A Decentralised System Approach for Controlling AGVs with ROS

Robert Walenta Twan Schellekens

AGVR GmbH
Auf der Hüls 197
Aachen, Germany

robert.walenta@alumni.fh-aachen.de
twan.schellekens@agvr.eu

Alexander Ferrein Stefan Schiffer

The Mobile Autonomous Systems
and Cognitive Robotics Institute
FH Aachen University of Applied Sciences
Aachen, Germany

ferrein@fh-aachen.de
s.schiffer@fh-aachen.de

***Abstract*—The current industrial state of the art for automated guided vehicles relies on centralised controllers dispatching transports to vehicles along predefined paths. The design of these paths is time-consuming and has to be done in advance, followed by an extensive testing phase. In the field of mobile robotics, robust path planning and navigation algorithms exist. However, they have not yet found their way into industrial applications in larger numbers. In this paper, we present a system architecture for a decentralised control of multiple automated guided vehicles performing material transportation tasks in intra-logistic applications which is based on mobile robotics solutions. The proposed system includes solutions for self-localisation, behaviour control, conflict-free routing and motion control. The non-centralised control of the system architecture allows for dynamic path planning and traffic coordination. Its implementation is based on the Robot Operating System, the de-facto standard middleware for robotics applications. We give an overview of the overall system architecture as well as the coordination mechanisms and show a first proof of concept in simulations.**

I. Introduction

Changing product lines, customised products and sophisticated production requirements demand flexible, adaptive and intelligent industrial manufacturing systems. The complexity of these systems is especially challenging for intra-logistic processes. Intra-logistics describes the organisation, realisation and optimisation of internal material flow. This requires efficient material transportation, which presumes non-blocking and non-empty runs.

Therefore, the use of automated guided vehicles (AGV) has been well-established in the last decades. AGVs are automated mobile platforms and are used since the 1950s for repetitive material handling tasks in industrial manufacturing and warehouse systems [1]. Running 24/7, AGVs are increasing the productivity while reducing labour costs. Their endurance, precision and reliability are ensuring predictable and traceable intra-logistic processes. Additionally, high safety standards are circumventing serious harm to human workers and help avoiding damages to the facility.

Simple transport tasks are usually rare in industrial applications. More common are requirements demanding for high payloads up to $65\,\mathrm{t}$, sophisticated pick-and-drop handling, high flow rates, temperature ranges between $-27\,^{\circ}\mathrm{C}$ to $50\,^{\circ}\mathrm{C}$, indoor and outdoor use cases and collaborative applications. In the past few years, the topic of Industry 4.0 becomes ever more important for the field of intra-logistics. The goal here is to have AGV systems fully integrated with the existing IT systems. Therefore, a new degree of autonomy is required which enables operating the fleet of UGVs from the office desk [2].

AGVs are designed to follow predefined tracks, similar to trains that are moving along the rails. These tracks are based on magnet- or inductive guide tapes, implemented on or in the site floor. This rather old-fashioned approach is still used in industrial applications, but is currently being replaced by virtual tracks designed in CAD programs based on the 2D layouts of the facility. These methods have a long preparation and implementation time, including a thorough testing phase at the facility site. This further increases the implementation cost. As a consequence common AGV systems are not flexible in adapting to changes. Also, their ability to perceive the environment is limited, so simple errors can lead to material flow interrupts.

In this paper, we propose a novel system concept that allows a higher degree of autonomy and enables the implementation of current developments for industrial usage. Our approach comes with more flexibility in the path planning and, hence, it is easier to adapt the approach to changes in the environment. The implementation is based on several standard packages that comes with the Robot Operating System (ROS), on the one hand; on the other hand, it relies on a number of custom-build path planning and routing algorithms which are tailored to the intralogistics scenario.

The rest of the paper is organised as follows. In the next section, we give an overview of different AGV types that are used in industrial intra-logistics applications and review some related approaches. In Section III, we give a brief introduction to the Robot Operating System, which the implementation of our approach is based on. In Section IV we present the architecture of our non-centralised control system before we conclude.

II. Automated Guided Vehicle

AGVs are deployed in a number of different application domains and the range of vehicle types has increased alongside customers' needs. Primary applications are in manufacturing, warehousing, automotive, chemical, paper-print, food and healthcare industry. The variety of applications specify the general system requirements, such as size, load-capacity, load-mechanism, navigation constraints, the number of deployed vehicles, and type of the environment. In the following, we discuss the different requirements.

The load is the most crucial requirement for material transportation and differ due to the use-case in size and shape, where the weight can raise up $65\,\mathrm{t}$. AGVs are therefore operating as a unit-loader or as a mobile manipulator. If only transportation of goods is required, the AGV is either loaded manually or by some automatic equipment. If interaction with stands or conveyors is needed, an effector will perform the material handling. For handling tasks where palletisation is not possible, effectors like clamps or industrial robots handle rolls, boxes and other raw materials. Otherwise, actuated forks lift and drop material units are deployed. Regarding the position accuracy when manipulating materials, usually only low tolerances within $10\,\mathrm{mm}$ or less are allowed. This constrains, in particular, the localisation system, but also has an influence on the type of kinematic drive chosen. Well-established are localisation systems relying on reflector beacons operating with a positioning accuracy of $4\,\mathrm{mm}$ [3]. The type of kinematic drive depends on the mass distribution, the required accuracy and the degree of freedom needed. Common in industrial applications are kinematic systems based on a tricycle or differential drive, skid steering or asynchronous drive mechanisms. In practice, the amount of actuated wheels is limited to reduce cost and, hence, increase the profitability of the system.

Automated guided vehicles (AGVs) is a rather long-established discipline in mechanical engineering. A quick web research yields numerous patents and a large number of publications from the 80's and the 90's. There are also a number of successful industrial players around who offer AGVs on the market for intra-logistics automation in production lines or warehouses. The market share of USD 810 million in the USA in 2015 is expected to grow by 7.3 % until 2024, as the summary of a market forecast suggests [4]. The technology used in many solutions is rather conservative using magnetic stripes on the shop-floor etc. On the other hand, in the last decade or so, many new innovative solutions appeared on the market that make use of mobile robotics technology. One of them is the KIVA system [1], now amazonrobotics. There, hundreds of so-called pods, small robots that can pick up inventory shelves, drive through a warehouse and bring the shelves to pick workers who commission online orders. Another rather novel approach is taken by Fetch Robotics. Founded in 2014 by a former Willow Garage employee, they just presented their latest development: the Freight500 (see, for instance, [5]). It resembles very much the Clearpath OTTO robot (e.g. [6]). What both these systems have in common is that they make use of the Robot Operating System [7] for their intelligent control software, as Melonee Wise, the founder of Freight Robotics, explains in an interview with the Industrial Robot Journal [8].

As our approach is likewise based on ROS, we outline the Robot Operating System in the next section.

III. The Robot Operating System

To implement a navigation system which fulfils the requirements of an industrial material transportation system, a software system has to be designed that is capable of coordinating all vehicles in the system, ensuring a conflict-free travelling and performing material handling actions. To do so, methods are required to localize the vehicle, perceive the environment and plan optimal paths through the environment. Additionally, a hardware abstraction layer is required to enable the communication with sensors and actuators. A driver software on top has to interface with high-level applications. This enables an evaluation of sensor readings and to control the robot system. ROS is an open-source middleware for robotic platforms. ROS provides all necessary features of an operating system and enables the development of applications in C++ and Python. ROS is published under the BSD license, allowing for commercial use of ROS [9]. It provides a large open-source robotics library with state-of-the-art solutions for robotics tasks such as localisation, path planning, image processing, collision avoidance and motion control. The ROS runtime environment manages the execution of applications and the inter-process communication. One of the basic goals of ROS is to enable small, mostly independent programs called *nodes* that all run at the same time and can be executed on different machines. That concept allows for a modularisation and a computational distribution [10].

IV. Control System Overview

A. System Outline

The control system is implemented completely in ROS. We made use of a number of already existing ROS components and integrated them with some packages tailor-made for the AGVR. Fig. 1 gives an overview of the overall system architecture. The control system consists of the following components:

- Warehouse Management System (WMS)
- Action Level Execution Agent (ALEA)
- Navigation Graph (NavGraph)
- ROS Move-Base (MoveBase)
- Time Elastic Band planner (TEB)

In the following, we describe the different components in detail and refer to their ROS implementations.

1) WMS: The Warehouse Management System controls the movement and the storage of materials within the warehouse. It is optimising the fragmented material storage by utilising the available resources making use of the current input and output demand. This is stimulating the material flow and so the transport system. Transport orders are published and

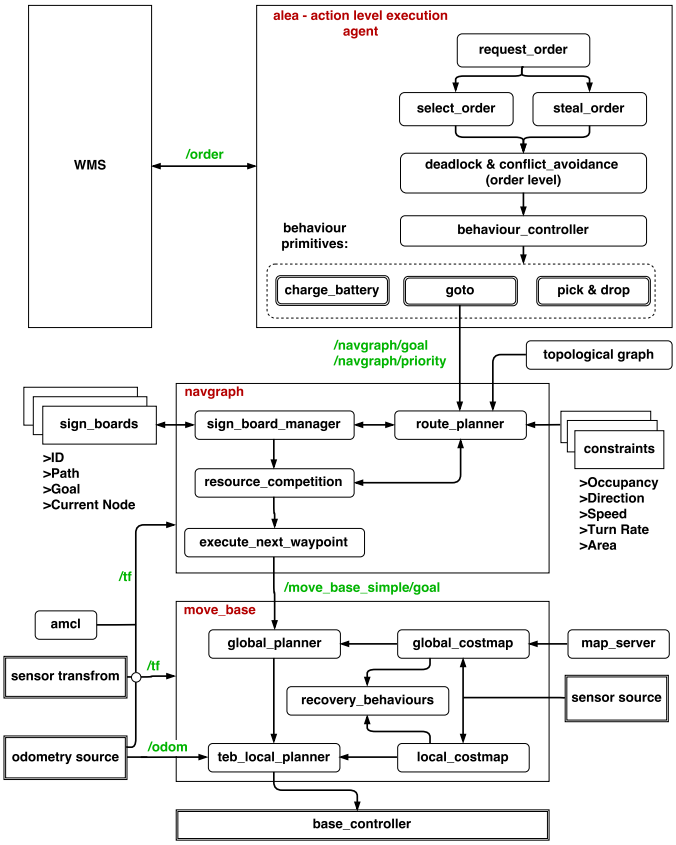


Fig. 1: The overall system components

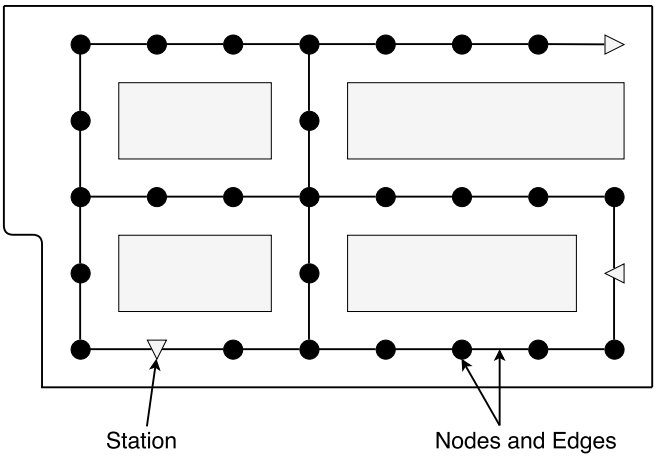


Fig. 2: Topological Graph

broadcasted to every transport vehicle. To deal with network issues all orders are stored on the AGVs itself.

2) ALEA: The Action Level Execution Agent is a behaviour controller which is monitoring the state of the AGV and controlling primitive actions such as *pick*, *drop*, *goto* or *chargeBattery*. It is continuously listening to broadcasted orders and selects the optimal order by priority, occurrence, distance and time constraints. Knowing the position and the assignment of all orders, ALEA assists in ensuring a blocking and deadlock-free traffic. So when priority or traffic state competitions determine a better vehicle assignment, order reallocation or "stealing" actions can be performed to improve the material flow performance.

3) NavGraph: The *NavGraph* is a topological graph-based route planner, initially developed for the robotic software framework Fawkes [11]. The navigation graph consists of nodes, segments and station elements (Fig. 2). In this work the graph is based on bidirectional segments to enable a flexible traffic flow. Each graph element is capable to hold properties that can either influence the behaviour controller or the path planning itself.

4) Navigation Stack: The ROS *Navigation Stack* is a software collection for self-localisation, path planning and motion control. Core application is the ROS *move_base* package. *move_base* is implemented as an ROS *action*, giving a goal, it will attempt to reach the destination using odometry and other sensor readings while sending velocity commands to the mobile robot base [12].

These are the base components for navigating the AGV with ROS. In the following we go more into the details on the navigation system.

B. Navigation Graph

A central component of the control architecture is the Navigation Graph package for allowing dynamic and flexible path planning. The *NavGraph* package was extended to coordinate the AGV fleet and is executed on each vehicle to achieve a decentralised system. Therefore inter-vehicle communication is required to exchange information and the current state between the vehicles. Considering upcoming traffic conflicts, the same rules are defined on each vehicle to ensure that every AGV comes up with the same solution. In the following we give more details on the mechanisms behind this concept.

NavGraph can annotate each graph element with constraints. Such constraints have a direct effect on the path planning and can, for example, force a vehicle to drive only forwards. Based on the navigation graph, an A* search is performed to determine the optimal path between a source and destination node. Through the inter-vehicle communication, *NavGraph* considers priority, current position, desired path and the final goal of each vehicle. Checking all published routes, *NavGraph* evaluates possible encounterings of vehicles. Based on a priority competition *NavGraph* decides, which vehicle has the right of way. This approach is based on [13].

NavGraph distinguishes between two phases for executing a *NavGraph* goal, an *offline* and an *online phase*. The former describes the period before the plan execution. Here, the *NavGraph* package determines plans with or without particular path constraints. The path without particular constraints represents the shortest possible route to the goal, a constrained path is considering blockings and cost increasing factors like speed or turning rates and represents a plan with possible detours that have to be taken. *NavGraph* trades off between the unconstrained and constrained path calculating a normed difference and relates the unconstrained plan as the reference

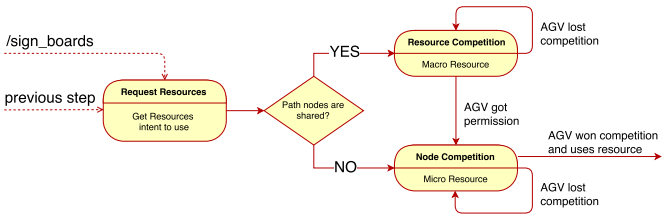


Fig. 3: Resource competition scheme as modelled in [13]

$d_c = \frac{(c_c - u_c)\cdot 100\%}{u_c}$, where, d_c equals the detour cost in percent, c_c stands for the constrained plan cost, and u_c resembles the unconstrained path costs. Based on a threshold value it is decided whether or not to execute a plan with a detour.

A detour is considered based on blocked pathways, but also several other constraints such as narrow pathways or crossings are taken into account.

Once a plan has been computed, all resources required by that plan have to be requested. *NavGraph* distinguishes between *micro resources* (M_r) and *macro resources* (M_R). Micro resources are nodes and edges which consist of a path between the current vehicle position and the destination. Macro resources are compositions of micro resources, typically a corridor, narrow passage, or a crossing. Micro resources can be blocked by vehicles that are close-by for avoiding collisions between vehicles. Macro resources serve the purpose of vehicle coordination. They can be *requested* or *allocated* by vehicles in the system in the online execution phase of the plan.

In the online phase, *NavGraph* executes the previously computed plan. To ensure a blocking-free execution of a plan, *NavGraph* needs to take each vehicle in the system into account. Therefore, state reports of a particular vehicle are published and broadcasted to all vehicles in the fleet. This inter-vehicle communication is based on so-called *sign board messages*, which consist of the vehicle ID, the order priority, the position of the currently occupied node, the target node, the current speed and the vehicle's intended path to the destination.

On the implementation level, the inter-vehicle communication is realised by a publisher/subscriber mechanism. For each vehicle, a separate callback is defined which monitors arriving messages and updates a local macro resource and micro resource repository. Once a sign board message arrives, an evaluation of the traversal progress is performed. All nodes that have been passed by a vehicle are released together with its corresponding macro resources. All resources that are intended to be used are requested. On the micro resource level, the callback listens to position updates from each vehicle. Nodes in the vicinity of a vehicle pose, will be determined and be marked as blocked to avoid collisions between vehicles.

The behaviour of the coordination system, which is executed on each vehicle, can be divided into the following steps: (1) request macro resources, (2) check for shared nodes, (3) compete for macro resources, (4) compete for micro resources.

Figure 3 visualises a simplified competition scheme, which describes that an AGV has to ask for permission before it

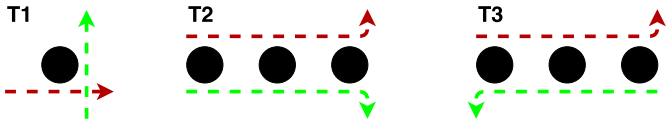


Fig. 4: Encounter types: (T1) Crossroad, (T2) Follower, (T3) Frontal (as given in [13])

Fig. 5: Time phases of two conflicting vehicles

can use a macro resource. That competition is priority based and is calculated for each macro resource allocation request. The priority is based on a weight sum function which has the following parameters: order priority, order maturity and accumulated speed that is allowed to drive on the path. The order priority gets assigned the highest weight in order to ensure on-demand commissioning; the order maturity is to avoid starvation. The macro resource competition is also used for determining conflicting encounters of AGVs. Conflicting situations occur when resources are shared between pairs of vehicles. There are three possible encounter types (see Fig. 4). The three encounter types can be seen as sub-sequences of nodes and characterised as follows according to the approach in [13]: **T1** occurs when only a single node is shared and the rest of the sequences are unaffected, **T2** occurs when at least two nodes are shared and the rest of both sequences are unaffected, **T3** occurs when at least two nodes are shared and one node sequence is inverted. The encounter determination leads to the following information abstraction:

1) Identify the first conflicting M_R and M_r
2) Identify the time to arrive at the conflict sequence
3) Identify the duration to pass the conflicting segments

Behaviour 1 above is used to stop a vehicle before the actual conflict; the higher prioritised vehicles is given the right of way. To avoid situations where a vehicle has to wait until all conflicting resources are released, a trade-off is between the arrival time and conflict duration using the behaviour 2 and 3 from above is used.

Figure 5 shows the different time phases of a conflict situation between two vehicles, where t_{AGV1} and t_{AGV2} stands for the time the vehicles need to arrive at the conflict sequence, and $t_{conflict}$ resembles the time it take to pass the conflict sequence. In the case of $t_{AGV1} + t_{conflict} \ll t_{AGV2}$, $AGV1$ is allowed the way of right despite a lower priority. This leads to a better transportation flow and decreases the overall vehicle waiting time. Another improvement is the determination of follower sequences, where two vehicles heading into the same direction. Instead of allowing only one vehicle to use the resource, in that case, both gets the permission regardless their priority. Is the leading vehicle delayed for some reason the following will stop due to the micro level competition.

The *NavGraph* has the capability to act not only as a global planner and determine paths in the topological graph, but is also suitable to plan, determine and resolve traffic management conflicts by exchanging information based on an inter-vehicle communication.

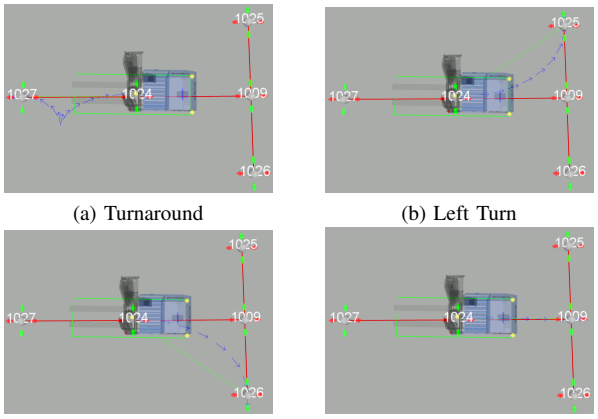
C. The MoveBase and Trajectory Planning with TEB

The *base_controller* handles the communication with the virtual hardware described in Section IV-D. Figure 1 shows the *move_base* as the last software level before that *base_controller*. The *move_base* is interfacing with the 2D LIDAR scanners to perceive the environment and provide a contour-based odometry (scan matching odometry). This data is combined with the wheel odometry provided by the VHD and enhanced with an adaptive particle filter called *amcl*. The resulting localisation is tracking the vehicle pose on a previously recorded map. The path planning part is divided into *global* and *local* planning. Regarding the incorporation with the *NavGraph*, the *global* path planner generates the shortest path between two nodes. This is reducing the *global* planner's effort due to the distance between two nodes, that is limited by the geometric layout of the facility. Ideally, the global planner matches the connecting edge between two nodes. This is always the case when no static obstacles interfere with the direct path. If an obstacle is intersecting the path, the global planner attempts to avoid it taking the vehicle geometric 2D shape into account. The *local* planner integrates the current sensor readings and also monitors dynamic changes in the environment. Following the global path, the local planner is also able to plan alternative paths around an obstacle. Due to the tricycle-wheel configuration, the local planner has also to consider the vehicle kinematic constraints, while sending velocity commands in the form $\vec{v} = (\dot{\vec{x}}\ \dot{\vec{y}}\ \dot{\theta})^T$.

AGVs are designed to follow predefined tracks, similar to trains that are moving along rails. In our approach, we deploy the *timed elastic band* planner (TEB) [14] for local path planning. TEB generates a trajectory considering vehicle dynamic and kinematic constraints. Adapting the trajectory constraints (limiting the velocity, acceleration or goal tolerance) leads to a customised behaviour with different local optimal solutions. Communicating directly with the underlying robot motion controller, TEB highly flexible and can be adapted to different robot kinematics and application requirements [15].

Regarding the incorporation with the *NavGraph*, TEB is responsible to travel from the current vehicle pose to an specified target node. Figure 7 visualises basic trajectory primitives that can be performed by the vehicle. *NavGraph* nodes are displayed as spheres and linking segments as red lines. Marked in green can be seen the resulting shortest path generated by the *global* planner and the TEB solution is visualised with blue vectors.

Considering that *NavGraph* provides the start and end position, the node constellation is specifying the geometric dependency of the *local* planner. If the node constellation is not matching the configured geometric and dynamic constraints, TEB will generate sub-optimal or infeasible trajectories.



(a) Turnaround (b) Left Turn

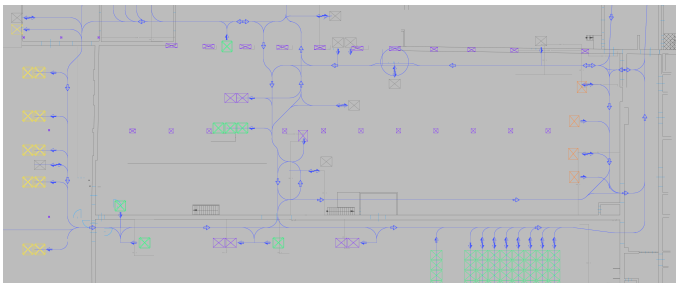
(c) Right Turn (d) Straight Move

Fig. 7: Trajectory primitives

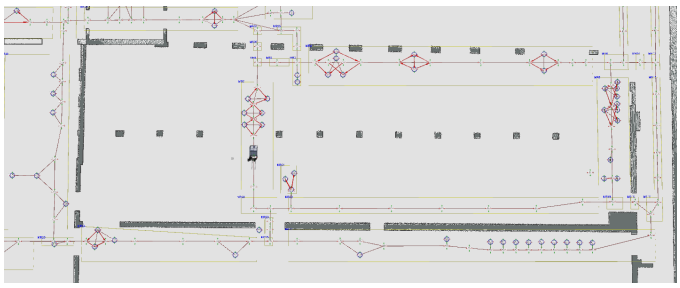
Therefore TEB has to be optimised for *kinematic saturation* and operate in proper limits of the actuation system before the node layout is designed. Due to the used tricycle drive, the vehicle is able to perform car-like motions and execute smooth trajectories which reduced the tire wear. Additionally, a turn in place is possible when actuating the steering wheel $-\pi$ to π, but tending the turning radius to zero would lead to very low translational velocities. Therefore, operating TEB for car-like behaviour requires a trade-off between optimising for linear velocity or maximum steering angle. Although the configuration is time consuming, TEB is significantly reduces the complexity of the topological graph. Because of the dynamic path generation, even complex trajectories are feasible with two node pairs considering multiple start and goal poses (see Fig. 6 and 6a). Another advantage is the possibility to avoid dynamic obstacles in the environment. For situations where the area space is sufficient this can guarantee a static martial flow instead of stopping the AGV and wait until an operator handles the blocking situation. This leads also to a new traffic management concept, where large areas are not constraint anymore by graph elements and the AGV fleet is driving without the discrete knowledge about the position of each vehicle in collision avoidance mode.

D. Simulation Model

The simulation model is realised as a virtual model in the simulation environment Gazebo that comes with ROS (see Fig. 8). Using the ODE physics engine, a realistic motion generation has been achieved. The AGV is based on a tricycle kinematic model, in Figure 8b marked in red. The frames *right_wheel* and *left_wheel* are passive and are used to calculate the vehicle odometry, while the frame *front_steering* can be actuated in linear x and angular z direction. A virtual hardware driver (VHD) is translating linear and angular velocity commands into corresponding joint rotational velocities to provide propulsion and steering. To simulate the dynamic behaviour the steering speed can be limited. Additionally, three virtual 2D LIDAR sensors are implemented to provide a perception of the environment with an $360°$ field of view. This



(a) (Part of the) layout from a real shop floor showing possible trajectories



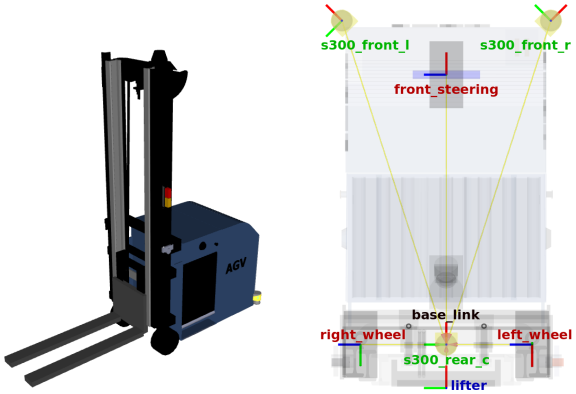
(b) (Part of the) shop floor layout modeled with *NavGraph*

Fig. 6: Example shop floor layout

enables to sense static and dynamic obstacles, and based on the contour information to localize the vehicle in the world. A more detailed explanation follows in section IV-A4. Last vehicle model element is the frame *lifter*, marked in 8b in blue. This frame is realised as an prismatic Joint and is also controllable through the VHD. The lifter is equipped with standard forks to pick euro pallets, but different effectors like clamps of custom mechanism are possible. All frames are attached to the *base_link* frame which is the local vehicle coordinate system, marked in Fig. 8b as black.

V. Summary and Perspective

In contrast to the current state of the art in implementing industrial AGV systems where the fleet of AGVs is controlled centrally, the architecture proposed in this paper realises a decentralised control with fully autonomous vehicles. The AGV is capable of localising itself in a known environment and finds its path through a discrete bi-directional topological graph. In addition, the proposed system enables a simplification of the global planning problem, limits the operating range of the vehicle, accelerates the implementation time, reduces the topological graph complexity and enables a dynamic adaption to environmental changes and enables a predictive traffic management based on an inter-vehicle communication attempting to maximize the material flow-rate. First results from simulation in Gazebo suggest that the position repeatability has to be improved. The current measured values imply values of 150mm (RMS), what is 15 times higher then the industrial standard. Therefore additional sensors could be used to decrease the position deviation. Considering the material flow the first results are promising and comparable with real implemented systems. This shows that ROS is a suitable framework to design systems for industrial acceptance.



(a) Perspective view (b) Top-Down view

Fig. 8: AGV Simulation Model

References

[1] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI magazine*, vol. 29, no. 1, p. 9, 2008.

[2] materialfluss.de, "Wenn Maschinen selber fahren," *Flurförderzeuge*, 2013.

[3] *Laser-Positioniersensor NAV350*, SICK, 1 2014, x041.

[4] G. V. Research, http://www.grandviewresearch.com/industry-analysis/automated-guided-vehicle-agv-market, 2016, last visited: 11/04/2017.

[5] E. Ackerman, "Fetch Robotics Introduces Burly New Freight Robots," *Robotics Blog, IEEE Spectrum*, 2017. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/industrial-robots/fetch-robotics-introduces-burly-new-freight-robots/

[6] ——, "Clearpath's OTTO Robot Can Autonomously Haul a Ton of Stuff," *Robotics Blog, IEEE Spectrum*, 2015. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/industrial-robots/clearpath-otto-can-autonomously-haul-a-ton-of-stuff

[7] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[8] J. Pransky, "The Pransky interview: Melonee Wise, CEO, Fetch Robotics," *Industrial Robot: An International Journal*, vol. 43, no. 3, pp. 253–257, 2016.

[9] M. Aaron, *Learning ROS for Robotics Programming*. Packt Publishing, 2013.

[10] P. Goebel, *ROS By Example Volume 2*. Lulu, 2014.

[11] T. Niemueller, A. Ferrein, D. Beck, and G. Lakemeyer, "Design principles of the component-based robot software framework fawkes," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 300–311.

[12] E. Marder-Eppstein. (2016) move_base package summary. [Online]. Available: http://http://wiki.ros.org/move_base

[13] S. Manca, A. Fagiolini, and L. Pallottino, "Decentralized coordination system for multiple agvs in a structured environment," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 6005–6010, 2011.

[14] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Efficient trajectory optimization using a sparse model," in *2013 European Conference on Mobile Robots*, Sept 2013, pp. 138–143.

[15] ——, "Trajectory modification considering dynamic constraints of autonomous robots," in *ROBOTIK 2012; 7th German Conference on Robotics*, 2012, pp. 1–6.