BVL *International*
The Global Supply Chain Network

RAWSim-O: A Simulation Framework for Robotic Mobile Fulfillment Systems

M. Merschformann[1] · L. Xie[2] · H. Li[3]

ABSTRACT

This paper deals with a new type of warehousing system, Robotic Mobile Fulfillment Systems (RMFS). In such systems, robots are sent to carry storage units, so-called "pods," from the inventory and bring them to human operators working at stations. At the stations, the items are picked according to customers' orders. There exist new decision problems in such systems, for example, the reallocation of pods after their visits at work stations or the selection of pods to fulfill orders. In order to analyze decision strategies for these decision problems and relations between them, we develop a simulation framework called "RAWSim-O" in this paper. Moreover, we show a real-world application of our simulation framework by integrating simple robot prototypes based on vacuum cleaning robots.

KEYWORDS: Robotic Mobile Fulfillment Systems · Simulation · Warehousing Systems · Parts-to-Picker Systems · Demonstration

✉ M. Sc. Marius Merschformann[1]
    merschformann@dsor.de

    Prof. Dr. Lin Xie[2] (Corresponding Author)
    xie@leuphana.de

    Dipl. Ing. Hanyi Li[3]

[1] University of Paderborn
    Warburger Str. 100, 33098 Paderborn, Germany
    http://www.dsor.de

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

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

1   INTRODUCTION

Due to the rise of e-commerce, the traditional manual picker-to-parts warehousing systems no longer work efficiently, and new types of warehousing systems are required, such as automated parts-to-picker systems. For details about the classification of different warehousing systems we refer to [8]. This paper studies one of the parts-to-picker systems, a so-called Robotic Mobile Fulfillment Systems (RMFS), such as the Kiva System ([4], nowadays Amazon Robotics), the GreyOrange Butler or the Swisslog CarryPick. The approach of an RMFS completely eliminates the need for travel within the warehouse, which accounts for approximately 50% of a picker's time in manual warehouse operations according to [14]. [15] indicates that the Kiva System increases the productivity two to three times, in contrast to a traditional manual picker-to-parts system. Compared to other kinds of warehousing systems, the biggest advantages of an RMFS are its flexibility as a result of having virtually no fixed installations, the scalability due to accessing the inventory in a parallel manner, and the reliability due to the use of only homogeneous components, i.e., redundant components may compensate for faulty ones (see [5] and [16]).

The first framework, "Alphabet Soup", was published by [5], and is a first simulation of the RMFS concept. In this work, we extend the work of [5] to develop our simulation framework, called "RAWSim-O" (Robotic Automatic Warehouse Simulation (for) Optimization). Similarly to "Alphabet Soup", we use an agent-based and event-driven simulation focusing on a detailed view of the system, but extend our simulation framework with the cases of multiple floors, which are connected by elevators. The reason for this extension is to avoid the lack of utilization of vertical space in an RMFS compared to other parts-to-picker systems, such as shuttle-based solutions (see [13]). Moreover, we integrate a more realistic robot movement emulation by considering the robot's turning time and adjusted acceleration or deceleration formulas. This enables a real world application of our simulation framework by using simple robot prototypes based on vacuum cleaning robots (iRobot Create 2). The implementation of the framework was done in C# and is compatible

with the Mono project to allow execution on high throughput clusters. All source codes are available at www.rawsim-o.de.

There exist many decision problems in an RMFS, and they influence each other. With the publication of RAWSim-O we provide a tool for analyzing effects of decision mechanisms for these problems and synergies of decision strategies. The framework enables a holistic look at RMFS which helps uncovering side-effects of strategies, e.g. deciding tasks for robot that cause congestion issues for path planning methods. Hence, RAWSim-O enables a more reliable assessment of the system's overall efficiency, e.g., in terms of customer order throughput. Thus, we hope to further support and push the research on RMFS with our work. We describe the RMFS in more detail and point out the decision problems in focus in Section 2. Next, our simulation framework is described in Section 3. After that, we show a demonstration application of our simulation in Section 4. Finnaly, we conclude our work in Section 5.

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 inventory is stored
– *storage locations* denoting the inventory area where the pods are stored
– workstations, where the pick order items are picked from pods (*pick stations*) or replenishment orders are stored to pods (*replenishment stations*)
– mobile *robots*, which can move underneath pods and carry them to workstations.

The pods are transported by robots between the inventory area and workstations. Figure 1 shows the storage and retrieval process: after the arrival of a replenishment order (consisting of a number of physical units of one stock keeping unit (SKU)), robots carry selected pods to a replenishment station to store units in pods. Similarly, after receiving a pick order (including a set of order lines, each for one SKU, with corresponding units necessary to fulfill the line), robots carry selected pods to a pick station, where the units for the order lines are picked. Note that, in order to fulfill pick orders, several pods may be needed, since orders may have multiple lines. Although pods typically contain multiple SKUs with many SKUs in stored in the system it is very unlikely that a pick order can be completed with only one pod. Then, after a pod has been processed at one or more stations, it is brought back to a storage location in the inventory area.

2.1 Decision Problems

In an RMFS environment, various optimization and allocation problems have to be solved in real time. The system aims at keeping human workers at the stations busy while minimizing the resources (e.g. robots) to fulfill the incoming pick orders. These problems were first described by [16]:
– For the bots, the planning of their tasks, paths, and motions
– For the stations, the management of the workflow for each station
– For the resources of the system, their planning, provisioning, and allocation to other components of the system, usually called the Resource Allocation Problem

As [16] note, the Resource Allocation Problem cannot be treated as a global optimization problem, but rather as a set of subproblems that should be solved using specialized methods to enable the decisions
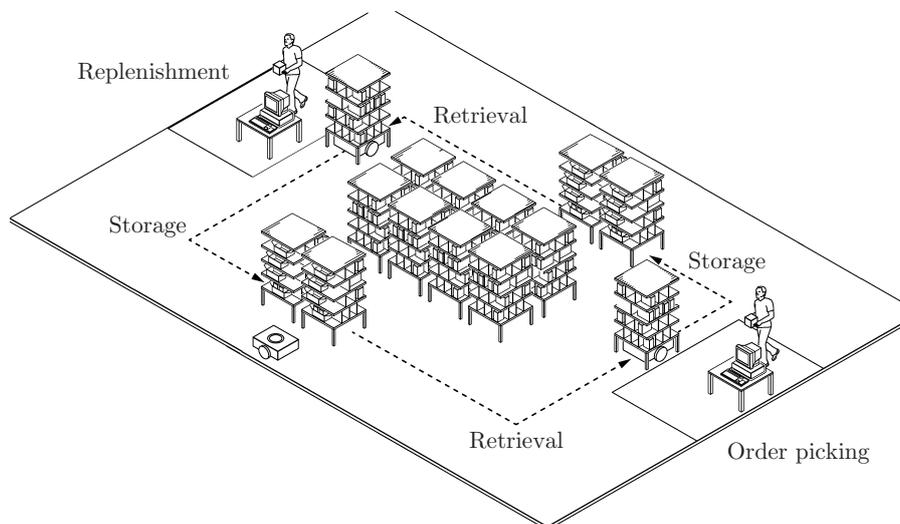

Figure 1: The central process of an RMFS (see [6])

in an online e-commerce environment. We reiterate these decision problems briefly in the context of the simulation framework developed in this work:

- *Order Assignment*
  - **Replenishment Order Assignment (ROA)**: assignment of replenishment orders to replenishment stations
  - **Pick Order Assignment (POA)**: assignment of pick orders to pick stations
- *Task Creation*
  - *Pod Selection*
    - **Replenishment Pod Selection (RPS)**: selection of the pod to store one replenishment order in
    - **Pick Pod Selection (PPS)**: selection of the pods to use for picking the pick orders assigned at a pick station
  - **Pod Storage Assignment (PSA)**: assignment of an available storage location to a pod that needs to be brought back to the inventory area
- **Task Allocation (TA)**: assignment of tasks from *Task Creation* and additional support tasks like idling to robots
- **Path Planning (PP)**: planning of the paths for the robots to execute
- **Station Activation (SA)**: controlling active times of the stations (e.g. switching some off based on work shifts or for emulation of "jumper" pickers that can be assigned to allow for temporarily increased throughput)
- **Method Management (MM)**: exchange of controlling mechanisms in a running system (e.g. replacing the PSA controller with a different one during runtime in order to adapt to changing conditions)

The decisions for the aforementioned operational problems influence each other. Here we sketch two relationships between decision problems that may exploit synergy effects or sabotage each others success:

- POA and PPS: one objective is to maximize the average number of picks per handled pod (called pile-on); in other words, the higher the pile-on is, the fewer pods are needed at pick stations. Therefore, the selection of a pod for an order in PPS is dependent on the selection of the pick station of other orders. If similar orders are assigned to the same pick station, a pod can be selected to maximize the number of picks.
- PSA, TA and PP: One objective is to minimize the travel times of robots to complete all orders, thus, reducing the number of robots needed to achieve a certain throughput. For this, the selected storage locations for pods impact the performance of the assignment of tasks and the pathfinding, because the trip destinations change.

Some of the operational decision problems described above have been addressed in the context of RMFS in previous publications. First, the sequencing of pick orders and pick pods for an integration of PPS and POA is studied in [1]. The authors propose methods for obtaining sequences minimizing the number of pod visits at one station. Second, the planning of paths in an RMFS is subject to the studies in [2], [3], and [10]. The state-of-the-art multi-agent path planning algorithms were implemented in [10] to suit PP in an RMFS. Besides the operational problems, we would like to note that more literature exists on more tactical to strategic decision problems, like layout planning, for example, in [9].

Furthermore, many of the problems discussed above share similarities with decision problems well-studied in the context of other warehouse systems or in theory itself. However, the successful application of policies or algorithms of other applications to an RMFS often is unclear. For an overview of literature about decision problems occurring when controlling other Automated Storage & Retrieval Systems' processes, we refer to [12]. The authors provide an overview about literature on decision problems, such as storage assignment (sharing similarities with RPS and PSA) and order batching (sharing similarities with POA). Another decision problem related to one occurring in RMFS is multi robot task allocation (sharing similarities with TA). Similar to TA in RMFS, a set of homogeneous robots needs to be matched with a set of tasks to maximize the overall system performance. For an overview of literature on the subject we refer to [7]. Furthermore, the problem of multi agent path planning (sometimes referred to as route planning or pathfinding) is a subject of research in multiple applications and theories. The studied formulations often differ in the way whether kinematic constraints and further additional constraints from the real-world applications are modeled. We refer to [11] for a detailed explanation of the problem and an overview of existing literature.

3   RAWSIM-O

RAWSim-O is an agent-based discrete-event simulation framework. It is designed to study the context of an RMFS while considering kinematic behavior of the mobile robots and evaluating multiple decision problems jointly. Hence, the main focus of the framework is the assessment of new control strategies for RMFS and their mutual effects. In the following, we describe the RAWSim-O simulation framework in more detail. At first, we describe the general structure of the framework, and then we give detailed information about how the robotic movement is emulated.
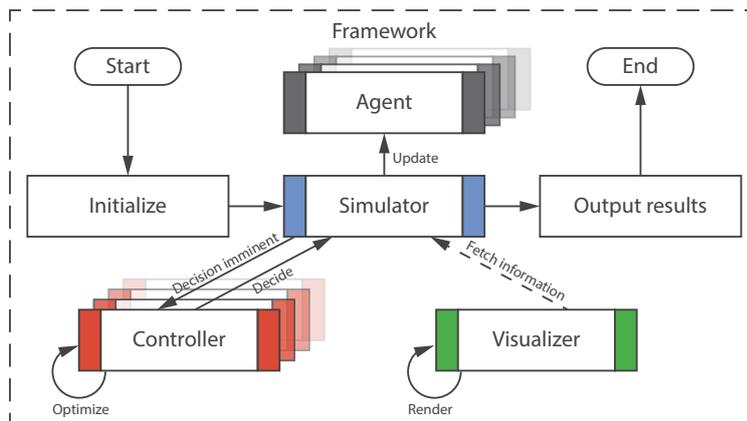
3.1 Simulation Process

Figure 2a shows an overview of our simulation process, which is managed by the core *simulator* instance. The tasks of the simulator contain:

- 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 *controllers*, 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 2b illustrates a screenshot of our simulation in 3D.

As mentioned before, RMFS usually lack utilization of vertical space when compared to other parts-to-picker systems, for example shuttle-based solutions (see [13]). Although the flexibility of the system due to the mobile components is an advantage over the many fixed components of a shuttle-based solution, in areas where land costs are high the lack of vertical space utilization can be a deal-breaker for the RMFS concept. To help mitigate this disadvantage RAWSim-O allows the study of multi-floor systems for applications where the integration of mezzanine floors is possible (see Fig. 2b for example). The floors are connected by elevators that can be used by one robot at a time. One elevator connects at least two waypoints of the underlying waypoint-system, which is used for guiding the robots. For each pair of connected waypoints a constant time is specified to capture the travel time of the lift transporting a robot. Conceptually, the waypoint system itself is a directed graph which robots travel along while adhering to their kinematic behavior.

I.e., turning is only allowed on-the-spot at waypoints (essentially nodes of the graph) and is executed by a specified angular speed. Furthermore, straight travel along the edges of the graph involves acceleration and deceleration, which is discussed in more detail in Sec. 3.2. To avoid blocked destination waypoints for path planning (many robots share the same destination when approaching stations or elevators) waypoints may form a queuing zone. After a robot reaches such a zone guided by a path planning engine a queue manager takes over handling of the robots paths within the zone. For this, the manager may make use of shortcuts definable within the queue, but mainly lets the robots move up until they reach the end waypoint (e.g., a pick station). On leaving the queuing zone again the defined path planning engine takes over again.

The framework allows easy exchange of controllers by implementing a base structure for all mentioned decision problems. For this, all controllers are also agents of the simulation that may expose a next event to jump to, and also get updated at each event. This enables the mechanisms to react to every change occurring in the simulated world as well as to steer it. As a shortcut, controllers may also subscribe to events that fire, if certain situations occur. Additionally, the controllers are called whenever a new decision needs to be made, e.g. the POA controller is called when a new pick order is submitted to the system, and the TA controller is called when a robot finished its task. In Tab. 1 we include a short outline of the calls to the event-driven decision controllers. The SA and MM controllers are not event-driven, because they are considered like management mechanisms. However, all controllers may subscribe to simulation-wide events.

(a) Overview of the simulation process.
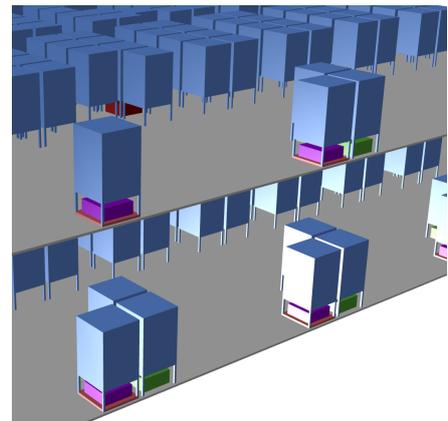
(b) Visualization screenshot

Figure 2: RAWSim-O simulation framework

*Table 1: Overview of the base events causing calls to the controller per decision problem
and the corresponding assignment responsibility*

Problem	Decision	Main triggers
ROA	repl. order → repl. station	New repl. order, repl. order stored in pod
POA	pick order → pick station	New pick order, pick order completed, repl. order stored in pod
RPS	repl. order → pod	New repl. order, SKU unit picked
PPS	pod → pick order(s)	New task is assigned to robot
PSA	pod → storage location	Pod needs to be brought back to inventory
TA	task → robot	Robot needs a new task
PP	robot → robot	Robot has a new destination

In addition to acting ad-hoc according to the triggers mentioned in Tab. 1 strategies may plan ahead, e.g., by sequencing tasks for robots instead of greedily selecting them as soon as a robot is done with its previous one. For this example it can be done by preparing tasks ahead and assigning them to the chosen robot as soon as it becomes ready. Furthermore, the framework also allows some controllers to run optimization algorithms in parallel. This means that the controllers are allowed to buffer certain decisions (like the assignment of pick orders), run an optimization procedure in parallel, and submit the decision later on. In this case the simulation is paced until the optimization algorithm returns a solution in order to synchronize the wall time of the algorithm with the simulation time. I.e., the simulation will only continue for the wall time that already passed for the optimization algorithm converted to simulation time (while it is still running), but not beyond it.

3.1.1 Core Decision Hierarchy

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 3).

For this, the SA and MM decision problems are neglected, since they have a more supportive role and can even be replaced with default mechanisms that keep the status quo. If a new replenishment order is received, first the controllers 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 controller. Similarly, after the POA controller 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 controllers 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 controller 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
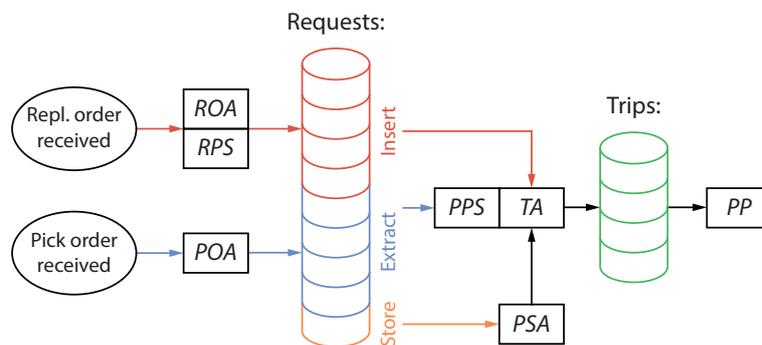

Fig. 3: Order of decisions to be done triggered by receiving a pick or replenishment order

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.

3.1.2   Input Information

The simulation framework conceptually consists of three different inputs. First, a layout configuration specifies the characteristics and dimensions of the system layout itself. Second, a scenario configuration describes how orders are generated and further settings of the system's surroundings. Finally, a controller configuration is given to specify the decision mechanisms for all previously described decision problems. We distinguish them as three different input files to enable easier assessment of control methods for different systems under diverse scenarios.

Layout Specification The layout specification can be either an explicit file specifying the exact positions and individual characteristics for all stations, the waypoint system, and the robots; or a file providing specifications leading to a default layout based on the concepts of [9]. This default layout can be seen in Fig. 4, with pick stations as red circles, replenishment stations as yellow circles, robots in green, and the pod storage locations in the middle as blue squares. It is based on the idea of circular flows around storage location blocks that also align with the entrances and exits of the stations and their queuing areas (dotted lines). Between the queuing area and the storage locations, a hallway area (dashed lines) allows the robots to cross between the stations. In order to generate such a layout, the numbers of pick and replenishment stations on the north, south, east, and west sides need to be set. Furthermore, the numbers of vertical and horizontal aisles and the block size need to be specified.

Scenario Specification The next input file specifies information about the scenario to simulate. This includes the duration of the simulation and settings about the inventory and order emulation. In contrast with the concept of colored word order emulation of "Alphabet Soup" in [5], we use a concept for SKU and pick order generation based on typical random distributions. For the generation of SKU popularity information, we implemented constant, uniform, normal, and gamma distributions to emulate simple scenarios up to more ABC-like popularity curves. The SKU for each pick order line is then selected using the chosen distribution, but limiting the choice to only in-stock products. This is done to avoid stock-out situations, which are not useful when simulating the control of an RMFS for such situation, i.e. we assume that no unfulfillable pick orders are submitted to the system. The number of order lines and the number of units per line are also chosen from a distribution previously specified by the user. For choosing the SKU for each replenishment order, we use the same popularity distribution connected to information about the order size in which a certain product is replenished. For this, we also allow setting a parameterized amount of replenishment orders to be return orders, i.e. single line and single unit replenishment orders. The space consumption of one unit of a SKU on a pod is a one-dimensional factor which is also set according to a user-specified distribution. This is necessary to enable the simulation of both pick and replenishment operations at the same time, while emulating the inventory situation in our system.

Moreover, we implement three procedures for generating new replenishment and pick orders. First, a constant order backlog scenario can be selected. This means that a completed order is immediately replaced by a new one. This is done to keep the system under constant pressure. To avoid completely overfilling or draining the inventory over time, we allow the specification of replenishment and pick order generation pauses according to given inventory level thresholds. E.g., if the inventory reaches a 95% filling, replenishment order generation is paused
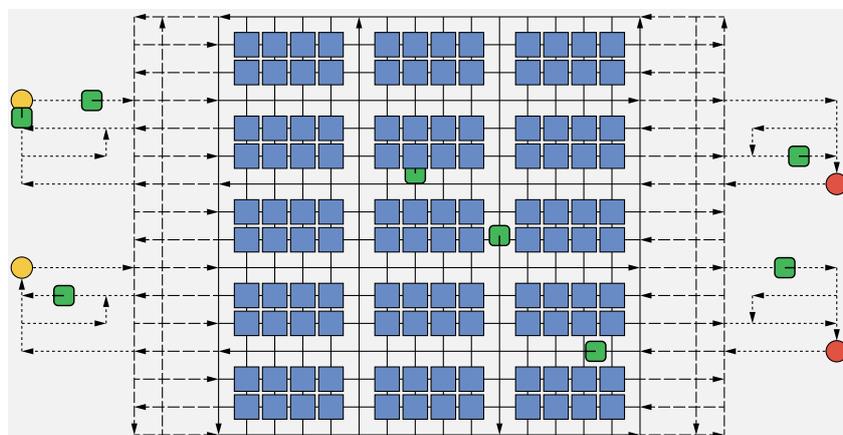

Fig. 4: The default layout, with two replenishment and two pick stations

until it drops below 85% again. Second, we generate orders according to arrivals of a configurable Poisson process. This Poisson process can be inhomogeneous to capture order peak situations during a day or match certain patterns observed at distribution centers. Finally, we allow the input of files to specify which orders are generated during the simulation horizon. Additionally, we allow a combined setting of one of the order generation scenarios above with the submission of order batches at given periodic time points. This is done to allow emulation of batch operations or hybrid scenarios. With all of these options, we aim to resemble most artificial and realistic scenarios.

Controller Configuration The last input file specifies the controller to use for each decision problem as well as the parameters. This enables a flexible configuration due to the modular controller concept. Controllers integrating multiple decision problems can be configured using dummy controllers for the other components. With the help of reflection most parameter structures only need to be defined once. Without additional implementation effort new parameters are serializable and configurable in the graphical user interface.

3.1.3 Output

In order to investigate the system's behavior in more detail, different output measures are tracked and logged over time. For this, we distinguish three main measure types. First, a footprint is written for each simulation execution to allow for an easy comparison of multiple executions. This footprint contains most basic performance measures for the simulation run like the order throughput rate or the distance traveled by the robots. Second, time-based information is logged to generate plots after execution of the simulation.

These are useful for post-experiment analysis of the simulation processes, especially if no visualization was attached. As an example, Figure 5a shows the pick order throughput and overall distance traveled per hour for a simulation in which the PSA strategy is switched after half of the simulation horizon. For this experiment, a random PSA strategy was replaced with a strategy selecting the nearest available storage location for the pod. This is reflected by the decrease in distance to cover by the robots, which immediately leads to an increased throughput in pick orders, because more pods are available at the pick stations. Finally, location-based information is logged to conduct heatmap analyses. This is useful, for example, to get insights about congestion effects when looking at robot movement behavior over time (see Fig. 5b). In this example, we can see the queuing happening at the pick stations at the top and the replenishment stations at the bottom. Furthermore, we can identify highly frequented areas which are prone to congestion. Note that the logarithm was applied to the heat values in order to increase the contrast in color in the resulting heatmap.
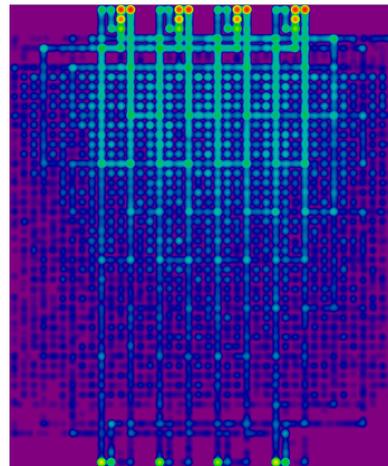
3.2 Robot Movement Emulation

Table 2: Symbol definitions

Symbol	Description
d	Drive distance
v_t	The speed at time t
s_t	The position at time t
$\overrightarrow{a}$	Acceleration in $\frac{m}{s^2}$
$\overleftarrow{a}$	Deceleration in $\frac{m}{s^2}$ (negative)
$\overline{v}$	Top-speed in $\frac{m}{s}$
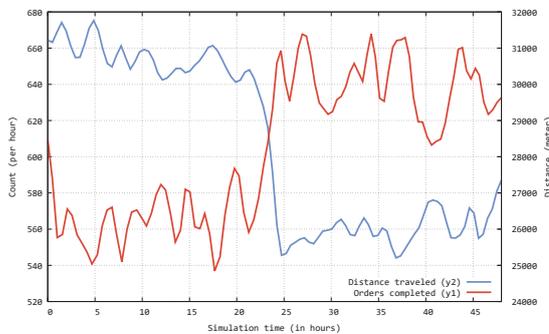
(a) Sample progression plot of picked orders (red) and distance traveled (blue) per hour

(b) Example of a heatmap showing robot movement behavior over time (red ≡ high, purple ≡ low)

Fig. 5: Examples of the system's output

In order to accurately capture the movement behavior of the robots we consider linear acceleration and deceleration for straight robot movement. Furthermore, we allow continuous turning of the robots while adhering to a certain turning speed. Thereby, turning can only be done on the spot while standing still. Hence, we assume that moving along a curve is not possible. Next, we describe the necessary calculations for modeling the described movement.

The computation of traveling times and distances for straight movement is based on uniform acceleration and deceleration. For this, the velocity of a robot has to be considered to determine its arrival time at a destination node. The symbols used in the following description are defined in Tab. 2. The fundamental formulas for all remaining definitions are given by the resulting speed v_t after accelerating by $\overrightarrow{a}$ for time t from initial speed v_0 (see Eq. 1), the new position s_t compared to the initial position s_0 after driving while accelerating for time t (see Eq. 2), and the new position s_t compared to the initial position s_0 after driving at top-speed for time t (see Eq. 3).

$$v_t = \overrightarrow{a} t + v_0 \tag{1}$$

$$s_t = \frac{\overrightarrow{a}}{2} t^2 + v_0 t + s_0 \tag{2}$$

$$s_t = \overline{v} t + s_0 \tag{3}$$

For the implementation of the simulation framework, it is required to not only calculate the time for covering a distance when initially standing still, but also to calculate distances and times based on an initial speed $v_0 > 0$, because simulation events may occur at any time during robot travel. For this, the time ($t_{v_0 \to \overline{v}}$) and distance ($s_{t_{v_0 \to \overline{v}}}$) to reach the top-speed are needed (see Eq. 4 & 5). This can analogously be defined for full deceleration (see Eq. 6 & 7).

$$\overline{v} = \overrightarrow{a} t_{v_0 \to \overline{v}} + v_0 \Leftrightarrow t_{v_0 \to \overline{v}} = \frac{\overline{v} - v_0}{\overrightarrow{a}} \tag{4}$$

$$s_{t_{v_0 \to \overline{v}}} = \frac{\overrightarrow{a}}{2} \left(t_{v_0 \to \overline{v}} \right)^2 + v_0 t_{v_0 \to \overline{v}} \tag{5}$$

$$0 = \overleftarrow{a} t_{v_0 \to 0} + v_0 \Leftrightarrow t_{v_0 \to 0} = \frac{0 - v_0}{\overleftarrow{a}} \tag{6}$$

$$s_{t_{v_0 \to 0}} = \frac{\overleftarrow{a}}{2} \left(t_{v_0 \to 0} \right)^2 + v_0 t_{v_0 \to 0} \tag{7}$$

For the calculation of time for traveling distance d when starting at an initial speed of v_0, four cases need to be considered. In the first case, only deceleration is possible. In the second case, cruising at top-speed and deceleration are possible. In the third case, acceleration up to top-speed, cruising at top-speed and deceleration are possible. In the fourth case, the distance is so short that only acceleration and deceleration phases are possible. The function defined in Alg. 1 calculates the remaining cruise time for all of the cases. Line 7 uses the time at which acceleration switches to deceleration, which is described in more detail below.

Algorithm 1: $CruiseTime(\overrightarrow{a}, \overleftarrow{a}, \overline{v}, v_0, d)$

---

1 **if** $d \leq s_{t_{v_0 \to 0}}$ **then**

2 $\quad$ **return** $t_{v_0 \to 0}$

3 **if** $v_0 = \overline{v}$ **then**

4 $\quad$ **return** $\frac{d - s_{t_{v_0 \to 0}}}{\overline{v}} + t_{v_0 \to 0}$

5 **if** $s_{t_{v_0 \to 0}} + s_{t_{v_0 \to 0}} \leq d$ **then**

6 $\quad$ **return** $t_{v_0 \to \overline{v}} + \frac{d - s_{t_{v_0 \to \overline{v}}} - s_{t_{v_0 \to 0}}}{\overline{v}} + t_{v_0 \to 0}$

7 **return** $\sqrt{\dfrac{d + \frac{\overrightarrow{a}}{2} \left(\frac{v_0}{\overrightarrow{a}} \right)^2}{\frac{\overrightarrow{a}}{2} + \frac{\overrightarrow{a}}{2\overleftarrow{a}}}} + \sqrt{\dfrac{d + \frac{\overrightarrow{a}}{2} \left(\frac{v_0}{\overrightarrow{a}} \right)^2}{\frac{\overleftarrow{a}}{2} + \frac{\overleftarrow{a}}{2\overrightarrow{a}}}} - \dfrac{v_0}{\overrightarrow{a}}$

Let d' be the full distance from the start node to the destination node, thus, starting with zero speed at the beginning of d' and stopping with zero speed at the end of d'. For this, the time of switching from acceleration to deceleration is given by Eq. 8, i.e., the time for driving while accelerating. To calculate this time we make use of the fact that the speed at which we switch from acceleration to deceleration must match ($\overrightarrow{a} t_1 = \overleftarrow{a} t_2$), hence, we can substitute t_2 with $\frac{\overrightarrow{a} t_1}{\overleftarrow{a}}$.

$$d' = \frac{\overrightarrow{a}}{2} t_1^2 + \frac{\overleftarrow{a}}{2} t_2^2$$

$$\Leftrightarrow d' = \frac{\overrightarrow{a}}{2} t_1^2 + \frac{\overleftarrow{a}}{2} \left(\frac{\overrightarrow{a} t_1}{\overleftarrow{a}} \right)^2$$

$$\Leftrightarrow d' = \frac{\overrightarrow{a}}{2} t_1^2 + \frac{\overleftarrow{a}}{2} \frac{(\overrightarrow{a} t_1)^2}{2\overleftarrow{a}}$$

$$\Leftrightarrow t_1^2 = \frac{d'}{\frac{\overrightarrow{a}}{2} + \frac{\overrightarrow{a}^2}{2\overleftarrow{a}}}$$

$$\Leftrightarrow t_1 = \sqrt{\frac{d'}{\frac{\overrightarrow{a}}{2} + \frac{\overrightarrow{a}^2}{2\overleftarrow{a}}}} \tag{8}$$

For the calculation, we assume that speed is currently zero and the movement just starts at the start node. If $v_0 > 0$ and $d < d'$, then d' has to be calculated by using d (see Eq. 9).

$$d' = d + \frac{\overrightarrow{a}}{2} \left(\frac{v_0}{\overrightarrow{a}} \right)^2 \tag{9}$$

Analogously to Eq. 8, it is also possible to solve for t_2, i.e., the time for driving while decelerating. The sum of t_1 and t_2 is the complete time for the cruise from start node to destination node. The time for accelerating to v_0 is being subtracted, such that the remaining time can be expressed as in line 7 of Alg. 1.
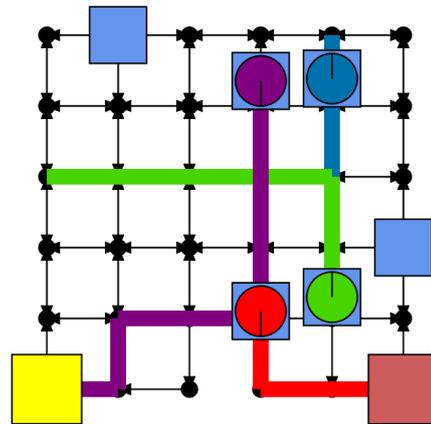
4   DEMONSTRATOR

We implemented a demonstrator functionality to investigate how our algorithms developed within RAWSim-O work with real robots. For example with the help of the iRobot Create 2, a mobile robot platform based on the Roomba vacuum cleaning robot. There are several reasons we choose the iRobot Create 2: ease of programming, low complexity, similar movement behavior like typical RMFS robots and low costs. The robots are equipped with ASUS Eee PCs through serial-to-USB cables for processing capabilities, webcams for line-following, blink(1) for visual feedback, and RFID tag readers mounted inside the former vacuum cleaning compartment for waypoint recognition (see Figure 6a). Although we cannot emulate the transportation of pods with the robots, we are still able to study the overall movement behavior in a real situation, which is more prone to errors and noise than a simulation. Technically, the demonstrator robots replace the simulated robots, hence, a hybrid of a real system and a simulation is built.

We demonstrate our RMFS with a simple running example of four robots in a grid-world with $0.45m \times 0.45m$ cells (totally 36 cells), see Figure 6b for the view from RAWSim-O's visualization for this example and Figure 6c for the view of the demonstration. One replenishment station is set on the left bottom side and one pick station is set on the right bottom side. In total, there are six pods (blue rectangles) available. The maximum velocity limit of each robot is $0.21\frac{m}{s}$, while the time it takes for each robot to do a complete turn is set to $5.5s$. And the maximum acceleration and deceleration of each robot are set to $0.5\frac{m}{s^2}$ and $-0.5\frac{m}{s^2}$. To adhere to the kinematic constraints (such as turning times and acceleration) in continious time, we use the MAPFWR-solver from [10] to generate time-efficient collision- and deadlock-free paths. Those paths are converted to a sequence of go straight, turn left, and turn right commands and sent to the robot via WiFi. The robot then executes these commands and sends back the RFID waypoint tag of each intersection it comes across. By doing this, the movement of the real robot and the expected movement are synchronized.

(a) Close-up of a demonstrator robot

(b) The view from RAWSim-O

(c) Multiple robots in an emulated RMFS

Figure 6: Demonstrator example of four robots running in an emulated RMFS

(a) Filtered image showing the line

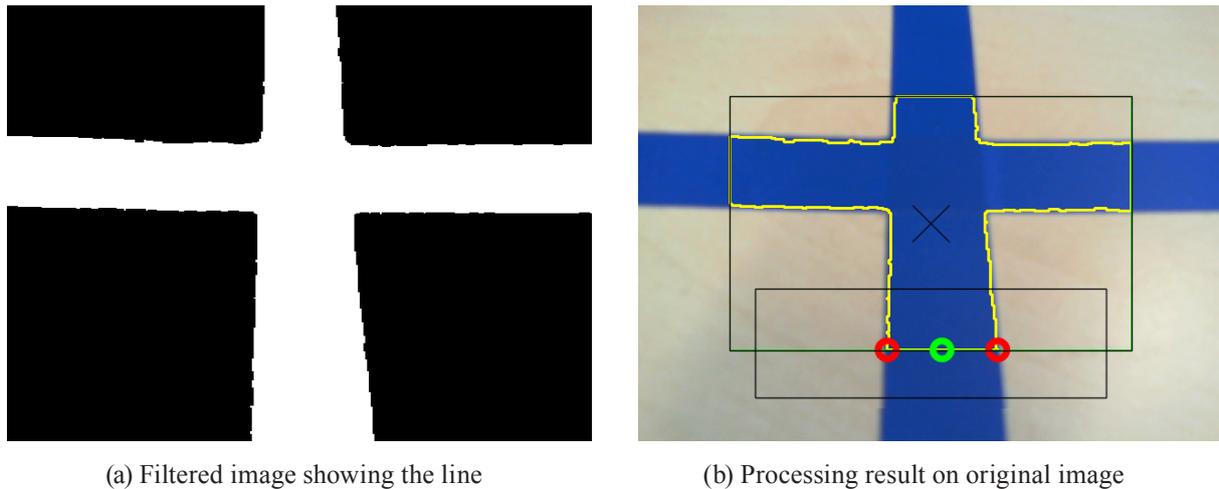(b) Processing result on original image

Figure 7: Line-following example using a blue line

This needs to be done in order to avoid errors from the noisy real-world setting to add up. Similarly to the simulated environment, pick and replenishment operations are emulated by blocking the robot at the station for a fixed time. With the published source code and similar components the demonstrator can be rebuild.

The robot line-following is implemented using a common PID (Proportional Integration Differential) controller method. The center of the line is extracted by using the OpenCV library. The line recognition first translates the image to a binary one by applying a range filter, and then applies the basic morphological operation "erosion" for removing noise (see resulting image in Figure 7a). Afterwards, contour extraction is used to find the line's borders within the gray image. Using these lines, the bottom center is estimated to allow a stable target (see green ring in Figure 7b) for the PID controller. Using the demonstrator, we are able to show the successful application of the path planning algorithms developed within RAWSim-O in a real-world situation. Furthermore, it can be used to demonstrate the basic idea of an RMFS at a small scale.

5   CONCLUSION

In this work, we outlined core real-time decision problems occurring when operating an RMFS. To investigate different solution approaches for those decision problems, we introduce the RMFS simulation framework RAWSim-O and describe its functionality and capabilities. Alongside this publication, we also publish the source code of the framework and already present controllers and algorithms at www.rawsim-o.de to support future research on RMFS. With this work we aim to support future research on RMFS. For example, a study of how to control systems involving multiple mezzanine floors can be done using RAWSim-O. For these, an efficient storage strategy and task allocation method is expected to be crucial in order to mitigate the bottle-neck effect introduced by the elevators. Furthermore, we have shown demonstrator application capabilities of the framework that enable the integration of simple robots for real world demonstrations.

REFERENCES

1. 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
2. Cohen, L., Uras, T., Koenig, S.: Feasibility study: Using highways for boundedsuboptimal multi-agent path finding. In: Eighth Annual Symposium on Combinatorial Search (2015)
3. 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)
4. Enright, J., Wurman, P.R.: Optimization and coordinated autonomy in mobile fulfillment systems. In: Sanem Sariel-Talay, Stephen F. Smith, Nilufer Onder (eds.) Automated Action Planning for Autonomous Mobile Robots (2011)
5. 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)
6. 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 inventory management using mobile drive units (2013). URL https://www.google.com/patents/ US20130103552
7. Khamis, A., Hussein, A., Elmogy, A.: Multi-robot task allocation: A review of the state-of-the-art. In: A. Koubâa (ed.) Cooperative robots and sensor networks 2015, *Studies in Computational Intelligence*, vol. 604, pp. 31–51. Springer, Cham (2015). DOI 10. 1007/978-3-319-18299-5 2
8. 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 10.1016/j.ejor.2006.07.009
9. Lamballais, T., Roy, D., de Koster, M.: Estimating performance in a robotic mobile fulfillment system. European Journal of Operational Research (2016). DOI 10.1016/j. ejor.2016.06.063
10. Merschformann, M., Xie, L., Erdmann, D.: Path planning for robotic mobile fulfillment systems. ArXiv e-prints (2017)
11. Mors, A.W.t.: The world according to MARP: Multi-Agent Route Planning. Technische Universiteit Delft, Delft (2010)
12. Roodbergen, K.J., Vis, I.F.: A survey of literature on automated storage and retrieval systems. European Journal of Operational Research 194(2), 343–362 (2009). DOI 10.1016/j.ejor.2008.01.038
13. Tappia, E., Roy, D., de Koster, R., Melacini, M.: Modeling, analysis, and design insights for shuttle-based compact storage systems. Transportation Science 51(1), 269–295 (2017). DOI 10.1287/trsc.2016.0699
14. Tompkins, J.A.: Facilities planning, 4th ed. edn. John Wiley & Sons, Hoboken, NJ and Chichester (2010)
15. Wulfraat, M.: Is kiva systems a good fit for your distribution center? an unbiased distribution consultant evaluation. URL http://www.mwpvl. com/html/kiva_systems.html
16. Wurman, P.R., D'Andrea, R., Mountz, M.: Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI Magazine 29(1), 9 (2008)