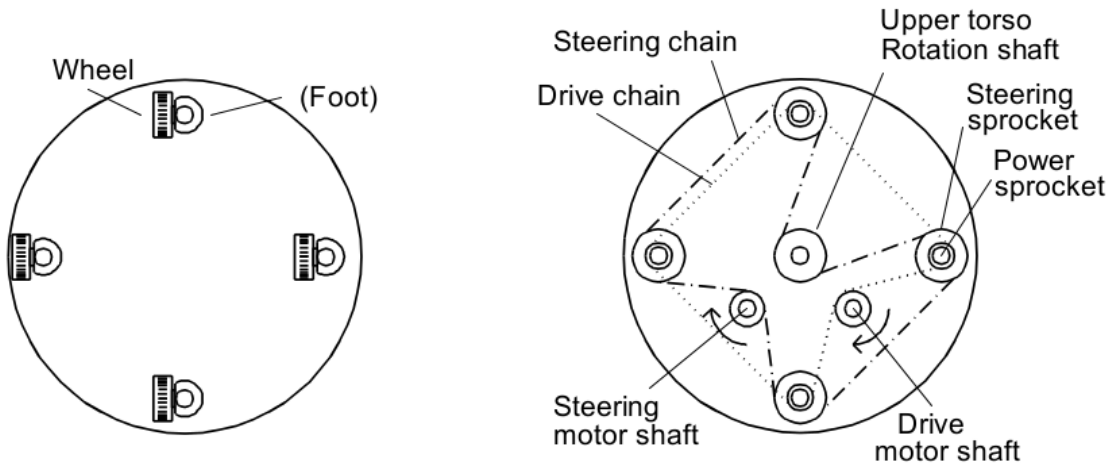


Figure 3: A synchronous drive system [20].

the stability of the machine.

Skid steered vehicles can be considered a variation of the differential drive configuration. Skid steering uses either a number of non-steering wheel axles or a set of tracks to move the machine; the wheels or tracks on either side of the vehicle are controlled separately. The main difference between these machines and differential drive systems is the increased ground contact area caused by the tracks or the larger number of wheels. This makes lateral slippage upon turning an inevitability for skid steered vehicles. Thus, skid steered vehicles are dynamic in nature and difficult to model accurately using kinematics [23, p. 49]. A skid steered vehicle is shown in Fig. 1c.

There are also a number of lesser used or more experimental wheel configurations, at least when it comes to mobile vehicles used in actual real-world engineering tasks. These include vehicles with synchronous or omnidirectional drive. As such configurations are mainly experimental in industry, they will be covered only briefly here.

Synchronous or synchro drive is a configuration where each wheel can both drive and steer [23, p. 43]. The wheels are mechanically coupled together so that they always turn and drive in unison. Synchro drives require a complex mechanical assembly of drive belts, for example, to achieve synchronization. This can be prone to faults or flaws such as degradation over time [20, p. 23]. A synchronous drive system is shown in Figure 3.

Omnidirectional drive is a highly maneuverable and increasingly popular type of locomotion for robots. Machines with omnidirectional drive have enough degrees of freedom to e.g. roll sideways without changing their heading. Controlling such a robot can be relatively easy, but it can also suffer from increased slippage. There are different ways to make a robot omnidirectional, from using special omnidirectional wheels that allow rolling in multiple directions [23, p. 47], to machines with multiple degrees of freedom that e.g. control each wheel separately [24]. The straddle carrier in Figure 1b is an omnidirectional vehicle even though it is not commonly controlled

Figure 4: KUKA youBot, an omni-wheeled mobile platform [27]

like one, as each of its wheels can be controlled separately. An example of an omni-wheeled robot is the KUKA youBot, shown in Figure 4.

Limbed locomotion has been perfected in nature over the course of hundreds of millions of years. From an engineering point of view, limbs have many attractive features, such as their ability to cope with rough, uneven terrain [25]. Unfortunately, limbs are a much more complex locomotion method than wheels or tracks. Vehicle stability becomes a much larger problem when using limbs [23, p. 49-51]. Different limb configurations have been researched for use in robotics, but currently their usage in mobile vehicles is limited. There has been some research on larger limbed outdoor vehicles, as e.g. with Mecant [25].

In addition to pre-existing forms of locomotion, entirely new types of movement are being developed for use in robotics. One such method is 'rolking', a hybridisation of wheeled and limbed locomotion [26]. The use of powered wheels gives the robot increased stability and speed, while having actuated, leg-like limbs as well allows it to move in difficult terrain or to climb over obstacles.

4.2 Ackerman steering and the bicycle model

Ackerman steering is a very common steering configuration used by virtually every car on the roads today. It has been in use at least since the 19th century [21, p. 69]. Ackerman steering is designed so that when turning, the inner steering wheels of a vehicle (the wheels closer to its center of rotation) are rotated at a slightly sharper angle than the outer ones. The angle of each wheel is calculated based on the steering angles so that when a line is drawn through the roll axis of the wheel, they all intersect at a common point. This is done to eliminate lateral tire slippage [20, p. 21]. However, while Ackerman steering is most commonly used in vehicles with only one steering axle (e.g. cars), it is valid for four-wheel steered vehicles as well [3]. With suitable control, it can also be used with vehicles that have more than two steering axles [28].

The Ackerman steering geometry for a vehicle with two steering axles is shown in Figure 5. Here, d is the axle width of the vehicle and l is its wheelbase. θ_F and θ_R are the steering angles for the front and rear axles. θ_{Fo}, θ_{Fi}, θ_{Ro} and θ_{Ri} are the Ackerman angles required for the outer and inner front and rear wheels respectively to make the vehicle follow the intended trajectory with no lateral tire slippage. The instantaneous center of rotation ICR of the vehicle is the point around which the vehicle travels on a circular path when there is no lateral slippage to its wheels, i.e. a point around which the motion of the vehicle can be described as a rotation with no translation. P_1 and P_2 are the midpoints of each axle. In this manner, each wheel in the vehicle lies tangential to concentric circles with their center points at ICR.

Assuming the steering angles for each steering axle, i.e. θ_F and θ_R, are known, the Ackerman angles for the outside and inside wheels of those axles can be solved geometrically [20, 28]:

$$l_{F/R} = r\tan(\theta_{F/R}) \tag{1}$$

$$\cot(\theta_{Fi\ /\ Ri}) = \cot(\theta_{F/R}) - \frac{d}{2l_{F/R}} \tag{2}$$

$$\cot(\theta_{Fo\ /\ Ro}) = \cot(\theta_{F/R}) + \frac{d}{2l_{F/R}} \tag{3}$$

where r is the perpendicular distance from the center of rotation at ICR to the central axis of the vehicle and l_F and l_R are the distances from the imaginary wheels to the point where r intersects the body. The perpendicular distance r can be solved e.g. by calculating the ICR in the coordinates of the machine using the well-known formulas for the intersection of two lines, in this case the extended imaginary axles from P_1 and P_2.

Ackerman steering is often used in conjunction with the so-called bicycle model of kinematics. In this model, the geometry of the vehicle is simplified by using imaginary wheels located on the central axis of the vehicle instead of the actual wheels as shown in Figure 5. Here the imaginary wheels are at P_1 and P_2. This approximation is accurate when used together with Ackerman steering, as the Ackerman angles ensure that the vehicle has a single center of rotation. Assuming that the velocity v and heading ϕ of the vehicle are known, the bicycle kinematic model for

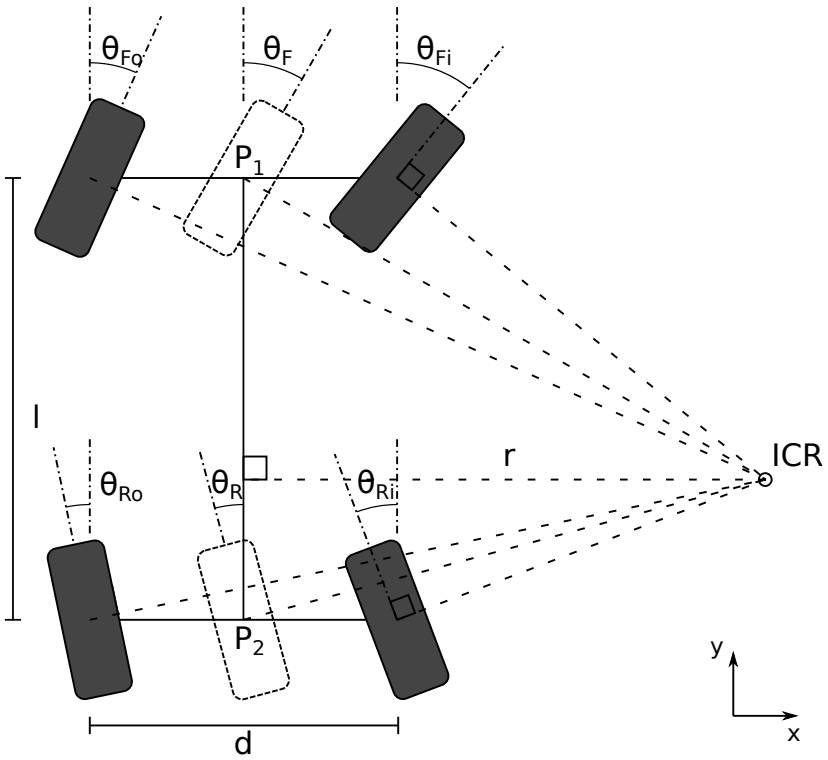


Figure 5: Ackerman steering geometry with a bicycle approximation for a vehicle with two steering axles. Adapted from [20, p. 22].

e.g. a vehicle with two axles as in Figure 5 becomes [3]:

$$\dot{x} = v\cos(\phi) \quad (4)$$

$$\dot{y} = v\sin(\phi) \quad (5)$$

$$\dot{\phi} = \frac{v}{l}(\tan(\theta_F) + \tan(\theta_R)) \quad (6)$$

The bicycle model has a nonholonomic constraint that states that the wheels of a vehicle can roll forward unimpeded but cannot slide laterally [21, p. 69]:

$$\dot{y}\cos(\phi) - \dot{x}\sin(\phi) = 0 \quad (7)$$

The bicycle model can be accurately used to estimate the kinematics of a vehicle with an arbitrary number of steering axles with some reservations. For vehicles with more than two axles, it is possible to have the axles intersect at more than one point, causing some amount of slipping in the vehicle [28]. In this case, the nonholonomic constraint is no longer observed. This can be prevented by using suitable control to ensure that the lines through the roll axis of the axles always intersect at a single common point.

4.3 Articulated vehicles

Articulated vehicles are basically divided into two or more parts by joints somewhere on their body. The joints can be either actuated, as with load-haul-dump (LHD) -trucks (e.g. Fig. 1a), or non-actuated, as would be the case e.g. with a tractor pulling a trailer. In the actuated case, the joint can be used to steer the vehicle instead of a steering axle as with car-like vehicles, giving such vehicles an improved turning radius over car-like vehicles [22]. The nonholonomic constraint in Equation 7 also applies to articulated vehicles [4].

Figure 6 shows a kinematic model of an LHD truck. Here, l_1 and l_2 are distances from the corresponding axles to the joint, P_1 and P_2 are their Cartesian coordinates, θ_1 and θ_2 are the headings of both vehicle sections, γ is the joint angle and r_1, r_2 and d are the distances from the axles and the joint to the center of rotation of the vehicle at *ICR* respectively. As is apparent from the figure, articulated vehicles can in the ideal case be treated as an extended type of bicycle model: with suitable control, the roll axes of each axle can be made to intersect at the same point, even if some of the axles are steering [29]. Scheding et al. [30] use a bicycle-like kinematic model for an articulated LHD similar to the one in Figure 6 where θ_2 is considered the heading of the machine. This gives the following kinematic model for P_2:

$$\dot{x} = v\cos(\theta_2) \quad (8)$$

$$\dot{y} = v\sin(\theta_2) \quad (9)$$

$$\dot{\theta}_2 = \frac{v\sin(\gamma)}{l_2 + l_1\cos(\gamma)} \quad (10)$$

Altafini [22] suggests an alternate model for the turn rate of the articulated vehicle:

$$\dot{\theta}_2 = \frac{v\sin(\gamma) - l_1\dot{\gamma}}{l_2 + l_1\cos(\gamma)} \quad (11)$$

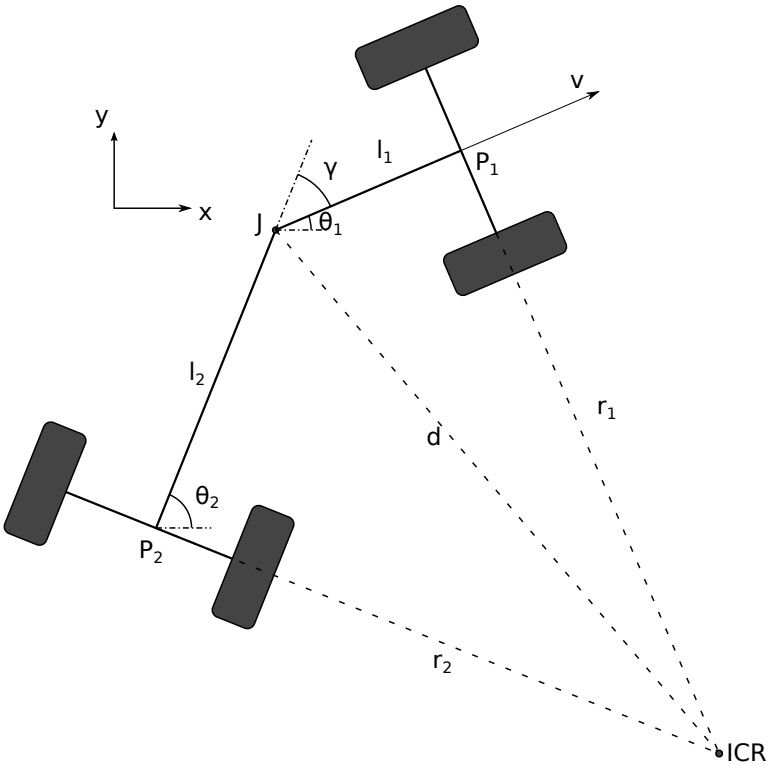


Figure 6: A kinematic model of an articulated load-haul-dump -truck. Adapted from [4].

Both of these models are for a vehicle that has a pair of axles and an actuated joint between them. An important difference between them is that in Scheding's model $\dot{\theta_2}$ is predicted to be zero when $v = 0$, meaning that in this model it would be impossible to articulate the vehicle while it is stationary [4]. In practice this is generally not the case if the joint is actuated. Because of the $\dot{\gamma}$ term in the numerator in Equation 11, the Altafini model allows for articulation even when stationary.

An articulated vehicle with more than two axles can also have more than one center of rotation as in Figure 7, where the tractor has one at ICR_1 and the trailer another at ICR_2 [31, 29]. As with car-like vehicles using the bicycle model, this breaks the nonholonomic constraint from Equation 7. If the joints of the vehicle joints are actuated, it is possible to prevent this with suitable control. In the case of a non-actuated joint, such control is not possible. Sampei et al. [31] seek to avoid this problem by focusing only on simple paths consisting of lines and arcs and by describing the kinematics of a tractor with a semi-trailer as a function of distance

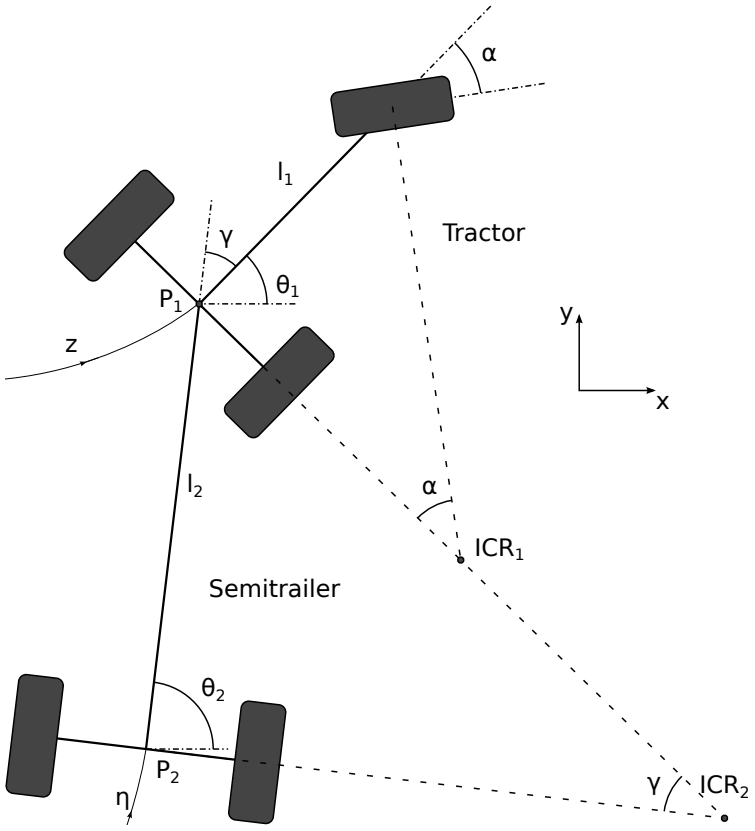


Figure 7: A kinematic model for a tractor with a semi-trailer connected by passive articulation. Adapted from [31].

travelled instead of time, giving the following for P_2:

$$\frac{d}{dz}(\gamma + \theta_2) = \frac{1}{l_1}\tan(\alpha) \tag{12}$$

$$\frac{dx_2}{d\eta} = \cos(\theta_2) \tag{13}$$

$$\frac{dy_2}{d\eta} = \sin(\theta_2) \tag{14}$$

$$\frac{d\theta_2}{d\eta} = \frac{1}{l_2}\tan(\gamma) \tag{15}$$

where z is the distance travelled by the tractor, η is the distance travelled by the trailer, α is the steering angle of the tractor and the other variables are as before. This model was originally intended for a very specific type of tractor-trailer-configuration, shown in Figure 7. A similar approach is used by Bolzern et al. [29] for a more generalized vehicle.

Another alternative for articulated vehicles with passive linkage is given by Larsson et al. [32]. They suggest a generalized kinematic model for a vehicle with an arbitrary number of axles and joints. The model for one link in a chain of axles and

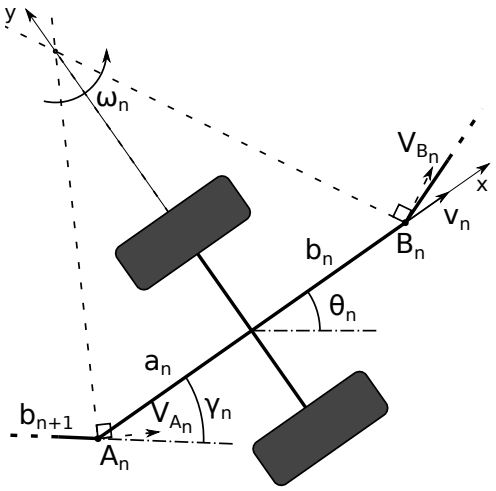


Figure 8: A generalized kinematic model for articulated vehicles. Adapted from [32].

joints is shown in Figure 8. Their model describes the kinematics of each axle:

$$\dot{x}_n = v_n \cos(\theta_n) \tag{16}$$
$$\dot{y}_n = v_n \sin(\theta_n) \tag{17}$$
$$\dot{\theta}_n = \omega_n \tag{18}$$

which gives the kinematic equations for the nth axle in the vehicle. ω_n is the angular velocity of the nth axle around its center of rotation, while v_n is its velocity. This model has a non-slipping assumption similar to Equation 7, because of which v_n is perpendicular to the wheel axis. The velocities can be calculated using [32]:

$$\begin{bmatrix} v_{n+1} \\ \omega_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & b_{n+1} \end{bmatrix}^{-1} \begin{bmatrix} \cos(\gamma_n) & -\sin(\gamma_n) \\ \sin(\gamma_n) & \cos(\gamma_n) \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -a_n \end{bmatrix} \begin{bmatrix} v_n \\ \omega_n \end{bmatrix} \tag{19}$$

where a_n is the distance from the nth axle to the next point of interest at A_n, i.e. a joint or the next axle. The distance from the next axle to B_{n+1} is b_{n+1}; B_{n+1} is the point of interest at A_n in the coordinate system of the $n+1$ axle. Finally, γ_n is the angle between nth axle and the $n+1$ one. In other words, if there is no joint between two consequent axles m and $m+1$, then $a_m = b_{m+1}$; if there is a joint, a_m and b_{m+1} are the distances from the axles to the joint. Each of these variables is in the coordinate system of that specific axle.

In general, there are several problems related to the control of articulated vehicles as opposed to other vehicle types, especially if the joints of the vehicle are not actuated. For example, a common problem when reversing with an actuated vehicle is the vehicle jackknifing at the joint [31]. However, because these types of vehicles are quite common in industrial applications, further research into their kinematics is important.

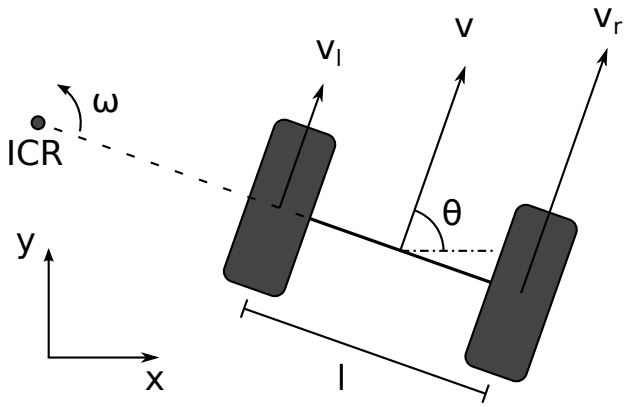


Figure 9: Differential drive kinematics. Adapted from [23, p. 39].

4.4 Differential drive

Differential drive is one of the simplest methods of locomotion used by mobile robots. Differentially driven robots use a pair of wheels mounted on a common axis but controlled by separate motors [23, p. 39]. Thus the wheels can be rotated at different speeds or in opposite directions from one another to move the robot in the desired manner. Because of this, differentially driven robots have a very simple control scheme that requires only the rotational velocities for each wheel as input. It is assumed that the wheels are ideal in nature, meaning that there is no lateral slipping and that they have exactly one point of contact with the ground. The basic geometry of a differential drive system is shown in Figure 9.

There are a number of other advantages to differential drive as well. The simple nature of a differential drive system makes it both easy and cheap to manufacture. Differentially driven machines can turn in place, which makes them an attractive solution for narrow or cluttered environments [23, p. 39]. For the same reason, the obstacle-free area around the robot can be easily calculated. However, the accuracy of the differential drive model is especially sensitive to variations in the ground plane, as a momentary loss of contact with the ground for just one wheel results in erroneous trajectory estimates [23, p. 40]. The ideal wheel assumption will also cause some inaccuracies, as in practice the contact area of the wheels with the ground varies dynamically. Finally, differential drive robots can only move bidirectionally [19].

In Figure 9, the forward velocity of the robot is v while the velocities for the left and right wheels are v_l and v_r respectively. The angular velocity of the robot around its center of rotation is ω. The *ICR* for a differentially driven robot depends upon its wheel velocities. The robot has a heading θ and the distance distance between its wheels is l. The kinematics for a differentially driven robot can be calculated using the assumption that the forward velocity of the robot is the average of its wheel

velocities [20, 23]:

$$\dot{x} = \frac{v_l + v_r}{2} \cos(\theta) \quad (20)$$

$$\dot{y} = \frac{v_l + v_r}{2} \sin(\theta) \quad (21)$$

$$\dot{\theta} = \frac{v_l + v_r}{l} \quad (22)$$

In practice, a robot with only two wheels would not be very stable. Because of this, differentially driven robots often make use of castor wheels for additional support, removing the need for dynamic balancing control [19]. In terms of kinematics, their effect on the motion of the robot is normally ignored [23, p. 40].

4.5 Skid steering and tracks

Skid steering is another relatively simple form of locomotion at least in terms of electromechanics. In principle it is similar to the differential drive configuration described in the previous chapter [33]. As with differential drive, skid steering uses two control inputs, the linear velocities of the left and right sides of the vehicle, to steer. However, unlike in differential drive, skid steering uses a number of powered wheels or tracks as opposed to the two on differential drive systems. One example of a skid steered vehicle is shown in Fig. 1c.

With differential drive, the assumption of a single point of contact for the wheels of the machine is usually adequate even with non-ideal wheels. With skid steered machines, on the other hand, the contact surface the machine has with the ground is much larger due to having many smaller separate contact areas from a larger number of wheels or a few large ones from tracks [34]. Different wheel or track configurations slip differently, further increasing the difficulty of accurately estimating skid steer systems.

A skid steered vehicle can be approximated using the differential drive kinematics (Equations 20-22). However, this ignores several important differences between skid steered and differentially driven vehicles, chiefly that because of the larger ground contact area of skid steered vehicles, they always have noticeable lateral slipping when turning. Because of these differences, motion estimates made of skid steered vehicles using differential drive kinematics tend to be inaccurate [23, p. 48]. Kinematic models for skid steered vehicles are in general difficult due to complex dynamic interactions with the environment of the machine. Error sources range from how slippery or soft the ground is to the state of the wheels or tracks of the vehicle [33].

Figure 10 shows a kinematic model for skid steered vehicles proposed by Nagatani et al. [35]. Comparing it with the differential drive model in Figure 9 the similarities are obvious. The main difference is the inclusion of the slip angle α caused by the lateral slip of the vehicle. Because of lateral slipping, the center of rotation of the vehicle is not necessarily on the central axis of the vehicle, nor is the direction of motion always the same as the heading of the vehicle [33]. The center of the machine moves at velocity v_c in the direction $\theta - \alpha$. The kinematic equations for the model

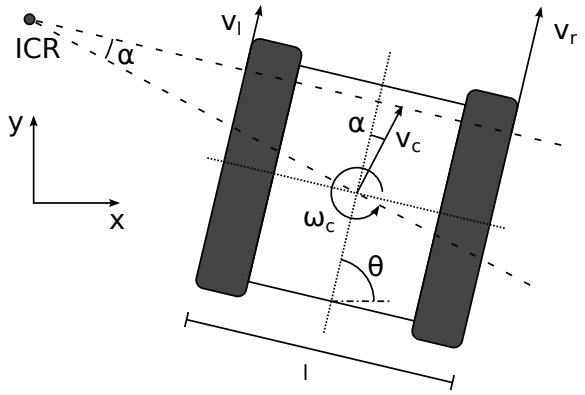


Figure 10: A kinematic model for skid steered vehicles. Adapted from [35].

in Figure 10 expand the ones for differential drive [35]:

$$a_l = \frac{v_l - v_l'}{v_l} \tag{23}$$

$$a_r = \frac{v_r - v_r'}{v_r} \tag{24}$$

$$\dot{x} = \frac{v_r(1-a_r) + v_l(1-a_l)}{2\cos(\alpha)} \cos(\theta - \alpha) \tag{25}$$

$$\dot{y} = \frac{v_r(1-a_r) + v_l(1-a_l)}{2\cos(\alpha)} \sin(\theta - \alpha) \tag{26}$$

$$\dot{\theta} = \frac{v_r(1-a_r) + v_l(1-a_l)}{l} \tag{27}$$

where a_l and a_r are slip ratios for the left and right tracks, v_l and v_r are their theoretical velocities determined from the angular velocities and radii of the sprockets of the tracks, and v_l' and v_r' are the ground speeds of the tracks. The axle width of the vehicle is l. Nagatani et al. reached considerably accurate estimates for trajectory and orientation using this method although the slip angle α has to be estimated separately [35].

A different approach to calculating skid steering kinematics is presented in [33], focusing on the ICRs of the skid steered vehicle instead of its slip angle. A diagram for this model is shown in Figure 11. The left and right sides of the vehicle have their own ICRs because they can be modelled as separate rigid bodies. The entire vehicle is a rigid body following a circular path around ICR_v. Each ICR is on the same line parallel to the X axis of the vehicle. The kinematic model of the vehicle

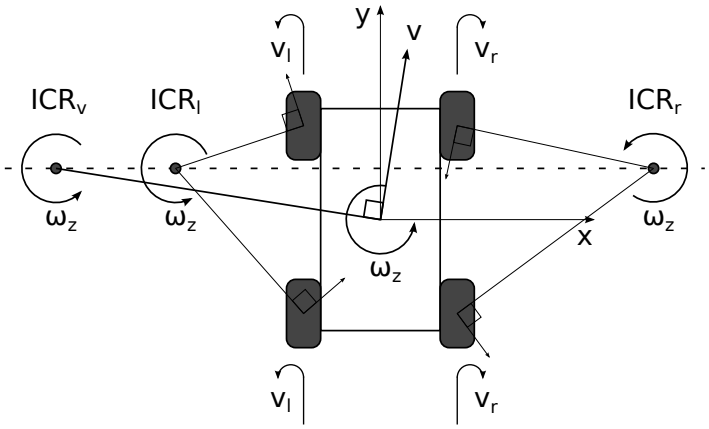


Figure 11: The instantaneous centres of rotation (ICR) of a skid steered vehicle. Adapted from [34].

is calculated using the coordinates of the ICRs in the local frame of the vehicle:

$$x_{ICRv} = \frac{-v_y}{\omega_z} \tag{28}$$

$$x_{ICRl} = \frac{v_l - v_y}{\omega_z} \tag{29}$$

$$x_{ICRr} = \frac{v_r - v_y}{\omega_z} \tag{30}$$

$$y_{ICRv} = y_{ICRl} = y_{ICRt} = \frac{v_x}{\omega_z} \tag{31}$$

where (x_{ICRr}, y_{ICRr}) and (x_{ICRl}, y_{ICRl}) are the coordinates for the left and right ICRs, v_x and v_y are the velocities of the vehicle along the corresponding axes in the local frame, v_l and v_r are the rolling speeds of the left and right sides of the vehicle and ω_z is its angular velocity. Using these coordinates, the kinematics can be estimated [33]:

$$\dot{x} = \frac{v_r - v_l}{x_{ICRr} - x_{ICRl}} y_{ICRv} \tag{32}$$

$$\dot{y} = \frac{v_r + v_l}{2} - \frac{v_r - v_l}{x_{ICRr} - x_{ICRl}} \left(\frac{x_{ICRr} + x_{ICRl}}{2}\right) \tag{33}$$

$$\dot{\theta} = \omega_z = \frac{v_r - v_l}{x_{ICRr} - x_{ICRl}} \tag{34}$$

Skid steering intentionally relies on dynamic slippage caused by its operating principle. Due to this inherently dynamic nature, it is challenging to create accurate kinematic models for skid steered vehicles. Research into the subject is still very much ongoing, as shown by the work done by Martinez et al. [33] and Nagatani et al. [35] among others. Alternatively, the inaccuracies inherent to skid steered vehicles can be compensated for by partially teleoperating the vehicles [20, p. 28].

5 Generic mobile machine definition software (MaDe)

A software tool for defining generic mobile machines is one of the two main results for this thesis work. The tool, named Machine Definer (MaDe), allows a user to quickly construct an abstract representation of a vehicle out of simple geometric building blocks. A kinematic model for the vehicle can then be generated based on the representation. This in turn can be further refined into a Stage simulator model, producing a fully functioning simulated vehicle in a short time. As the kinematics themselves depend mainly on certain key parameters such as axle width and wheelbase, a simple, low detail, highly abstract model is entirely sufficient to deduce them. As such, the main purpose of MaDe is the rapid definition of the key kinematics parameters of a vehicle, while the actual kinematics calculations are handled by external software, in this case StageMaCI. StageMaCI will be introduced in Chapter 6.

MaDe was created in its entirety during this thesis work and was programmed using C++. It uses the open source wxWidgets library [36] for windowing and OpenGL [37] for graphics rendering. In addition, it features an implementation of the Command behavioral design pattern [38, p. 233] to allow certain functionalities. This chapter describes the structure of MaDe and its various components, functionalities and features.

5.1 Overview

The MaDe program window consists mainly of a large view area surrounded by various tools. In design mode, the view screen is split into two equally sized, orthographic top and side views showing the same section of the workspace from two different viewpoints. MaDe also has a perspective view mode, which allows the user to see a three-dimensional view of the current vehicle model from a movable viewpoint. The user can move the camera or zoom in and out in both view modes. MaDe's main design view is shown in Figure 12 and perspective view in Figure 13.

In its current version, MaDe comes with 11 different tools. In order from top to bottom (see the panel on the left-hand side in Figure 12), they are the selection, move, box selection, box, polygon, polygon modification, axle, track, joint, sensor and link tools. Not shown are MaDe's various drop-down menus, through which models are saved, loaded and exported, objects are copied, pasted or deleted, and various features such as grid lines or whether to draw objects solid are toggled. The color palette in the lower left corner of the screen allows the user to set the color of objects while the information bar below it shows the current XYZ-coordinates of the mouse cursor. The program is mainly mouse operated though many actions also have keyboard short-cuts. Actions such as copying, pasting, saving and undoing use the same common short-cuts that hundreds of other software use while different tools can be quickly selected with the number keys. The tool set in MaDe fulfils the Requirements R1–R4.

The selection and move tools work very similarly to each other. They can both be used to move the camera in the design view or to select individual or multiple

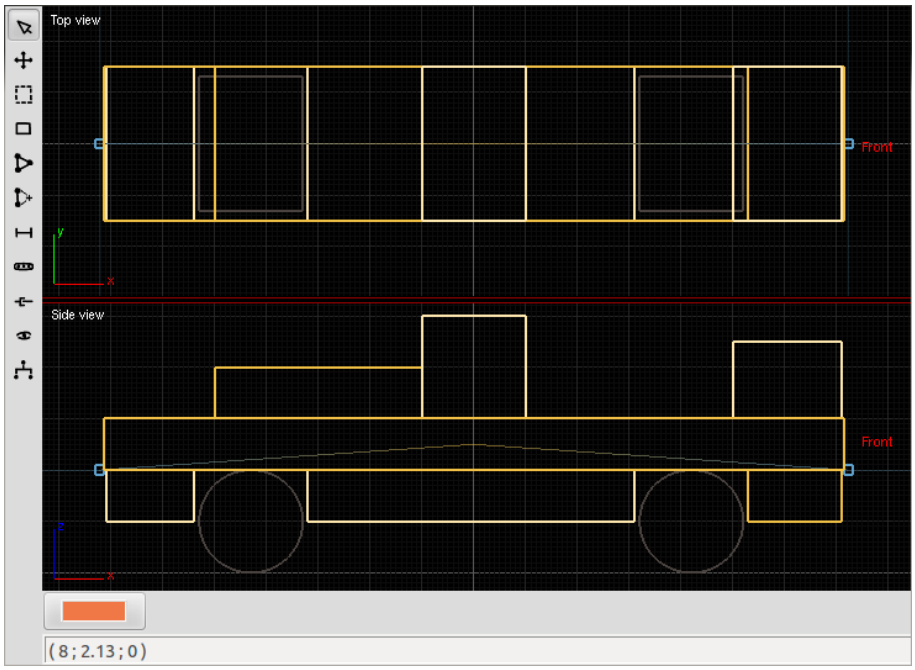


Figure 12: MaDe's main design view consisting of orthographic top and side views, with a toolbar on the left-hand side. The pictured vehicle is used as an example throughout this thesis.

objects. Selected objects or object groups can be moved or resized using a tool bar that appears in the bar below the view screen once an object is selected. The move tool can additionally be used to drag objects around the screen. When an object is moved, any objects linked to it are moved as well, though this can be modified to include only the ancestors, descendants of the object or neither with the shift and control meta keys. The box selection tool is essentially a bulk version of the selection tool, allowing the user to drag a three-dimensional box in the design area. Any objects inside the box will be selected.

The box, polygon, axle and track tools are all used to create the main structural parts of a vehicle. The box and polygon tools create objects that are considered the body of a vehicle while axles and tracks are its means of motion. Boxes, axles and tracks are created by clicking and dragging on either of the design view ports, whereas polygons are created point by point on the top view port. For the sake of simplicity, polygons are limited to 2D surfaces extruded along the Z axis to a certain height. Boxes and axles are geometric primitives, with axles being cylinders with their length along the Y axis, whereas tracks are rounded boxes. Axles also have operational parameters for whether they are steering or not, allowing the user to define their maximum turn angle and rate.

Like its name implies, the polygon modification tool is used to modify existing boxes and polygons. With it, new vertices can be added to polygons and existing ones can be moved on the XY-plane or removed. If the removal of a vertex reduces the number of vertices in a polygon to less than three, the polygon is removed

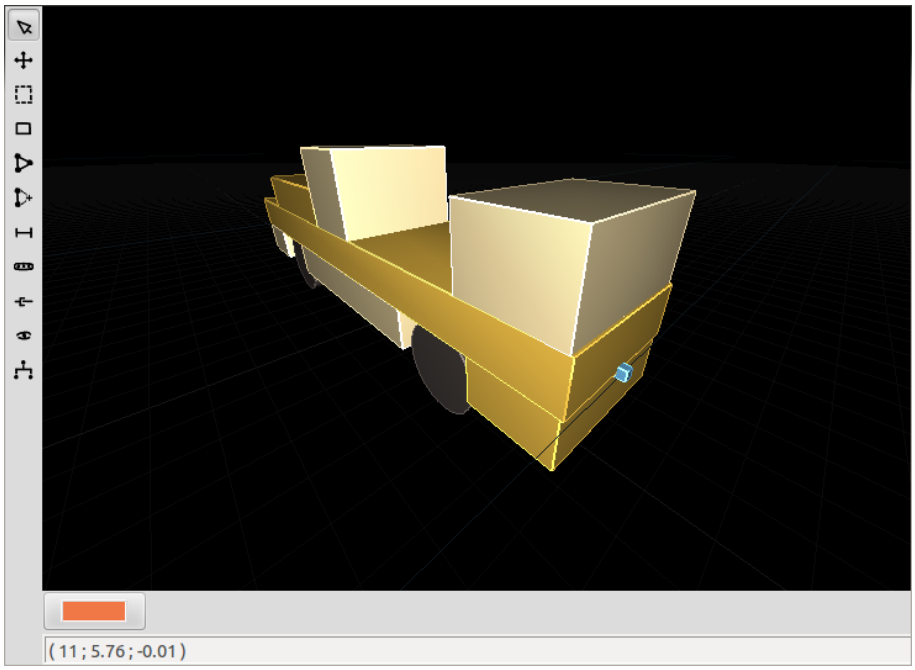


Figure 13: MaDe's perspective view.

entirely.

Both the joint and range sensor tools require a pre-existing vehicle body (i.e. boxes or polygons) before they can be used. The joint tool adds a hull joint to the vehicle body, splitting it along the joint position into two polygonal segments. The segments are linked to the newly-created joint object so that a tree structure is formed with joints forming each non-terminal vertex and objects as the leaves. This makes it easy to divide the vehicle into sub-models for simulation purposes. The joints can be actuated or not, and in the actuated case the user can set parameters for their maximum turn angle and rate similar to axles. Also similar to axles, joints are depicted as cylinders with their length along Z.

Range sensors are a special type of box object with parameters for the facing direction, range and field of vision. The default dimensions of the sensors are based on commonly used laser range scanners. Sensors can be attached to anywhere on the vehicle body, i.e. on any box or polygon. For example, the vehicle in Figure 12 has two light blue sensors at its front and rear attached to its central body object as indicated by the lines representing linkage. While sensors serve no function from a kinematics point of view, they were included in MaDe as they are a fundamentally important tool in robotics. As models generated with MaDe are used in simulations, support for range sensors allows the models to be used in e.g. mapping and localization tools.

Finally, the link tool allows users to manually edit the linkage between different objects. It can both add and remove connections to structure the object tree as the user sees fit. To maintain tree validity, it is not possible to link two objects if a relation between them already exists through their ancestors or descendants.

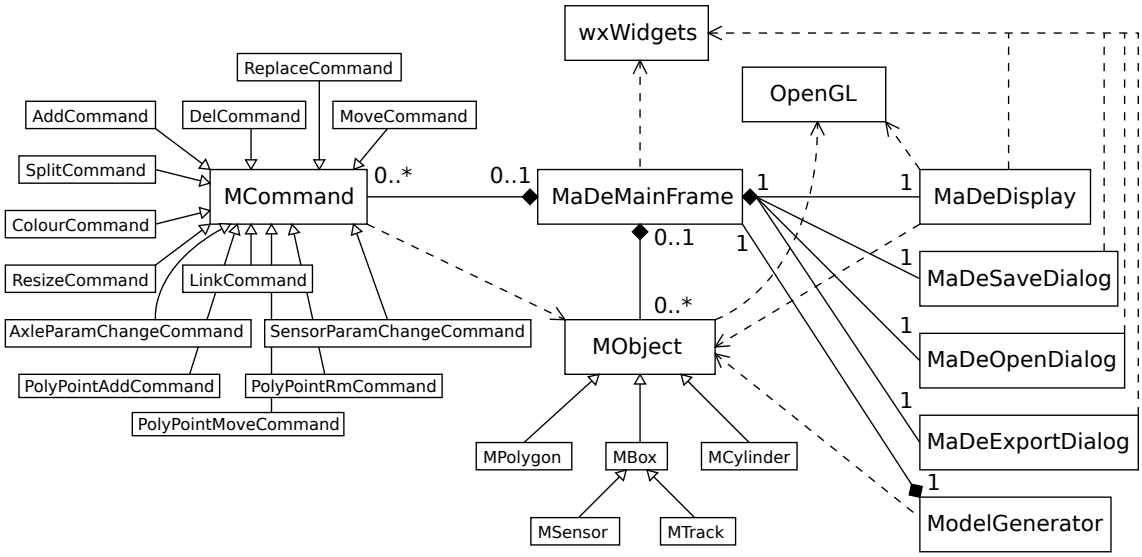


Figure 14: MaDe's class structure.

5.2 Program structure

MaDe consists of a number of classes that together implement its various functionalities. Its class structure is shown in Figure 14. The main program window class, MaDeMainFrame, keeps track of all the commands given by the user and the objects currently on the canvas. Different object types are implementations of the MObject virtual object class, while commands are similarly implementations of the MCommand class. MaDe consists of the following classes:

MaDeMainFrame: The main window class in MaDe, implementing the wxFrame class of wxWidgets. Keeps track of all the objects on the canvas as well as undo and redo stacks of commands given by the user. Contains an instance of an OpenGL canvas widget and calls different saving, loading and exporting features through its menus.

MaDeDisplay: An implementation of the wxWidgets wxGLCanvas widget, it handles all graphics rendering operations in its own process thread. Depending on view mode, it renders either the design or the perspective view. It calls the rendering function for each object and handles any event callbacks for the keyboard and the mouse not handled by MaDeMainFrame.

MObject: A virtual base class for objects, i.e. vehicle parts. The base class implements functions related to object colour and position handling as well as object linking. The position of an object is defined as its center coordinate. Each object knows its parent and any children connected to it. Object type specific rendering and dimension calculation functions are implemented in the inheriting classes. Because it is possible for a single object class to be used to represent different parts, each object has a type attribute to specify their actual function. The base class also contains a unique identifier for each object. The current version of MaDe has the following object types:

MBox: The simplest object type, it represents a cuboid defined by length, width and height values along the X, Y and Z axes respectively.

MSensor: A special type of box object, extending it with range, field of vision and facing.

MTrack: A special type of box object representing a pair of tracks. In essence, it is a box object with rounded ends. It has the same parameters as a box object, but implements a custom render function.

MCylinder: A cylindrical object type. It is used to represent both axles and joints. Cylinders are defined by their height and radius as well as the axis along which they extend, defined by a rotation value about the X axis. Cylinder objects have parametrised values for whether they can be steered as well as their maximum turn rates and turn angles.

MPolygon: Polygons are a more complex type of object for depicting the body of a vehicle. Polygons mainly consist of a list of coordinate points for each of their vertexes and a height value. MPolygon also implements a range of helper functions to add or remove points or to calculate a triangulation of the polygon upon its creation or when its points are modified.

MCommand: Similar to MObject, MCommand is a virtual base class for various commands recognized by MaDe. The command classes are a part of MaDe's implementation for the command pattern. Their main purpose is the implementation of an undo/redo-functionality. Every command that can be undone has its own command class, instances of which are added to undo/redo stacks in the main frame object instance when the commands are called. Each command contains a vector of the objects it affects. The command pattern and the structure of commands are covered more thoroughly in Chapter 5.4. Currently, the different commands are:

Add and Del: Two commands that complement each other, AddCommand is used to add new objects to the design canvas, while DelCommand is used to remove existing ones.

Move: MoveCommand is used to move objects by a given amount in any direction.

Replace: ReplaceCommand swaps an object on the design canvas with another.

Split: This command takes a coordinate point as one of its parameters. Affected objects are split into two new polygons along a plane that intersects that point.

Colour: A command that changes the colour of affected objects.

Resize: This command takes length, width and height ratios as its parameters and resizes all affected objects accordingly. When resizing a group of objects, the objects retain their relative positions.

Link: A command used to link or unlink two objects together. It first ensures that the two objects are not already linked before linking them.

Axle and SensorParamChange: Two very similar commands used to change the parameters of axles or joints in the first case and sensors in the second.

PolyPointAdd/Rm/Move: These commands are all related polygon modification, either adding, removing or moving vertices on a polygon.

MaDeSaveDialog: An implementation of wxDialog, this class represents a pop-up window for saving a vehicle model. The user chooses a file name and location, as well as a file format for the saved model file.

MaDeOpenDialog: An implementation of wxDialog, this class represents a pop-up window for opening an existing vehicle model. The user chooses a file of a supported type and MaDe attempts to read a vehicle model from it.

MaDeExportDialog: An implementation of wxDialog, this class represents a pop-up window for exporting a model as a simulator model. The user selects a file name and location, as well as setting some parameters such as top speed and acceleration for the vehicle.

ModelGenerator: ModelGenerator reads the object data from an instance of MaDeMainFrame and generates a Stage simulator model of a vehicle based on it. The model is written into a specified file with certain parameters gained from an instance of MaDeExportDialog used by the main frame.

To make the vehicle model easier to perceive in the perspective mode, MaDe uses the lighting features of OpenGL. For each planar surface, e.g. the sides of a box, a surface normal is calculated. OpenGL uses the normal vector to efficiently deduce how light falls on the surface when it arrives from a given source. In MaDe, the light source is always at the location of the perspective camera. Objects are rendered using the built-in functions of OpenGL for geometric primitives though polygons and the round areas of cylinders and tracks require construction from multiple triangle primitives. For circular surfaces, this is a simple matter of rendering a suitable number of triangle primitives arranged in a circle. Polygons, on the other hand, require triangulation to find the triangles making up their structure. The triangulation algorithm used in MaDe is described in the next section.

Object selection also utilizes OpenGL by writing the ID value of each object to each pixel of the stencil mask of OpenGL covered by the object. When the user tries to select an object, the value in the stencil mask at the mouse coordinates is retrieved and set as the active object.

The two viewpoints shown in the design view (see Figure 12) show orthographic projections of the top and side of the model. While these views may appear two-dimensional, they are in fact complete three-dimensional renders of the vehicle. As the vehicle is thus rendered twice, the design view may become computationally heavy with extremely complex models. However, it was determined that with

MaDe's current settings on a regular office computer, MaDe can render hundreds of objects without significant slowdowns. Similarly, having to calculate lighting effects on a large number of multi-faceted objects may slow down the perspective view. To help MaDe run on slower computers, lighting can be disabled. MaDe's computational requirements were also reduced by e.g. reducing the number of facets on round objects such as axles. Currently, cylindrical objects are in most cases the most computationally intensive objects supported, as each cylinder is rendered by drawing a large enough number of facets to appear circular.

5.3 Polygon triangulation using ear clipping

Polygons can be highly complex structurally and thus cannot be trivially rendered using simple geometric primitives the same way e.g. boxes can. Instead of drawing a polygon as a single solid shape, it needs to be divided into triangles which are then in turn rendered on-screen, filling the polygon shape out. In MaDe, this triangulation is done using the ear clipping method [39].

Ear clipping is relatively simple and effective, provided that the polygons that are triangulated are not self-intersecting or degenerate as defined in [39]. While these days triangulation algorithms faster than ear clipping exist, MaDe is used on regular computers and with such highly abstract, simplified shapes that it was assumed that computational speed was not an issue. An example of triangulation by ear clipping is shown in Figure 15.

The ear clipping algorithm begins by picking three consecutive vertices, v_{i-1}, v_i and v_{i+1}, from the polygon. These vertices form an ear, i.e. one of the triangles used to fill the polygon, if:

1. v_i is a convex vertex, i.e. if the internal angle formed by these vertices is less than 180°

2. the segment consisting of these vertices does not intersect any of the edges of the polygon

3. a triangle formed from these vertices does not contain any of the other vertices of the polygon

If the segment $< v_{i-1}, v_i, v_{i+1} >$ is an ear, the vertex v_i (the ear tip) is removed from the polygon and another set of vertices is selected from the reduced polygon. If it is not, i is incremented to select the next three consecutive vertices. This is repeated until it is no longer possible to form new triangles from the polygon. In a valid, non-self-intersecting and non-degenerate polygon, this will always produce a triangulation of the polygon. The triangulation results depend to some extent on which vertex of the polygon the algorithm is started from. [39]

In the case shown in Figure 15, the polygon in question is a star shape with ten vertices. The algorithm starts from the vertex labelled A at the top point of the star and proceeds from there in the clockwise direction. The first possible triangle is therefore $\triangle ABC$, which however is not a valid choice as it is not inside the star

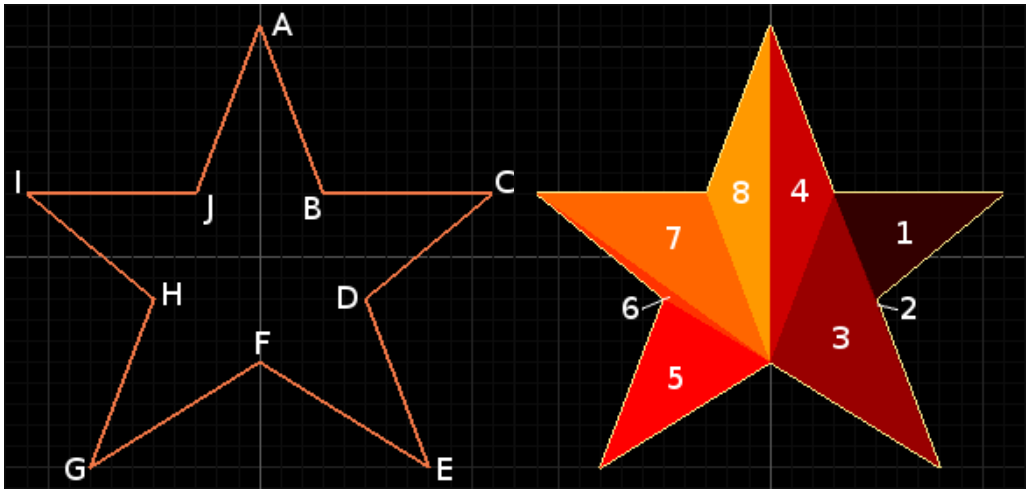


Figure 15: A polygon and one possible triangulation using ear clipping.

polygon. The algorithm moves on and considers the triangle $\triangle BCD$ next. This time, all the requirements for an ear are met. The vertex C is removed from the list of unused vertices and $\triangle BCD$ is marked as one of the triangles making up the triangulation of the polygon. The algorithm resets back to A, but the first triangle of this round, $\triangle ABD$, is again invalid. The next ear to be found is $\triangle BDE$, even if it is flat enough to be indiscernible from the figure. Thus the algorithm goes on, eliminating vertices one at a time, until it finishes with the ear $\triangle AFJ$, producing eight distinct triangles.

As triangulation is a relatively expensive operation, MaDe tries to perform it on each polygon as few times as possible. Polygons need to be re-triangulated every time points are modified, added to or removed from them. MaDe allows polygons to be modified using a specific tool.

5.4 Command pattern

The Command pattern is a behavioral design pattern, where all the information necessary to perform an action is encapsulated in an object [38, p. 233]. This allows the action to be performed at a later time, repeated as often as necessary or, as in MaDe, to perform undo and redo operations for specific commands. The pattern functions by using so-called command, receiver, invoker and client objects.

The command object is a container for the action to be done itself. It knows all the parameters necessary to perform a certain action within the program. As is shown in Figure 14, in MaDe every action that can be undone has its own corresponding MCommand object class. The command object also has a pointer to a receiver object, which is in practice the target of the action the command performs. Commands consist mainly of their parameters and an execute function that performs a specific set of actions and function calls on the receiver object using said parameters. To undo the command, it also has an unexecute function that performs the exact opposite actions on the receiver to revert it to its state prior to the exe-

cution of the command. Depending on the type of the command, this may require storing additional data of the initial state of the receiver in the command object. [38]

Representing each reversible action with a separate command class is also the downside of the pattern. As the number of functions that can be undone increases, so too does the number of command classes required. This is a fairly negligible problem, as using inheritance and pointers to the base command class ensures proper functioning. However, this does complicate the required program code and its upkeep.

In MaDe, most of the commands can have multiple receivers, which are different MObject instances (Figure 14). This allows the user to e.g. move multiple objects around the design canvas or to recolour a batch of objects at once. This also makes it easier to e.g. maintain the relative positions of different objects during a resizing.

The invoker is the object that at some point during the execution of the program creates new commands and asks for them to be executed. It is often the same object as the client object, which is usually that part of the program which contains the receiver [38]. The terminology for the command pattern is somewhat vague for these objects, as clients and invokers are at times used as synonyms for each other. In MaDe's implementation of the Command pattern, the client is always the main frame object of the program while the invoker is either the main frame or its OpenGL canvas component.

MaDe stores commands in undo and redo stacks. When a command is executed, it is pushed onto the undo stack, making it the first to be undone. Once this happens, the command is moved from the undo stack to the redo stack. The redo stack is emptied if a new command is executed while there are commands in it. Thus the Command pattern functionality gives MaDe a multi-level undo-redo-mechanism that can remember a very large number of recent actions performed by the user. This fulfils Requirement R5.

5.5 Saving and Exporting

After considering the alternatives (see Chapter 3.2), it was decided that MaDe shall initially support the XML-based X3D and SVG as its chosen formats for storing vehicle models. Both are open source file formats and especially SVG is in widespread use. A number of open-source editors are available that support them. Both of them also feature simple, easy to understand syntax. This makes models created with MaDe easily portable and modifiable. Using these formats meant that only minor modifications needed to be made to them to correctly store the models. The modifications will not affect the functioning of the files in software other than MaDe.

Saving vehicles in the X3D format is very straightforward. The vehicle model is represented by a single X3D Scene element. In X3D, a Scene contains all the relevant information for a specific X3D model ranging from individual objects to lighting settings. As each geometric shape supported by MaDe is also supported by the X3D schema, storing shape data is trivially easy.

Objects in X3D form a branch in the XML element tree. At the lowest level

of the branch is the three-dimensional representation of the object, such as a box, a cylinder or, for polygons, an extruded cross section. These are defined by their attributes, such as height or a list of points for the cross section. The geometric form of the object is coupled with an Appearance element that defines its color and material. Together they form a Shape element. Shapes are positioned on the canvas by appending them as children to Transform elements consisting of a translation and a rotation along the XYZ axes. Finally, on the highest level of the branch is the Group element, which can be used to link a number of different shapes. Each Group is named using the unique identifier of the corresponding vehicle part, ensuring that upon loading from the save file, the loaded parts have the same ID as the original ones. The type of an object is preserved using the class attribute for Group elements.

Because of this high level of compatibility, the only additions to the X3D schema required to store a vehicle model are the properties of special parts such as axles, joints and sensors. They are written as attributes for the Group element corresponding to the object. Also, links between objects are stored by giving the Group element a parent attribute, with the unique identifier of the parent as the value of the attribute. While these attributes are not recognized as valid X3D, they should not cause any errors in other software. Usually, software that does not recognize certain XML attributes, for example, will simply ignore them instead. X3D also has a number of features not supported by MaDe, such as scene lighting. If MaDe encounters any unfamiliar elements or attributes, it simply ignores them.

The resulting X3D file can be imported as is into Blender, for instance. However, it is important to remember that not all programs handle data importing in the same way. Blender, for example, seems to discard any data it does not use such as the ID values, and X3D files exported from Blender do not use the same X3D standard features to e.g. export box primitives as MaDe. Thus, while it is possible to view and even edit vehicle models made with MaDe, it may not be possible to load the modified models back into MaDe. A X3D representation of the example vehicle used in e.g. Figure 12 is shown in Appendix A.

As mentioned in Chapter 3.2, the two-dimensional vector graphics format SVG can also be used to represent a three-dimensional object. In MaDe, this is done by describing each vehicle part with two SVG graphic objects, one showing the part from the top, the other from the side. Thus any three-dimensional object can be described using two two-dimensional ones; For example a cylinder can be described with a circle and a rectangle. The end result is a vehicle shown from two viewpoints similar to MaDe's orthographic design view in Figure 12. The two SVG objects are grouped together and given the ID and type of the part as identifiers. The two views are separated from each other along the Y axis of the SVG image by a known value to make the resulting image more informative for the human viewer.

Structurally the SVG file is very similar to its X3D counterpart. As with the X3D format, SVG supports MaDe's part types well right from the start. The only non-standard SVG attributes required are the axle, joint and sensor parameters and the ID of the parent object. The resulting SVG files are actually simpler than the X3D ones, as the same vehicle can be described using far fewer XML elements without losing any information. However, this approach requires that certain factors

such as the magnitude of the separation between the two views are known. This can be problematic when attempting to modify the files with software other than MaDe.

An example SVG image representation of a vehicle model, generated by MaDe from the vehicle used throughout this thesis, is shown in Figure 16. The SVG presentation of the same vehicle is shown in Appendix B. While there is some unwanted overlap between different parts of the vehicle, the vehicle itself is immediately recognizable and because of the widespread use of SVG, can be viewed e.g. in a web browser. The SVG image can also be freely scaled, due to it being in a vector graphic format. The only restriction to scaling is that MaDe assumes a certain scale ratio is used when it opens an SVG file. Similar to the X3D files, the SVG files created by MaDe can be viewed and edited in other software, but doing so may make them incompatible with MaDe.

Currently, vehicles can only be exported as models for the StageMaCI simulator. In practice, this means that MaDe generates a Stage model file for the vehicle, defining the structure of the vehicle in Stage's own syntax. Such a model file is required by Stage for each object in the simulation. Because of this, the SVG or X3D save files cannot be used in the simulator software as they are. Stage model generation is not a trivial, straightforward operation, as Stage does not normally support e.g. articulated vehicles or parametrised, steering axles. However, once the Stage model file has been created, users need only to include it in a Stage world file and create an instance of it to simulate the vehicle.

To simulate vehicle articulation, the vehicle model needed to be split into a tree-like structure of sub-models in the same way that MaDe splits jointed objects. Each non-terminal vertex of the sub-model tree is a joint, whereas the body, wheels and other components of the vehicle form larger sub-models at the leaves of the tree. This makes it possible to manipulate individual sections of the vehicle in StageMaCI according to a kinematic model. Axles and sensors are exported as sub-models regardless of whether the model has joints or not, as they have their own special functionalities in the simulator software and use specific Stage features. A Stage model file for the example vehicle is shown in Appendix C.

To pass various machine and part parameters on to StageMaCI, MaDe creates an additional parameter file upon exporting. The file contains the parameters for each of the axles and joints of the vehicle, ranging from whether they can steer to their maximum angle and turn rate. It also specifies a top speed and acceleration for the vehicle. Parameter files use regular comma-separated value (CSV) syntax.

The saving, loading and exporting functionalities in MaDe fulfil the Requirements R6–R8. At this point, it has been demonstrated that MaDe fulfils all of the requirements set for it in Chapter 2.

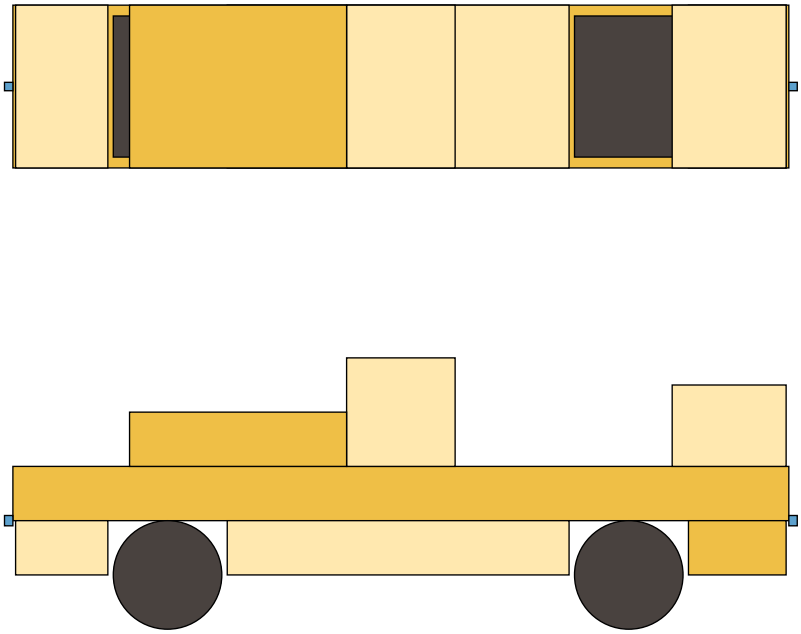


Figure 16: An example SVG image of a MaDe vehicle model.

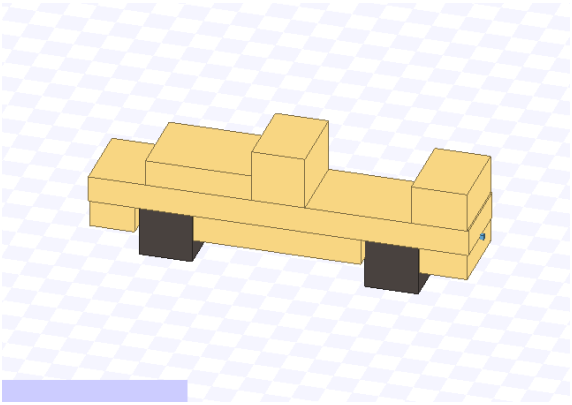


Figure 17: A vehicle defined with MaDe in the Stage simulator environment.

6 Model of a generic mobile machine

The machine definition tool MaDe is meant to be used in conjunction with the modified version of the Stage simulator environment called StageMaCI or Motti Simulator. While the ground work for StageMaCI was done in [40], this thesis contributes to it with the creation of a simulator model for generic mobile machines, which is in practice a generic kinematic model. In essence, the generic model is a single C++ class extension to the StageMaCI software. The model is used to estimate the kinematics for the vehicles defined in MaDe. This model is one of the main results of this thesis work alongside the MaDe program. A simulated version of a vehicle defined in MaDe is shown in Figure 17.

A separate, more general purpose module for generating kinematic models for vehicles defined in MaDe is currently in development. The module will allow the kinematic models to be used even without StageMaCI. In practice, it will be a standalone generic kinematic model class that will make use of the model files generated by MaDe to calculate kinematics based on velocity and angular velocity given as inputs. It will use the same kinematic models as the generic StageMaCI model.

Like MaDe, the generic kinematic model is a part of the tool chain of the Motti project. With the help of MaDe, it can be used to quickly generate kinematic models for arbitrary vehicle configurations. Kinematic models are useful in a wide range of tasks from the generation of viable routes for a vehicle to task planning, mission control and overall system simulation. As defining kinematic models by hand can be time consuming, having a generic one can also reduce the time required to deploy an automated vehicle fleet.

In this chapter, the generic mobile machine model, the algorithms it uses and the pre-existing software it depends on are presented in detail. First, a brief introduction to both the GIMnet communications framework and the machine control interface MaCI is given. A second brief introduction is given to the StageMaCI simulator environment. Finally, an in-depth description of the generic mobile machine model is presented.

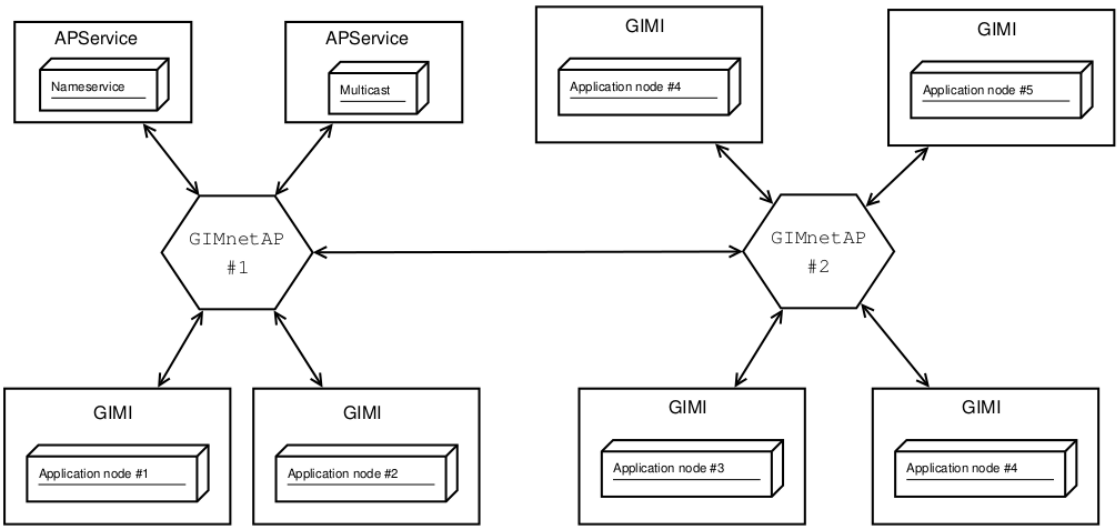


Figure 18: The structure of GIMnet. GIMnetAPs are data hubs, while GIMI is the interface through which application nodes communicate with them. [41]

6.1 GIMnet and MaCI

GIMnet and MaCI are both part of a control interface for fleets of generic machines. By using them, it is possible to connect and control a machine or even a fleet of machines that are unknown in advance. GIMnet handles the communication between various machines and their human operators while MaCI is a hardware abstraction layer on top of GIMnet. The StageMaCI simulator environment, as its name implies, uses MaCI for abstracting simulated machines.

GIMnet is a service-based communication middleware that essentially combines aspects of peer-to-peer and server-client communication architectures [41]. A GIMnet network consists of a number of GIMnet Access Points through which different services and applications send data to each other. The network topology is hidden from the user. All the application nodes can see and communicate with each other if their access points are connected despite real network topology or firewalls between them [42]. The basic structure of GIMnet with access points and application nodes is shown in Figure 18.

MaCI defines a number of interfaces as abstract representations of different machine components, such as range scanners, position, speed control and joint control. Using MaCI interfaces it is possible to control a generic machine through GIMnet. A machine is usually represented as a group of components that provide several MaCI interfaces [43]. In addition to controlling individual machines, it is also possible to connect to and control entire fleets of different types of machines through MaCI [44]. In this work MaCI interfaces are used in the StageMaCI simulator to describe the machine to the network and to enable controlling it in a generic way.

6.2 StageMaCI

Stage [45] is a lightweight, 2D kinematic simulator for multiple robots. It features support for very large maps, huge numbers of robots of different types and a number of ready-made models for different devices and sensors. Machine motions are estimated using one of three different drive modes, namely differential drive, holonomic drive or car-like drive. Stage uses simplified, low fidelity, computationally cheap kinematic models for each drive mode. This makes the models cheap to simulate but limits their ability to represent more complex real-world machines. Interaction between objects is limited to three-dimensional collision detection. While lacking the ability to simulate dynamic interactions between robots and objects, Stage is a useful tool e.g. for designing robot control.

Stage uses its own syntax to define both maps and the models populating them. The maps are highly modifiable, as Stage allows the user to e.g. read map layouts from image files, define the height and width in the simulation world of each map pixel or adjust the pose of the map. Models are constructed primarily from simple extruded polygons. A model can contain other models, making it possible to render objects with varying degrees of fidelity. As models can have specific functionalities such as representing robots or sensors, this can also be used to create increasingly complex machines.

Stage has been previously modified to work with MaCI as StageMaCI, also known as Motti Simulator [40]. In addition to connecting Stage simulations to the MaCI control interface, StageMaCI can be expanded with more complex, customized vehicles. Any new vehicles can be controlled through MaCI once they were set up for it. The original version of StageMaCI has the downside that each new vehicle requires its own C++ vehicle class that defines the kinematics and properties of the vehicle before it is recognized by StageMaCI. Each of these classes is highly complex, as they are required to define every important feature of the vehicle from steering to sensor data handling.

In this thesis work, StageMaCI is further expanded with the addition of a generic vehicle class. The main purpose of this is the elimination of the costly, time-consuming creation of separate classes for each individual vehicle configuration. Instances of the class make deductions based on the structure of a Stage vehicle model to estimate kinematics. The generic vehicle class also supports vehicles with multiple rotating axles or joints (Requirement R17), an arbitrary body structure and any amount of range sensors (R19). Thus, various common vehicle types and configurations can be simulated using just one class of vehicles. Each MaDe-generated vehicle can be controlled manually or given a path of coordinate points which it will follow autonomously. The only thing a user needs to do manually for these vehicles to be simulated in StageMaCI is to include them in a Stage world file.

Upon initialization, instances of the generic vehicle class connect to GIMnet and set up MaCI servers for parts of the vehicle. Another server is set up for the coordinate drive mode of the vehicle to pass path points through MaCI. MaCI servers providing positional data are created for the center point of the vehicle as well as the center of each track or axle it has. To pass sensor data to MaCI, each sensor

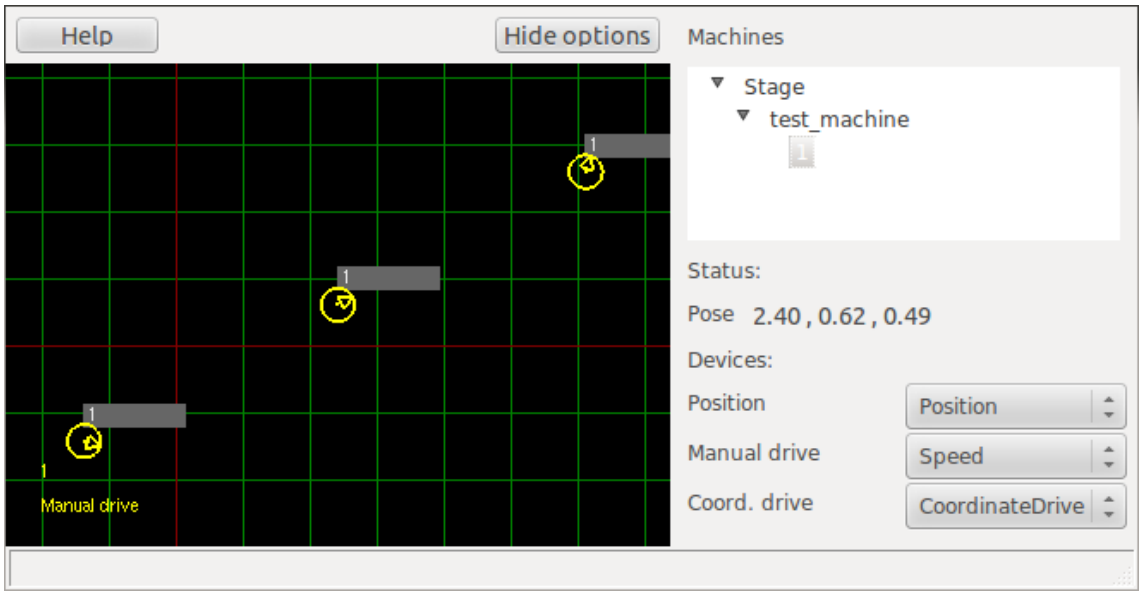


Figure 19: Controlling the example vehicle through MuRo. The circles represent the midpoint of the vehicle and its two axles, both of which have been rotated 45°.

is connected to a MaCI ranging server. Finally, separate rotational joint control servers are specified for axles and joints. Any steerable axles or joints are connected to their corresponding control servers, allowing their poses to be controlled through MaCI. The MaCI connection also makes it possible for a user to manually control the simulated vehicles e.g. through the MuRo control program as shown in Figure 19 by passing velocity and angular velocity control values through MaCI. Using MaCI also fulfils Requirements R20–R22.

After experimenting with a number of different vehicle configurations, it was determined that StageMaCI is capable of simulating at least 50 and up to 100 instances of the generic vehicle class at the same time on a regular desktop computer without slowing down significantly. The processor load of each simulated vehicle depends upon its complexity, with vehicles that have multiple range scanners being the most taxing for the CPU. While the generic vehicle class with its MaCI compatibility can be heavier to simulate than regular Stage models, these numbers of vehicles easily cover the needs of most work sites. Multiple types of generic vehicle configurations can be simulated in the same simulation instance. This is easily enough to cover the needs of Requirement R18.

6.3 Kinematic model

To support different vehicle types, the generic vehicle model has to have the capability to accurately estimate the kinematics for the large number of different vehicle configurations that can be generated with MaDe. The model created for this thesis work supports at least the most common vehicle types: car-like vehicles with N axles, differential drive machines and articulated vehicles with either passive or ac-

tive articulation, fulfilling Requirement R15. Simulating an unknown vehicle with a highly variable structure poses challenges in terms of simulation accuracy, especially as the Stage simulator sets its own constraints for the simulation. Accurately estimating the motion of a vehicle with multiple centres of rotation is not a trivial task though the model developed for this thesis attempts to limit the number of centres by e.g. controlling axle positions so that extended imaginary axles always intersect at a single point as described in Chapter 4.

The simulator model for generic vehicles was designed from the ground up to function together with MaDe. At the start of the simulation, the vehicle model is read from the Stage model file. Different sub-models and special components such as axles are recognized either by their model name or by their name attribute. It is assumed that any articulated vehicles are constructed as a tree of sub-models in the way used by MaDe as described previously. Some of the parameters of the vehicle such as its top speed are read from a separate file created by MaDe, whereas attributes related to the structure of the vehicle such as its wheelbase are derived from the Stage model.

Once the Stage model of the vehicle has been processed, a rough estimate is made of the drive type of the vehicle. Based on the number of steerable and non-steerable axles as well as joints, the vehicle is categorized as being either differentially driven, skid steered, car-like or joint steered. In essence, if a vehicle only has one axle, it is assumed to have differential drive. Vehicles with more than one non-steering axle or set of tracks are marked as skid steered, whereas if the vehicle has more than one axle or track and some of them are steering, it is considered car-like. If the vehicle has any actuated joints, it is joint steered. Vehicles with passive joints are a special case, as the poses of passive joints are calculated separately. For these vehicles, their type is derived from the parts in the first vehicle section attached to the first passive joint, which is assumed to be the driving front section of a vehicle. An example of such a vehicle is a tractor-trailer combination with passive linkage, where the tractor is the front section.

The estimate for the drive type of the vehicle is used to decide which kinematics calculations are performed on the vehicle. Here, Stage's model for differentially driven vehicles is used extensively, because its simplicity allows for easy modifications. In practice, the generic vehicle model needs to mainly focus on calculating the center of rotation and the forward and angular velocities for the vehicle, as Stage is capable of estimating its motion once these values are known.

For a generic mobile vehicle, the kinematic model requires the heading, velocity and angular velocity of the vehicle measured at some known point in vehicle coordinates as well as some information on the pose of each component used to steer it. As such, the current poses of each axle and joint on the vehicle are used in the generic kinematic model when calculating the trajectory of a vehicle. The generic kinematic model uses several of the kinematic models covered in Chapter 4 to construct a model for a specific vehicle; the models used depend on the estimate for the drive type of the vehicle. This fulfils the Requirement R16.

Differentially driven vehicles are the simplest group of mobile machines supported by the generic vehicle model. Since Stage already supports differential drive, all that

remains for the generic vehicle model to do is deducing the point around which the machine revolves. Because differentially driven vehicles are assumed to only have one wheeled axle, this point is naturally the center point of that axle.

As covered in Chapter 4, skid steering is very similar to differential drive. The main difference between the two is the fact that skid steering always has a certain amount of lateral slippage when turning, which cannot be modelled properly with a kinematic model. The amount and direction of slippage depend largely on driving conditions and is dynamic in nature. Since Stage is a kinematic simulator, skid steered vehicles use the same model as differentially driven ones as information on the environment is not available in the generic case. The sole difference between the models used for differential drive and skid steering is that with skid steered vehicles, the center of rotation is placed at the midpoint of all the axles and tracks on the vehicle. Essentially, the skid steering model assumes that the vehicle operates in a highly ideal environment. While this means the skid steering model is more inaccurate than e.g. the differential drive model, the difficulties of modelling skid steered vehicles are well known.

Car-like vehicles and those with actuated joints use the common bicycle model for calculating their centres of rotation. The biggest challenge here is that as stated, vehicles with more than two axles can have more than one center of rotation, while Stage only supports one center per vehicle. This problem is overcome by estimating a single center of rotation for the vehicle using the equations to find an intersection of N lines, the lines being the imaginary continuations of the axles of the vehicle. As it is unlikely for more than two lines to intersect at a common point, a least-squares approach is used with the sum of squared distances as the cost to find a point at a minimum distance from at least two of the lines. The used model also supports driving vehicles diagonally by turning each axle by the same amount.

To solve the N line intersection problem, each line i is represented as a point p_i and a unit normal vector $\hat{n}_i$. In this case, the problem is two-dimensional in nature, as Stage is a 2D simulator. Given two points along the line, x_{i1} and x_{i2}, the normal vector becomes:

$$\hat{n}_i = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} (x_{i2} - x_{i1}) / \|x_{i2} - x_{i1}\| \tag{35}$$

In the generic model, the ends of the ith axle in vehicle coordinates were chosen as x_{i1} and x_{i2}. The model also takes into account changes in axle pose and location caused by any joints along the body of the vehicle. As p_i is a point along the i line, let $p_i = x_{i1}$. The squared distance from a given point x to the line $(p, \hat{n})$ is:

$$d(x, (p, \hat{n}))^2 = (x - p)^\intercal (\hat{n}\hat{n}^\intercal)(x - p) \tag{36}$$

This leads to the cost function $E(x)$. It can be minimized to solve the point x, which is the least-squares solution to the N line intersection problem:

$$E(x) = \sum_i (x - p_i)^\intercal (\hat{n}_i \hat{n}_i^\intercal)(x - p_i) \tag{37}$$

$$x = \left(\sum_i \hat{n}_i \hat{n}_i^\intercal \right)^{-1} \left(\sum_i \hat{n}_i \hat{n}_i^\intercal p_i \right) \tag{38}$$

The N line intersection produces a good estimate for a single center of rotation and, as will be shown in Chapter 7, the estimate remains good even for vehicles with multiple centers of rotation. To further minimize inaccuracies caused by using such an estimate, whenever the StageMaCI model is in manual or coordinate drive mode, steerable axle and joint poses are controlled so that vehicles always have as few centres of rotation as possible. This is done by calculating suitable Ackerman angles for each steerable axle. For certain vehicles, such as ones with only steering axles, the number of centres of rotation can thus be reduced down to just one. Naturally, if this is the case, the equations for N line intersection produce that point.

Vehicles with passive joints require the most complex kinematics calculations of the vehicle types supported by this model, making use of two separate models. The first is one of the models for differential drive, car-like or articulated vehicles, used to calculate the motion of the front section of the vehicle. The second is the generalized model for a vehicle consisting of a chain of passively linked sections by Larsson et al. [32] (see Chapter 4.3). It is used to calculate the poses for each passive joint on the vehicle based on the motion of the front section. The movement of passive joints is naturally limited by their physical maximum angle.

Axles or tracks connected with passive linkage to the driving front section of the vehicle are not considered for the purpose of determining the center of rotation, unless the linkage has reached its maximum angle. It is assumed that each passive joint has zero friction and thus does not affect the trajectory of the vehicle as long as the joint can still rotate in the required direction. This is of course an idealization, but the dynamic forces involved in such a situation in the real world cannot be reliably modelled with kinematics alone. Sections connected to the front of the vehicle by joints that have reached their maximum angle are included in the kinematics calculations of the front section. This is to account for their effect on the motion of the vehicle configuration. As such joints cannot turn further, they would essentially behave like non-articulated vehicle sections at a fixed angle relative to the front, thus becoming a part of the regular kinematics calculations.

Initially, the only variables the chain model needs to know are the velocity and the angular velocity of the driving front section and the distances between each link in the chain of vehicle sections. These are all known from calculations made prior to updating passive joint poses. Because the actual motion of the vehicle in the simulator is dependent upon the front section, the algorithm aims to solve the current angle for each passive joint based on the velocities of any parts immediately before and after its position in the part chain. Said velocities can be solved using Equation 19 and the joint angles from the previous time step. The algorithm starts from the front-most passive joint on the vehicle and moves from there towards the rear of the vehicle while updating the poses for the joints. This makes it possible to simulate indefinitely long vehicles with an arbitrary number of passive joints, as demonstrated in Figure 20.

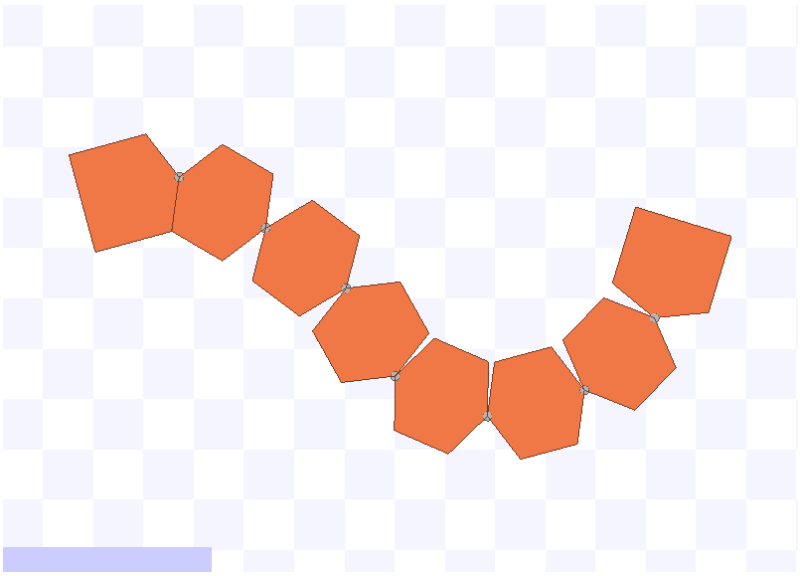


Figure 20: Driving a vehicle with 7 passive joints in StageMaCI.

7 Simulation results

As with any simulated system, it was important to ensure that the kinematic models generated with MaDe and StageMaCI were accurate enough to be useful. In this case, this meant a generally realistic behaviour for different vehicle types. Cumulative errors in the trajectory were expected, as the StageMaCI simulations would be incapable of matching the dynamic behaviour of their real-world counterparts. In actual use cases, such cumulative errors would be either compensated for with suitable control or ignored as unimportant. For example, one of the planned uses for the generic kinematic model is in route generation, where it is more important that the resulting routes are drivable for the modelled vehicle than it is to follow a certain trajectory with high accuracy.

To verify the kinematic models, three real-world vehicles were first defined in MaDe and then simulated in StageMaCI using datasets gained from the real-world counterparts of the vehicles. The first vehicle was a differentially driven robot used in indoor research. The second was a four-wheel steered radioactive waste deposition vehicle while the third was a tractor-trailer combination. These last two vehicle configurations are comparable to the majority of mobile vehicles in use today, as car-like vehicles with or without trailers are easily the most common vehicle configuration in the world.

Although it would have been preferable to make comparisons between simulated and real data with more than three vehicle configurations, unfortunately this was not possible due to lack of datasets. In an ideal scenario, simulations would have been run at least once for each of the most common steering configurations. Especially the lack of datasets for center-articulated vehicles was unfortunate.

7.1 Test case 1: Differential drive robot

As the first exercise of the developed tools, a relatively simple differential drive robot called J2B2 was defined in MaDe and simulated. J2B2 is an example of a very common mobile robot configuration, consisting of a cylindrical body supported by two differentially driven wheels and a third castor wheel. J2B2 has been used in both education and research, as a result of which a number of datasets were available for it. The robot is shown in Figure 21.

The dataset used here was compiled using J2B2's on-board sensors while the robot was teleoperated around an indoor laboratory track. Estimates for the velocity and angular velocity of the robot were provided by its odometry sensors while its position was estimated using its laser range scanner. The robot was operated in an indoor laboratory. As J2B2 was differentially driven, the only control variables it required were its velocity and angular velocity. The state of the real-world robot was logged every 50 ms while the simulator is updated every 100 ms. Because of this, the simulated robot was steered by using the average of two consequent control commands. The results of the J2B2 simulation are shown in Figures 22 and 23.

Figure 23 shows the trajectory of the simulated and real vehicles, the trajectory estimated from the odometry data of the robot as well as its heading over time.

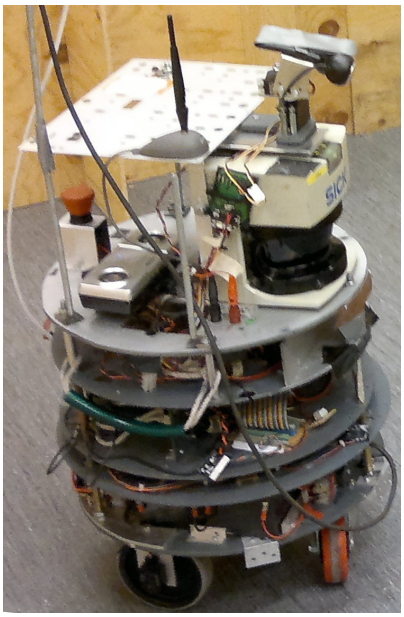


Figure 21: The J2B2 differential drive robot.

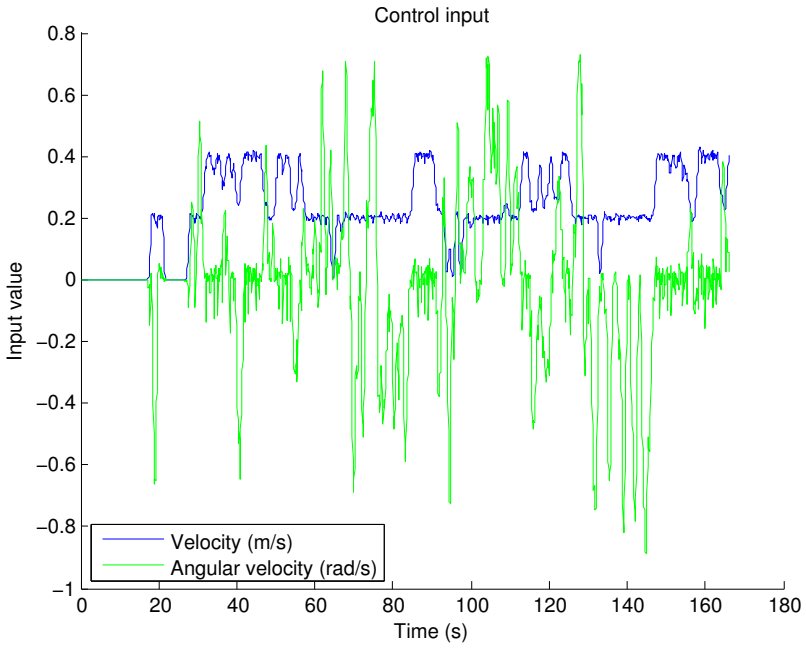


Figure 22: J2B2's control input over time.

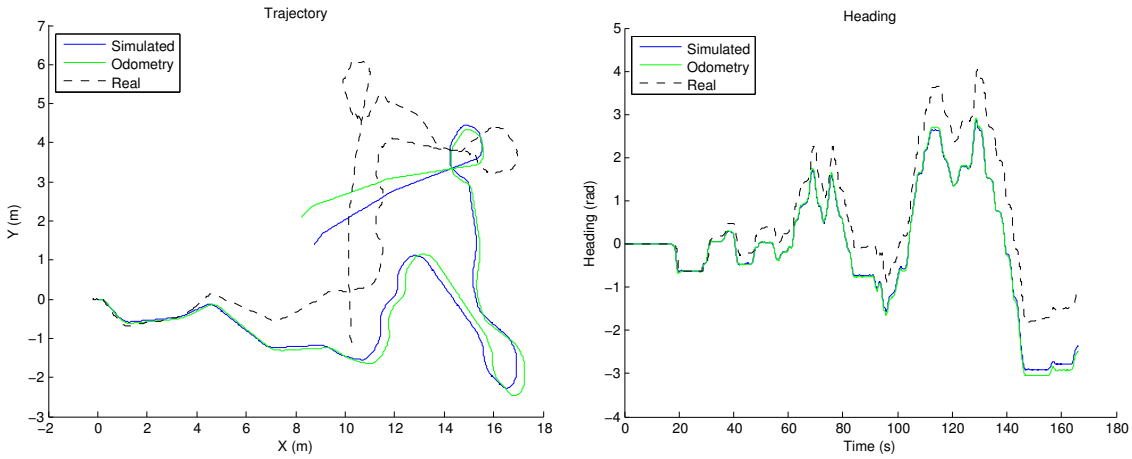


Figure 23: J2B2's trajectory and heading.

It is apparent that the trajectory and heading of the simulated vehicle match the odometry estimates very well, while differing by a large amount from the behaviour of the real robot. This was expectable, as the control input of the robot, shown in Figure 22, was gained from its odometry measurements. As is apparent, the odometry data has a high degree of noise and inaccuracy. The real robot also had a number of mechanical error sources, such as a tendency to over steer when turning due to a limited breaking capability in its wheels, as well as the castor wheel which could affect steering. These error sources together with the inherently inaccurate nature of odometry measurements lead to the odometry estimates differing by quite a large amount from the real-world behaviour of the robot.

As a measure of the accuracy of the kinematic model for the J2B2 robot, the root mean square error (RMSE) for the heading of the simulated robot was calculated. The RMSE represents the standard deviation between the predicted and real values. In this case, the RMSE of the heading is the average rate of growth of the difference between the simulated and real headings; in other words, it is the rate of growth of the heading error. It is calculated using:

$$\sigma_t = \sqrt{\frac{1}{N}\sum_i (\theta_i - \hat{\theta}_i)^2} \tag{39}$$

$$\sigma_t = \sigma\sqrt{t} \quad \rightarrow \quad \sigma = \frac{\sigma_t}{\sqrt{t}} \tag{40}$$

where σ_t is the RMSE, N is the total number of data samples, θ_i is the simulated heading in the i sample and $\hat{\theta}_i$ is the real heading. σ is the rate of growth of the difference. Because σ_t, the variance of the heading, is relative to the square root of the simulation time t, σ is gained by dividing σ_t with the root of the simulation time.

For J2B2, the difference between the simulated and real headings grew at an average rate of 0.0044 rad/s (0.25°/s) throughout the 166 second long simulation. At such a slow rate of difference growth, the simulated heading would have remained

reasonably close to the real value even with much longer simulations. This is further indication of the high accuracy of the model generated for J2B2.

Despite the mismatch between simulated and real world trajectories, these results show that the kinematic model for the differential drive robot was very accurate. When given the control input resulting in the odometry estimate for trajectory, a very closely matching simulated trajectory was produced. The simulated robot matches the heading changes of the odometry estimate almost step for step as shown in Figure 23. Some of the remaining difference can be explained with the filtering nature of averaging the control input.

It would be interesting to see how the model performs with more accurately measured control input. Unfortunately, no accurate measurements of J2B2's velocity or angular velocity were available. However, it is reasonable to conclude that if the simulated robot had been given control input more closely matching what the real robot used, the resulting trajectory would have been very close to that of the real robot as well. The generated kinematic model could definitely be used with good results to simulate the robot in other situations. In the other test cases, more accurate control input measurements were available for simulation purposes.

7.2 Test case 2: Radioactive waste deposition vehicle

The second test case for models generated with MaDe was for Magne, a radioactive waste deposition vehicle designed to drive in narrow underground tunnels. The vehicle carries sealed canisters of waste to a repository deep within a network of tunnels before depositing them inside previously prepared holes [46]. Magne is shown in Figure 24. The vehicle in question was equipped with four-wheel steering and drive as well as two laser range scanners, one at the front and one at the rear of the vehicle. It is the same vehicle used as the example in Figure 12.

Datasets for the vehicle were provided by Navitec Systems Ltd. [47], a partner in the Motti project, who are working on automating the vehicle in question. The datasets were created by driving the real-world vehicle slowly along a test track. Unfortunately, the datasets are somewhat limited in scope, as the work using the Magne vehicle is still in its initial stages with a planned start of operations in the mid-2020s [46]. The data consists of the control input given to the vehicle as well as measurements of its position and velocity. The position of the vehicle was calculated using laser measurements from two range scanners on the vehicle itself.

Parameters for the dimensions and performance of the vehicle were gained from both real-world measurements and the specifications of the vehicle. Using these parameters, a version of the vehicle was defined in MaDe as shown in Figure 12. This vehicle was then simulated in StageMaCI and the results were logged. As Magne is a relatively simple four-wheeled, car-like vehicle, the generated kinematic model for the vehicle was essentially the bicycle model (see Chapter 4.2) with two steering axles, using Magne's dimensions as parameters.

As control input, the simulated vehicle was provided the same target control values as its real-world counterpart. Because the vehicle had four-wheel steering, this meant separate target angle values for both its front and rear axles as well as a

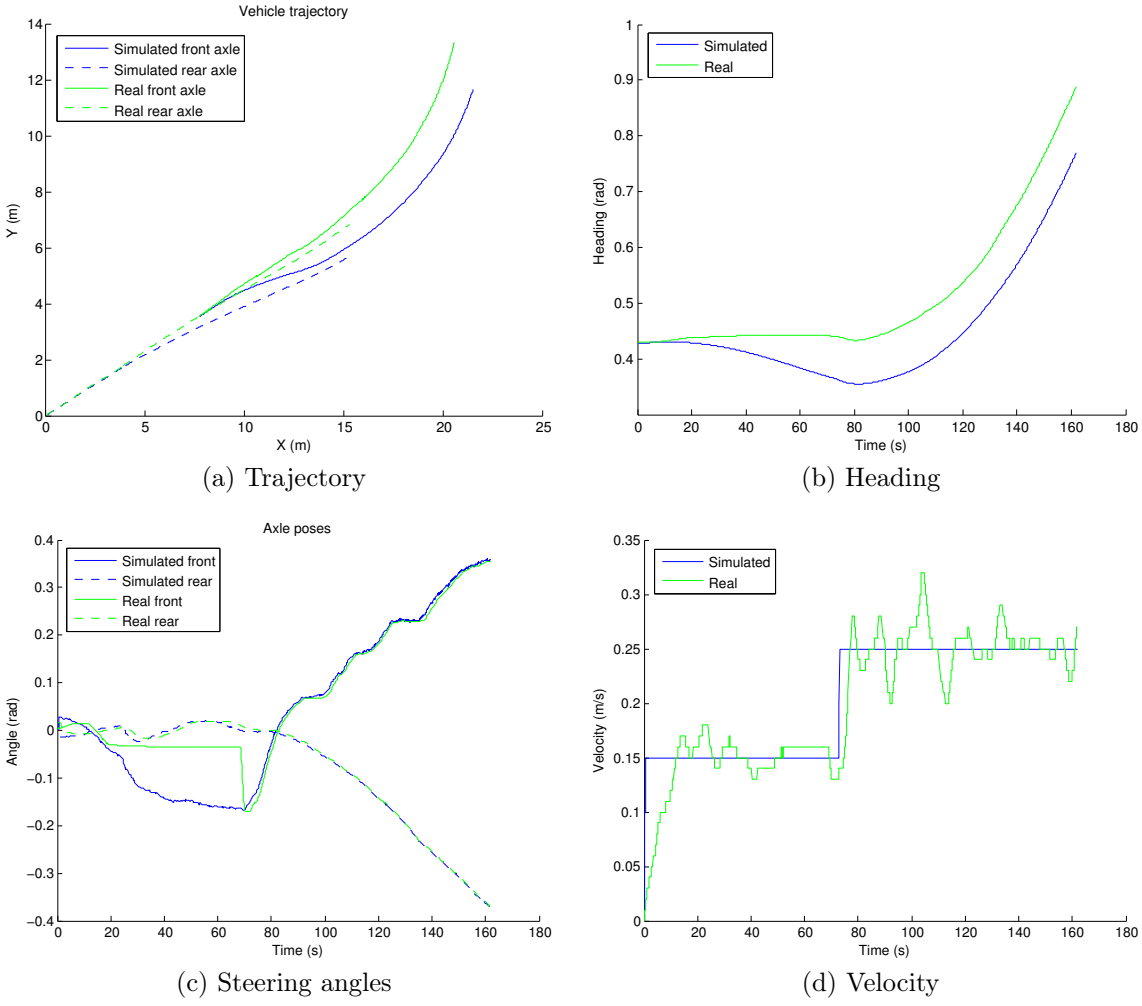
target driving speed. The same input had been given to the actual Magne vehicle during a real-world autonomous driving exercise. Data was logged once every 100 ms, the same frequency as used by the real-world vehicle. The total durations of the dataset and thus the simulation were 2.7 minutes. Simulation results are shown in Figure 25.

As is often the case, the simulated version of the vehicle reaches the target input values immediately and with perfect accuracy, whereas there is considerable noise and variation in the real-world response. This is especially apparent in the velocity values shown in Figure 25d, where the simulated vehicle matches target velocities almost instantly while the real-world velocity varies considerably. These differences in response naturally lead to further differences in the vehicle trajectory.

However, while the simulated axles match the reference input angle value very well, it is apparent from Figure 25c that for some unknown reason, the front axle in the real vehicle data became stuck in the same position during the 20-80 s time frame. It is safe to assume that this causes a larger than normal difference between the simulated and real-world trajectory of the vehicle. By simulating the vehicle again, this time giving the model exactly the same steering and velocity values as used by the real world vehicle as input, the simulation accuracy improves as shown in Figure 26.

The performance of the simulated vehicle now matches more closely that of its real-world counterpart. The remaining differences between the simulated and real-world results can be explained at least in part with inaccurate measurements of the dimensions of the vehicle and possibly some small amount of dynamic interaction with the environment of the vehicle. There was a mismatch between the measured and specified dimensions of the vehicle and it was not possible to verify them with new measurements. Dynamics in this case would have been minimized, though, due

Figure 24: The radioactive waste deposition vehicle Magne. [47]



(a) Trajectory (b) Heading

(c) Steering angles (d) Velocity

Figure 25: Initial simulation results for Magne.

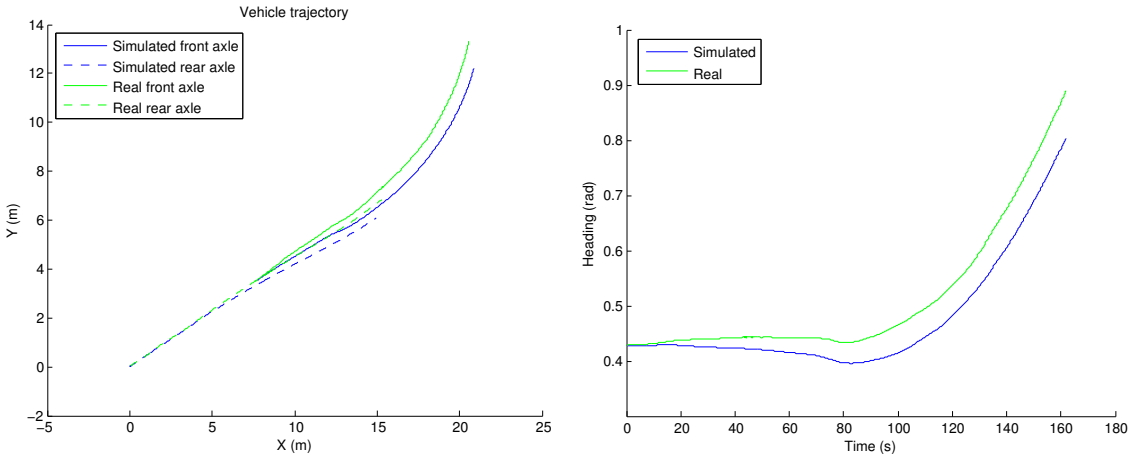


Figure 26: Magne trajectory and heading using precise control input.

to the low driving speeds of the vehicle. Unfortunately, no lengthier, more complex datasets were available for the Magne vehicle at the time of the writing of this thesis.

As in the previous test case, the RMSE for the heading of Magne was calculated. Using the improved results, the average rate of growth for the difference between the simulated and real headings was 0.016 rad/s (0.92°/s). While it is apparent that this model is less accurate than the one for J2B2, the rate of difference growth remains small despite the uncertainties encountered while modelling and simulating Magne. The value is certainly low enough for the model to remain reasonably accurate even during longer simulations.

The simulation results for the Magne vehicle show that the kinematic models generated by MaDe and StageMaCI are reasonably accurate. This shows that the tools can be used to quickly provide a reliable kinematic model for at least the most common type of mobile vehicle in the world. Accurate kinematics support for car-like vehicles is already enough to make MaDe and the generic kinematic model useful in a wide variety of scenarios such as predictive safety features or path planning.

7.3 Test case 3: Tractor-trailer combination

The third vehicle to be simulated was a tractor-trailer combination previously studied by Backman et al. for path tracking purposes [48]. The vehicle configuration consisted of a standard tractor and a seed drill. The seed drill was attached to the tractor with two separate joints, one of which was actuated. Thus, this vehicle was rather more complex structurally than the relatively simple, car-like vehicle in the second test case. Several datasets gained from field research were graciously provided for use in this thesis by the original research team. The vehicle configuration and its simulated counterpart are shown in Figure 27.

The datasets are largely similar to the one used in the previous test case, containing the control input values for vehicle velocity, the front steering angle of the tractor as well as the target angle for the actuated joint. Additionally, the position of the vehicle was estimated using GPS and laser measurements. The vehicle configuration also included a method for estimating the passive joint angle and the position of the trailer as explained in [48].

While a kinematic model for the vehicle was presented in [48], it was somewhat different from the one used in the simulation. It included the addition of a slipping factor for the front steering wheels of the tractor, whereas the generated simulator model could not make any assumptions of slippage and thus used a regular bicycle model for the tractor. As explained in Chapter 6.3, the simulated trailer section used the generalized model for passive articulation presented by Larsson et al. [32]. Unlike in the previous test case, this vehicle was not automated, instead being operated by a human driver. Thus, no precise reference values for control were available. Instead, the control values given to the simulated vehicle were the measured velocity and front wheel steering angle of the real vehicle over time. All of the parameters and physical constraints for the vehicle configuration were taken from [48].

The dataset used in this test case was of a simple driving exercise with the real vehicle driving a single back and forth run as when ploughing a field. For this case,

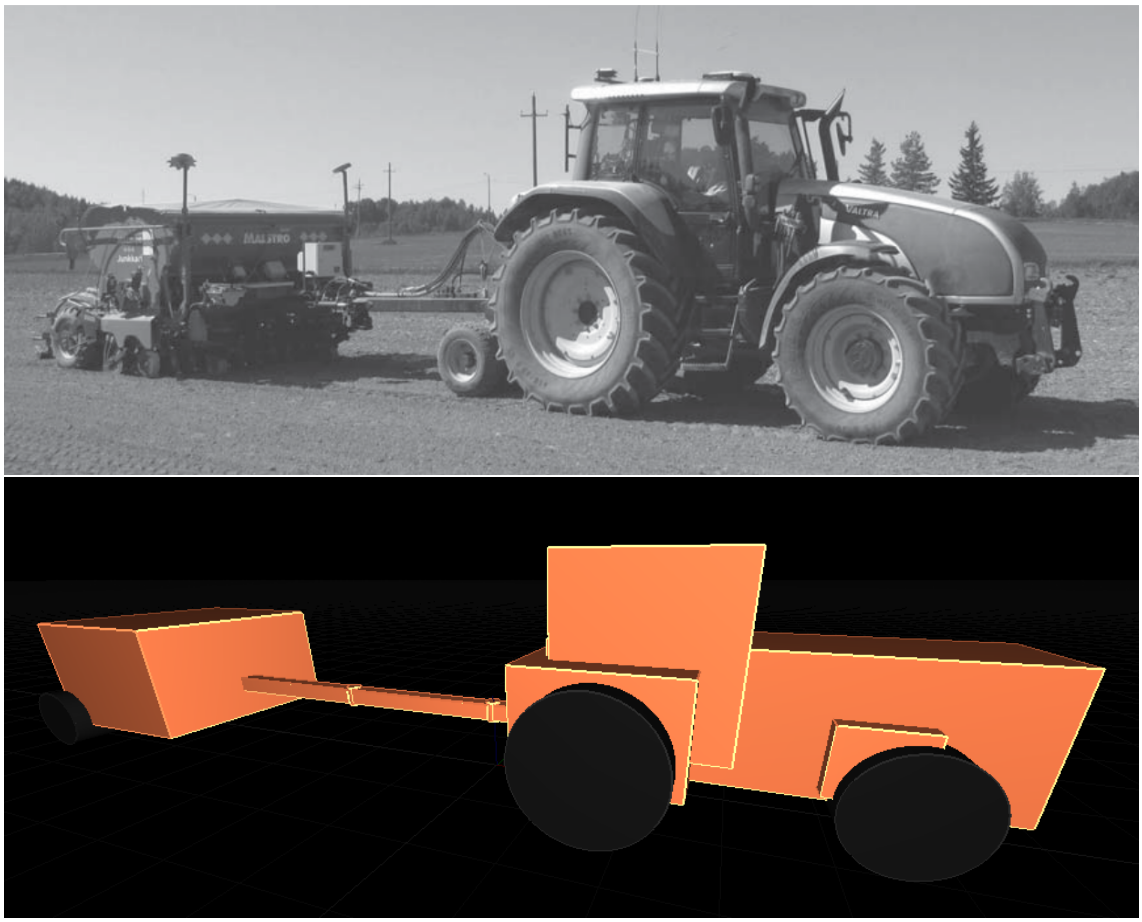
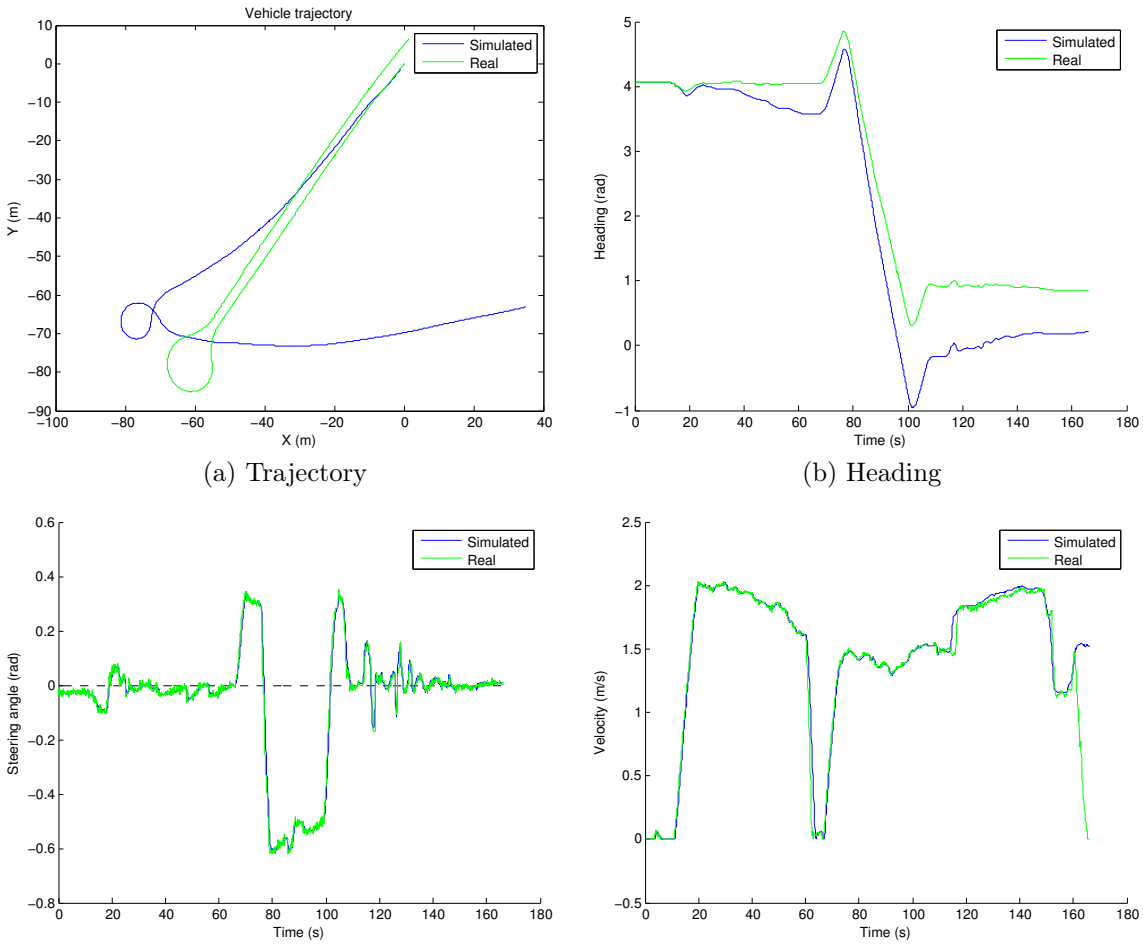


Figure 27: The tractor-trailer used in the second test case. [48]



(a) Trajectory (b) Heading

(c) Steering angle (d) Velocity

Figure 28: Initial tractor-trailer simulation results.

the actuated joint was set at a constant zero angle, effectively removing it from consideration. The vehicle was simulated using a slightly simplified model with the actuated joint removed. The simulation results are presented in Figures 28 and 29.

As is apparent from the figures, the simulated model of the tractor-trailer configuration is relatively accurate. However, due to cumulative error over time and e.g. lack of position correction in the steering behaviour, the end location of the simulated vehicle differs by a large amount from that of the real vehicle. Errors are caused by a number of sources, such as a certain amount of inherent inaccuracy in the simulation model itself, and the dynamics affecting the real-world vehicle as it drives around in an uneven, somewhat slippery field. The simulated vehicle has a tighter turn radius than the real-world one and ends up turning too much during the turnaround midway in the drive. This is caused by the behaviour of the simulated trailer section; It does not affect the motion of the vehicle until the passive joint reaches its maximum angle, which happens fairly late into the turn. The simulated vehicle also appears to react strongly to slight corrective movements made over time by the real vehicle, which cause the largest cumulative trajectory errors.

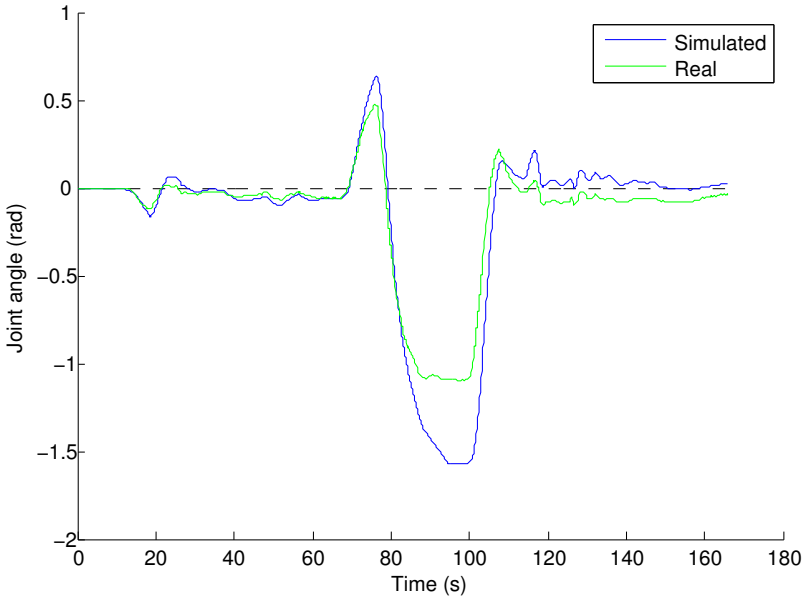


Figure 29: Tractor-trailer passive joint angle over time.

The model for the passive joint pose is quite accurate, as can be seen in Figure 29. Only the tighter turn radius of the simulated vehicle causes any real difference between the simulated and real joint poses. This is apparently caused by a difference in specified and actual maximum angles for the passive joint: While the real-world joint never reaches its stated maximum value of $\frac{\pi}{2}$ rad (90°), instead being quite clearly limited to roughly 1.1 rad (63°), the simulated joint quickly reaches its specified maximum. However, once the vehicle straightens out the model quickly returns close to the real-world values aside from a small offset in poses. As the simulated joint returns close to zero angle when driving straight while the real one does not, it is likely the difference is caused by the driving conditions of the real vehicle.

After the initial simulation, it was clear that by reducing the maximum joint angle to the value displayed by the real vehicle, i.e. roughly 1.1 rad (63°), the accuracy of the simulation could be improved. By lowering the maximum, the joint reaches its maximum angle sooner into the turn and thus alters the behaviour of the vehicle, leading to increases in simulation accuracy. The results from this altered simulation are shown in Figures 30 and 31.

As can be seen from Figure 30, the passive joint is still simulated quite accurately even after the reduction in maximum angle. The most significant improvement in simulation accuracy is apparent in vehicle heading during the clockwise turn. Once the passive joint is locked in its maximum pose, the behaviour of the vehicle very closely follows that of the real vehicle. The locking occurs about 85 seconds into the simulation as shown in Figure 30.

Figure 31 shows that the amount of deviation from the performance of the real-world vehicle was drastically reduced though a distinct curve to the left in the trajectory of the simulated vehicle is still apparent during the straight section. Given the consistency of the curve, this could be explained by slippage or a slight sloping

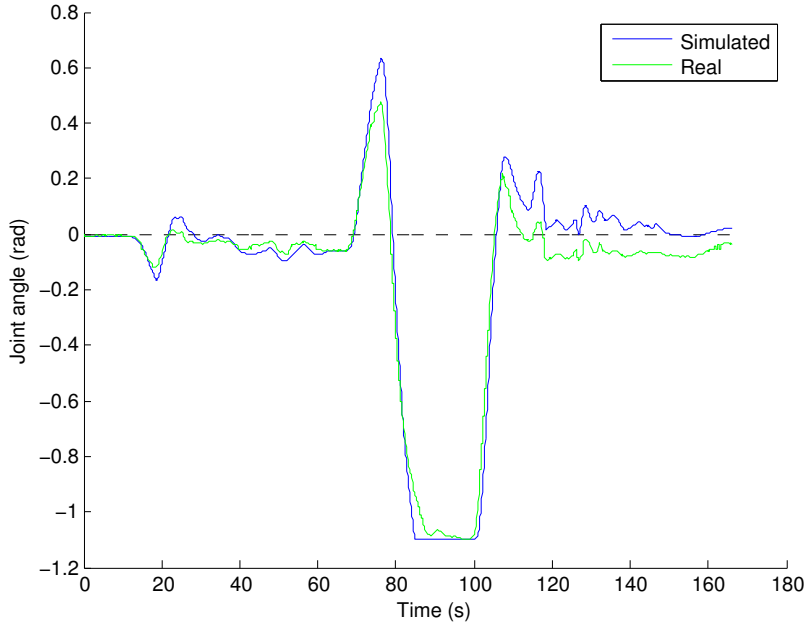


Figure 30: Tractor-trailer passive joint angle over time with a reduced maximum joint angle.

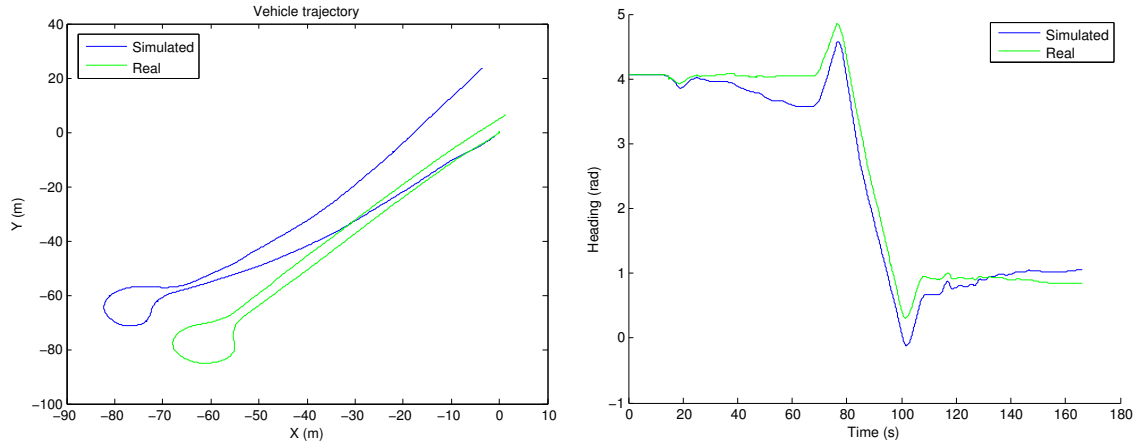


Figure 31: Tractor-trailer trajectory and heading with a reduced maximum joint angle.

of the field used to test drive the vehicle. The human driver operating the real world vehicle would have compensated by steering suitably. As the control input used by the driver was given as input to the simulation model, the compensatory steering would show up as deviation in the simulated trajectory. Indeed, the steering angle of the tractor is non-zero in the direction of the curve throughout the dataset as can be seen in Figure 28c. Remaining errors can likely be explained with inaccuracies in the model; while it is quite accurate, the model is still a simplified estimate of dynamic conditions.

The RMSE for the heading of the tractor-trailer combination was 0.02 rad/s

(1.15°/s). Considering the relative complexity of the vehicle, it is a good indicator of the accuracy of the model that the rate of difference growth between the headings was in this case very close to that in the previous test case, where the simulated vehicle was considerably less complex. This makes this model arguably more accurate than the one for Magne in the previous test case, as Magne operated in a significantly less dynamic environment. In this case, the consistent curve in the trajectory is the single largest source of heading error in this model. It is likely that in less dynamic conditions, the growth rate of the heading error would have been significantly smaller. Despite the differences in the simulated and real-world behaviours caused by environmental dynamics, the model for the tractor-trailer remained quite accurate.

These results show that aside from cumulative errors caused by indeterminate dynamics, the kinematic model generated for the tractor-trailer configuration is quite accurate. Even without the reduction in the maximum angle of the passive joint, the model is accurate enough to be useful in virtually any situation requiring one. Further refining of vehicle parameters to better match real-world scenarios may also be used to improve simulations when possible. Out of the three test cases, this one took place in the most dynamic conditions. The susceptibility of kinematic models to dynamic events shows in a moderate degree of error in the vehicle trajectory. Despite this, the model for the tractor-trailer is accurate enough to be usable in most situations requiring one especially in less dynamic conditions.

8 Summary and future work

In this thesis, a software tool for quickly and easily defining a kinematic model of a generic mobile vehicle was presented. Such a tool was necessary because despite their usefulness in a number of tasks such as path planning, kinematic models for specific vehicles are rarely available. The tool needed to support the definition of the most common mobile vehicle types used in industrial tasks, such as car-like, skid-steered or articulated vehicles. No tool suitable for this task was found, thus requiring the creation of new software.

Because calculating the kinematics for such a varied collection of vehicle types requires a number of different kinematics equations, a literature study of up-to-date research on mobile machine kinematics was presented. The study covered a number of different kinematics solutions for the vehicle types supported by the software. Some of them were then used to calculate the kinematics for certain vehicle configurations generated with the software.

The machine definition tool MaDe allows a user to define the basic structure and configuration of a vehicle through a simple, intuitive 3D user interface. The vehicle definition itself is done using two orthographic views from the top and side of the design canvas. It is also possible to view a 3D perspective presentation of the vehicle model. The tool supports the creation of boxes and point-to-point polygons as the chassis of a vehicle and a number of special parts such as axles, joints and range scanners to provide the main operational parameters of the vehicle. Many of the standard user commands supported by common design software, such as copying, pasting, undo and redo, are also supported by MaDe. These were programmed using the Command design pattern.

A number of different file formats were studied to find ones suitable for storing vehicle model data. MaDe stores its own models as XML-based X3D or SVG files, allowing for small file sizes, easy portability and viewing models in software other than MaDe. It is also possible to export vehicles from MaDe into the StageMaCI simulator environment as functioning kinematic simulator models.

The actual kinematics calculations for vehicles defined with MaDe are handled by a StageMaCI simulator component created as a part of this thesis work. The component is essentially a generic kinematic model that deduces the main features of the structure of a vehicle based on the model generated with MaDe and then forms a suitable kinematic model for it. This makes it possible to simulate arbitrary vehicle configurations in StageMaCI. Vehicle pose and range scanner data are broadcast over the GIM network and can be used in other software. Vehicles can be controlled through the MaCI interface.

The generic kinematic model is based on well-established kinematics knowledge. It uses a number of separate kinematic models for specific vehicle types to construct models for generic vehicles. The use of previously studied kinematics in the generic model makes it and its simulation results reliable in at least those cases where vehicles use a common vehicle configuration. Although the generic model also supports more complex configurations with e.g. multiple passive joints, the accuracy of the model in such cases could not be verified due to a lack of suitable datasets. While

it is possible that other similar generic kinematic models exist, open-source models or tools suitable for defining them or their parameters were not found during this thesis work, making it necessary to create the tools presented in this work.

A separate module for calculating kinematics for generic vehicles defined in MaDe is in development. It will be a standalone component not dependent on Stage. It will have all the same kinematics features as the StageMaCI component. A standalone generic kinematic model is necessary as decoupling the generic model from the StageMaCI simulator will make it useful in a wider range of applications than currently.

Kinematic models generated for three different real-world vehicle configurations were verified using datasets collected from field exercises conducted with said vehicles. The vehicles in question were a differential drive robot called J2B2, a car-like radioactive waste deposition vehicle called Magne and a tractor-trailer configuration with passive linkage. The kinematic models were found to be accurate in all cases, making them suitable for any work requiring such models. In each case, the rate of growth for vehicle heading error was low enough to keep trajectory errors reasonably small over fairly long periods of time. The downside of the models is a limited tolerance to dynamic environments with e.g. slippery or uneven terrain.

The examined test cases covered common vehicle configurations that are similar to those used in a large percentage of mobile industrial applications. It can be assumed that similar levels of accuracy can be achieved for other vehicles that use the same configurations. As such, the results are quite generalizable.

Unfortunately, suitable datasets were available only for the three vehicles covered in this thesis. Therefore it will be necessary to further verify kinematic models generated for vehicle configurations that were not covered. While these three cases cover the most commonly used mobile industrial vehicles, models for other common vehicle types such as center articulated vehicles were not verified. Further development may be required to add more support for vehicles that combine features from different vehicle configurations, such as vehicles with a mix of both actuated and passive joints.

The high level of accuracy in the kinematic models defined with MaDe and generated using the simulator component makes the models usable in a wide range of applications. Some planned uses for the MaDe and the generic model aside from vehicle simulations include route network and traffic rule generation for arbitrary vehicles. Additionally, the generic kinematic models could be used e.g. to predict vehicle trajectories in a certain time frame. This has many potential uses ranging from predictive safety systems aboard vehicles to making state predictions in Kalman filters. The models are also accurate enough to be used in any simulated scenario for the vehicles.

While the current version of MaDe fulfils all of its requirements, there are certain features missing from MaDe that could improve its usability. MaDe could benefit from certain additional tools, such as a tool for reshaping or resizing objects. It may also be useful to expand the file format support of MaDe to cover formats specifically intended for kinematics calculation. In the generic case kinematics parameters can already be deduced from suitable X3D or SVG representations of machines, but

support for more specialized formats could offer some benefits.

The stated goals for this thesis work were the creation of a tool for defining the kinematic parameters of generic vehicles as well as a generic kinematic simulator model that can calculate the kinematics for the vehicles using the defined parameters. As has been shown, both of these goals were met entirely and with good results. Both tools also fulfil all of their requirements.

MaDe and its accompanying generic kinematic model component allow for the quick and easy generation of kinematic models for arbitrary vehicle configurations as required. With MaDe, it is entirely possible to define and simulate a given vehicle within minutes. The tools eliminate the need for highly detailed CAD models and individual kinematic models for vehicles, leading to potentially considerable time saving whenever kinematic models for vehicles are required.

References

[1] Sandvik Mining. AutoMine, 2013. http://mediabase.sandvik.com/, Accessed: 22.10.2013.

[2] Konecranes. Straddle carrier, 2013. http://www.konecranes.com/industries/port-equipment-services, Accessed: 22.10.2013.

[3] D. Wang and F. Qi. Trajectory planning for a four-wheel-steering vehicle. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation*, volume 4, pages 3320–3325. IEEE, 2001.

[4] P. Corke and P. Ridley. Steering kinematics for a center-articulated mobile robot. *IEEE Transactions on Robotics and Automation*, 17(2):215–218, April 2001.

[5] Finnish Centre of Excellence in Generic Intelligent Machines Research (GIM). Motti. http://gim.aalto.fi/Motti, Accessed: 12.12.2013.

[6] Autodesk. AutoCAD. http://www.autodesk.com/products/autodesk-autocad/overview, Accessed: 11.12.2013.

[7] Blender Foundation. Blender. http://www.blender.org/, Accessed: 11.12.2013.

[8] The Inkscape Team. Inkscape. http://www.inkscape.org/en/, Accessed: 11.12.2013.

[9] MathWorks. MATLAB. http://www.mathworks.se/products/matlab/, Accessed: 11.12.2013.

[10] MathWorks. Simulink. http://www.mathworks.se/products/simulink/index.html, Accessed: 11.12.2013.

[11] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0 (Fifth Edition). http://www.w3.org/TR/REC-xml/, Accessed: 25.2.2014.

[12] Khronos Group. Collada. http://collada.org/, Accessed: 11.12.2013.

[13] Web 3D Consortium. X3D. http://www.web3d.org/realtime-3d/x3d/what-x3d, Accessed: 11.12.2013.

[14] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, 2010.

[15] World Wide Web Consortium (W3C). Scalable Vector Graphics (SVG). http://www.w3.org/Graphics/SVG/, Accessed: 11.12.2013.

[16] Open Source Robotics Foundation. Gazebo. http://gazebosim.org/, Accessed: 27.3.2014.

[17] K. Waldron and J. Schmiedeler. Kinematics. In *Springer Handbook of Robotics*, pages 9–31. Springer Berlin Heidelberg, 2008.

[18] R. Featherstone and D. E. Orin. Dynamics. In *Springer Handbook of Robotics*, pages 35–62. Springer Berlin Heidelberg, 2008.

[19] G. Campion and W. Chung. Wheeled Robots. In *Springer Handbook of Robotics*, pages 391–410. Springer Berlin Heidelberg, 2008.

[20] J. Borenstein, H. R. Everett, and L. Feng. *Where am I? Sensors and methods for mobile robot positioning.* University of Michigan, 1996.

[21] P. Corke. *Robotics, Vision and Control*, volume 73 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg, 2011.

[22] C. Altafini. A Path-Tracking Criterion for an LHD Articulated Vehicle. *The International Journal of Robotics Research*, 18(5):435–441, May 1999.

[23] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, 2010.

[24] J. Borenstein. Control and kinematic design of multi-degree-of freedom mobile robots with compliant linkage. *IEEE Transactions on Robotics and Automation*, 11(1):21–35, 1995.

[25] S. Salmi and A. Halme. Implementing and testing a reasoning-based free gait algorithm in the six-legged walking machine, "MECANT". *Control Engineering Practice*, 4(4):487–492, 1996.

[26] S.J. Ylonen and A.J. Halme. WorkPartner - centaur like service robot. In *IEEE/RSJ International Conference on Intelligent Robots and System*, volume 1, pages 727–732. IEEE, 2002.

[27] KUKA. KUKA youBot. `http://www.kuka-labs.com/en/service_robotics/research_education/youbot/`, Accessed: 27.2.2014.

[28] S. J. An, K. Yi, G. Jung, K. I. Lee, and Y. W. Kim. Desired yaw rate and steering control method during cornering for a six-wheeled vehicle. *International Journal of Automotive Technology*, 9(2):173–181, 2008.

[29] P. Bolzern, R. M. DeSantis, A. Locatelli, and D. Masciocchi. Path-tracking for articulated vehicles with off-axle hitching. *IEEE Transactions on Control Systems Technology*, 6(4):515–523, July 1998.

[30] S. Scheding, G. Dissanayake, E. M. Nebot, and H. Durrant-Whyte. An experiment in autonomous navigation of an underground mining vehicle. *IEEE Transactions on Robotics and Automation*, 15(1):85–95, 1999.

[31] M. Sampei, T. Tamura, T. Kobayashi, and N. Shibui. Arbitrary path tracking control of articulated vehicles using nonlinear control theory. *IEEE Transactions on Control Systems Technology*, 3(1):125–131, March 1995.

[32] U. Larsson, C. Zell, K. Hyyppa, and A. Wernersson. Navigating an articulated vehicle and reversing with a trailer. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2398–2404. IEEE Comput. Soc. Press, 1994.

[33] J. L. Martinez, A. Mandow, J. Morales, S. Pedraza, and A. García-Cerezo. Approximating Kinematics for Tracked Mobile Robots. *The International Journal of Robotics Research*, 24(10):867–878, October 2005.

[34] A. Mandow, J. L. Martinez, J. Morales, J. L. Blanco, A. Garcia-Cerezo, and J. Gonzalez. Experimental kinematics for wheeled skid-steer mobile robots. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1222–1227. IEEE, October 2007.

[35] K. Nagatani, D. Endo, and K. Yoshida. Improvement of the Odometry Accuracy of a Crawler Vehicle with Consideration of Slippage. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2752–2757. IEEE, April 2007.

[36] J. Smart. wxWidgets. `http://wxwidgets.org/`, Accessed: 9.1.2014.

[37] Silicon Graphics and Khronos Group. OpenGL. `http://www.opengl.org/`, Accessed: 9.1.2014.

[38] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.

[39] M. Held. FIST: Fast Industrial-Strength Triangulation of Polygons. *Algorithmica*, 30(4):563–596, October 2001.

[40] J. Grönholm. Developing a generic multi-machine simulator environment. Master's thesis, Helsinki University of Technology, Espoo, Finland, 2012.

[41] A. Maula, M. Myrsky, and J. Saarinen. GIMnet 2.0 - Enhanced Communication Framework for Distributed Control of Generic Intelligent Machines. *Intelligence and Telematics in Control*, 2012.

[42] J. Saarinen, A. Maula, R. Nissinen, H. Kukkonen, J. Suomela, and A. Halme. GIMnet - Infrastructure for Distributed Control of Generic Intelligent Machines.

[43] M. Myrsky. Hardware abstraction interfaces for mobile robotic applications. Master's thesis, Helsinki University of Technology, Espoo, Finland, 2010.

[44] M. Myrsky, J. Saarinen, and A. Maula. Interface for controlling a fleet of generic machines. In *Multivehicle Systems*, volume 2, pages 60–65, 2012.

[45] B. Gerkey, R. T. Vaughan, and A. Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. *Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003)*, pages 317–323, 2003.

[46] Joni M Sundholm. Centralized Traffic Control of Underground Work Machine Fleet. In *Proc. the 2012 IFAC Workshop on Multivehicle Systems*, 2012.

[47] Navitec Systems. `http://www.navitecsystems.com/wordpress/`, Accessed: 5.2.2014.

[48] J. Backman, T. Oksanen, and A. Visala. Navigation system for agricultural machines: Nonlinear Model Predictive path tracking. *Computers and Electronics in Agriculture*, 82:32–43, 2012.

A An example vehicle in X3D format

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.3//EN"
"http://www.web3d.org/specifications/x3d-3.3.dtd">
<X3D profile="Interchange" version="3.3"
xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
xsd:noNamespaceSchemaLocation=
"http://www.web3d.org/specifications/x3d-3.3.xsd">
  <!-- X3D model generated by MaDe -->
  <Scene>
    <Group DEF="0" class="box">
      <Transform translation="0 2.5 0" rotation="0 1 0 0">
        <Shape>
          <Box size="14.3 1 3"/>
          <Appearance>
            <Material diffuseColor="0.937255 0.74902 0.27451"/>
          </Appearance>
        </Shape>
      </Transform>
    </Group>
    <Group DEF="1" class="axle" canTurn="1"
    turnRate="0.698132" turnAngle="0.785398">
      <Transform translation="-4.3 1 0" rotation="1 0 0 1.5708">
        <Shape>
          <Cylinder radius="1" height="2.6" top="true"/>
          <Appearance>
            <Material diffuseColor="0.286275 0.258824 0.247059"/>
          </Appearance>
        </Shape>
      </Transform>
    </Group>
    <Group DEF="2" class="axle" canTurn="1"
    turnRate="0.698132" turnAngle="0.785398">
      <Transform translation="4.2 1 0" rotation="1 0 0 1.5708">
        <Shape>
          <Cylinder radius="1" height="2.6" top="true"/>
          <Appearance>
            <Material diffuseColor="0.286275 0.258824 0.247059"/>
          </Appearance>
        </Shape>
      </Transform>
    </Group>
    <Group DEF="3" parent="0" class="sensor" fov="3.14159" range="80">
      <Transform translation="7.2275 2 0"
      rotation="0 1 0 0">
        <Shape>
          <Box size="0.155 0.19 0.155"/>
```

```
      <Appearance>
        <Material diffuseColor="0.368627 0.631373 0.803922"/>
      </Appearance>
    </Shape>
  </Transform>
</Group>
<Group DEF="4" parent="0" class="sensor" fov="3.14159" range="80">
  <Transform translation="-7.2275 2 0" rotation="0 1 0 3.14159">
    <Shape>
      <Box size="0.155 0.19 0.155"/>
      <Appearance>
        <Material diffuseColor="0.368627 0.631373 0.803922"/>
      </Appearance>
    </Shape>
  </Transform>
</Group>
<Group DEF="5" class="box">
  <Transform translation="-6.25 1.5 0" rotation="0 1 0 0">
    <Shape>
      <Box size="1.7 1 3"/>
      <Appearance>
        <Material diffuseColor="1 0.909804 0.686275"/>
      </Appearance>
    </Shape>
  </Transform>
</Group>
<Group DEF="6" class="box">
  <Transform translation="6.2 1.5 0" rotation="0 1 0 0">
    <Shape>
      <Box size="1.8 1 3"/>
      <Appearance>
        <Material diffuseColor="0.937255 0.74902 0.27451"/>
      </Appearance>
    </Shape>
  </Transform>
</Group>
<Group DEF="8" class="box">
  <Transform translation="-0.05 1.5 0" rotation="0 1 0 0">
    <Shape>
      <Box size="6.3 1 3"/>
      <Appearance>
        <Material diffuseColor="1 0.909804 0.686275"/>
      </Appearance>
    </Shape>
  </Transform>
</Group>
<Group DEF="17" class="box">
  <Transform translation="6.05 3.75 0" rotation="0 1 0 0">
```

```
        <Shape>
          <Box size="2.1 1.5 3"/>
          <Appearance>
            <Material diffuseColor="1 0.909804 0.686275"/>
          </Appearance>
        </Shape>
      </Transform>
    </Group>
    <Group DEF="18" class="box">
      <Transform translation="0 4 0" rotation="0 1 0 0">
        <Shape>
          <Box size="2 2 3"/>
          <Appearance>
            <Material diffuseColor="1 0.909804 0.686275"/>
          </Appearance>
        </Shape>
      </Transform>
    </Group>
    <Group DEF="19" class="box">
      <Transform translation="-3 3.5 0" rotation="0 1 0 0">
        <Shape>
          <Box size="4 1 3"/>
          <Appearance>
            <Material diffuseColor="0.937255 0.74902 0.27451"/>
          </Appearance>
        </Shape>
      </Transform>
    </Group>
  </Scene>
</X3D>
```

B An example vehicle in SVG format

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:svg="http://www.w3.org/2000/svg"
  xmlns="http://www.w3.org/2000/svg" width="292.2" height="200"
  viewBox="-146.1 -112.5 292.2 200" version="1.1">
  <!-- SVG model generated by MaDe -->
  <g class="box" id="0" stroke="black" fill="#efbf46">
    <rect class="xy" x="-71.5" y="-65" width="143" height="30"/>
    <rect class="xz" x="-71.5" y="20" width="143" height="10"/>
  </g>
  <g class="axle" id="1" stroke="black" fill="#49423f"
  canTurn="1" turnRate="40" turnAngle="45">
    <circle cx="-43" cy="40" r="10"/>
    <rect class="xy" x="-53" y="-63" width="20" height="26"/>
  </g>
  <g class="axle" id="2" stroke="black" fill="#49423f"
  canTurn="1" turnRate="40" turnAngle="45">
    <circle cx="42" cy="40" r="10"/>
    <rect class="xy" x="32" y="-63" width="20" height="26"/>
  </g>
  <g class="sensor" id="3" parent="0" stroke="black"
  fill="#5ea1cd" fov="180" range="80">
    <rect class="xy" transform="rotate(0, 72.275, -50)"
    x="71.5" y="-50.775" width="1.55" height="1.55"/>
    <rect class="xz" x="71.5" y="29.05" width="1.55" height="1.9"/>
  </g>
  <g class="sensor" id="4" parent="0" stroke="black"
  fill="#5ea1cd" fov="180" range="80">
    <rect class="xy" transform="rotate(180, -72.275, -50)"
    x="-73.05" y="-50.775" width="1.55" height="1.55"/>
    <rect class="xz" x="-73.05" y="29.05" width="1.55" height="1.9"/>
  </g>
  <g class="box" id="5" stroke="black" fill="#ffe8af">
    <rect class="xy" x="-71" y="-65" width="17" height="30"/>
    <rect class="xz" x="-71" y="30" width="17" height="10"/>
  </g>
  <g class="box" id="6" stroke="black" fill="#efbf46">
    <rect class="xy" x="53" y="-65" width="18" height="30"/>
    <rect class="xz" x="53" y="30" width="18" height="10"/>
  </g>
  <g class="box" id="8" stroke="black" fill="#ffe8af">
    <rect class="xy" x="-32" y="-65" width="63" height="30"/>
    <rect class="xz" x="-32" y="30" width="63" height="10"/>
  </g>
```

```
  <g class="box" id="17" stroke="black" fill="#ffe8af">
    <rect class="xy" x="50" y="-65" width="21" height="30"/>
    <rect class="xz" x="50" y="5" width="21" height="15"/>
  </g>
  <g class="box" id="18" stroke="black" fill="#ffe8af">
    <rect class="xy" x="-10" y="-65" width="20" height="30"/>
    <rect class="xz" x="-10" y="0" width="20" height="20"/>
  </g>
  <g class="box" id="19" stroke="black" fill="#efbf46">
    <rect class="xy" x="-50" y="-65" width="40" height="30"/>
    <rect class="xz" x="-50" y="10" width="40" height="10"/>
  </g>
</svg>
```

C The Stage model of an example vehicle

```
define test_machine position
(

    gui_nose 1
    obstacle_return 0
    ranger_return 1
    blob_return 1
    fiducial_return 1

    localization "gps"
    localization_origin [0 0 0 0]
    color_rgba [ 0.97311 0.84090 0.50980 1 ]
    origin [ 0.00000 0.00000 0 0 ]
    size [ 14.30000 3.00000 5.00000 ]

    test_scanner1( pose [ 7.22750 0.00000 -3.09500 0.00000 ] )

    test_scanner1( pose [ -7.22750 0.00000 -3.09500 180.00001 ] )

    test_axle1( pose [ -4.30000 0.00000 -5.00000 0 ] )

    test_axle1( pose [ 4.20000 0.00000 -5.00000 0 ] )

    block
    (
        points 4
        point[0] [ -7.15000 -1.50000 ]
        point[1] [ 7.15000 -1.50000 ]
        point[2] [ 7.15000 1.50000 ]
        point[3] [ -7.15000 1.50000 ]
        z [ 2.00000 3.00000 ]
    )

    block
    (
        points 4
        point[0] [ -7.10000 -1.50000 ]
        point[1] [ -5.40000 -1.50000 ]
        point[2] [ -5.40000 1.50000 ]
        point[3] [ -7.10000 1.50000 ]
        z [ 1.00000 2.00000 ]
    )

    block
    (
        points 4
```

```
        point[0] [ 5.30000 -1.50000 ]
        point[1] [ 7.10000 -1.50000 ]
        point[2] [ 7.10000 1.50000 ]
        point[3] [ 5.30000 1.50000 ]
        z [ 1.00000 2.00000 ]
    )

    block
    (
        points 4
        point[0] [ -3.20000 -1.50000 ]
        point[1] [ 3.10000 -1.50000 ]
        point[2] [ 3.10000 1.50000 ]
        point[3] [ -3.20000 1.50000 ]
        z [ 1.00000 2.00000 ]
    )

    block
    (
        points 4
        point[0] [ 5.00000 -1.50000 ]
        point[1] [ 7.10000 -1.50000 ]
        point[2] [ 7.10000 1.50000 ]
        point[3] [ 5.00000 1.50000 ]
        z [ 3.00000 4.50000 ]
    )

    block
    (
        points 4
        point[0] [ -1.00000 -1.50000 ]
        point[1] [ 1.00000 -1.50000 ]
        point[2] [ 1.00000 1.50000 ]
        point[3] [ -1.00000 1.50000 ]
        z [ 3.00000 5.00000 ]
    )

    block
    (
        points 4
        point[0] [ -5.00000 -1.50000 ]
        point[1] [ -1.00000 -1.50000 ]
        point[2] [ -1.00000 1.50000 ]
        point[3] [ -5.00000 1.50000 ]
        z [ 3.00000 4.00000 ]
    )

)
```

```
define test_sensor1 sensor
(
    obstacle_return 0
    range [ 0.0 80.00000 ]
    pose [ 0 0 0 0 ]
    fov 180.00000
    samples 361.00000
    color_rgba [ 0.36863 0.63137 0.80392 0.15 ]
)

define test_scanner1 ranger
(
    obstacle_return 0
    color_rgba [ 0.36863 0.63137 0.80392 1 ]
    size [ 0.15500 0.15500 0.19000 ]

    block
    (
        points 4
        point[0] [ -0.07750 -0.07750 ]
        point[1] [ -0.07750 0.07750 ]
        point[2] [ 0.07750 0.07750 ]
        point[3] [ 0.07750 -0.07750 ]
        z [ 0 0.19000 ]
    )

    test_sensor1()
)

define test_axle1 model
(
    name "test_axle1"
    obstacle_return 0
    color_rgba [ 0.28627 0.25882 0.24706 1 ]
    size [ 2.00000 2.60000 2.00000 ]

    block
    (
        points 4
        point[0] [ 0 0 ]
        point[1] [ 0 2.60000 ]
        point[2] [ 2.00000 2.60000 ]
        point[3] [ 2.00000 0 ]
        z [ 0 2.00000 ]
    )
)
```