Decentralized Collision Avoidance, Deadlock Detection, and Deadlock Resolution for Multiple Mobile Robots

Markus Jäger
Corporate Technology
Siemens AG
81739 Munich, Germany
markus.jaeger@mchp.siemens.de

Bernhard Nebel
Institut für Informatik
Albert-Ludwigs-Universität
79100 Freiburg, Germany
nebel@informatik.uni-freiburg.de

Abstract

This paper describes a method for coordinating the independently planned trajectories of multiple mobile robots to avoid collisions and deadlocks among them.

Whenever the distance between two robots drops below a certain value, they exchange information about their planned trajectories and determine whether they are in danger of a collision. If a possible collision is detected, they monitor their movements and, if necessary, insert idle times between certain segments of their trajectories in order to avoid the collision.

Deadlocks among two or more robots occur if a number of robots block each other in a way such that none of them is able to continue along its trajectory without causing a collision. These deadlocks are reliably detected. After a deadlock is detected, the trajectory planners of each of the involved robots are successively asked to plan an alternative trajectory until the deadlock is resolved.

We use a combination of three fully distributed algorithms to reliably solve the task. They do not use any global synchronization and do not interfere with each other.

1 Introduction

Whenever multiple mobile robots share the same workspace, the potential for collisions among them must be taken into account. This can be done by using a centralized component to plan collision free trajectories of all the robots simultaneously [1] or by planning the trajectories of all the robots independently and using a centralized component to coordinate these trajectories, so that no collision is possible [2, 3].

Centralized approaches, however, have the disadvantage that they are computationally demanding, inflexible and presuppose that there is a global communication network. We therefore omit centralized components and, in contrast to centralized approaches, achieve global coordination by distributed algorithms and assume only local communication between pairs of physically close robots. This permits less demanding communication frameworks and allows easier and more adaptive coordination between the robots.

Whenever the distance between two robots drops below a certain value they exchange information about their planned trajectories and determine whether they are in danger of a collision. If a possible collision is detected, they monitor their movements and, if necessary, insert idle times between certain segments of their trajectories in order to avoid the collision.

Deadlocks among two or more robots occur if a number of robots block each other in a way such that none of them is able to continue along its trajectory without causing a collision. These deadlocks are reliably detected. After a deadlock is detected, the trajectory planners of each of the involved robots are successively asked to plan an alternative trajectory until the deadlock is resolved.

We use a combination of three fully distributed algorithms to reliably solve the task. They do not use any global synchronization and do not interfere with each other.

This paper does not deal with the trajectory planning itself. It concentrates on the coordination methods for the collision avoidance, the deadlock detection, and the deadlock resolution. The methods provided in the paper can therefore be seen as an intermediate layer between the trajectory planning layer and the trajectory execution layer, as shown in Figure 1.

Some related work, which also concentrates on independent planning of the trajectories, is summarized in the next section. Sections 3 to 6 describe the coordination. They focus on general assumptions, collision avoidance, deadlock detection, and deadlock resolution, respectively. The last two sections provide some simulation results and give a summary, draw some conclusions and mention some future work.

2 Related Work

The work dealing with independently planned trajectories differs mainly in the strategies which are used to avoid collisions and the strategies to deal with deadlocks.

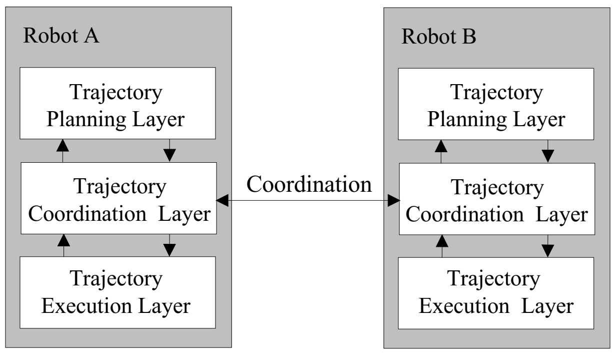
Robot A
Trajectory Planning Layer
Trajectory Coordination Layer
Trajectory Execution Layer
Coordination
Robot B
Trajectory Planning Layer
Trajectory Coordination Layer
Trajectory Execution Layer

Figure 1: Each robot in the system consists of three layers. The task of the trajectory planning layer is to plan a trajectory for a certain robot, the trajectory coordination layer is responsible for coordinating the independently planned trajectories, and the trajectory execution layer moves the robot along its trajectory.

Aguilar *et al.*[4] plan the trajectories independently, but use a centralized mechanism to merge the trajectories, so that no collisions and deadlocks can occur. The centralized mechanism requires that a robot is able to send broadcast messages to all other robots, leading to high demands on communication abilities of the robots.

Kato *et al.*[5], in contrast, demand no communication abilities of the robots at all. They use traffic rules which, if obeyed by all robots, ensure collision and deadlock free operation of the whole system. They, however, make some very special assumptions about the environment.

Chun *et al.*[6] also do not demand communication abilities at all, but require that the sensors of a robot can detect other robots within a certain range. Whenever a robot detects another robot it computes the probable collision point and replans its trajectory, in order to avoid it. Since the replanning is done on a local basis, it might lead to deadlocks, which are neither detected nor resolved.

The problem of avoiding collisions among two robots was discussed by Kant and Zucker [7], O'Donnell and Lozano-Periz [8], and Lee *et al.*[9]. They solve the problem by inserting idle times between trajectory segments. This approach has largely influenced our collision avoidance strategy.

Wang and Premvuti [10, 11] use a deadlock detection algorithm which requires communication only among a robot and its immediate neighbors. Our deadlock detection strategy is similar to theirs, but our deadlock resolution strategy differs. They assume, that each passage in the environment is unidirectional and that the environment contains *buffering-areas* at each intersection. Since a robot, which is involved in a deadlock, can use a *buffering-area* to dodge, it is possible to resolve deadlocks without replanning trajectories.

3 General Assumptions

This section presents some general assumptions made in the rest of the paper.

- Each robot is able to communicate with all other robots which are within a certain range r_c.
- If two robots are able to communicate, they regularly exchange information about their current position.
- The deviation between a computed distance of two robots, based on position information, and the real distance is assumed to be less than δ_d.
- The distance between two robots must never be less than d_{min}, since this means a probable collision.
- A planned trajectory is represented as a sequence of trajectory segments.

4 Collision Avoidance

When the distance between two robots is larger than a certain distance d_{safe} they are considered to be safe, i.e. not facing a possible collision in the near future. Each robot evaluates position information received from other robots to determine its distance to them.

When the position evaluation of a robot determines that its distance d_{rob} to another robot might be less than d_{safe}, i.e. $d_{rob} - \delta_d < d_{safe}$, it instantiates a coordination link to the other robot. The coordination link is only removed after the distance of the robots is safe again. Since for the maintenance of the coordination links communication is needed, the equation $d_{safe} + \delta_d < r_c$ must hold.

The task of a coordination link is to coordinate the movements of two robots along their trajectories, so that no collision is possible. The interaction between a robot and a coordination link is very simple. The link either gives permission to the robot to move on or not. A robot has to ask each of its coordination links for permission before it is allowed to give the next segment of its trajectory to its trajectory execution layer.

To establish a coordination link between two robots, one robot is elected coordinator and the other robot becomes partner. After the coordinator is elected, he requests the trajectory segments from his partner. Based on his own and the partners trajectory, the coordinator determines a schedule for the two robots. The schedule consists of the following entries: both robots are allowed to move, only the coordinator is allowed to move, and only the partner is allowed to move. Using the schedule, the coordinator then gives permissions to himself and his partner.

The determination of the schedule is based on work of Kant and Zucker [7] and O'Donnell and Lozano-Periz [8]. At first, a so-called *task-completion-diagram (TCD)* is constructed, see Figure 2. The trajectory segments of the coordinator are shown on the horizontal axis and the segments

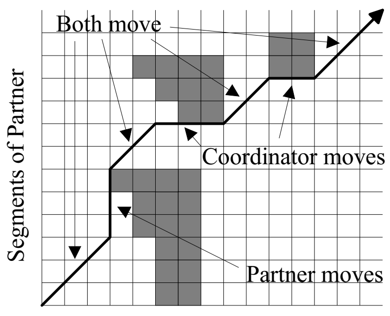

Figure 2: This figure shows a *task-completion-diagram (TCD)* with collision regions and an execution path from the lower left to the upper right corner.

of the partner on the vertical. The sizes of the segments in the diagram are shown as constant, independent of the execution time of the segments. The current position of the robots is in the lower left corner and the end of the trajectories is in the upper right corner.

The gray areas in the diagram correspond to collision regions. They are obtained by checking whether a certain segment of the coordinator and a certain segment of the partner lead to a collision, i.e. checking whether any two positions of the robots on the segments would lead to a distance of the robots less than d_{min}. If so, the area in the diagram which corresponds to the segments is marked gray. Collision checking has to be done for all combinations of coordinator and partner segments.

After the *TCD* is completed, an execution path which avoids all gray regions starting at the lower left corner and going to the upper right corner is constructed. If this is not possible, a special entry meaning that none of the robots is allowed to move is added to the schedule.

The execution path is finally used to construct the schedule. Starting from the lower left corner, the path is cut into straight pieces and for each piece entries are added to the schedule. For each horizontal piece an entry meaning that the coordinator is allowed to move, for each vertical piece an entry saying that the partner is allowed to move, and for each sloped piece an entry denoting that both robots are allowed to move.

Giving only one robot the permission to execute its next trajectory segment means that one robot has to wait until the other robot has finished its trajectory segment. This corresponds to inserting idle times between trajectory segments of the waiting robot.

As it can be seen in Figure 2, many different execution paths which avoid all gray regions starting at the lower left corner and going to the upper right corner could be constructed for a certain *TCD*. The algorithm we use determines the shortest possible execution path [9], since this path causes the least overall delay of the robots. The algorithm furthermore has the property that it finds a solution, i.e. an execution path, if one exists.

The different execution times of the trajectory segments and unforseen delays of one robot along its segments should also be taken into account when the execution path is constructed [7, 8, 9].

As described above, a coordination link is only used to coordinate the movement of two robots. The coordination of more than two robots results from the fact that a robot can have more than one coordination link. Since a robot needs the permission from all its coordination links before it is allowed to move on, the coordination links connect all the involved robots in a global coordination structure. This global coordination structure, however, can contain deadlocks - some robots mutually wait for each other - which have to be detected and resolved, see Section 5 and 6.

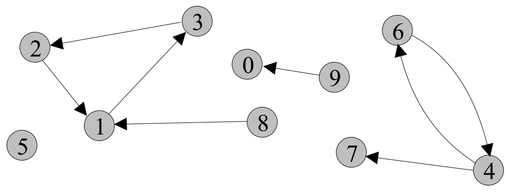

Figure 3: The nodes of the graph shown in this figure correspond to robots and a directed edge between two nodes is indicating that one robot is waiting for another one. The node from which an edge is originating corresponds to the waiting robot.

5 Deadlock Detection

Deadlocks can only occur if a coordination link forces a robot to stop, i.e. if a coordination link does not give the permission to process the next trajectory segment. This occurs when the robot has to wait until the other robot has completed some of its trajectory segments. Whenever a robot is forced to wait for another one, a deadlock detection is initiated.

The case in which one robot has to wait for another can be seen as a directed edge from the waiting robot to the other robot in a graph in which every robot is represented by a graph node. Such a graph is depicted in Figure 3. Here, for example, robot 9 has to wait for robot 0. The case in which the construction of the execution path fails and in which none of the two robots is given the permission to move on, is represented as two edges connecting the two robots and pointing in opposite directions. Figure 3 shows this for robot 4 and 6.

In the context of the graph, a deadlock among the robots exists if and only if the graph contains a circle. Therefore deadlock detection means to detect a circle in the graph. Since this is a common problem in distributed deadlock detection, a lot of algorithms for that purpose have been developed. We basically use the algorithm proposed by

Chandy *et al.*[12], which only requires that each robot knows its outgoing edges to detect deadlocks. The algorithm works as follows:

- When a new directed edge is established the robot from which the edge originates initiates a new *probe* message and sends it along the edge.
- Whenever a robot receives a *probe* message it forwards the *probe* message along all its outgoing edges. The forwarded *probe* message contains the id of the visited robot so that deadlock cycles can be identified.
- If a robot receives a *probe* message which itself has initiated, it knows that the *probe* must have been forwarded along a circle and therefore detects a deadlock. Since the traveled path of a probe is stored in the probe, the cycle causing the deadlock is known at the time when the deadlock is detected.

A disadvantage of this algorithm is that it is possible that more than one robot detect the same deadlock, e.g. if two robots initiate a deadlock detection for the same circle at the same time. Sinha and Natarajan [13] have proposed an improvement of the algorithm which ensures that each deadlock is only detected by exactly one robot. We do not use this improvement, since our deadlock resolution strategy is able to cope with multiple detections of the same deadlock, resulting in multiple deadlock resolutions operating on the same edges.

Using the improvement would not simplify the deadlock resolution strategy, since multiple deadlock resolutions which operate on the same edges can also occur if the graph contains multiple cycles which have nodes and edges in common.

The algorithm described above ensures that each deadlock is detected by at least one robot. It is however possible that phantom deadlocks are detected. This means, that a deadlock is detected which has already been resolved. This can happen if the deadlock has edges in common with another deadlock and if the deadlock resolution of the other deadlock modifies these edges. The deadlock resolution strategy is able to cope with such phantom deadlocks.

6 Deadlock Resolution

A deadlock resolution, which is initiated by the robot detecting a deadlock, works by sending messages along the deadlock circle. Whenever it is not possible to forward a message along the circle, this indicates a phantom deadlock and the deadlock resolution terminates.

A deadlock resolution comprises two steps. Initially an attempt is made to change the direction of one edge in the circle, since this would destroy the circle and therefore resolve the deadlock. If no edge direction can be changed, robots are chosen and asked to plan alternative trajectories. If none of the two steps succeeds, which is considered to be very unlikely for real applications, all robots blocked by the deadlock are informed. These robots then do not participate any more in any deadlock detection or deadlock resolution process.

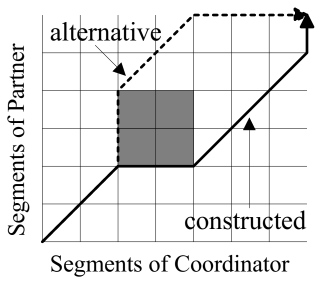

Figure 4: This figure shows a *task-completion-diagram (TCD)* with a collision regions, a constructed execution path, and an alternative execution path.

6.1 Edge Direction Change

As stated above, the direction of an edge defines which of the two robots connected by the edge has to move first. Sometimes, however, it is not important that a certain robot moves first. It is only important that both robots do not move at the same time. Figure 4 shows such a case. There are two possibilities to avoid the collision region. The solid line is the execution path constructed by the algorithm - the coordinator moves first - and the dashed line is an alternative one - the partner would move first.

A deadlock resolution is rather simple if one of the edges causing the deadlock has the above stated property. The coordinator of the coordination link to which this edge belongs must only be asked to use the alternative execution path. This results in a new schedule and a change of the edge direction. After the direction of one edge in a circle is changed, the circle does not exist any more and the deadlock is resolved.

To determine an edge whose direction can be changed and to actually change the direction, a *change* message is sent around the circle. When a robot receives a *change* message, it asks the coordinator of its outgoing edge, which belongs to the circle, to change the direction of the edge. If the coordinator is able to change the direction, the deadlock resolution was successful and is therefore terminated, i.e. the *change* message is discarded. If the *change* message travels around the whole circle, no edge direction could be changed and the second step of the deadlock resolution, described in the next section, is executed.

The change of the direction of an edge can cause new deadlocks. This is shown in Figure 5. If the deadlock which consists of the edge e and the semi-circle 1 is resolved by changing the direction of the edge e, a new deadlock consisting of the edge e and the semi-circle 2 is caused. If one of the new deadlock is resolved by again

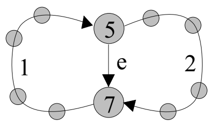

Figure 5: If the direction of the edge e is changed, the deadlock consisting of e and 1 is resolved, but a new deadlock consisting of e and 2 is caused.

changing the direction of the edge, the original deadlock is reestablished. This might lead to an oscillation of the edge direction.

Proposition 6.1: An oscillation does not continue forever.
Proof: If the change of the direction of an edge causes a new deadlock, the graph must contain an outer circle containing all the nodes of the original and the caused deadlock. Figure 5 shows such an outer circle consisting of the semi-circles 1 and 2. If the deadlock resolution of the outer circle breaks the circle, which means that either the semi-circle 1 or the semi-circle 2 is broken, the oscillation stops. If the outer circle cannot be broken, this is detected after finite time and all robots involved in the deadlock are informed. Since these robots then do not participate in any deadlock detection or deadlock resolution process any more, the oscillation stops. (See also Section 6.2.) □

6.2 Planning of Alternative Trajectories

If no schedule which avoids the deadlock can be found for the robots, i.e. it is not possible to resolve a deadlock by changing an edge direction, it is necessary to plan alternative trajectories for one or more robots. The replanning has to be done in a controlled manner. It should be avoided that all robots plan alternative trajectories at the same time.

The algorithm provided here tries to keep the number of robots which are forced to plan alternative trajectories low. It comprises two steps which are iterated until the deadlock is resolved or until it is determined that the deadlock cannot be resolved by asking individual robots to plan alternative trajectories. The latter case is considered to be very unlikely for real applications and depends solely on the abilities of the trajectory planning layer and the properties of the environment.

The first step is to send a *replan* message around the circle, just as it is done with the *change* message. If a robot receives a *replan* message it asks its trajectory planning layer to plan an alternative trajectory. The robot can inform its trajectory planning layer about the positions of other robots surrounding it, i.e. other robots to which it has a coordination link, to obtain better results. After a robot was able to plan an alternative trajectory, all its outgoing edges are removed. Since this breaks the circle, the deadlock is resolved and the *replan* message can be discarded. If a robot receives a *replan* message which itself has initiated, it knows that none of the robots in the circle is able to plan

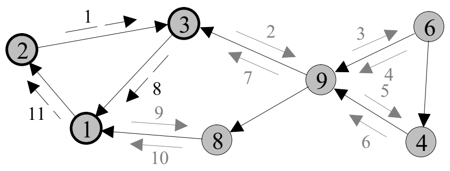

Figure 6: This figure shows a deadlock circle of the robots 1,2, and 3, and a number of other affected robots. Furthermore, it shows the path of a *free* message initiated by robot 2.

an alternative trajectory.

If the deadlock could not be resolved during the first step, this is either because it is not possible to resolve the deadlock by asking individual robots to plan an alternative trajectory or because it is temporarily not possible. The latter could be due to other robots in the neighborhood of the robots of the circles which hinder them to plan an alternative trajectory. The two cases can be distinguished by

Criteria 6.1: If none of the affected robots is able to plan an alternative trajectory and if none of the affected robots has an unaffected robot in its neighborhood, then the deadlock cannot be resolved by asking individual robots to plan alternative trajectories.

A robot is considered to be affected if it is part of the circle or if it is attached to the circle by an outgoing edge, either directly or transitive. Figure 6 shows a deadlock circle of the robots 1,2, and 3, and a number of other affected robots.

The idea of the second step now is to check Criteria 6.1 and to ask affected robots which are not part of the circle to plan alternative trajectories. Criteria 6.1 is used to stop the iteration of the two steps in hopeless cases. The affected robots which are not part of the circle are asked to plan alternative trajectories in order to free as much robots as possible from the deadlock.

During the second step a *free* message is sent to all affected robots. This is done by on the one hand sending the *free* message around the circle and on the other hand allowing the *free* message to leave the circle and traverse the affected robots in a depth-first way using incoming edges. Robots which have already received the *free* message once are omitted. Figure 6 shows the path of a *free* message initiated by robot 2, where the dashed arrows show the path around the circle and the grayed arrows show the path during traversal.

The *free* message is used to tell the robots, which are not part of the circle, to plan alternative trajectories and to collect information whether non affected robots are still in the neighborhood. If a robot receives a *free* message which itself has initiated, it knows that all the affected robots have received the message. The robot then evaluates Criteria 6.1 and stops the deadlock resolution, if the conditions to do so are met.

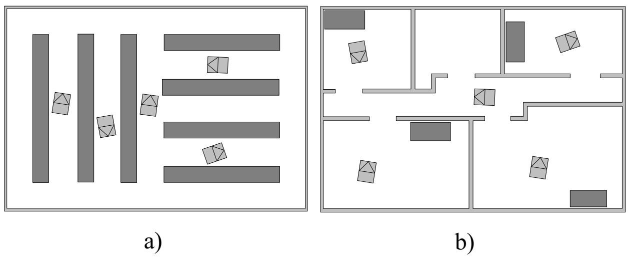
Figure 7: Figure a) shows a supermarket like environment with a lot of shelves and figure b) shows an office like environment with some desks. Each environment contains five robots.

7 Simulation Results

We implemented the described strategies and tested them in simulation. We used two different environments. A supermarket and an office like environment. They are shown in Figure 7. The supermarket consists of one huge room with a lot of obstacles, i.e. the shelves of the supermarket. The office consist of five smaller rooms with only some obstacles. The environments are 10x15m in size.

The robots we used for the simulation have a length of 1m and a width of 0.8m and are moving at a speed of 0.3m/s. Figure 7 shows some robots.

We carried out 6 simulation runs for each environment, with 1 to 6 robots respectively. Each of the 12 simulations was running for 10 minutes. During the simulations the robots followed randomly planned trajectories. The average number of coordination links a robot maintained, the number of deadlocks which occurred, and the average trajectory length of the robots were determined after each simulation.

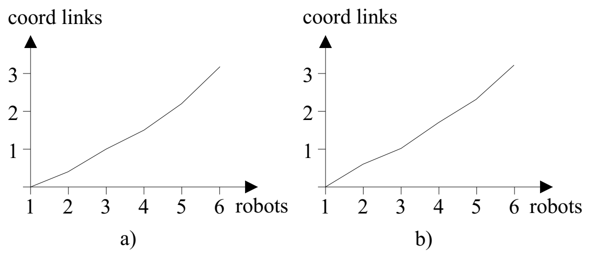

Figure 8: Average number of coordination links.

Figure 8 shows the average number of coordination links a robot maintained. Figure 8 a) for the supermarket and Figure 8 b) for the office. As it can be seen, the average number of coordination links increases almost linear with the number of robots. This is due to the fact that it mainly depends on the amount of robots in the neighborhood of a robot, which depends linear on the amount of robots used.

The number of deadlocks which occurred is depicted in Figure 9. It increases significantly when more than four robots are used. In the case of two and three robots the amount of collisions in the office like environment - Figure 9 b) - is two to three times higher than in the supermarket like environment - Figure 9 a). This is due to the fact that the office like environment contains a hot spot - the corridor connecting the rooms - which is regularly passed by the robots. The hot spot has its greatest impact when two or three robots are used since an increasing number of robots transforms the whole environment into a hot spot.

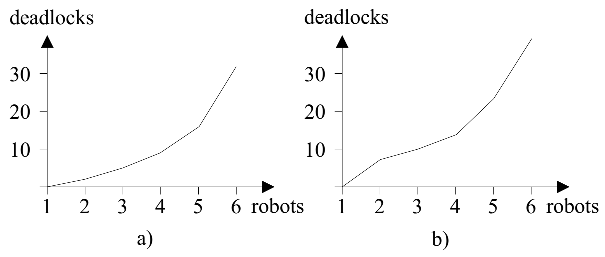

Figure 9: Number of deadlocks.

Almost all of the deadlocks had to be resolved by re-planning. Only 3% of the deadlocks could be resolved by changing the direction of an edge.

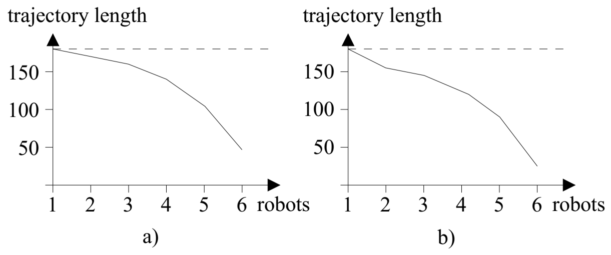

Figure 10: Average trajectory length.

Since a robot moves at a speed of 0.3m/s and a simulation is running for 10 minutes, a robot could move 180 meter at the best. The actual values, however, are lower since the robots have to wait for each other and are involved in deadlocks and deadlock resolution processes. The actual average trajectory length of the robots is shown in Figure 10. It significantly decreases if more than four robots are used.

A basic result of the simulations is that the collision avoidance, described in Section 4, is sufficient to coordinate the robots in most cases if only a few robots are used. The necessity for deadlock detection and deadlock resolution significantly increases with the number of robots. If more than four robots are used, the system is still able to resolve the deadlocks, but the overall performance, i.e. the length of the robots trajectories, decreases rapidly. Five robots in combination with the environments of the given size and structure seems to be the critical value for the coordination methods described in this paper.

8 Summary, Conclusion, and Future Work

In this paper we described a decentralized approach for collision avoidance, deadlock detection, and deadlock resolution among multiple mobile robots which follow independently planned trajectories. We introduced a combination of three fully distributed algorithms which reliably solve the task. They do not use any global synchronization, do not interfere with each other, and demand only local inter robot communication.

The global coordination of a fleet of robots is achieved by using only local coordination of pairs of robots. The idea to achieve this is to allow more than one coordination link for each robot. This links a set of robots together in a global control structure.

Deadlocks in the global coordination, which cannot be avoided when only local coordination is used, are reliably detected. The deadlocks are resolved by changing the direction of coordination links and asking robots to plan alternative trajectories. The only deadlocks which cannot be resolved are those where the trajectory planning layers of the involved robots are not able to plan alternative trajectories. This, however, depends solely on the abilities of the trajectory planning layer and the properties of the environment.

The strict separation of trajectory planning and collision avoidance/deadlock handling makes it possible to use a lot of different trajectory planners with our system. The only restriction is, that they should be able to plan alternative trajectories. There are no other special assumptions about the environment or the application made.

As a conclusion, we can say that our approach is very suitable to coordinate a number of independently moving robots through local inter robot communication in a lot of applications. Our simulation results confirm this.

For further work, we consider it interesting to investigate different strategies for the construction of the execution paths of the TCDs, see Section 4. If, for example, the execution path would be constructed in a way that situations like the one shown in Figure 4 are favored, obviously more deadlocks could be resolved by changing edge directions and less deadlocks would have to be resolved by replanning trajectories. Furthermore, different strategies for the collision avoidance could be evaluated.

We intend to use the methods described in this paper to coordinate a fleet of real robots. The task of the robots will be to cooperatively clean a large room, e.g. a large supermarket or an airport. Some field test results, concerning navigation etc., for one robot can be found in [14].

References

[1] J. Barraquand, B. Langlois, and J.-C. Latombe, Numerical Potential Field Techniques for Robot Path Planning, IEEE Trans. on System, Man, and Cybernetics, vol. 22(2), pp. 224-241, 1992

[2] S. Leroy, J. P. Laumond, and T. Simeon, Multiple Path Coordination for Mobile Robots: A Geometric Algorithm, *International Joint Conference on Artificial Intelligence (IJCAI)*, 1999

[3] M. Bennewitz and W. Burgard, Coordinating the Motions of Multiple Mobile Robots Using a Probabilistic Model. *8th International Symposium on Intelligent Robotic Systems (SIRS)*, 2000

[4] L. Aguilar, R. Alimi, S. Fleury, M. Herrb, F. Ingrand, and F. Robert, Ten Autonomous Mobile Robots (and even more) in a Route Network Like Environment, *Int. Conf. on Intelligent Robots and Systems (IROS)*, Vol. 2, pp. 260-267, 1995

[5] S. Kato, S. Nishiyama, and J. Takeno, Coordinating Mobile Robots by Applying Traffic Rules, *International Conference on Intelligent Robots and Systems (IROS)*, pp. 1535-1541, 1992

[6] L. Chun, Z. Zheng, and W. Chang, A Decentralized Approach to the Conflict-Free Motion Planning for Multiple Mobile Robots, *Int. Conf. on Robotics and Automation (ICRA)*, pp. 1544-1549, 1999

[7] K. Kant and S. W. Zucker, Towards Efficient Trajectory Planning: The Path-Velocity-Decomposition, *The International Journal of Robotics Research*, no. 5, pp. 72-89, 1986

[8] P.A. O'Donnell and T. Lozano-Periz, Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators, *Int. Conf. on Robotics and Automation (ICRA)*, pp. 484-489, 1989

[9] J. Lee, H. S. Nam, and J. Lyou, A Practical Collision-Free Trajectory Planning for Two Robot Systems, *Int. Conf. on Robotics and Automation (ICRA)*, pp. 2439-2445, 1995

[10] J. Wang and V.Premvuti, Distributed traffic regulation and control for multiple autonomous mobile robots operating in discrete space, *Int. Conf. on Robotics and Automation (ICRA)*, pp. 1619-1624, 1995

[11] J. Wang, Operating Primitives Supporting Traffic Regulation and Control of Mobile Robots under Distributed Robotic Systems, *Int. Conf. on Robotics and Automation (ICRA)*, pp. 1613-1618, 1995

[12] K. M. Chandy, J. Misra, and L. M. Haas, Distributed Deadlock Detection, *ACM Trans. on Computer Systems*, May 1983

[13] M. K. Sinha and N. Natarajan, A Priority Based Distributed Deadlock Detection Algorithm, *IEEE Trans. on Software Engineering*, Jan. 1985

[14] H. Endres, W. Feiten, and G. Lawitzky, Field Test of a Navigation System: Autonomous Cleaning in Supermarkets, *Int. Conf. on Robotics and Automation (ICRA)*, pp. 1779–1781, 1998