CrossMark

ORIGINAL PAPER

Mechanical feasibility and decentralized control algorithms of small-scale, multi-directional transport modules

Tobias Krühn[1] · Simon Sohrt[1] · Ludger Overmeyer[1]

© The Author(s) 2016. This article is published with open access at Springerlink.com

Abstract In this paper, we describe a conveyor matrix consisting of small-scale, multi-directional transport modules that are considerably smaller than the transported packets. If a large number of these modules are combined into a matrix, the emerging network solves transport tasks through cooperation of the modules. The control of the system is decentralized: Each module has its own control and derives its actions only from its own sensor and received messages from neighboring modules. Both the mechanical feasibility of the modules and the control algorithms are presented. We show that collision-free routes can be planned by the decentralized controlled system. Lastly, we present the necessary algorithms to detect and prevent deadlocks.

Keywords Decentralized control · Distributed control · Intralogistics · Warehousing · Material handling · Multiagent systems · Deadlock prevention

1 The need for flexibility as a result of modern distribution structures

Warehousing systems face new challenges. Smaller and smaller batch sizes and quantities, caused by product individualization (Mass Customization) and e-commerce, lead to considerable demands on the internal material flow [1, 13]. Product life cycles are becoming shorter, while the marketing supported by modern means of communication and online trading becomes faster. As a result, warehouse systems need to become more flexible to ensure quick response times to faster changing requirements [2].

In contrast to these requirements, today's material handling systems are inflexible from the mechanical and control perspective. Individual elements in conveyor systems are dedicated to specific tasks such as sorting, sequencing, rotating or transportation of goods. A short-term adjustment of the layout of a conveyor system causes high costs and downtime [5]. In addition, layout changes to existing systems necessitate a reconfiguration of the control. Therefore, conveyor systems need to become more flexible. This relates both to the mechanical implementation as well as to the control of these systems [2, 6, 11].

In this paper, we present the latest results for a new kind of continuous conveyor, which consists of small-scale, multi-directional transport modules [12]. Each of these modules is considerably smaller than the packets and can convey in any direction. If a large number of these modules are combined into a matrix, the emerging network solves transport tasks through cooperation of the modules. The resulting conveyor system is able to perform tasks such as transporting, rotating, separating and merging as well as buffering and sequencing [7]. In this article, the mechanical realization and algorithms for controlling the modules are shown.

2 Related research

The modules presented in this paper belong to the class of plug-and-work material handling systems which design principles are defined in [2]. In this paper, we will only look into plug-and-work conveyor systems that have both

---

✉ Simon Sohrt
simon.sohrt@ita.uni-hannover.de

[1] Institute of Transport and Automation Technology, Leibniz Universität Hannover, An der Universität 2, 30823 Hannover, Germany

proven their mechanical feasibility and disclosed their control algorithms.

MAYER and FURMANS describe in [10] algorithms for a completely decentralized controlled conveyor system. The FlexConveyor platform is built out of multiple, identical modules that can be easily plugged and unplugged. One module is at least the size of one packet. The platform is capable of decentralized generation of topological information, routing and deadlock avoidance. Complex transport tasks of multiple items with different sources and destinations can be performed. The platform is used as the basis for a sorting system [23] and a storage system [22].

During the initialization phase, every FlexConveyor module compiles a routing table using a *Distance-Vector Routing* approach. In this table, the topology information and the distance to the sinks (also called metrics) are stored. After the creation of the routing table, routes can be created for individual packages. Deadlocks are detected and avoided during the transportation phase. Due to the similarities to the modules described in this paper, the control algorithms of the FlexConveyor modules were used as a basis for designing our control algorithms.

KARIS PRO [15] combines the aspects of conveyors and autonomous guided vehicles. The vehicles cannot only perform the transport of single items but are designed to form two different functional clusters. As a discontinuous cluster, KARIS vehicles connect to each other in order to transport items that are multiple times larger than the vehicles. As a continuous cluster, several KARIS vehicles form a conveyor line to realize high throughput of goods. Two different algorithms for creating continuous clusters have been implemented and analyzed in a simulation environment [27]. The algorithms are capable of creating continuous cluster on maps where only the global coordinates of the packet source and sink are known. In contrast to the FlexConveyor, the control algorithms have only been shown to work for maps with a single source and a single sink. To the knowledge of the authors, no investigation has been performed about the avoidance of collisions and deadlocks of packets.

Apart from continuous conveyors, decentralized controlled automated guided vehicles (AGV) can be used to conduct the transport process. An example is the cellular transport system: swarm algorithms are used to control the autonomous vehicles' behavior. Moreover, optimization can be attained by using swarm algorithms for localization, navigation, collision avoidance, task allocation and transportation tasks [8, 20].

An overview of control algorithms for AGVs can be found in [16–18]. Most AGVs used in the industry are still controlled centrally [24]. The degree of decentralization of the control depends on the type of path planning that is used. [18] defines three different kinds of path planning:

open path, closed path and autonomous navigation. Both open path and closed path rely on the a priori creation of the paths by humans. The decentralized controlled AGVs then follow the predefined paths. Fully autonomous navigated vehicles can be realized by using potential fields [18, 19, 21]. In this case, the AGVs start with a topological map of their surroundings. The AGVs then calculated a potential field using the map and alter the field every time an obstacle is detected. Potential deadlock situations are resolved by backing up [25] or by replanning the route [17].

Since the FlexConveyor platform is conceptually comparable to the modules described in this paper, our approach is modeled after the algorithms described in [10]. Because the control algorithms for the KARIS vehicles can only deal with single sources and sinks, they are dismissed as a basis for our algorithms. The utilization of the routing algorithms for AGVs was considered, but ultimately not pursued. The main reason was that the deadlock avoidance algorithms for AGVs need unoccupied space around them to work. This requirement is not given for our modules. The available space is defined beforehand by the number of deployed modules.

3 Mechanical feasibility of small-scale conveyor modules

A matrix consisting of multiple identical conveyor modules is only able to perform tasks like transporting and rotating of packets, if the modules are sufficiently small enough. A first prototype was shown in [14] that demonstrated the ability of modules to convey and rotate goods. These prototypes lacked the necessary electronics to control the motors and to communicate with neighboring modules. Before any control algorithms can be developed, it is necessary to show the feasibility of small-scale conveyor modules both from a mechanical and a control point of view.

The dimensions of the realized prototype modules are 65×65 mm. They consist of a tilt disk driven by two identical, collinear mounted motors for varying the tilt angle and for propulsion. Figure 1 shows the module matrix integrated in an evaluation setup and the detailed view of a prototype. The evaluation setup consists of multiple conventional conveyors along with a matrix consisting of 16 active and 20 passive small-scale modules. Figure 2 shows the interior of a module. The active modules are all identical and dynamically exchangeable during the runtime of the system.

Each module contains a microcontroller (8-bit RISC, 8 kB RAM) and four optical communication interfaces with a full-duplex bandwidth of 115 kBit/s each. The drivers are
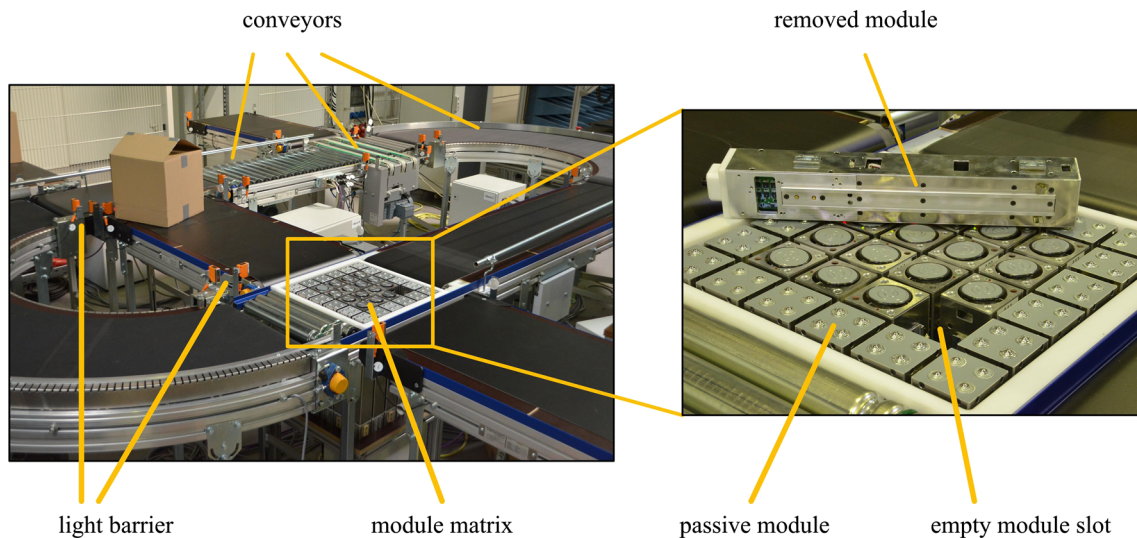
conveyors   removed module

light barrier   module matrix   passive module   empty module slot

Fig. 1 Evaluation setup of multiple conveyors and view of the module matrix

motor power amplifier   motors   guide rail   light sensor

power (+24 V)   control board   programmer interface (JTAG/ISP)   communication interface
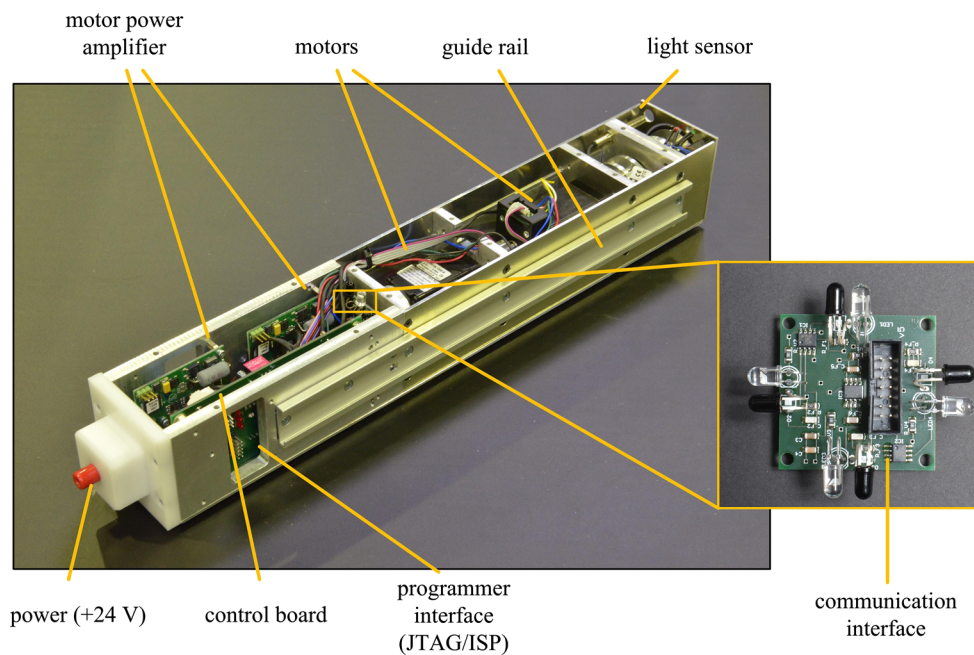
Fig. 2 Detailed view of a module

servo motors with attached power amplifiers. The maximum conveying speed is 2 m/s. When several modules are combined to form a conveyor matrix, it has a carrying capacity of 125 kg/m^2. For a typical packet of 600 mm × 400 mm, this corresponds to a weight of 30 kg. The demonstrator shows that the mechanical design of the matrix is possible.

The implemented software contains the communication system, engine control and functions such as feeding, separation or turning of packets. The communication within the matrix is decentralized. Via the adapter module,

the matrix is given a sequence of actions. An example: accept a packet from direction A, turn the packet around 90° and eject the packet in direction B. The execution of these actions is carried out automatically by the module matrix. Modules determine where the packet is on the matrix using light sensors. More details of the realized prototypes can be found in [26]. As a critical element regarding the computational power, the communication interface has been identified. Since the mechanical feasibility of the modules has been proven, it is possible to use the modules to implement complex routing algorithms.

4 Guaranteeing the control's scalability by local neighborhoods

Since the number of modules can become very large, the control of a module must not depend on the states of all other modules in the matrix to keep the system scalable. Problems must be solved on the basis of local information, coming from a limited number of modules in the local neighborhood. The size and shape of neighborhoods depend on the maximum size of the packets that need to be conveyed. Therefore, it is independent of the system size.

Since each transport module is smaller than the packets, modules must form groups to conduct the transport process. To transport a packet, a route reservation from source to sink is necessary (see Fig. 3). If multiple packets are moving on the matrix without a previous route reservation, they can meet each other where an evasion is not possible. The packets might not be able to move back due to subsequent packets, so that the system comes to a deadlock.

Route reservation appears as a classic route planning problem as it occurs in networked communications. However, the consideration of the physical dimensions of packets represents a fundamental difference to previously known algorithms. If packets should be transported through a bottleneck where they physically do not fit, then transport is not possible. In communication networks, this problem is not taken into account. Therefore, the modules have to consider the size of the packets.

Moreover, routes in the module matrix are arbitrarily long and thus depend indirectly on the system size. The algorithms working on the routes may therefore not be dependent on the state of the entire route. To achieve this independence, information along the route can only be passed from module to module. This results in a significant
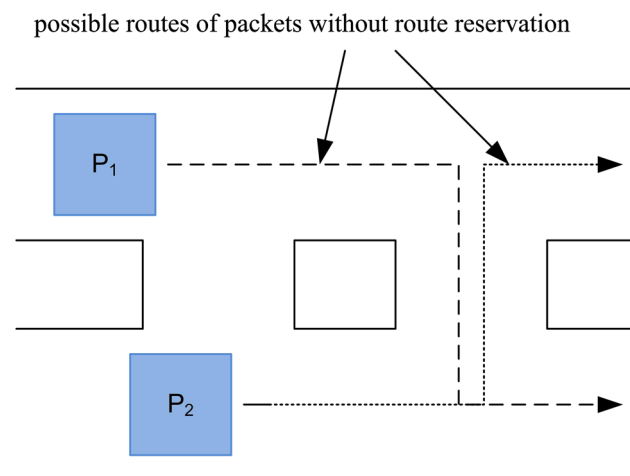
possible routes of packets without route reservation

Fig. 3 Without route reservation, the system will run into deadlocks

delay in the exchange of information between more distant parts of the module matrix. The developed algorithms are thus designed so that delays in the exchange of messages are considered.

Altogether, large scale tasks must be reduced to spatial and temporal local problems. The following sections explain the implemented algorithms for routing and deadlock prevention.

5 Calculating the routing table considering packet dimensions

Before a packet can be conveyed, the topological information must be created. During the initialization process, every module generates a routing table. For every combination of packet size and sink, a metric is generated. Creation of the metric is modeled after the creation of the routing table for the FlexConveyor modules [10]. The main difference is that the FlexConveyor modules do not take the packet size into account.

First each module determines, for all four directions, the estimated virtual cost of moving a transportation unit in the respective direction. The calculated costs are added to the remaining costs of the adjacent module. After the remaining costs for all four directions are calculated, the best direction is selected and the remaining costs and directions are stored. This procedure extends the distance-vector algorithm. The extension of JAFFE and MOSS [4] is used to provide better convergence and to avoid the count-to-infinity problem. GARCIA–LUNA–ACEVES showed that this algorithm can be implemented in local neighborhoods [3]. Due to the two-dimensional propagation of the metric and the small size of the modules relative to the packets size, the term *field* is used to describe the semi-continuous metric in this work.

Figure 5 shows an example of the development of the cost field for a sink in a transport matrix. To distinguish various sinks, the sinks and the corresponding fields are marked with colors. All blue color scales and directional arrows shall apply mutatis mutandis for the blue marked sink.

Initially, all modules have infinitely high residual costs, since no sink is known (Fig. 4a). After starting the system, a sink module is initialized that creates a metric field for every packet format that it can accept. In order to keep the example simple, only the creation of the metric field for a 5×5 packet is shown and only the distance is used as a metric. The destination module shares its remaining costs with its neighbors, which then recalculates their residual cost (Fig. 4b). In the image, a light color of a field stands for a low value of the residual cost and a dark for a high value. With each time step, the information continues to spread.
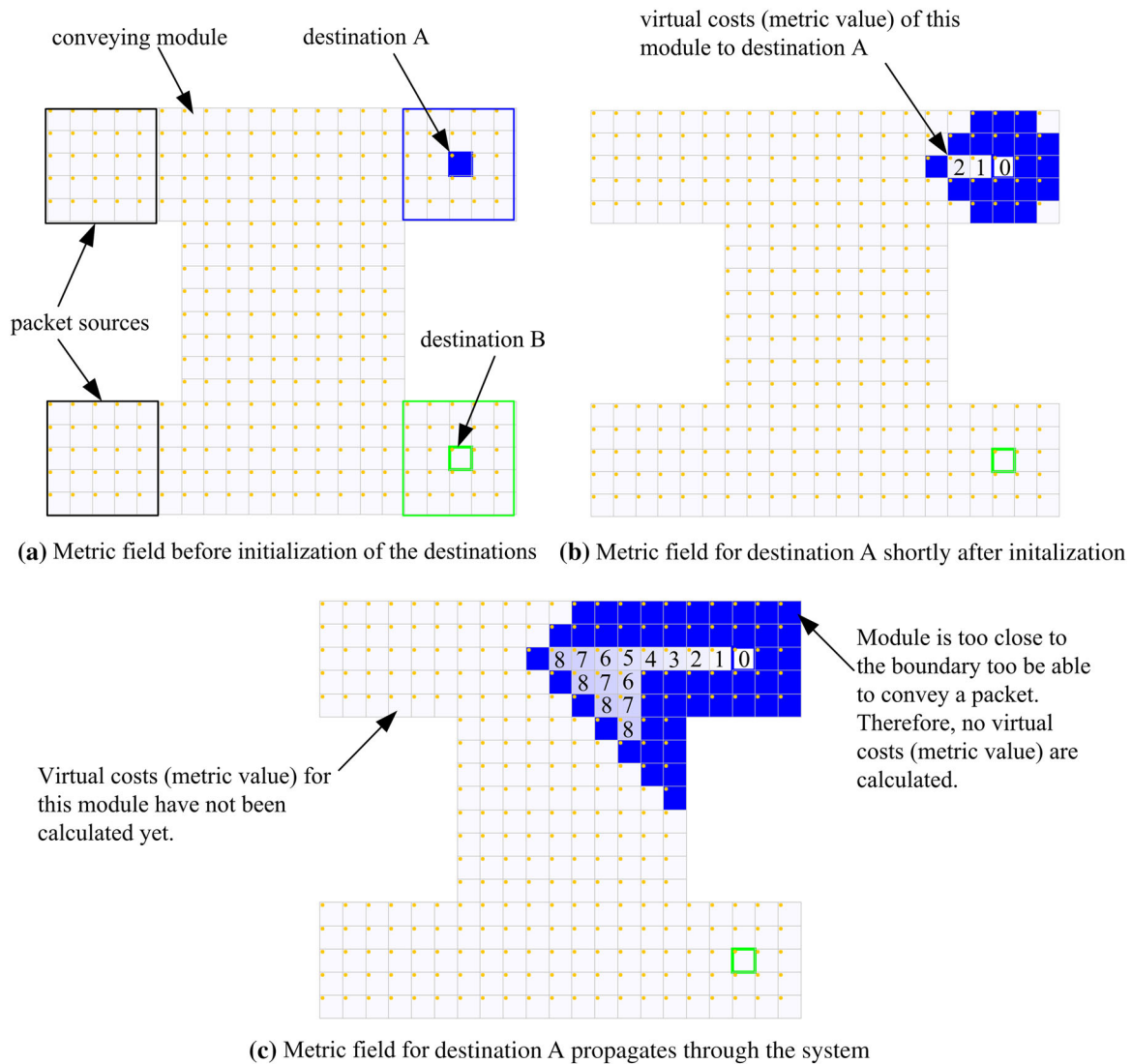
(a) Metric field before initialization of the destinations   (b) Metric field for destination A shortly after initalization

(c) Metric field for destination A propagates through the system

Fig. 4 Field propagation in a module matrix

For each module, the metric is calculated in the assumption that the packet is located centrally above the module. The modules at the edge of the matrix are therefore not suitable for the planning of routes, because the packet would hit the system boundary. Consequently, they get a high metric value and are shown in dark (Fig. 4).

The procedure described requires that for each packet format, a separate metric is calculated. However, formats can be combined. If no suitable metric exists for a packet format, the next largest is used. Since all metrics need the information about the distance of the module to the nearest system boundary, this information is calculated a priori and stored in a base metric (Fig. 5).

New metrics can not only be created by sinks, but by regular modules on the matrix as well. The process is described as follows: A module is informed by one of its neighboring modules of the existence of a metric for sink 1

and the packet format 5×7. It now checks if the distance to the system boundaries allows the creation of both a metric for sink 1 with the packet format 5×7 and a metric for sink 1 with the packet format 7×5. After creating the metrics, it calculates their respective values.

Figure 6 shows the necessity for creating new metrics on the transport matrix: The displayed system includes a bottleneck that can only be passed by packets with the format 5×7. The effect of the bottleneck on the metric field for the packet format 7×5 is shown in Fig. 6a: A local minimum is created between the packet source and the bottleneck. The calculated gradients for this metric are shown in Fig. 6c. All packets with the format 7×5 are transported according to these gradients. Once the local minimum of the metric is reached, the packet is rotated. After rotating the packet, the metric depicted in Fig. 6b is relevant for calculating the gradients for the movement

Fig. 5 A new metric is
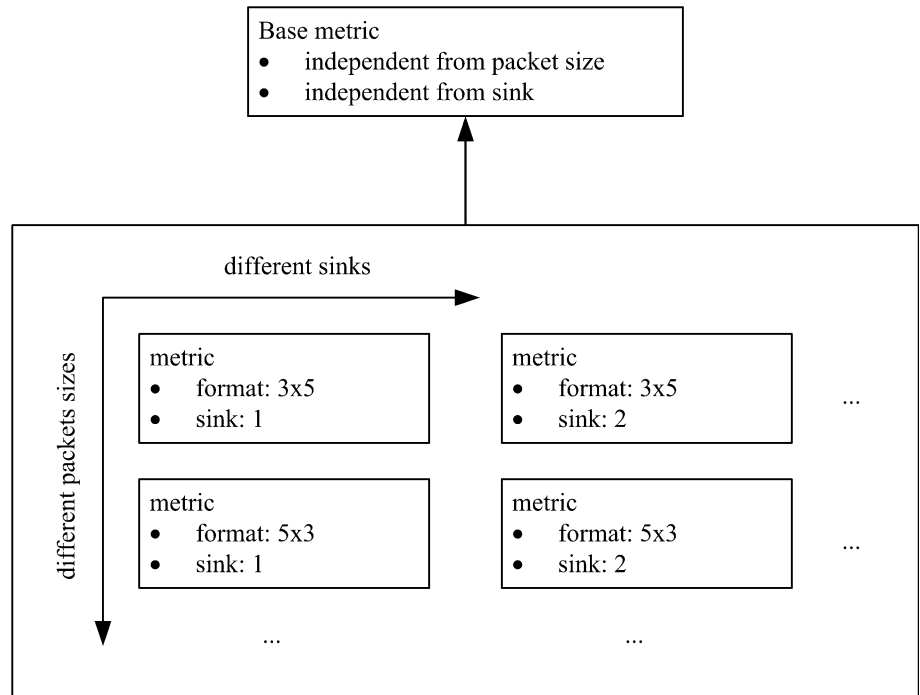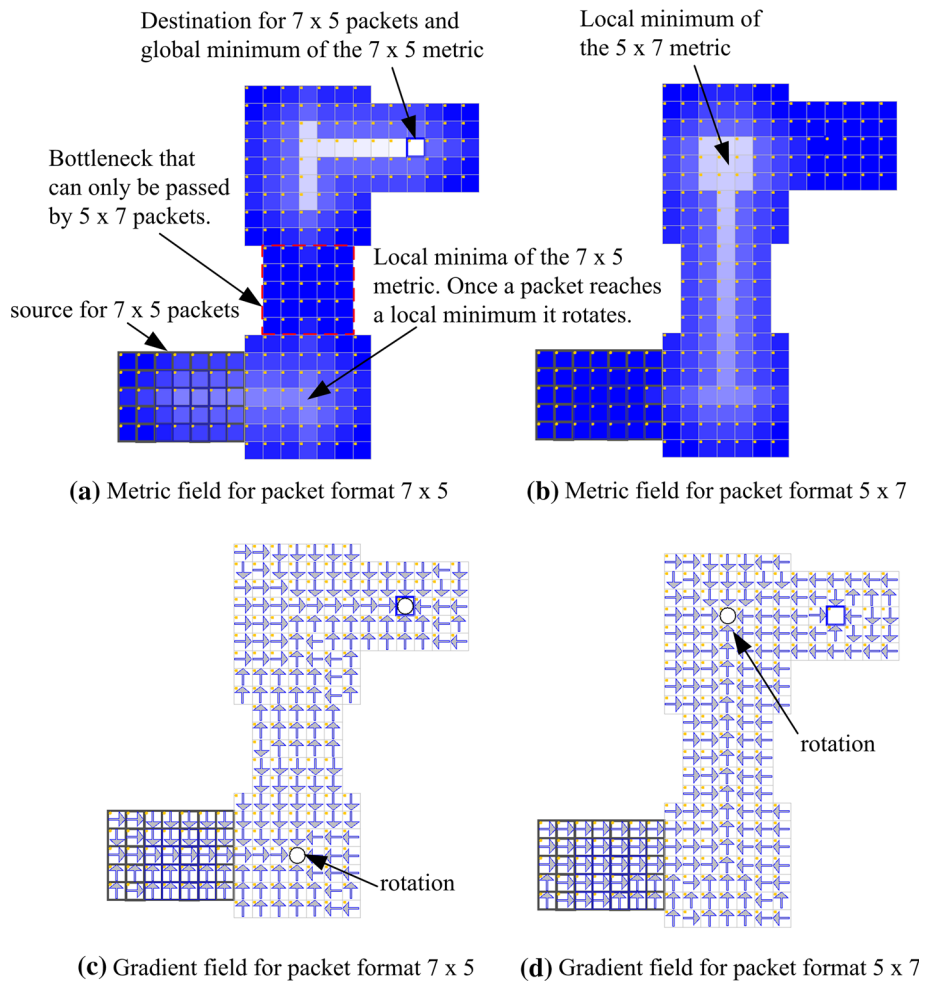generated for each sink and
format

Base metric
- independent from packet size
- independent from sink

different packets sizes

different sinks

metric
- format: 3x5
- sink: 1

metric
- format: 3x5
- sink: 2

...

metric
- format: 5x3
- sink: 1

metric
- format: 5x3
- sink: 2

...

...   ...

Fig. 6 Metric and Gradient
fields shown for different packet
formats

Destination for 7 x 5 packets and
global minimum of the 7 x 5 metric

Local minimum of
the 5 x 7 metric

Bottleneck that
can only be passed
by 5 x 7 packets.

Local minima of the 7 x 5
metric. Once a packet reaches
a local minimum it rotates.

source for 7 x 5 packets

(a) Metric field for packet format 7 x 5

(b) Metric field for packet format 5 x 7

rotation

rotation

(c) Gradient field for packet format 7 x 5

(d) Gradient field for packet format 5 x 7

(shown in Fig. 6d). Since the destination of the system can only accept packets with the format 7 × 5, another rotation is necessary to complete the conveying process.

Since routes in the matrix can be planned freely, it is first necessary to determine how multiple routes may relate to one another. Figure 7 shows the basic situations for which an assessment is determined. Each arrow represents a route segment. For example, a route may be branched (a) or merged (b). Thus, the situations shown in Fig. 7a and Fig. 7b are acceptable: Fig. 7a shows a route coming from the west, branching into two routes, running north and south. Figure 7b shows two route segments, coming from the east and west, merging into a route running south. Because, these situations are normal for routes, low costs arise. Furthermore, multiple routes in the same direction may exist on a module (c). A straight route also results in low costs (d).

In addition, situations exist that may present a valid plan, but reduce system efficiency. Route branches close to one another result in intersections whose neighborhoods overlap (e). The superposition of intersections place increased demands on the avoiding of deadlocks. The same applies for route merges that take place not in the same module, but in a neighborhood (f), or for routes with closely spaced direction changes (g).

Routes in opposite directions, as in situation (h), are not allowed. Figure 7h shows two route segments, coming from east and west, very close to each other. No matter if these route segments belong to the same route or not, this situation is illegal. Further route planning would result in a situation where overlapping routes run in different directions. Packets would not be able to path both routes. To implement this criterion, the target module checks if the route overlaps with the neighborhood of another route. Parallel running routes whose neighborhoods overlap

should be excluded as (i). The reason for the two prohibitions is that possible deadlocks arise from these situations which are not covered by the implemented deadlock prevention. For the deadlock prevention, only intersections and direction changes are monitored.

For every pending packet, the module matrix plans a route a priori (i.e., before the conveying process on the matrix begins). A route through the matrix is made up of multiple route segments. A single module does not know the complete route through the system, but only its own route segment and the route segment of the neighbors of its von Neumann neighborhood.

Route reservation is done in a 2-way handshake. First, the route is planned and kept in the state *in progress*. The planning runs from source to sink from module to module. The next route segment is created according to the matching metric (see Fig. 5). If the route reaches the sink, it is reserved and the route segments switches from *in progress* to *reserved*. When the reservation reaches the source, the route is confirmed, and the packet can be transported. Since an exclusive route is planned for every packet, it is deleted as soon as the packet was conveyed.

Before switching a route segment to *reserved*, the modules have to ensure that this does not create a prohibited routing situation according to Fig. 8. Furthermore, the planning of an undesirable route should be avoided. Since route segments are planed according to the metric value, the calculation of the metric can be altered to take the distance to other routes into account. A route segment with the state *in progress* alters the metric considerably less than a route segment with the state *reserved*.

A first calculation of the metric value M was described in [7]. By taking the state of the routes into account the formula is as follows:

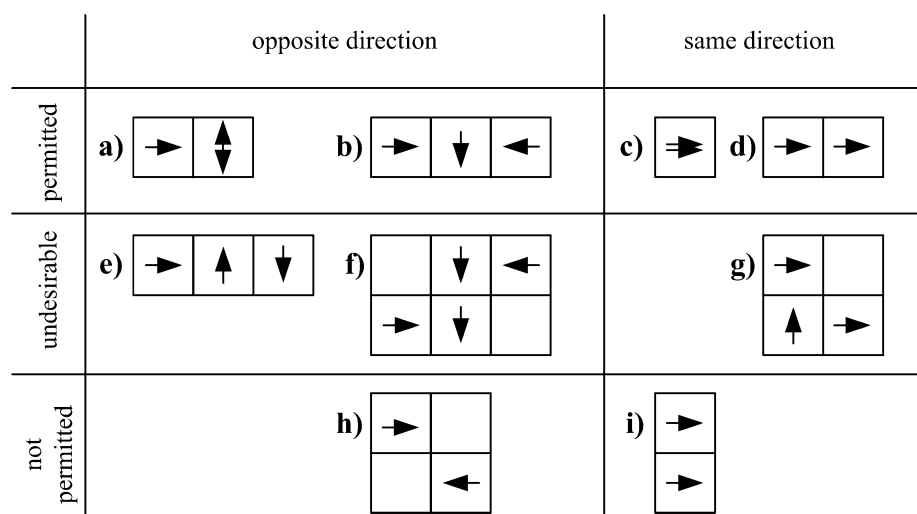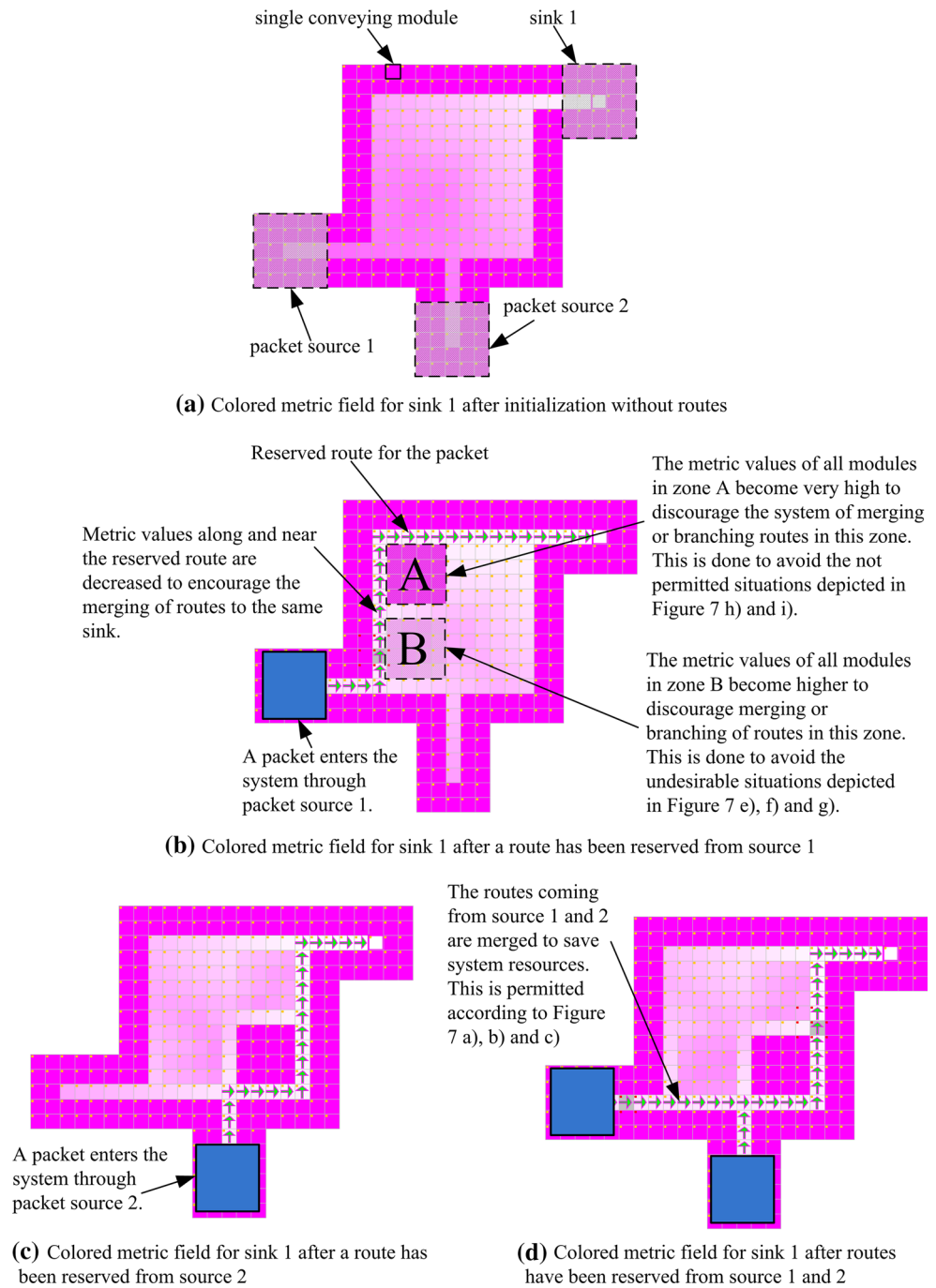**Fig. 7** Permitted, undesirable and not permitted routing situations—named **a** to **i**

Fig. 8 Influence of routes on the metric values

single conveying module   sink 1

(a) Colored metric field for sink 1 after initialization without routes

Reserved route for the packet

The metric values of all modules in zone A become very high to discourage the system of merging or branching routes in this zone. This is done to avoid the not permitted situations depicted in Figure 7 h) and i).

Metric values along and near the reserved route are decreased to encourage the merging of routes to the same sink.

A

B

A packet enters the system through packet source 1.

The metric values of all modules in zone B become higher to discourage merging or branching of routes in this zone. This is done to avoid the undesirable situations depicted in Figure 7 e), f) and g).

(b) Colored metric field for sink 1 after a route has been reserved from source 1

The routes coming from source 1 and 2 are merged to save system resources. This is permitted according to Figure 7 a), b) and c)

A packet enters the system through packet source 2.

(c) Colored metric field for sink 1 after a route has been reserved from source 2

(d) Colored metric field for sink 1 after routes have been reserved from source 1 and 2

$$M = \min(M_R \vee M_{N,i}) \quad \text{with}$$

$$M_{N,i} = M_D + M_{s,i} + \begin{cases} S_R \cdot M_{0,i} & \text{if route exists} \\ S_R \cdot M_{G,i} & \text{if opposing route exists} \end{cases}$$

$M_{s,i}$ Depends on the neighboring modules distance to the system boundary. $M_{s,i}$ becomes ∞ if the boundary is too close to convey a packet

M_R Are the costs for rotating the packet if necessary

$M_{N,i}$ Is the metric value of neighbor i from von Neumann neighborhood

$M_{0,i}$ Is the transport cost to move a packet by one module in the direction of an existing route for this neighbor i

$M_{G,i}$ Is the transport cost to move a packet by one module against the direction of an existing route for this neighbor i

M_D Routing costs only based on the physical distance of one module to another

S_R Factor to consider the state of the route. S_R for *reserved* routes is bigger than S_R for *in progress* routes

The quotient of $M_{G,i}/M_{0,i}$ defines the maximum detour a planned route will accept before the system tries to plan a new route conflicting with an existing route. The already existing route will then be re-planned. If this is not desired, the value of $M_{G,i}/M_{0,i}$ must be larger than the longest route in the system. The determination of the specific values depends on the desired system behavior.

Routes that are near to each other and run in the same direction are combined to save system resources. The quotient $M_D/M_{0,i}$ defines what near means in that case, because the routing costs along existing routes are smaller than on free modules. The larger the quotient, the more the system tries to merge routes in larger areas.

Every time a module changes its metric value, the neighbors are informed and hence update their metric. Thus, each time a packet is inserted into the system and a new route is planned, the metric values of the modules may change.

The computation and memory requirements for the calculation of the metrics depends upon the number of observed sizes n_{Format} and the number of possible sinks n_{Sink}, since for each combination, a metric is calculated. The cost of computing all metrics is thus:

$$f_{\text{metric}}(n_{\text{format}}, n_{\text{Sink}}) \in O(n_{\text{format}}, n_{\text{Sink}}).$$

The effects of newly planned routes on the metric values are depicted in Fig. 8. Figure 8a shows the metric values of the matrix without any routes. In Fig. 8b, a route was successfully reserved from the packet source 1 to the sink 1. The metric value of all modules with a route segment becomes lower since $M_{0,i} \leq M_{D,i}$. Modules in the vicinity of the route are affected as well: If the merging and branching of routes is permitted according to Fig. 7, the metric value of these modules is consequently lowered.

However, if the merging or branching of the route would lead to an undesirable or unpermitted situation as depicted in Fig. 7, the metric value is increased accordingly. This increase is depicted in Fig. 8b: Since merging or branching a route in zone A would directly oppose the already existing route, the metric value is altered to prohibit merging and branching. The metric value in zone B is only increased slightly since merging and branching in this zone would not create an unpermitted situation, but only an undesirable situation.

Figure 8c shows the metric field if a route from source 2 to sink 1 is reserved. The metric values on and near the route are altered accordingly. Figure 8d shows the successful merging of two routes to the same sink. The route coming from source 2 was reserved slightly before the route coming from source 1. Therefore, the route coming from source 1 was merged with the already existing route to save system resources. In contrast to Fig. 8b, the route from source 1 is now vastly different.

After a route has been reserved, the transport process starts. Since the reservation of a route only guarantees that a packet can be conveyed without a collision, possible deadlock situations have to be identified and prevented during the transport process.

6 Local and global deadlock prevention

After routes are planned, packets may be moved across the conveying matrix. In the next step, the absence of deadlocks must be ensured.

To avoid deadlocks, two different cases have to be considered. On the one hand, global deadlocks can occur [9]. This refers to a system state in which multiple packets wait on another cyclically. On the other hand, local deadlocks can occur at intersections. Figure 9 demonstrates a local deadlock situation. If both packets are moved in the
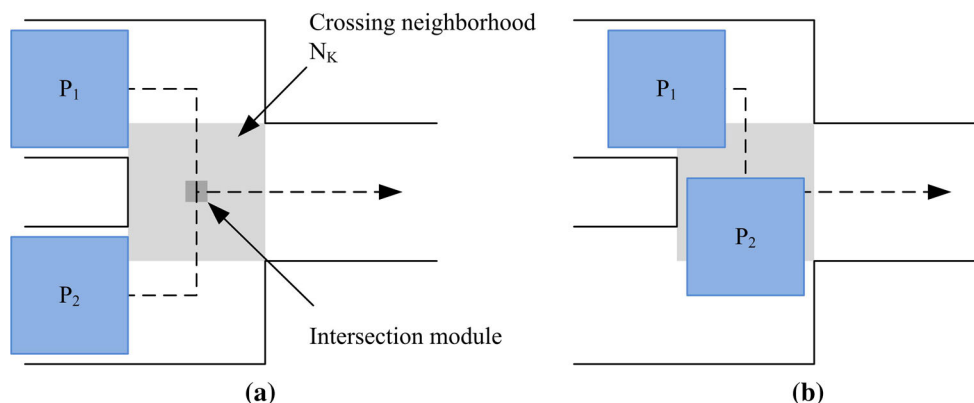

Fig. 9 Local deadlock situation

crossing region, neither of the two packets can pass through the area. A packet might not be able to back off due to following packets.

The example shows that on the conveying matrix, deadlocks are possible that do not occur in conventional conveyor systems and which are not solved by MAYER. This is due to the fact that crossing areas are created dynamically during runtime and can overlap each other.

To guarantee the absence of deadlocks, the neighborhood of a crossing is always occupied by a maximum of one packet. The dimension of the crossing area depends on the dimensions of the involved packets. Since information on all scheduled routes is present at the central module at the crossing, this one is defined as the crucial element. A module is called intersection module if it contains routes in different directions or a route changes its direction on the module. Since this module has the route information, it also defines the size of the neighborhood. For this purpose, it sends a message to all modules in the neighborhood $N_{Crossing}$ belong to the crossing area.

Acceptance of a packet at a crossing occurs in two steps: First, the intersection module accepts a packet and informs its neighborhood. If a module is a member of several crossing neighborhoods, it has to decide in the second step for the adoption of an acceptance from exactly one of the two intersections.

At any time, the intersection module can only accept one pending packet. After the decision is made, it remains until the packet leaves the crossing. The decision can only be reverted, if the already accepted packet is blocked. Otherwise, it may happen that the last accepted packet enters the crossing, while another is released (race condition).

The entrance of packets in a crossing region is defined by the following rules:

1. If a module is in no crossing neighborhood, a packet may enter it at any time.
2. If a module is in exactly one crossing neighborhood, it may be entered by a packet only if the intersection module has accepted the entrance of that packet.
3. If a module is in multiple crossing neighborhoods and all crossings have accepted the same packet, the packet may enter.
4. If a module is in multiple crossing neighborhoods and the associated intersection's modules have accepted different packets, one packet must be chosen. The packet is selected by the intersection module, which is aware of the routes of all packets involved.
5. If there is no module that knows all the routes, the older packet is selected. This is possible since every packet introduced into the system is provided with a time stamp by its respective source. If the time stamp is identical, the system randomly selects one of the packets.

Figure 10 shows an example where the rules 3 and 4 are explained. In Fig. 10a, P_1 is already on crossing 1 and wants to enter crossing region 2. At the same time, P_2 wants to enter crossing 2. The intersection module 2 now can choose freely one of the packets. If it chooses P_1, rule 3 applies. P_1 may enter the overlapping area; there is no risk of a deadlock.

However, if intersection module 2 accepts P_2, rule 4 applies. Figure 10b shows the situation. The intersection module 1 knows all routes. In accordance with rule 4, P_1 is accepted. Otherwise, a deadlock occurs.
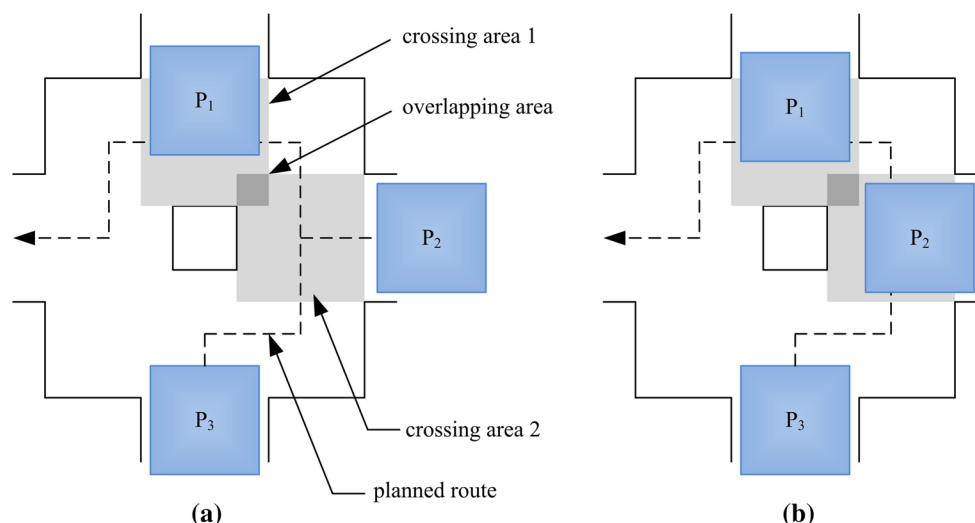

Fig. 10 Danger of local deadlock: If P_2 is accepted by the overlapping area, the system comes to a deadlock
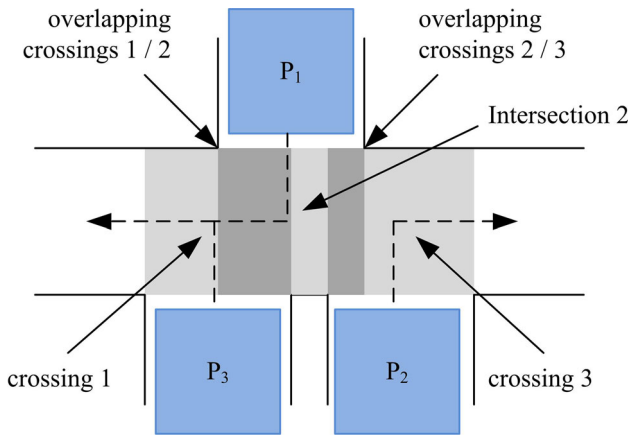
Fig. 11 Complex situation of overlapping crossings and multiple pending packets

If two crossings are overlapping, but none of the modules knows all the routes, usually rule 5 applies. Figure 11 shows the situation. As long as all modules in the overlap region 2 behave the same way, there is no risk of a deadlock. The global deadlock prevention was implemented as it is described by Mayer with extensions to fit the module matrix [10].

7 Verifying the control algorithms in a simulation environment

Since only up to four conventional conveyors can be attached to the prototype described in chapter 3, a simulation environment was written to simulate the behavior of conveyor matrices with different sizes and a different number of sources and sinks. The simulation is based on the multi-agent simulation framework MASON [28]. It was used to verify the propagation of metric values, the planning and reservation of routes and the detection and avoidance of deadlocks. Figures 4, 6 and 8 were created by using the simulation environment.

In the simulation, both space and time are discrete. Conveyor modules are positioned in cells in a grid. Controls of the modules are modeled as state machines. The states of the individual controls of the modules are all updated simultaneously. This is achieved by utilizing a global scheduler that informs the individual controls to update their state. The global scheduler only informs the controls to update their state after all controls from the previous update cycle have finished updating.

Different topologies are created by placing modules on the grid. Packet sources and sinks are created by assigning a special property to modules: These special modules are allowed to create or delete packets from the simulation environment. Inertia of packets is neglected. Packets either do not move at all or move with the maximum speed which is 10 % of the width of a module per update step. Force transmission between the modules and the packets is assumed to be ideal: Effects of slippage are neglected.

Modules can send messages to all modules in their neighborhood in every update step. A neighborhood is defined as a coherent group of modules without gaps. Figure 12 depicts the von Neumann, the Moore and a rectangular neighborhood. A module can be a member of several overlapping neighborhoods at the same time. The communication lag that would occur on real hardware is neglected, since communication between modules in a neighborhood only takes one update step. We allow this neglect, because the speed of sending data packets is several orders of magnitudes faster than the speed of conveying physical packets.

The simulation environment was used to determine the effect of both control parameters (described in chapter 5) and deployment parameters on the throughput of the system. The topology of the matrix, the rate at which packets

Fig. 12 Different module neighborhoods

Module does not belong to neighborhood
Module does belong to neighborhood
Central Module

Moore-neighborhood　　　von-Neumann-neighborhood　　　rectangular neighborhood
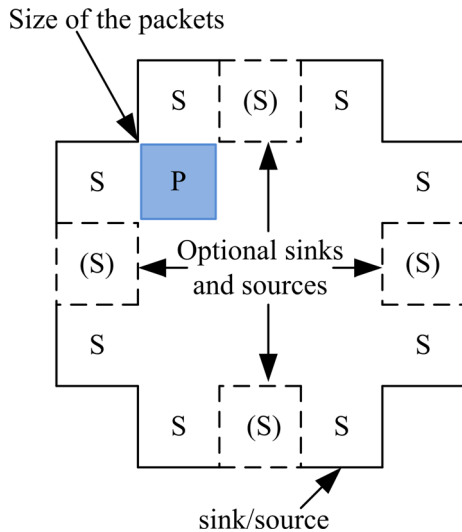
Size of the packets
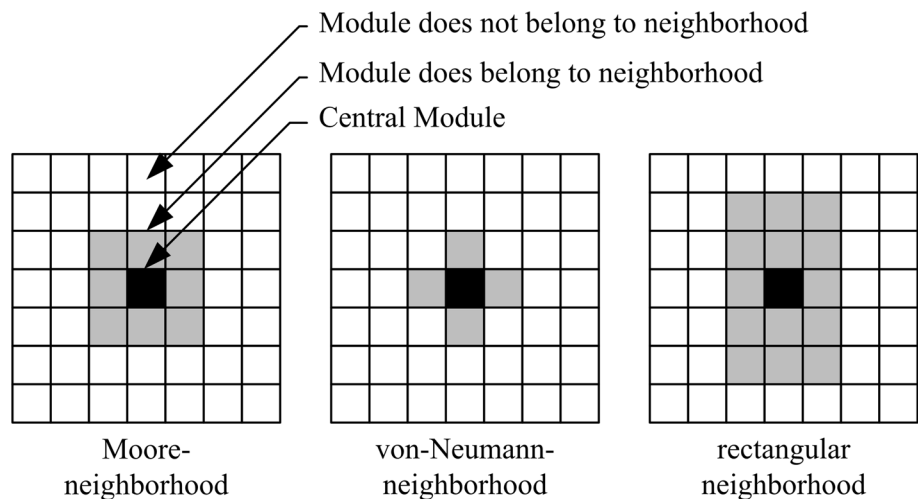
Fig. 13 Conveyor matrix as a hub

are created, packet shapes and packet sizes are defined as deployment parameters. Creation rate for packets is modeled as follows:

$$P_A = \frac{\text{Packets}}{\text{Simulation step}}.$$

In this paper, we will give a short overview of the influence of deployment and control parameters on packet throughput for an example layout. A more detailed throughput analysis was performed by KRÜHN and can be found in [26]. The examined layout is shown in Fig. 13. Deployment parameters that were constant are the size and the shape of both the matrix and the packets: Packets and the matrix in this first setup were always square, and the sides of the matrix were three times as large as the packets sides. Altered deployment parameters were the packet creation rate and the number of simulated conveyors that are connected to the matrix. Either 8 conveyors were used or 12. A simulated conveyor can act both as a packet source and as a packet sink.
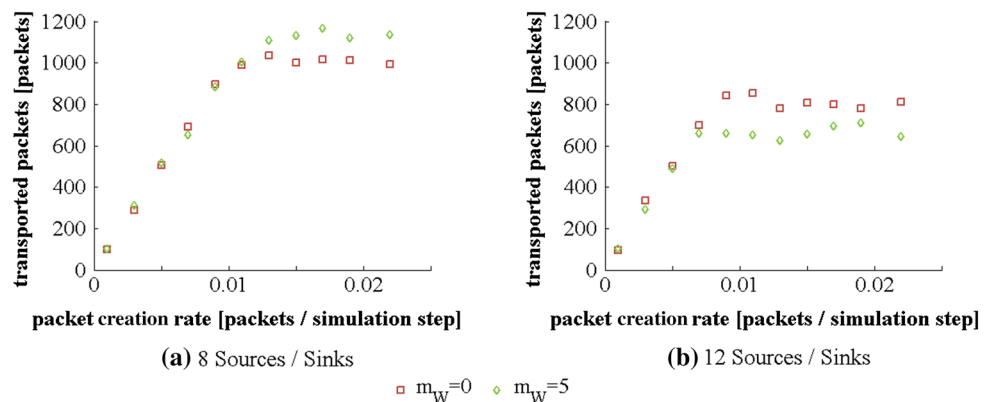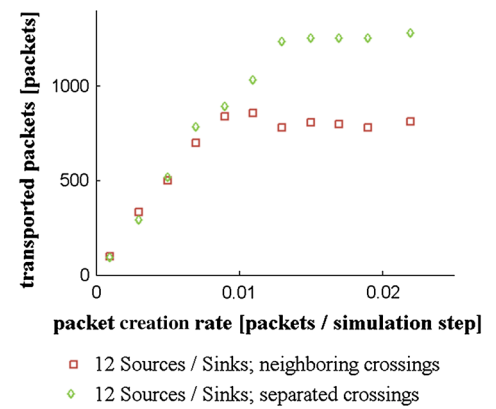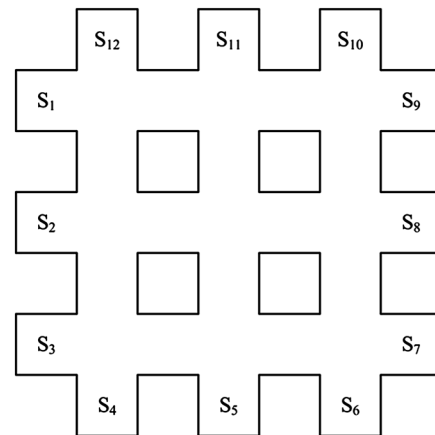
Control parameters were varied for both the layout with 8 sources and sinks and 12. Since the packets were quadratic, no rotation of packets was necessary. Consequently M_R had no influence on the metric value. The value of M_G/M_0 was set larger than the longest route in the system; therefore, the system never replans reserved routes. M_D/M_0 was set to match the size of the packets to prevent the overlapping of routes and instead encourage the merging of routes. S_R was twice as big for *reserved* routes than for *in progress* routes. The only control value that was altered during the simulation was the influence of the distance to the system boundaries by altering M_s. Figure 14 shows the result from 100,000 simulation steps.

The throughput of (a) is never reached in the scenario (b). If M_s has a high value packets are conveyed closer to the boundary of the matrix and a circular packet flow emerges; the center of the matrix is not used. As seen in Fig. 14, the influence of M_s on the throughput is significant. It is remarkable that a high M_s has a positive effect on the throughput in (a) and a negative one in (b). One possible explanation for this effect could be the number of crossings: The circular packet flow that emerges for high M_s values leads to the creation of 4 crossings in the scenario (a). If the value for M_s is decreased and routes are planned more freely, up to 9 crossings can develop. Since every crossing has to be checked for possible deadlock scenarios and has to synchronize itself with neighboring crossings (see chapter 6), the throughput is decreased. A high value for M_s leads to a circular flow with eight crossings for scenario (b) while a low value leads to a route network with nine. The higher number of crossings seems to be compensated by the additionally used space.

To test the hypothesis that the number of crossings and their synchronization is the reason for the decrease in throughput, a second layout (see Fig. 15) was simulated. In this second layout, no crossing is a neighbor of another one.

Figure 15 shows that the throughput with separated crossings is higher than without separated crossings. The

Fig. 14 Throughput for a matrix with the size 3×3

(a) 8 Sources / Sinks

(b) 12 Sources / Sinks

□ $m_W = 0$ ◇ $m_W = 5$

Fig. 15 Conveyor matrix with separated crossing areas

synchronization of crossings is thus the main factor of the decrease in throughput. The simulation results show that the control algorithms are capable of conveying packets without the occurrence of collisions or deadlocks.

8 Summary

Today, installed intralogisitic systems are designed to run for many years. Extensions and modifications of these systems are expensive and time consuming and therefore often uneconomic. Powered by recent developments such as shorter product life cycles and mass customization, system manufacturers are looking for ways to improve future delivery technology significantly. The aim is both autonomous and flexible adjustment of a conveyor system to changing conditions as well as their improved versatility.

As shown in this paper, small-scale, multi-directional conveyor modules have been developed which, fitted together to form a larger matrix, can solve material handling tasks. Mechanically they are able to do all regular tasks, such as sorting, merging, turning and sequencing, and thus offer a unique opportunity to map all the tasks on the same base module. However, so far we have lacked the ability to control a large number of these modules.

This paper discussed how a scalable, decentralized control can be implemented. The steps were:

- The dynamic formation of module neighborhoods that jointly solve a task,
- Planning and reservation of routes for packets, including the ability to reschedule these over time,
- Synchronization of crossing areas to prevent local deadlocks,
- The implementation of an algorithm to prevent global deadlocks,
- Verification of the algorithms in a simulation environment.

The essential core of the algorithms is that decisions are made solely on the basis of local information. Interactions over longer distances can be achieved by information passing from module to module. The resulting time delay is taken into account by the algorithms. It follows that an exchange of information over long distances on the module matrix is possible, but is decoupled in time. The system remains scalable.

As a first step toward the realization of the module matrix, an experimental system has been developed. This demonstrates the feasibility and shows first mechanical functions, such as the transportation and the turning of packets. At the same time, it is clear that further development work in the area of hardware is necessary.

To reduce the design space and to reduce costs further, integration of the electronics is desirable in the future. For example, two motor amplifiers are currently installed per module, which are significantly oversized. An integration of the motor amplifiers and the control is possible on a single board. Our current works show that a drastic reduction of manufacturing costs in the field of mass production is possible.

In summary, it can be stated that the design, manufacture and control of a very large number of small-scale modules is possible by reducing complex transport tasks to spatial and temporal local problems.

References

1. Cox WM, Alm R (1998) The right stuff: America's move to mass customization. In: Federal Reserve Bank of Dallas, Annual Report, pp 3–26

2. Furmans K, Schönung F, Gue KR (2010) Plug-and-work material handling systems. In: Progress in material handling research, pp 132–142

3. Garcia-Luna-Aceves JJ. (1988) A distributed, loop-free, shortest-path routing algorithm. In: INFOCOM'88. Networks: evolution or revolution, Proceedings of the Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies, IEEE, pp 1125–1137

4. Jaffe J, Moss F (1982) A responsive distributed routing algorithm for computer networks: communications, IEEE transactions on. In: Communications, IEEE transactions on. doi:10.1109/TCOM.1982.1095632 30, Nr. 7, pp 1758–1762

5. Jodin D, Hompel MT (2006) Sortier- und Verteilsysteme: Grundlagen, Aufbau, Berechnung und Realisierung. Springer (VDI-Buch). ISBN 978–3–540–29071–1

6. Kamagaew A, Stenzel J, Nettstrater A, ten Hompel M (2011) Concept of cellular transport systems in facility logistics. Automation, Robotics and Applications (ICARA), 2011 5th International Conference on, pp 40, 45. doi: 10.1109/ICARA.2011.6144853

7. Krühn T, Radosavac M, Shchekutin N, Overmeyer L (2013) Decentralized and dynamic routing for a cognitive conveyor. In: International Conference on Advanced Intelligent Mechatronics (AIM), pp 436–441. IEEE/ASME, Wollongong

8. Günther WA (2012) Algorithmen und Kommunikationssysteme für die Zellulare Fördertechnik Forschungsbericht zum IGF-Vorhaben 16166 N der AiF-Forschungsvereinigung Bundesvereinigung Logistik (BVL) e.V. München, Dortmund. ISBN/ISSN 978-3-941702-26-4

9. Mayer S, Furmans K (2010) Deadlock prevention in a completely decentralized controlled materials flow systems. Logist Res 2:147–158. doi:10.1007/s12159-010-0035-4

10. Mayer S, Furmans K (2009) Wissenschaftliche Berichte des Institutes für Fördertechnik und Logistiksysteme der Universität Karlsruhe (TH). Bd. 73: Development of a completely decentralized control system for modular continuous conveyors. Universitätsverlag Karlsruhe

11. Nettstraeter A, Nopper JR, Prasse C, Hompel MT (2010) The internet of things in logistics. In: Smart objects: systems, technologies and applications (RFID Sys Tech), 2010 European Workshop on, pp 1–8

12. Overmeyer L, Ventz K, Falkenberg S, Krühn T (2010) Interfaced multidirectional small-scaled modules for intralogistics operations, logistics research, vol 2. Springer, Heidelberg, pp 123–133

13. Seibold Z, Stoll T, Furmans K (2013) Layout-optimized sorting of goods with decentralized controlled conveying modules. Systems Conference (SysCon), IEEE

14. Ventz K, Hachicha MB, Radosavac M, Krühn T, Overmeyer L (2012) Aufbau hochfunktionaler Intralogistik-Knoten mittels kleinskaliger Module als Cognitive Conveyor, 8. Fachkolloquium der Wissenschaftlichen Gesellschaft für Technische Logistik e.V. (WGTL), S. 19–36: Otto-von-Guericke-Universität Magdeburg. doi:10.2195/lj_Proc_ventz_de_201210_01

15. Furmans K, Seibold Z, Trenkle A, Stoll T (2014) Future requirements for small-scaled autonomous transportation systems. In: Production Environments, 7th International Scientific Symposium on Logistics Proceeding

16. Taghaboni-Dutta F, Tanchoco JMA (1995) Comparison of dynamic routing techniques for automated guided vehicle systems. Int J Prod Res 33(10). doi:10.1080/00207549508945352

17. Le-Anh T, De Koster MBM (2004) A review of design and control of automated guided vehicle systems. Eur J Oper Res 171(1):1–23. doi:10.1016/j.ejor.2005.01.036

18. Qiu L, Hsu WJ, Huang SY, Wang H (2002) Scheduling and routing algorithms for AGVs: a survey. Int J Prod Res 40(3):745–760. doi:10.1080/00207540110091712

19. Arambula Cosío F, Padilla Castañeda MA (2004) Autonomous robot navigation using adaptive potential fields. Math comput Model 40(9–10):1141. doi:10.1016/j.mcm.2004.05.001

20. Kamagaew A, Stenzel J, Nettstrater A, Hompel M (2011) Ten.:concept of cellular transport systems in facility logistics. Automation, Robotics and Applications (ICARA), 2011 5th International conference on. IEEE, Wellington, pp 40–45. ISBN: 978-1-4577-0329-4. doi:10.1109/ICARA.2011.6144853

21. Weyns D, Boucke N, Holvoet T (2006) Gradient field-based task assignment in an AGV transportation system. AAMAS '06 proceedings of the fifth international joint conference on autonomous agents and multiagent systems, pp 842–849. ISBN:1-59593-303-4. doi:10.1145/1160633.1160785

22. Uludag O (2014) GridPick: a high density puzzle based order picking system with decentralized control. Karlsruhe Institute of Technology, Auburn University, Karlsruhe

23. Seibold Z, Stoll T, Furmans K (2013) Layout-optimized sorting of goods with decentralized controlled conveying modules. In: Systems Conference (SysCon), 2013 IEEE International, pp 628–633, 978-1-4673-3107-4

24. Smolic-Rocak N, Bogdan S, Kovacic Z, Petrovic T (2010) Time Windows based dynamic routing in multi-agv systems. IEEE Trans Automat Sci Eng 7(1):151–155. doi:10.1109/TASE.2009.2016350

25. Schwarz C, Schachmanow J, Sauer J, Overmeyer O, Ullmann G (2013) Coupling of a discrete event multi agent simulation with a real time multi agent simulation. In: Proceedings of the 11th International Industrial Simulation Conference

26. Krühn T (2015) Dezentrale, verteilte Steuerungen flächiger Fördersysteme für den innerbetrieblichen Materialfluss, Berichte aus dem ITA, 01/2015, ISBN: 978-3-95900-014-7

27. Stoll T (2012) Dezentral gesteuerter Aufbau von Stetigförderern mittels autonomer Materialflusselemente, Wissenschaftliche Berichte des Instituts für Fördertechnik und Logistiksysteme des Karlsruher Instituts für Technologie; 78, KIT Scientific Publishing, Karlsruhe. ISBN: 9783866448667

28. Luke S (2015) Multiagent simulation and the MASON library, manual version 19. http://cs.gmu.edu/~eclab/projects/mason/manual.pdf