BVL

From Simulation to Real-World Robotic Mobile Fulfillment Systems

L. Xie[1], H. Li[2], N. Thieme[1]

ABSTRACT

In a new type of automated parts-to-picker warehouse system – a Robotic Mobile Fulfillment System (RMFS) – robots are sent to transport pods (movable shelves) to human operators at stations to pick/put items from/to pods. There are many operational decision problems in such a system, and some of them are interdependent and influence each other. In order to analyze the decision problems and the relationships between them, there are two open-source simulation frameworks in the literature, Alphabet Soup and RAWSim-O. However, the steps between simulation and real-world RMFS are not clear in the literature. Therefore, this paper aims to bridge this gap. The simulator is firstly transferred as core software. The core software is connected with an open-source ERP system, called Odoo, while it is also connected with real robots and stations through an XOR-bench. The XOR-bench enables the RMFS to be integrated with several mini-robots and mobile industrial robots in (removed) experiments for the purpose of research and education.

KEYWORDS: ERP System · Simulation · Robotic Mobile Fulfillment Systems · Warehousing Systems

✉ Lin Xie[1] (corresponding author for proof copy)
   xie@leuphana.de

   Nils Thieme[1]

   Hanyi Li[2]

   [1] Institute of Information Systems
   Leuphana University of Lüneburg
   Faculty of Business and Economics
   Universitätsallee 1
   21335 Lüneburg Germany
   http://www.leuphana.de/iis

   [2] Beijing HANNING ZN Tech Co.,Ltd,
   Beijing, China

1. INTRODUCTION

A Robotic Mobile Fulfillment System (RMFS) is a new type of automated parts-to-picker warehousing system, in which robots are used to transport movable shelves (also called pods), containing the inventory items, back and forth between the storage area and the replenishment or picking stations. Human operators work only at stations, either picking or replenishing items. In the former, items are picked from pods to fulfill customers' orders, while items are refilled in pods in the latter. An RMFS aims to keep human workers at the stations busy while minimizing the resources (e.g. robots, stations, pods) to fulfill the incoming pick orders.

RMFS has received more attention in the last decade, and some real-world RMFSs are currently in operation (see Figure 1). There are numerous operational decision problems in this system, such as the decisions as to which robots will carry which pods to which station to fulfill an incoming order. Most of the research about such systems is focused on these problems and algorithms to improve system performance (see an overview in [16] and [9]). Also, some of them are tested with a simulation framework (such as Alphabet Soup of [7] or RAWSim-O of [16]). This simulates the process of an RMFS, while it controls the resources. However, the steps between simulation and real-world RMFS are not clear in the literature, since these steps are highly complex and require interdisciplinary work between robotics engineering, software engineering and management science / operations research. We can also formulate the problem as this way: How can we make sure that the algorithms we implement and test in the simulator are still applicable for the real-world scenarios? This is the question we focus on in this paper. In order to answer this, four steps are required from simulation to real-world RMFS as illustrated in Figure 2. The first step is changing an RMFS simulator into RMFS core software. We use here our developed open-source simulator RAWSim-O. The second step is connecting the RMFS core software with an ERP

system and station apps. We use here an open-source ERP system – Odoo. The third step is the integration of the RMFS core software with mini-robots in an XOR-bench. The XOR-bench is a teleoperated platform for experiments. After the successful testing with mini-robots, we use the XOR-bench to test amd validate algorithms and programs remotely on a mobile industrial robot. Moreover, it is worth to mention that SAP provides extended warehouse management system (EWM) for automated warehouses to connect SAP ERP and robots (or conveyors systems, or automated guided vehicles (AGVs)) using defined interfaces. The applications are for example the AutoStore compact warehouse system and the CarryPick mobile solution of Swisslog (see [1]). Our RMFS core software shares similar priciple of SAP EWM to connect ERP's and robot's interfaces.

Therefore, there are three main contributions in this work, which are summarized as follows:

– We designed a process which aims to provide a seamless transition from the RMFS simulator to real-world RMFS.
– We connect the RMFS core software with an open-source ERP system (Odoo); moreover, the developed station apps are connected as well.
– We designed an XOR-bench, which aims to integrate the RMFS core software with the real robot and test and validate the programs on a mobile industrial robot.

We describe the RMFS and RAWSim-O in more detail in Sections 2 and 3. After that we explain the four steps of the transition process from RMFS simulator to real-world RMFS shown in Figure 2 in Sections 4, 5, 6 and 7, respectively. Section 9 concludes our work and provides directions for future research.
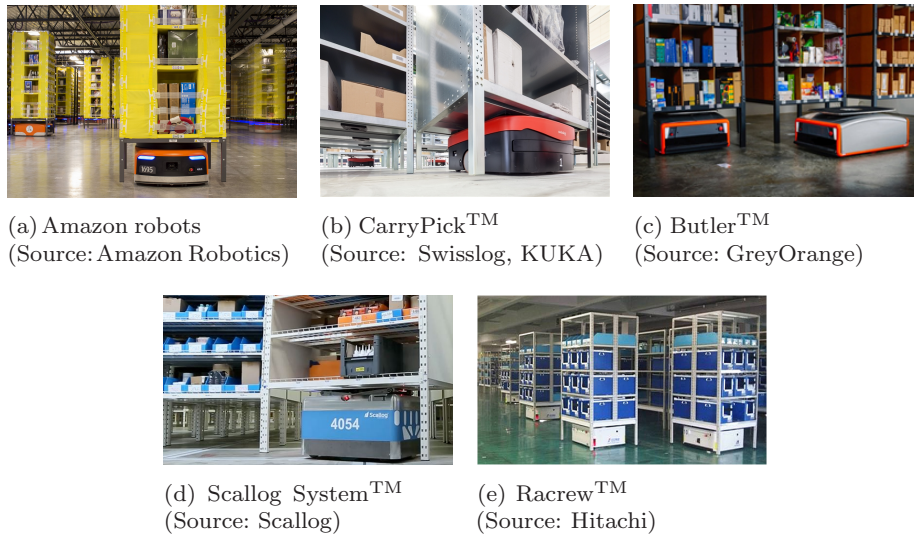

(a) Amazon robots
(Source: Amazon Robotics)

(b) CarryPick^{TM}
(Source: Swisslog, KUKA)

(c) Butler^{TM}
(Source: GreyOrange)

(d) Scallog System^{TM}
(Source: Scallog)

(e) Racrew^{TM}
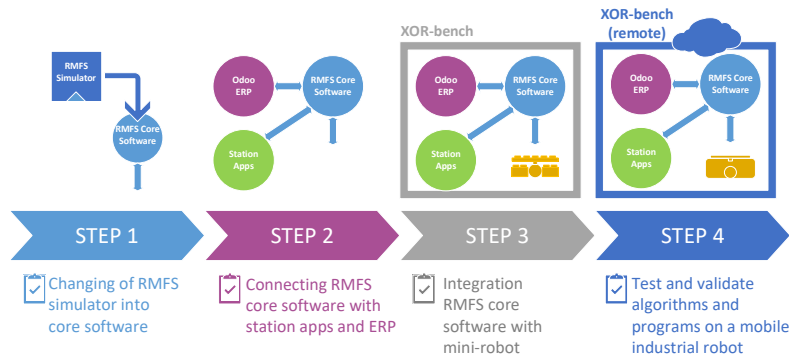(Source: Hitachi)

Fig. 1: Real-world RMFSs.

Fig. 2: From simulation to real-world RMFS in four steps.

2. THE ROBOTIC MOBILE FULFILLMENT SYSTEM

Instead of using a system of shelves and conveyors as in traditional parts-to-picker warehouses, the central components of an RMFS are:
- movable shelves, called *pods*, on which the inventory is stored
- *storage area* denoting the inventory area where the pods are stored
- workstations, where the pick order items are picked from pods (*pick stations*) or replenishment items are stored to pods (*replenishment stations*)
- mobile *robots*, which can move underneath pods and carry them to workstations.

Firstly, we need to define some terms related to orders before explaining the processes in an RMFS, as follows:
- stock keeping unit (*SKU*)
- an *order line* including one SKU with number
- a *pick order* including a set of order lines from a customer's order
- a *replenishment order* consisting of a number of physical units of one SKU

The process of an RMFS is illustrated in Figure 3. The pods are transported by robots between the storage area and workstations. Two processes are included:
- *Retrieval process:* After the arrival of a replenishment order, robots carry selected pods to a replenishment station to store units in pods. Similarly, after receiving a pick order, robots carry selected pods to a picking station, where the items for the order lines are picked. We assume it is unlikely that a pick order can be completed with only one pod, unless there is only one order line or the association policy was applied in the replenishment process (in other words, all SKUs are stored together in one pod, if they are often ordered together by the same customer.)
- *Storage process:* After a pod has been processed at one or more stations, it is brought back to a storage location in the storage area. The retrieval and storage processes are based on [8].

Then, after a pod has been processed at one or more stations, it is brought back to the storage area.

2.1. Decision Problems
In an RMFS environment, three levels of decision problems are required, namely the strategic, tactical and operational levels. The decision problems at the strategic level include storage area dimensioning and workstation placement (see [11]), while the problems at the tactical level include decisions regarding the number of robots (see [22]), and the numbers of pods and stations (see [12]). Our simulation concentrates only on various operational decision problems for pods, robots and stations that have to be solved in an online e-commerce environment, including:

- *Order Assignment* (orders to stations)
    - **Replenishment Order Assignment (ROA)**: assignment of replenishment orders to replenishment stations
    - **Pick Order Assignment (POA)**: assignment of pick orders to pick stations

- *Task Creation* (for pods)
    - *Pod Selection*
        - **Replenishment Pod Selection (RPS)**: selection of the pod to store one replenishment order (see [17])
        - **Pick Pod Selection (PPS)**: selection of the pods to use for picking the pick orders assigned at a pick station (see [3] and [23])
    - **Pod Repositioning (PR)**: assignment of an available storage location to a pod that needs to be brought back to the storage area (see [14] and [10])
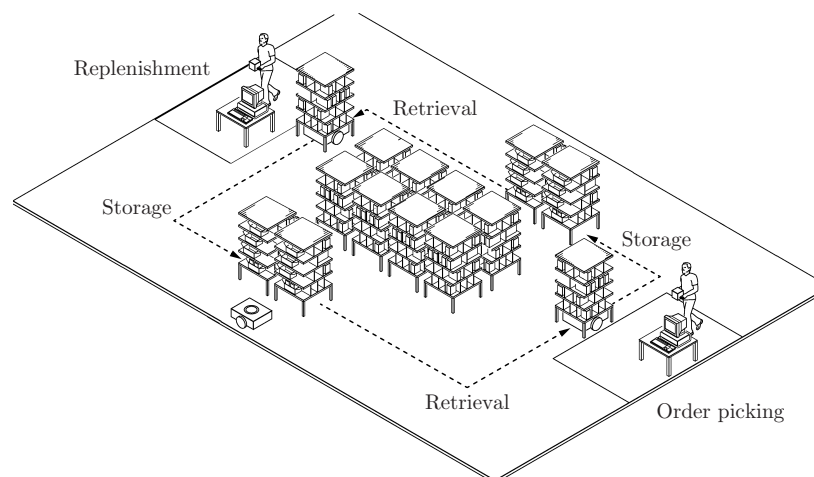

Fig. 3: The central process of an RMFS (see [8]).

– **Task Allocation (TA)** (for robots): assignment of tasks from *Task Creation* and additional support tasks like idling to robots
– **Path Planning (PP)** (for robots): planning of the paths for the robots to execute (see [4] and [5])

3. SIMULATION FRAMEWORK – RAWSIM-O

RAWSim-O is an agent-based discrete-event simulation framework. It is designed to study the context of an RMFS while evaluating multiple decision problems jointly. Figure 4a shows an overview of our simulation process, which is managed by the core *simulator* instance. The tasks of the simulator include:

– Updating *agents*, which can resemble either real entities, such as robots and stations, or virtual entities like managers, e.g. for emulating order processes.
– Passing decisions to *optimizers*, which can either decide immediately or buffer multiple requests and release the decision later.
– Exposing information to a *visualizer*, which allows optional visual feedback in 2D or 3D. Figure 4b illustrates a screenshot of our simulation in 3D.

In the following, we describe the hierarchy of all core decision problems after new replenishment or pick orders are submitted to the system (see Figure 5). If a new replenishment order is received, first the optimizers of ROA and RPS are responsible for choosing a replenishment station and a pod. This technically results in an insertion request, i.e. a request for a robot to bring the selected pod to the given workstation. A number of these requests are then combined in an insertion task and assigned to a robot by a TA optimizer. Similarly, after the POA optimizer selects a pick order from the backlog and assigns it to a pick station, an extraction request is generated, i.e. a request to bring a suitable pod to the chosen station. Up to this point, the physical units of SKUs for fulfilling the pick order are not yet chosen. Instead, the decision is postponed and taken just before PPS combines different requests into extraction tasks and TA assigns these tasks to robots. This allows the implemented optimizers to exploit more information when choosing a pod for picking. Hence, in this work we consider PPS as a decision closely interlinked with TA. Furthermore, the system generates store requests each time a pod is required to be transported to a storage location, and the PSA optimizer decides the storage location for that pod. The idle robots are located at dwelling points, which are located in the middle of the storage area to avoid blocking prominent storage locations next to the stations. Another possible type of task is charging, if the battery of a robot runs low; however, for this work we assume the battery capacity to be infinite. All of the tasks result in trips, which are planned by a PP algorithm. The only exception is when a pod can be used for another task at the same station, thus, not requiring the robot to move.
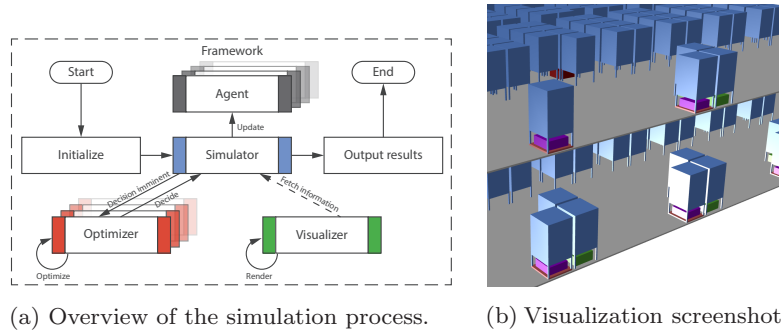

(a) Overview of the simulation process. (b) Visualization screenshot
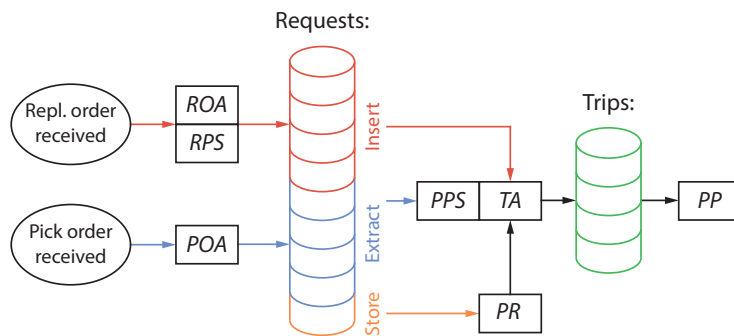
Fig. 4: RAWSim-O simulation framework.

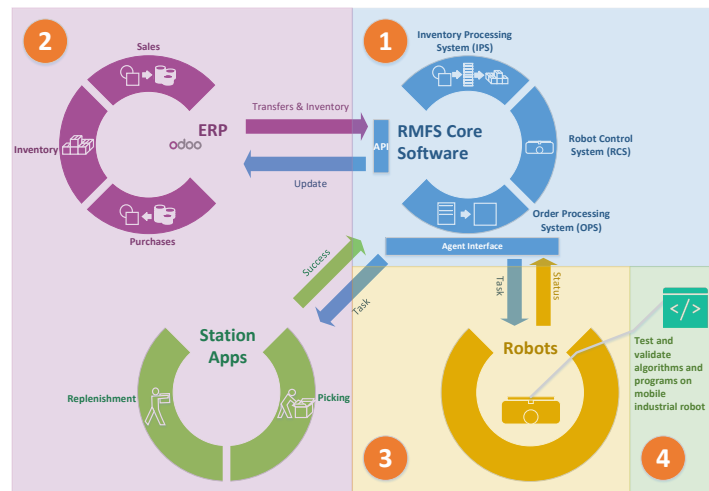*Fig. 5: Order of decisions to be done triggered by receiving a pick or replenishment order.*

Fig. 6: Relationships between the RMFS core software and the ERP system, the station apps, robots.

Figure 6 gives an overview of an RMFS, which consists of an RMFS software system and robots. The simplest RMFS software system includes the RMFS core software, ERP and station apps. The numbers in Figure 6 represent the four steps in the process for transforming, which is described in Figure 2.

4. STEP 1: CHANGING RMFS SIMULATOR INTO THE RMFS CORE SOFTWARE

The RMFS core software has three main parts, the inventory processing system (IPS), order processing system (OPS) and robot control system (RCS). The IPS functions as inventory management (the corresponding optimizers are RPS, PPS and PR), the OPS does transaction processing (the corresponding optimizers are ROA and POA), and the RCS operates and directs the robots (the corresponding optimizers are TA and PP). In this step, we extend the RMFS simulator to the RMFS core software. In total, there are two following extensions:

– Implementation of application programming interface (API) for integrating with ERP and other enterprise systems.
– Implementation of the agent interface for robots and station apps.

The API of the RMFS core software is used for integrating with the ERP or other stand-alone systems, such as WMS (warehouse management system) and OMS (order management system). A simple description of the relationship between the ERP and the RMFS core software is that by using the API, the operational status and transactions will be reported back to the ERP, and the order information and the information of items stored in pods are passed from the ERP to the RMFS

core software for processing. The API is designed for handling a variety of XML-RPC requests and XML-RPC responses (a remote procedure call (RPC) protocol [13]), because the ERP system we used – Odoo – is easily available over XML-RPC with Odoo's external API.

Task message	Meaning
Go	execute move commands
Turn	execute turning commands for a defined number of degrees to the right or left
Rest	immediately stop all actions and rest where it is currently
Pickup	pick up a pod at the current position
Setdown	set down a pod at the current position
GetItem	its carrying pod is currently used for extraction
PutItem	its carrying pod is currently used for insertion

Table 1: Meanings of task messages sent to robots.

Status message	Meaning
Error	explains an error that has occurred
WaypointTag	the current position of the robot
Orientation	the orientation of the robot
PickupSuccess	indicates whether the pickup operation was successful
SetdownSuccess	indicates whether the setdown operation was successful

Table 2: Meanings of status messages received from robots.

With an agent interface (see Figure 6) the core software can receive the status information from the robots. This includes the robot ID, the current robot position information, the orientation of the robot in radians, whether the pickup pod operation was

successful and whether the setdown pod operation was successful etc. The core software can send task messages through the agent interface to the robots, which are listed in Table 1, while it can receive some status messages from robots as shown in Table 2.

The agent interface is also used for exchanging various types of information between the core software and station apps. Table 3 lists the important information for input/output stations, which is passed from the RMFS core software to the station apps for displaying to the replenisher or picker. The confirmation or error messages from the station apps are passed back to the RMFS core software to notify the operations in the input/output stations.

5. STEP 2: CONNECTING THE RMFS CORE SOFTWARE WITH ERP AND STATION APPS

5.1. Connecting station apps

In the automation of a real warehouse, the robot moves and carries the pods to the specified input/output stations. The pickers follow instructions on the output station app, grabbing items off the pod, while the replenishers follow instructions on the input station app and stuff products onto the pods for replenishing the inventory. Here, we explain how to connect station apps with our RMFS core software.

Station information	Meaning
BundleID	ID of current bundle for replenishment (input station)
OrderID	ID of current order for picking (output station)
ItemID	product ID (input/output station)
Name	name of product/item (input/output station)
Quantity	indicates quantity of product to put or get (input/output station)
Pod Modeling Info	vertical and horizontal compartment placement within pod
Stock Level Info	optimum stock level, maximum stock level and minimum presentation quantity (input station)
Compartment Info	ID of product/item, filling rate or current count in this compartment (input/output station)
Compartment to Pick	indicates the compartment in the pod to pick (output station)
Compartment to Replenish	indicates the best compartment in the pod to replenish and the possible compartment to replenish for the current item (input station)

Table 3: Meaning of input/output station information received from the core software.

First, we develop graphical user interfaces in the programming language C# that can be used at the input/output stations of the warehouse. To allow the use of several input and output stations in an automated warehouse, each station can be identified by its ID. Second, both applications connect to the agent interface via TCP and can receive, decode and display messages containing all the necessary information for the picker/replenisher. For example, as shown in the input station app for the replenishers in Figure 7a, the selected pod compartment is marked in green. Moreover, some other possible compartments are marked in blue. Similar to the picker, the selected pod compartment is marked as shown in Figure 7b. Pickers and replenishers can also send messages containing information about the success or failure of the picking/replenishment operation over the TCP connection by pressing the OK or Error Button.

5.2 Connecting Odoo

Additionally, the API of IPS and the OPS in the RMFS core software are defined and implemented to exchange information with Odoo's external API (https://odoo.com/documentation/11.0/webservices/odoo.html), which can send and receive information using XML-RPC. We mainly use three parts of Odoo, namely purchases, sales and inventory. We use the Inventory module of Odoo to manage the location and contents of all pods and all kinds of transfers (item movement from a to b), whether incoming, outgoing or internal. We also use the Sales and Purchase modules to create sales orders and purchase orders.
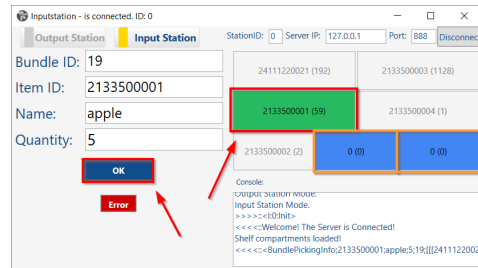
(a) Input station GUI
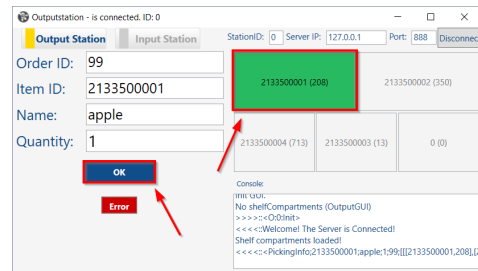
(b) Output station GUI

Fig. 7: The GUI of the station apps.

5.2.1 Order to Picking

In this part we will describe the process of an order passing through the system, from receiving an order in Odoo to the package leaving the warehouse. We describe this process firstly with a small example shown in Figure 8. After an order is created with a demand for five apples (Figure 8a) in Odoo, the sale is confirmed for sale in Figure 8b. The RMFS core software then recognizes the incoming order and selects a pod (or several, if needed) containing at least five apples and the pod is moved by a robot to the designated picking station (Figure 8c). The picker then taks the items from the pod and confirm the sucessful picking operations by pressing the OK buttom in the output station app (Figure 8d). The green marked position is the pod compartment where the items are located in the pod. The output station app gets this information from a JSON (JavaScript Object Notation)-formatted message from the RMFS core software agent interface. After a successful picking operation, the core software adjusts the inventory of the pod that was used.

We will now explain more about the technical details, referring to Figure 9. We will use some Odoo-specific terms, and some descriptions in the picking process, such as enqueuing and dequeuing of extract requests, are simplified. On startup, the RMFS core software retrieves the current positions and contents of all pods from Odoo. When an order is placed (1), Odoo creates a new planned, outgoing transfer, from Stock to Customer (2). Whenever the Update method in the RMFS core software is called (3), it scans the Odoo Transfer database for new planned, outgoing transfers (4). The transfers are converted to orders (5), for which the RMFS core software then generates and enqueues extract requests (6). An extract request is a request to pick a specific item belonging to a certain order at a chosen output station. Concurrently, the RMFS core software dequeues an extract request (7) whenever the Update method is called (3), chooses a pod to fulfill the request and moves the pod to the designated output station (8), unless it is already there because it was used to fulfill the previous extract request. Once the pod arrives at the output station, the RMFS core software sends a message containing the ID of the output station and information about the item and the affiliated order over a previously established TCP/IP connection to the agent interface (9). Once the interface receives a message (10), it decodes the message (11) and identifies its type. As in this case the type is "Pickinginfo", it then forwards the item's name and ID and the order ID to the output station that was specified in the message (12). Once the output station app receives the message (13), it displays the received information (14) and waits for the picker to finish the picking operation (15) and to respond with "OK" if it was successful (16) or "Error" if the item could not be picked (17). The information about the success of the picking operation is then sent to the agent interface (18/19). The agent interface forwards the output station's response and ID to the RMFS core software (10→11→20). If the output station's response (21) is an "OK", the RMFS core software moves the item from the pod to the station in the visualization/simulation (22), sets the pod as the move's source location in Odoo and increments the done quantity of the move by 1 (23). If this completes the last move of a transfer, the transfer is marked as done in Odoo. Odoo reacts to an increase of the done quantity (24) by removing the same quantity of the item from the source location (25). If the output station has sent an error message, the RMFS core software skips the extract request and moves it to the end of the queue (26). Finally, the RMFS core software checks if it can use the same pod again to fulfill the next extract request. If the same pod cannot be used again, a storage location will be chosen and the pod will be moved (27). Once the robot sets the pod down, the RMFS core software updates the pod's location in Odoo (28).
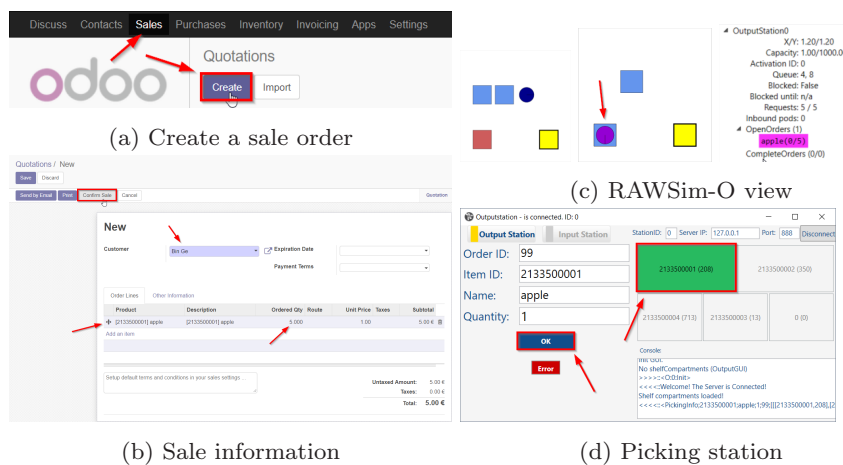

(a) Create a sale order

(b) Sale information

(c) RAWSim-O view

(d) Picking station

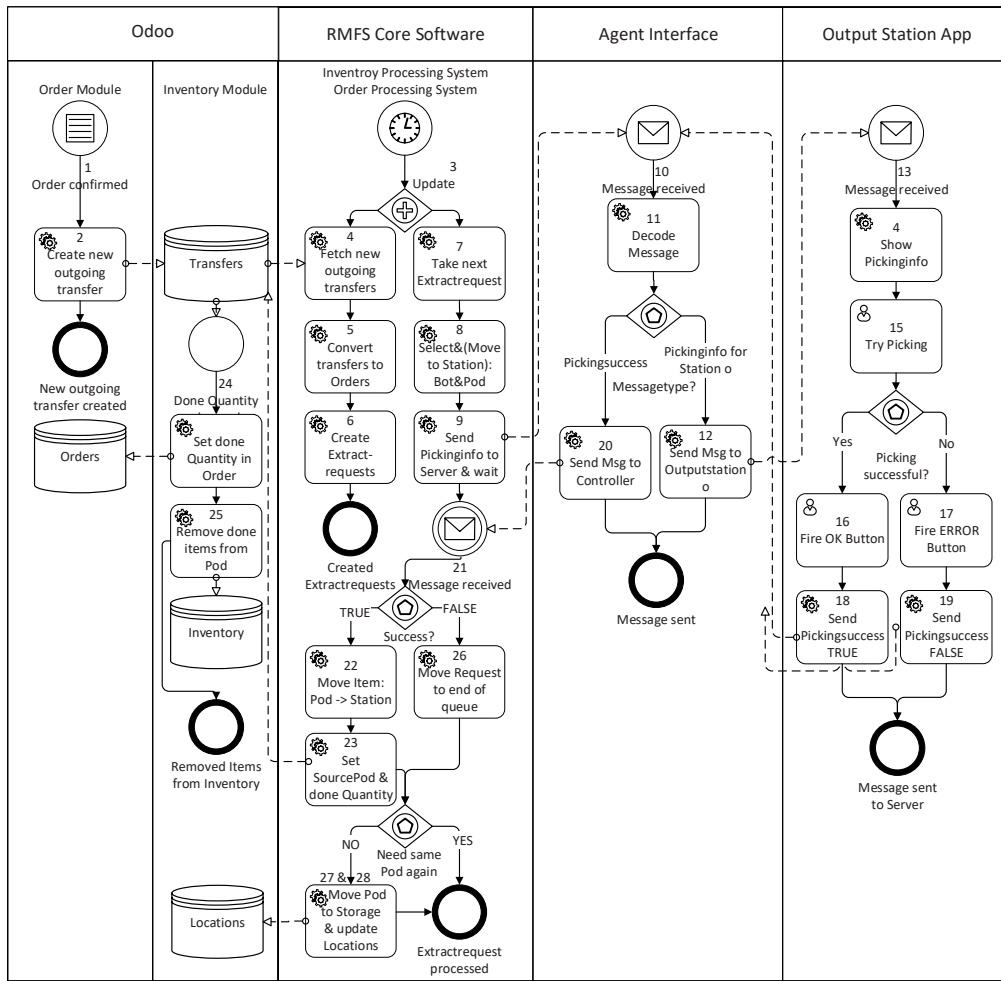*Fig. 8: An example from generating a sale order in Odoo to picking.*

Fig. 9: The process diagramm of an order passing through the system.

5.2.2. Purchase Order to Replenishment

The replenishment process's implementation is rather similar to the picking process described above, therefore only the main differences will be described. The event triggering the start of the replenishment process in the RMFS core software is the receipt of the replenishment items. After that, an internal transfer is automatically created to transfer the received items from the receiving area into pods. When this transfer is detected by the RMFS core software, this transfer is split into its moves and they are converted to item bundles (amount x of sku s). So an insert request for each item bundle is generated (as opposed to an extract request for each order line), which is then handled by the RMFS core software.

6. STEP 3: INTEGRATING THE RMFS CORE SOFTWARE WITH MINI-ROBOTS IN XOR-BENCH

Teleworkbench [19] and [20] is a platform or an infrastructure for conducting, analyzing and evaluating experiments using a number of mini-robots. It offers a controlled environment in which users can execute and test the robot programs using real robots [20]. Similar to Teleworkbench, we designed a teleoperated platform (call XOR-bench), but its features are specially designed for an RMFS. The XOR-bench includes the following functionalities:

– Live video of experiment
– Program-download to robot
– Computer vision-based robot positioning system
– Events and messages logger
– Wireless communication system
– Internet connectivity

The real-time experiment video will be streamed through an IP-based camera. The user can download the developed robot program to the robots through XOR-bench. The robots' positions will be obtained by using computer vision technology from the captured live video. All events and messages from the RMFS software system (including the RMFS core software, ERP and station apps) are recorded and accessible.
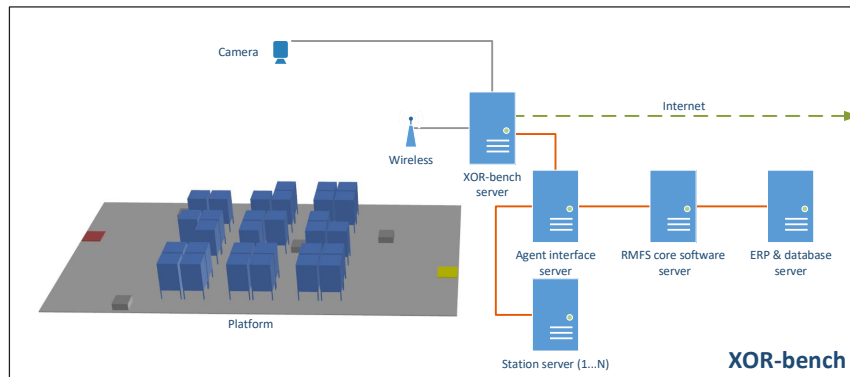
Fig. 10: The general system architecture of the XOR-bench system.

With the logger data the researchers and developers can analyze the results of the experiments. The XOR-bench provides wireless communication between the robots and the RMFS core software. The XOR-bench is connected to the Internet to enable easy access for researchers and developers.

Figure 10 illustrates the general system architecture of the XOR-bench system. The XOR-bench server provides a TCP/IP service, which can be used to connect the robots (the gray rectangles in the platform) with the agent interface sever. The agent interface server is connected with the RMFS core software server and the station servers. The station apps and ERP (including the database) are located in the station servers and ERP & database server respectively. Each station app is located in a station server.

In this step, we use mini-robots, because they are affordable and support high programmability and expandability in the research and teaching. Unlike using mobile industrial robots, using mini-robots for the RMFS system integration experiment, there are some significant advantages, for instance, the experiment is easy to set up, execute, monitor and analyze in a short time. Several RMFS system integration experiments can be done rapidly.

Two different mini robotic platforms are currently used on the XOR-bench. In our previous research work [16], we used the iRobot Create 2, a mobile robot platform based on the Roomba vacuum cleaning robot. The robots are equipped with ASUS Eee PCs through serial-to-USB cables for processing capabilities, and web-cams for line-following. The RFID tag reader is mounted inside the former vacuum cleaning compartment for waypoint recognition. The robot program was written in C#. The main drawback of using iRobot Create 2 for the RMFS experiment is that the robot does not have the lifting mechanism to elevate the pods off the floor when transporting them.

The new mini-robot (see Figure 11a) we use on the XOR-bench is built by LEGO Mindstorms EV3 (third-generation robotics kit in LEGO's Mindstorms line). The robotics kit makes it easy to build the robot, with ease of programming, low complexity and low costs both for the research and teaching. The robot has three motors, with one middle motor used for the lifting mechanical structure (the robot can pick up or set down the pod: see Figure 11b), while two large motors are differential motors for the movement. We also program the robot similar to typical RMFS robots, such as line-following, turning right and turning left. A color sensor is responsible for the function of line-following; moreover, an ultrasonic sensor is used for avoiding obstacle and distance detection. The program was written in the EV3 Python programming language to
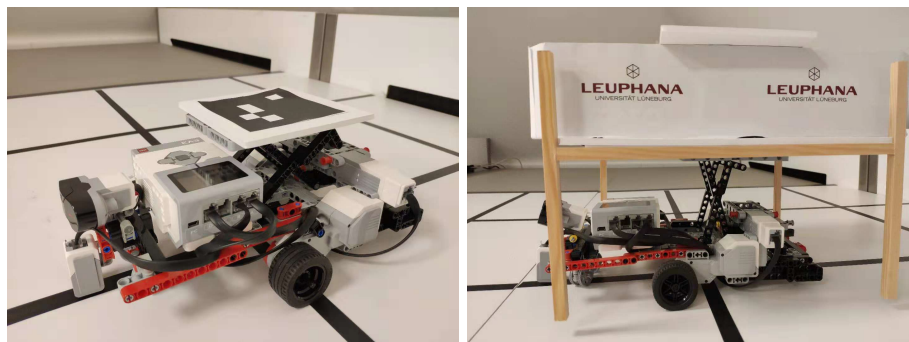
(a) Close-up of the robot   (b) Robot carries the pod

Fig. 11: LEGO-RMFS robot.

(a) Robots in the experiment
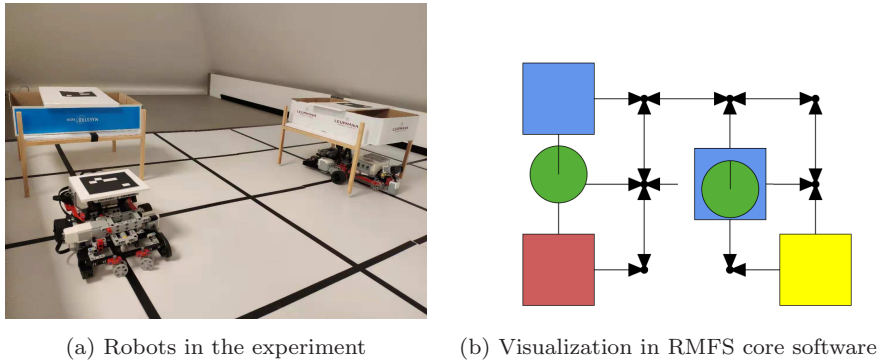
(b) Visualization in RMFS core software

Fig. 12: Experiment with LEGO-RMFS robots.

control the robot, which runs on the EV3dev operating system (a modified version of the Linux Debian Jessie operating system).

We set the maximum velocity limit of each robot to 0.05 m/s, while the time it takes for each robot to do a complete turn is set to 3 s. The time for the robot to set down and pick up the pod is about 3 s. The commands are sent to the robot via WiFi.

In the experiment as shown in Figure 12, we use a 3 × 4 grid layout. One replenishment station is set in the bottom-left corner and one pick station is set on the bottom-right corner. Two robots are used; the pods on the field are made with straight sticks. Figure 12b shows the visualization of the experiment in our RMFS core software. The robot is in the green circle, which carries the pod in the blue rectangle, while the red rectangle is the output station and the yellow one is the input station.

As shown in Figure 13a, the XOR-bench in the lab has a computer visionbased multi-robot positioning system similar to that in [18], so the robot position information as well as identification of different robots can be captured. The software components of the robot positioning are written in Python with OpenCV (Open Source Computer Vision Library). On each pod and robot, an ArUco marker in Figure 13b is used for localization. We use an open-source algorithm for detecting the ArUco marker as in [6]. The robot positioning software runs on XOR-bench server and sends the robot position information to the RMFS core software.

7. STEP 4: TEST AND VALIDATE ALGORITHMS AND PROGRAMS ON A MOBILE INDUSTRIAL ROBOT IN XOR-BENCH

In the last step of the process from RMFS simulation to the real world, we perform the experiment in the XOR-bench for testing and validating the on-board algorithms and programs of a mobile industrial robot. The mobile industrial robots are expensive; moreover,

they need a large place to execute the experiments. Therefore, the test and validation of algorithms and programs on them is highly complex. Ideally using the Internet connectivity feature of the XOR-bench, we can remotely perform experiments for industrial robots by using real-time information streamed over the Internet, such as video, and exchanged messages between the robots and the RMFS core software. We can do the robot tele-programming (in other words, we can develop a robot program locally and it is possible to download it to remote robots).
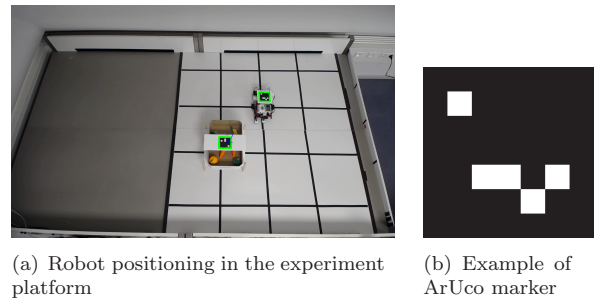

(a) Robot positioning in the experiment platform

(b) Example of ArUco marker

Fig. 13: LEGO-RMFS robot positioning by using of computer vision technology.

The mobile industrial robot for the remote experiment is the Xiellog-Z series from Hanning ZN Tech Beijing, China. The payload of the robot is 650 kg; the speed of the robot is 1.2 m/s. The robot measures about 830mm × 650mm × 350mm and about 150 kg. The robot mounts an advanced industrial positioning tracking and control system on the base, which can be used for data matrix-based code positioning tracking and guiding the robot along a colored path. In other words, the system looks down at the ground to recognize matrixbased codes and color lines on the floor. Figure 14a shows a data matrix-based position tracking and control system called PGV from the German sensor manufacturing company Pepperl+Fuchs. Using an industrial 2-D camera, the PGV can also guides a robot along its colored path. Figure 14b shows a data matrix tag containing position information in addition

to a specific number (https://www.pepperl-fuchs.com/global/en/classid_3334.htm). Because of the robot's on-board positioning system, for this remote experiment, we do not need the video-based positioning system in the XOR-bench. Moreover, a multiray LED scanner (2-D LiDAR Sensor) from Pepperl+Fuchs is installed in each robot to detect potential obstacles and to avoid the collisions between robots.

The robot hasa scissor-lift mechanism to lift the pods. Moreover, they have an industrial WiFi module, which can be connected with the application server through the TCP/IP protocol. The controller on the robot is a Lenze C300 PLC (Programmable Logic Controller), which requires high reliability control. The program on the robot was written in high-level-language-structured text under the IEC 61131-3 standard.

The infrastructure we designed allows people around the world to remotely test and benchmark the robot programs, without spending a lot of effort on handling communication and experiment settings. Figure 15 shows the system architecture of the XOR-bench system in the remote test situation. The XOR-bench is split into two parts: one part located in the robotics lab in Beijing, China, which includes the XOR-bench server, four IP-based cameras and mobile industrial robots;

another part located in our lab in Germany, which has the RMFS core software server agent interface server and the station server. Note that the station server is in the real world located with the station; however, it is in our case located in the lab in Germany for the experiment. The Odoo ERP system and databases are hosted on the Tencent cloud.

As with the experiment layout in the previous section, a 3×4 grid layout is used for performing the experiment with a mobile industrial robot. Figures 16a 16b show the mobile industrial robot. Figure 16c gives a view of the IP-based camera in the lab during the experiments, while Figure 16d shows the visualization of the information in the RMFS core software. Recall that the robot is in the green circle, which carries the pod in the blue rectangle, while the red rectangle is the output station and the yellow one is the input station.
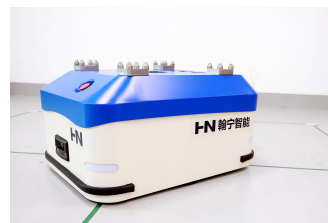
(a) Close-up of the robot
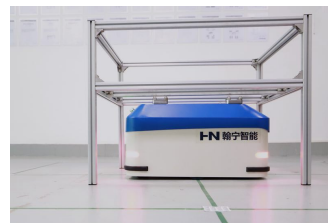
(b) Robot carries the pod

(c) Robot in the experiment
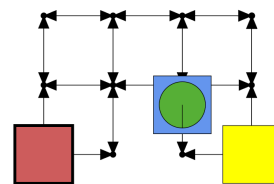
(d) Visualization in RMFS core software

Fig. 16: Remote experiment with mobile industrial robot.

(a) Pepperl+Fuchs PGV position tracking and control system

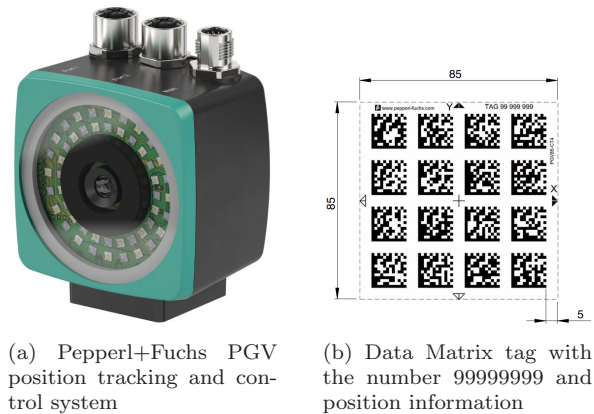(b) Data Matrix tag with the number 99999999 and position information

Fig. 14: Real robot positioning by using of position tracking and control system.
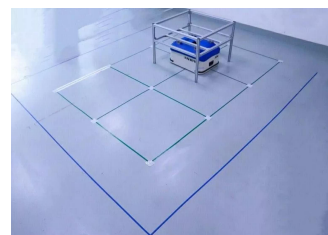

Fig. 15: The system architecture of the XOR-bench system in remote testing situation.

8. AN EXAMPLE – PATH PLANNING OPTIMIZER

In order to answer the question at the beginning of this paper: "how can we make sure that the algorithms we implement and test in the simulator are still applicable for the real-world scenarios?", we give you in this section an example of an optimizer we developed in the simulation, the path planning optimizer. And we will explain shortly how we make this optimizer applicable for the real-world scenarios.

The path planning algorithm is called Multi-Agent Pathfinding (MAPF) in the literature. This is a challenging problem with many applications such as robotics, disaster rescue and video games (see [21]). The goal in MAPF is to compute collision-free paths along a grid-based graph for all agents from their start to their unique destination nodes. There are many MAPF solvers from artificial intelligence which typically work for agents, but they do not consider kinematic constraints, such as maximum velocity limits, maximum acceleration and deceleration, and turning times. Moreover, they consider the movement of robots only in discretized environments. It is a similar case with the existing path planning algorithms for an RMFS (see [4] and [5]). However, we can not use those MAPF solvers directly if we want to test the path planning algorithms with real robots. Moreover, there are MAPF solvers from robotics, which consider the kinematic constraints, but they work only for a small number of robots; therefore, they are too slow for a massive search like in an RMFS. In order to make path planning algorithms work for real robots, we have implemented new MAPF solvers in our path planning optimizer that consider kinematic constraints in a continuous environment for the RMFS. More details can be found in [15]. It is worth to mention that if there are delays of robots during the execution of the planned routes, the collisions of robots might occur. In such cases, the sensor installed in the robot will detect that, and one robot will stop and wait until another robot go through and continue its path.

9. SUMMARY AND OUTLOOK

In order to make it easy to test the developed algorithms, such as the ones in the previous section, in the simulator with real-world data and robots, we have presented in this work a novel development process flow by using our XOR-bench. This process flow includes changing the RMFS simulator into core software and integrating the RMFS core software with Odoo ERP and input/output station app.

Moreover, we have shown that geographically distributed users can use our designed XOR-bench to test and validate the RMFS system and algorithms on real robots, either educational robots (such as iRobots or LEGO robots) or industrial robots. This is beneficial for both research and teaching.

Due to the existing interfaces with the ERP system and the robots, we can use the XOR-bench in the future to test other types of automated warehousing systems, such as an automated system [2], where humans work in zones, and robots are sent to humans to load items from zones to input/output stations; in this case, robots work alongside humans (traveling in the same space with humans).

10. ACKNOWLEDGEMENTS

We would like to thank Beijing Hanning ZN Tech for providing the robots and robotics research lab and performing the RMFS system experiment with us.

REFERENCES

1. SAP EWM: Integrated Warehouse Management Solution by Swisslog. URL https://www.swisslog.com/en-us/warehouse-logistics-distribution-center-automation/software-inventory-management/sap-ewm-extended-warehouse-management
2. Ackerman, E.: How locus robotics plans to build a successor to Amazon's Kiva robots (2016). URL https://spectrum.ieee.org/automaton/robotics/industrial-robots/locus-robotics-warehouse-automation-robots
3. Boysen, N., Briskorn, D., Emde, S.: Parts-to-picker based order processing in a rackmoving mobile robots environment. European Journal of Operational Research 262(2), 550–562 (2017). DOI 10.1016/j.ejor.2017.03.053
4. Cohen, L., Uras, T., Koenig, S.: Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In: Eighth Annual Symposium on Combinatorial Search (2015)
5. Cohen, L., Wagner, G., Satish Kumar, T.K., Choset, H., Koenig, S.: Rapid randomized restarts for multi-agent path finding solvers. ArXiv e-prints (2017)
6. Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F.J., Marín-Jiménez, M.J.: Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition 47(6), 2280–2292 (2014)
7. Hazard, C.J., Wurman, P.R., D'Andrea, R.: Alphabet Soup: A testbed for studying resource allocation in multi-vehicle systems. In: Proceedings of AAAI Workshop on Auction Mechanisms for Robot Coordination, pp. 23-30. Citeseer (2006)
8. Hoffman, A.E., Mountz, M.C., Barbehenn, M.T., Allard, J.R., Kimmel, M.E., Santini, F., Decker, M.H., D'Andrea, R., Wurman, P.R.: System and

method for inventorymanagement using mobile drive units (2013). URL https://www.google.com/patents/US20130103552

9. de Koster, R., Le-Duc, T., Roodbergen, K.J.: Design and control of warehouse order picking: A literature review. European Journal of Operational Research 182(2),481–501 (2007). DOI https://doi.org/10.1016/j.ejor.2006.07.009. URL http://www.sciencedirect.com/science/article/pii/S0377221706006473

10. Krenzler, R., Xie, L., Li, H.: Deterministic pod repositioning problem in robotic mobile fulfillment systems (2018)

11. Lamballais, T., Roy, D., De Koster, M.: Estimating performance in a robotic mobile fulfillment system. European Journal of Operational Research 256(3), 976–990 (2017)

12. Lamballais, T., Roy, D., De Koster, M.: Inventory allocation in robotic mobile fulfillment systems (2017)

13. Laurent, S.S., Johnston, J., Wilder-James, E., Winer, D.: Programming Web Services with XML-RPC: Creating Web Application Gateways. O'Reilly Media, Inc. (2001)

14. Merschformann, M.: Active repositioning of storage units in robotic mobile fulfillment systems. In: Operations Research Proceedings 2017, pp. 379–385. Springer (2018)

15. Merschformann, M., Xie, L., Erdmann, D.: Path planning for robotic mobile fulfillment systems. arXiv preprint arXiv:1706.09347

16. Merschformann, M., Xie, L., Li, H.: RAWSim-O: A simulation framework for robotic mobile fulfillment systems. Logistics Research 11(1) (2018). DOI doi:10.23773/2018 8

17. Nigam, S., Roy, D., de Koster, R., Adan, I.: Analysis of class-based storage strategies for the mobile shelf-based order pick system (2014)

18. Tanoto, A., Li, H., Rückert, U., Sitte, J.: Scalable and flexible vision-based multi-robot tracking system. In: 2012 IEEE International Symposium on Intelligent Control (ISIC), pp. 19–24. IEEE (2012)

19. Tanoto, A., Rückert, U., Witkowski, U.: Teleworkbench: A teleoperated platform for experiments in multi-robotics. In: Web-Based Control and Robotics Education, pp.267–296. Springer (2009)

20. Tanoto, A., Werner, F., Rückert, U., Li, H.: Teleworkbench: validating robot programs from simulation to prototyping with minirobots. In: The 10th International Conference on Autonomous Agents and Multiagent Systems, vol. 3, pp. 1303–1304. International Foundation for Autonomous Agents and Multiagent Systems (2011)

21. Wang, K.H.C., Botea, A.: MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. Journal of Artificial Intelligence Research 42, 55–90 (2011)

22. Yuan, Z., Gong, Y.Y.: Bot-in-time delivery for robotic mobile fulfillment systems. IEEE Transactions on Engineering Management 64(1), 83–93 (2017)

23. Zou, B., Gong, Y., Xu, X., Yuan, Z.: Assignment rules in robotic mobile fulfilment systems for online retailers. International Journal of Production Research 55(20), 6175–6192 (2017)