2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Behavior-based Multi-Robot Collision Avoidance

Dali Sun, Alexander Kleiner, Bernhard Nebel

Abstract— **Autonomous robot teams that simultaneously dispatch transportation tasks are playing a more and more important role in the industry. In this paper we consider the multi-robot motion planning problem in large robot teams and present a decoupled approach by combining decentralized path planning methods and swarm technologies. Instead of a central coordination, a proper behavior which is directly selected according to the context is used by the robot to keep cooperating with others and to resolve path collisions. We show experimentally that the quality of solutions and the scalability of our method are significantly better than those of conventional decoupled path planning methods. Furthermore, compared to conventional swarm approaches, our method can be widely applied in large-scale environments.**

I. INTRODUCTION

With shortening product life cycles and increasing demands for diversification of products, more flexibility and reconfigurability are needed in the field of logistics. However, fixed conveyor installations that have to be rebuilt when production lines change, are nowadays still be widely used for the material flow. Therefore, autonomous systems which can organize transportation tasks autonomously are playing an increasingly important role in today's logistic centers and manufacturing plants.

Besides the task assignment problem, i.e., allocating robots to different tasks [10], another challenge in this domain is to efficiently coordinate the simultaneous navigation of large robot teams in confined and cluttered environments. In general, multi-robot motion planning can be solved by either considering the joint configuration space of the robots [1] or by deploying decoupled techniques that separate the problems of motion planning and coordination [6]. The first approach is intractable for large robot teams because the dimension of the joint configuration space is growing exponentially with increasing number of robots. The decoupled approach is generally incomplete and yields in most cases to sub-optimal solutions.

In this paper, we propose a behavior-based multi-robot collision avoidance (BBMRCA) method in large robot teams inspired by the concept of swarm intelligence [5], [8]. Through biological research on insects, ants and birds in nature, it was found that the coordination problem in a large numbers of such animals can be very efficiently solved by using swarm behavior without any central coordination. The key point is that swarm behavior can be triggered automatically by relatively simple rules followed by individuals. Although lots of applications have been developed for robotics and video games, almost none of these perfectly fits for structured and heavily crowded environments. Combining with traditional path planning methods and swarm intelligence, our approach focuses on how to solve the problem in dynamic, structured and crowded environments. In short, a path for each robot is computed without considering any of the other robots. All robots execute their paths simultaneously. If a path collision occurs during execution time, a situation specific behavior is automatically selected that avoids that collision by issuing appropriate behaviors that might temporarily deviate from the original and optimal path. In our application behaviors are selected according to specific traffic rules that directly emerge from the context. The path will be re-planned, when the avoiding behavior has failed or a deadlock has been detected. Re-planning will also be triggered whenever the next waypoint in the path is fully blocked by obstacles, which can, for example, occur when the real environment deviates too much from the initial grid map. Note that intra-logistics and production environments cannot be considered as static since, for example, objects such as pallets and crates can be replaced.

Silver et al. introduced a decentralized approach for large robot team (WHCA) for reducing computation time [9]. However, their method is incomplete and there is no guarantee on the quality of the computed solutions. Wang et al. introduced a tractable algorithm for multi-agent path planning (MAPP) on undirected graphs [13]. Even though the algorithm is incomplete in the general case, it provides formal completeness guarantees on a class of so-called *slidable* problems. Luna et al. proposed a fast algorithm that can provide completeness guarantees for a general class of problems without any assumptions about the graphs topology [7]. Although the solution quality is noticeably better when compared to WHCA, there are still no solution quality guarantees. Rubenstein et al. presented a low-cost robot (Kilobot) to provide a testing platform for research on swarm algorithms [8]. They demonstrated some popular swarm behaviors with 29 Kilobots. Kushleyev et al. described the architecture and algorithms to coordinate a team of quad-rotors in known three-dimensional environments [5]. They demonstrated with 20 quad-rotors how a team perform a tight formation flight and how four groups of four quad-rotors fly through a window. Kiva System is widely recognized as a whole solution concerned with fulfilling orders in a distribution environment, however Kiva System can neither be integrated into existing company infrastructure nor operate in the presence of humans.

The reminder of this paper is organized as follows. In Section II the problem is formally described and in Section III a description of the target system is provided. In

D. Sun and B. Nebel are with the Department of Computer Science, University of Freiburg, Germany

A. Kleiner is with the Department of Computer and Information Science, Linköping University, Sweden

Section IV the method for collision avoidance is described. In Section V results from experiments are presented and we finally conclude in Section VI. Additionally, the video attachment "bbmrca.mpg" illustrates this method and helps the people better understand the concept.

II. PROBLEM FORMULATION

We consider the problem of coordinating the execution of delivery tasks by a team of autonomous robots, e.g., the transportation of crates containing goods, between a set of fixed stations $\mathcal{S}$. For each delivery task $d^{kl} \in \mathcal{D}(t)$ a robot has to be assigned to finalize the delivery by transporting the corresponding crate from station $k \in \mathcal{S}$ to station $l \in \mathcal{S}$. We assume that the assignment problem has been solved (e.g. by DHHT as shown in our previous work [10]), and hence restrict our attention to the problem of solving the multiple robot motion planning problem as defined in the following. Let $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ be the set of n robots navigating simultaneously on a two-dimensional grid map. During planning, each robot has a start configuration $s_i \in \mathcal{C}_{free}$ and a goal configuration $g_i \in \mathcal{C}_{free}$, where $\mathcal{C}_{free}$ is the subset of configurations robots can take on without colliding with static obstacles. For each robot $R_i \in \mathcal{R}$ a path $\pi_i : [0, T_i] \rightarrow \mathcal{C}_{free}$ will be computed by executing the A* algorithm on a grid map with obstacles grown according to the outer radius of the robot footprint. Note that T_i denotes the individual path length of robot R_i. If R_i does not have any potential collision with other robots while following path π_i, T_i is equal to the final path length. In case of a path collision between R_i and R_j, R_i or R_j has to take a detour or hold its position to avoid the collision. We denote by Π_{ij} the increased cost (in our implementation increased time) for solving a collision between two robots R_i and R_j either due to the taken detour or induced wait steps. The problem is to develop a decoupled collision avoidance system to minimize the overall costs for avoiding collisions during the entire transportation process of all robots that is given by

$$\min \sum_{(i,j)} \Pi_{ij}.$$

III. SYSTEM OVERVIEW

Our system is based on the KARIS (*Kleinskalige Autonomes Redundantes Intralogistiksystem*) [3] platform (see Figure 1) developed by a joint effort of several companies and universities of the "Intralogistic Network" in south Germany. The long-term goal of this project is to deploy hundreds of these elements to solve tasks such as autonomously organizing the material flow between stations in intra-logistics and production environments. The element has a size of 50×50 cm, a payload of 60 kg, and is capable of recharging its batteries via contact-less rechargers let into the ground. Each element is equipped with a holonomic drive to facilitate docking behaviors and a conveyor for loading and unloading crates when docked with a loading station. The conveyor has an integrated RFID reader that allows to directly read out from the crate the ID of the destination station when the crate is placed on the conveyor.

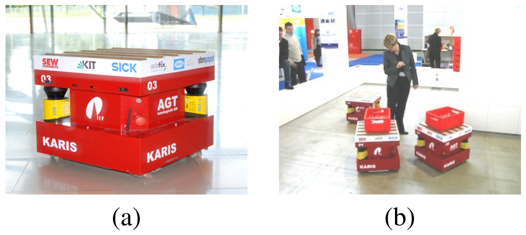

Fig. 1. (a) The KARIS element. (b) Safe navigation among humans.

For the purpose of autonomous navigation, each element is equipped with two SICK S300 laser range finders (LRFs) mounted in two opposing corners, wheel odometry, and an inertial measurement unit (IMU). Navigation is based on grid maps, which are generated from data collected by once steering a single robot manually through the environment. We use Monte-Carlo localization [2] with wheel odometry, IMU, and range readings from the two LRFs for localizing robots on the grid map. Furthermore, the typical hybrid architecture is deployed consisting of two components, which are a deliberative planning layer based on the grid map and a reactive safety layer based on the LRF data directly. Figure 1 (b) depicts the demonstration of the system during the *Logimat* fair in Stuttgart where it was deployed to deliver cups with freshly made coffee and to collect cups that have been used by the visitors.

IV. BEHAVIOR-BASED COLLISION AVOIDANCE

As state above, we are assuming that the job assignment problem to be solved (e.g. by DHHT [10]) and each robot $R_i \in \mathcal{R}$ gets a job assignment. For each robot a path will be computed with A* algorithm and the paths are executed simultaneously by the robots. During this process, robots are constantly exchanging information about their position, path, and status whenever they are within direct communication range. The robots are synchronized not with a global clock but messages and all communications are local (no broadcast but to neighbors only). With help of the *FrontAreaCheck* function, each robot continuously checks whether other robots are situated within its way. The front area is defined by a rectangle as showed in Figure 2. The origin of this area is the current position of the robot. The orientation of the front area depends on the current position of the robot and the next waypoint from the path to be reached, whereas the current orientation of the robot is ignored. In other words, if the robot rotates at its current location, the orientation of the front area will not change. The critical area is located in the front area and corresponds to the drive channel needed by the robot.

The closest robot to the origin in the front area will be selected by the front area check and the robot in the critical area will be preferably selected. That means, if any robot is located in the critical area, the robots which are not in the critical area will be ignored.

A. Definition of behaviors

To solve the multi-robot coordination problem, we defined the following 8 behaviors.

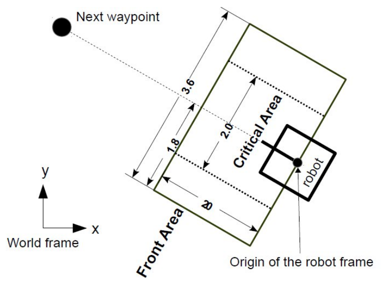

Fig. 2. Definition of the front area and the critical area.

1) *FollowWayPoint:* The robot follows the next waypoint one by one in the path until the robot reaches the goal station. After loading or unloading a crate at the station, the robot will plan a new path to new goal and execute FollowWayPoint again.
2) *Avoid:* The robot drives round the other robot which is its partner, or an unknown obstacle which is considered as its partner. When the partner is not in the front Area any more, this behavior is then completed and the robot will execute FollowWayPoint again.
3) *Exchange:* For a head-on collision, two robots pass each other and they are each others partners. When the partner is not in the front area any more, this behavior is completed and the FollowWayPoint behavior will be started again.
4) *GoThrough:* For a side collision, this robot is going through the intersection, the other has to wait until this robot passed. After passing through the intersection, the behavior is completed and the FollowWayPoint behavior will be executed again.
5) *Dock:* The robot reached the docking region of a station and starts to dock at the station.
6) *WaitKeepDistance:* The robot wait for a partner and keeps a certain distance to the partner. When the partner leaves its front area, the behavior is completed and the FollowWayPoint behavior will be executed again.
7) *WaitForGoThrough:* When the partner is just doing GoThrough behavior, this robot must wait and if necessary, make the intersection free for the partner to go through. If the partner stop his behavior or leaves its critical area, the robot will change to do FollowWayPoint again.
8) *WaitForDocking:* The robot must wait when another robot is just docking at the same station.

The FollowWayPoint behavior uses the data from the laser range finder (LRF) in order to avoid the unexpected and unknown obstacles. The LRF data is also used in addition to robot position data by the behaviors *Avoid* and *Exchange*.

In this paper we focus on intra-logistics problems and therefore particular behaviors are needed for handling the docking at stations. When a robot is docking at a station and another robot arrives nearby, the latter should stop at a certain distance from the station and wait until the former leaves. When no robot is docking at the station, the robot should check whether other robots besides arrived and are ready to dock at the same time. The robot first comes first serves. In case of the same arrival time, the robot with the higher priority will dock at first. The priorities are defined according to the robot IDs, which are unique in our system. Only one robot can dock at a station at a time.

B. Definition of the traffic rules

Like the FollowWayPoint behavior, the robot should continuously check the front area for a new possible collision in executing other behaviors. Once a new possible collision is found in the front area, the robot will select a new behavior using the traffic rules. The traffic rules for each behavior are defined as following.

1) FollowWayPoint: Algorithm 1.
2) Avoid: Algorithm 2.
3) Exchange: Algorithm 2.
4) GoThrough: Algorithm 3.
5) Dock: When a path collision happens, the robot will keep moving towards the goal instead of selecting a new behavior. In this case, the other robots will automatically give way.
6) WaitKeepDistance: Algorithm 4.
7) WaitForGoThrough: Algorithm 4.
8) WaitForDocking: In stead of changing the behavior, the robot will move back or move into a less crowded area when another robot is too close to it. However, the robot will stop WaitForDocking and execute FollowWayPoint again when it is too far away from the station because of the avoiding movement.

Algorithm 1: SolveFollowWayPointCollision

```
Data: the robot in the front area
Result: select a new behavior
begin
    switch behavior of the robot do
        case $FollowWayPoint \vee Avoid \vee Exchange \vee GoThrough$
            WaitKeepDistance;
        case $Dock$
            if $pathCollision$ then
                Avoid;
            else
                WaitKeepDistance;
        case $WaitKeepDistance \vee WaitForGoThrough \vee WaitForDocking$
            SolveActiveAndWaitCollision(robot);
end
```

The path collision consists of rear-end collision, head-on collision and side collision. The function *SolveActiveAndWaitCollision* is used to decide whether to circumnavigate another robot or not. In case other robots have the same goal station or are also within the critical area, the circumnavigation has to be delayed. Otherwise, if no other robots are within the front area doing an active behavior, the circumnavigation can proceed.

```
Algorithm 2: SolveAvoidCollision
Data: the robot the front area
Result: select a new behavior
begin
    switch behavior of the Robot do
        case $FollowWayPoint \vee Avoid \vee Exchange$
            if rearEndCollision then
                WaitKeepDistance;
            else if $(headOnCollision \vee sideCollision) \wedge lowOrder(robot)$ then
                WaitKeepDistance;
        case GoThrough
            if $pathCollision$ then
                WaitKeepDistance;
        case Dock
            if $pathCollision$ then
                Avoid;
            else
                WaitKeepDistance;
        case $WaitKeepDistance \vee WaitForGoThrough \vee WaitForDocking$
            SolveActiveAndWaitCollision(robot);
end
```

The function *lowOrder* determines the priority of robots. The robot with a shorter distance to the last goal station has a higher priority. In case of the same distance, the priority is determined by the IDs. The idea behind this function is that if a station is crowded by many robots, the robot closest to the station should first start to leave the crowded region.

The function *noOtherActive* is used to check whether other robots in the front area are executing an active behavior. The idea behind this function is to prevent additional possible collisions when other robots in the same area are already engaged in avoiding collisions.

C. Deadlock and Re-plan

In case of a failure to execute the behaviors, the robot will plan a new path to the goal. For this purpose, a virtual wall will be built between the current position of the robot and its next waypoint in the search space expanded by A*. In this way, A* will avoid using the original waypoint in the path and find a new path. The failure of different behaviors is defined as follows:

1) *FollowWayPoint:* If the robot found an unexpected and unknown obstacles that blocks the entire front area, but no other robots are located in the front area.
2) *Avoid:* If the robot can't find any free space in the front area to execute this behavior and there are no other robots except its partner in the clear area.
3) *Exchange:* If both the robot and its partner can't find free space to execute this behavior. The robot with lower ID will re-plan its path and the robot with higher ID will change to WaitKeepDistance behavior.
4) *GoThrough:* The execution time exceeds 30.0 sec.

```
Algorithm 3: SolveGoThroughCollision
Data: the robot in the front area
Result: select a new behavior
begin
    switch behavior of the Robot do
        case $FollowWayPoint \vee Avoid \vee Exchange$
            if rearEndCollision then
                WaitKeepDistance;
        case GoThrough
            if rearEndCollision then
                WaitKeepDistance;
            else if $(headOnCollision \vee sideCollision) \wedge lowOrder(robot)$ then
                WaitKeepDistance;
        case Dock
            if $pathCollision$ then
                Avoid;
            else
                WaitKeepDistance;
        case $WaitKeepDistance \vee WaitForGoThrough \vee WaitForDocking$
            SolveActiveAndWaitCollision(robot);
end
```

```
Algorithm 4: SolveWaitKeepDistanceCollision
Data: the robot in the front area
Result: select a new behavior
begin
    switch behavior of the robot do
        case $Avoid \vee Exchange \vee GoThrough$
            if Avoid me then
                Cooperate with the robot with a proper
                active behavior;
        case Dock
            if pathCollision then
                Avoid;
        case $WaitKeepDistance \vee WaitForGoThrough \vee WaitForDocking$
            if Wait for me then /* Deadlock */
                if lowOrder(robot) then
                    Drive around the robot with a proper
                    active behavior;
            else
                if $\neg sameGoalStation \wedge \neg inCriticalArea \wedge noOtherActive(robot)$ then
                    Drive around the robot with a proper
                    active behavior;
end
```

It is possible that robots run into deadlocks by waiting for each within circular chain. The problem with two robots is solved by algorithm 4. However, the problem with more than two robots can't be solve with the traffic rules only. Hence, a deadlock detection mechanism is used for automatically detecting this problem. Each robot that executes a "Wait" behavior sends a so-called "WaitMessage" at a certain frequency to it's partner (we used 2 second intervals during our experiments). The ID of this robot is added to the "WaitMessage". If the partner is not doing a Wait behavior, it will ignore this "WaitMessage". Otherwise, the partner will first check whether the "WaitMessage" already contains the ID information about itself. If that is the case, a deadlock will be detected and the partner has to plan an alternative path. If not, the partner will add its own ID to the "WaitMessage" and send the message to the robot which the partner is waiting for.

Now we discuss the method's capability in avoiding deadlocks. Outside of the docking region, the path collision within a certain group of robots executing *FollowWayPoint*, *Avoid*, *Exchange*, and *GoThrough* behavior will be solved by the traffic rules. Only one robot can execute the *GoThrough* behavior while the other robots have to wait. Inside of the docking region, the *Dock* behavior dominates over all other active behaviors. On the assumption that there is no overlap between the docking regions of any two stations, it is impossible that two robots are doing the *Dock* behavior in the same region and at the same time. Therefore, a cycling of the active behaviors within the same group of robots cannot occur. This will ensure that at least one robot in this group will keep moving toward its goal.

All active behaviors except the *Dock* behavior can degrade to the *Wait* behavior. In this case the robots can build a cyclic dependency and wait for each other. This problem is solved by the deadlock detection and re-planing method as described above. If the robot that executed the re-planning can always find an alternative path from any position to any station, then at least one robot will be released from the Deadlock and moving towards its goal.

V. EXPERIMENTAL RESULTS

For evaluating our approach we conducted a series of experiments in three different environments. Figure 3 depicts those three environments used for testing our approach. The size of the Building_78 map is 33m x 30m, the ASE map is 94m x 82m, and the KNO map is 88m x 43m. On each map we defined several loading stations (green rectangle): 4 stations on Building_78, 9 on ASE, and 6 on KNO.

We modified and used the *Stage* software library [12] for simulating large robot teams. One advantage of the Stage is that it allows us to build simulation worlds directly from grid maps, which in our case were generated from real environments.

A comparison between the Adaptive Road Map Optimization (ARMO), introduced in our previous work [4], and the decoupled technique based on prioritized planning (PRIOR) from Berg and colleagues [11], will be presented in the following sections. Based on linear programming, ARMO

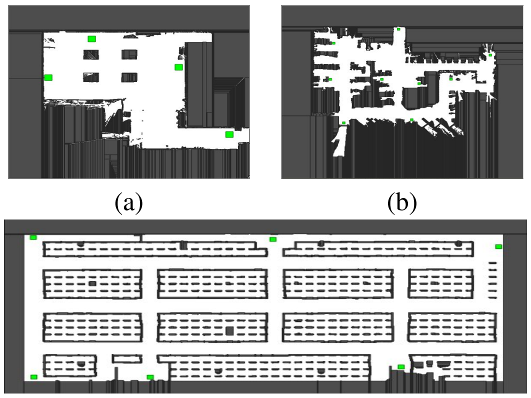

Fig. 3. Snapshot pictures captured from Stage utilized for experiments: (a) the *Building_78* map generated in robot experimental area of University of Freiburg, (b) the *ASE* map generated in a real logistic center. (c) *KNO* map generated in a large distribution center.

computes an optimal road map according to environmental constrains and the demand for transportation tasks from stations. After each robot planed the path based on the road map, the path is executed by the navigation module. In decoupled prioritized planning, the path is planned iteratively after a pro-defined priority scheme. When the ith robot plans its path, then the paths of the $i-1, i-2, \ldots, i-k$ robot that were planned beforehand are considered as dynamic obstacles. In order to guarantee completeness with decoupled planners, it is required that start and goal locations of each robot are so-called garage configurations, i.e., configurations that are not part of $\mathcal{C}_{free}$ of any other robot.

Under the same settings several experiments with the same number and identical sequence of delivery tasks (#Del.) were conducted for each planning method. The maximum allowed velocity of the robot was set to 0.8 m/s for all three methods. The completion times (CTime) was measured in seconds, which mainly reflects the efficiency of the different methods. In addition, the average velocity over all robots (avg. v) was also measured in meter per second. The average velocity not only provide a different perspective to the efficiency, but also indicates the energy consumption of robots. Normally, the moving needs more energy than the standing; in other words, lower velocity means less energy consumption at the same period. All experiments were tested with a machine configured with Intel(R) Core i7-3770 (3.4 GHz) processor, 8 GB main memory and Ubuntu 12.04 64-Bit version.

Building_78				
#R	M	#Del.	avg. v (m/s)	CTime (s)
20	ARMO	144	0.23	1416
	PRIOR	144	0.13	4544
	BBMRCA	144	0.12	2560
40	ARMO	216	0.12	2144
	PRIOR	216	0.09	9144
	BBMRCA	216	0.09	3889

TABLE I
EXPERIMENT RESULTS OF MAP BUILDING_78.

Table I shows the results of experiments on the map "Building_78". The free space of this environment is rela-

Map Name ASE				
#R	M	#Del.	avg. v (m/s)	CTime(s)
20	ARMO	196	0.35	2438
	PRIOR	196	0.24	4636
	BBMRCA	196	0.31	2401
50	ARMO	294	0.20	2732
	PRIOR	294	0.15	8229
	BBMRCA	294	0.20	3250
100	ARMO	294	0.11	2826
	PRIOR	294	0.14	10206
	BBMRCA	294	0.16	2685

TABLE II
EXPERIMENT RESULTS OF MAP ASE.

KNO				
#R	M	#Del.	avg. v (m/s)	CTime(s)
20	ARMO	112	0.40	1746
	PRIOR	112	0.25	4184
	BBMRCA	112	0.36	1902
50	ARMO	196	0.29	1743
	PRIOR	196	0.16	8131
	BBMRCA	196	0.25	2604
100	ARMO	252	0.16	2114
	PRIOR	252	0.14	17977
	BBMRCA	252	0.19	2612

TABLE III
EXPERIMENT RESULTS OF MAP KNO.

tively small, particularly when it is crowded with 20 to 40 robots. Therefore, it is not surprising that ARMO had the best results. However, ARMO requires much more time for computing the optimal road map. Moreover, since the robots can only move on the road map, another weakness of ARMO is that it cannot efficiently deal with the situation of dynamic changes of the map. Compared to ARMO, BBMRCA is more flexible. Furthermore, with almost same energy consumption, the efficiency of BBMRCA is nearly double of PRIOR (44% with 20 robots and 57% with 40 robots); That means, BBMRCA is more energy efficiency than PRIOR.

As is shown in Table II for the map "ASE", the performance of BBMRCA is nearly as well as that of ARMO and in some cases (with 20 robots and 100 robots) even better than ARMO. One reason for this is that ARMO generates only orthogonal road map and in this environment, the orthogonal lines may not the shortest path between stations. Compared to PRIOR, BBMRCA has a significant better performance and the results have shown up to 70% improvement. Additionally, BBMRCA is better scalable by varying the number of the robots.

The results of experiments on the map "KNO" are shown in Table III. Although the drivable space of this environment is very limited, the performance of our method is at the same level as ARMO. The average velocities of ARMO and BBMRCA were also nearly identical. Furthermore, by comparisons with PRIOR, the efficiency of BBMRCA had led to an improvement up to 85%. And as can be seen in this table, the completion time of PRIOR was raised significantly with increasing number of robots, while the number of robots had hardly any impact on the efficiency of BBMRCA.

VI. CONCLUSION

We proposed a behavior-based multi-robot collision avoidance in large robot teams. Experimental results have shown that the quality of solutions and the scalability are at same level as ARMO and significant better than those of conventional decoupled prioritized planning methods. The main advantage of BBMRCA compared to ARMO is that BBMRCA doesn't need any pre-processing such as computing the optimal road-map, which requires much time for large and complex environments. BBMRCA is as flexible as PRIOR, when an unexpected and unknown obstacles appears. Compared to common swarm technologies, our method is more suitable in complicated environments, such as confined and cluttered logistic centers and manufacturing plants.

Although at present our method is specially used to implement a solution to the intra-logistic problems, in the future, it can also be adopted to solve other multi-agent motion planning problems in similar scenarios, for example, for distributed agents in video games. Furthermore, the experiment on the real KARIS system will be planned as the next step, when a large number of real KARIS robots are available to us.

ACKNOWLEDGMENT

This work has been funded by a BMBF grant 02PJ2667 as part of the KARIS PRO project, and supported by ELLIIT, the Strategic Area for ICT research, funded by the Swedish Government.

REFERENCES

[1] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *International journal of robotics research*, 10:628–649, 1991.
[2] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.
[3] H. Hippenmeyer, K. Furmans, T. Stoll, and F. Schönung. Ein neuartiges Element für zukünftige Materialflusssysteme. *Hebezeuge Fördermittel: Fachzeitschrift für Technische Logistik*, (6), 2009.
[4] A. Kleiner, D. Sun, and D. Meyer-Delius. Armo: Adaptive road map optimization for large robot teams. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, pages 3276–3282, San Francisco, California, 2011.
[5] Alex Kushleyev, Daniel Mellinger, and Vijay Kumar. Towards a swarm of agile micro quadrotors. *Robotics: Science and Systems*, July 2012.
[6] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.
[7] Ryan Luna and Kostas E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In Toby Walsh, editor, *IJCAI*, pages 294–300. IJCAI/AAAI, 2011.
[8] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *ICRA*, pages 3293–3298. IEEE, 2012.
[9] David Silver. Cooperative pathfinding. In R. Michael Young and John E. Laird, editors, *AIIDE*, pages 117–122. AAAI Press, 2005.
[10] D. Sun, A. Kleiner, and C. Schindelhauer. Decentralized hash tables for mobile robot teams solving intra-logistics tasks. In *Proc. of the 9th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 923–930, Toronto, Canada, 2010.
[11] J.P. van den Berg and M.H. Overmars. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, pages 430–435, 2005.
[12] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2):189–208, 2008.
[13] Ko-Hsin Cindy Wang and Adi Botea. Tractable multi-agent path planning on grid maps. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1870–1875, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.