Universität Stuttgart
Fakultät Informatik

Institut für parallele und verteilte
Höchstleistungsrechnersysteme
Breitwiesenstraße 20-22
D-70565 Stuttgart

Asynchronous and Synchronous Cooperation — Demonstrated by Deadlock Resolution in a Distributed Robot System

W. A. Rausch, P. Levi

Report Nr. 1996/07

May 24, 1996

Abstract

Based upon an hierarchical world model with (virtual) roads we describe two different forms of cooperation. Asynchronous cooperation arises if preplanned trajectories of autonomous mobile robots with individual goals overlap. Synchronous cooperation takes place if several robots coordinate their actions in order to fulfill a collective task. We concentrate on all effects that result in the course of intersection passing. If a robot intends to pass an intersection and its destination road is occupied at its beginning the robot must not enter the intersection to keep it free. If there are mutual dependencies that result in a cycle no robot may proceed. We are faced with a deadlock that must be detected by all involved robots and resolved. While intersection passing in its regular form is an example for asynchronous cooperation, deadlock resolution shows cooperative synchronous behavior. The synchronized robots constitute a vehicle formation that moves as a unit at a higher organization level. We examine possible dependency graphs in traffic scenarios and explain the resulting data flow between the affected robots. We have taken care about the usability of our concepts in manufacturing and traffic.

Keywords: Decentralized robot system, asynchronous versus synchronous cooperation, deadlock detection and resolution, navigation, vehicle formation

1 Introduction

Autonomy of mobile robots is still an important focus of research. The use of multiple robots in a cooperative manner requires consideration on robot architectures as well as on communication and synchronization between different robot control systems. Figure 1 illustrates the different stages of cooperation.

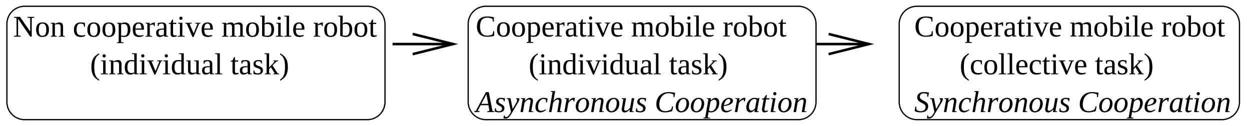


Figure 1: Stages of cooperation

Autonomous *non cooperative* robots show a certain degree of implicit cooperation capabilities. Autonomy requires a robot control structure to interlink its sensor devices with actor devices by several control cycles (figure 2). All cycles are operating simultaneously and they are interlinked with one another to fulfill the robot's mission. Thus they constitute the overall robot control system as an hierarchical net of control cycles where each cycle is in duty of a different task. We have already introduced our so-called "autonomy cycle" and our matrix-shaped architecture in previous publications [Lev89, ROL95]. Each line of the architectural matrix contains cycles of the same abstract level while the lines themselves mirror the division into the strategical, tactical and reflexive levels.

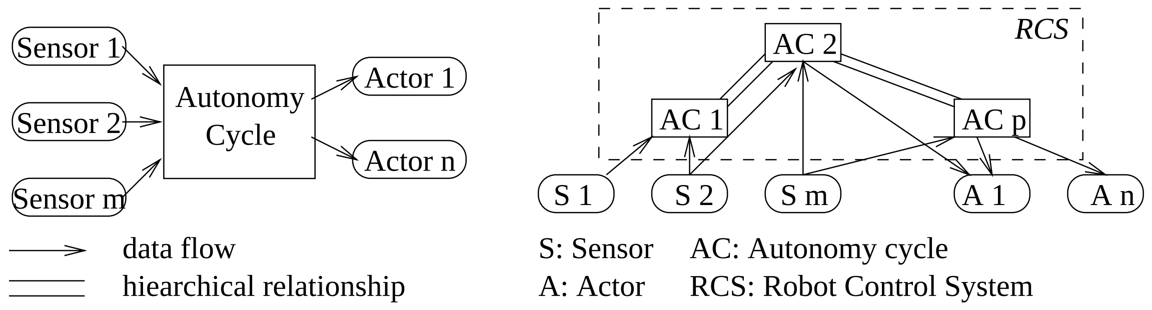


Figure 2: A single autonomy cycle and its embedding in a robot control system

An autonomous mobile robot is able to cope with unexpected events in its environment like obstacle avoidance. The autonomy of multiple robots operating in the same environment evolves into a certain degree of cooperative capabilities even if the robots' control systems are not enhanced with respect to cooperation issues. For example, simple rules like "pass right of obstacle" enables robots facing each other to go on. Usually each robot follows a deterministic set of rules [Gro88] and eventually postpones its genuine intention to give way to unknown dynamic obstacles. For instance, two robots approaching the same intersection will come close each other until they must stop because the respective other "dynamic obstacle" intrudes into their safety areas. The robots will be mutually blocked and wait forever if they do not cooperate.

The key for managing symmetrical behavior of the robots lies in exchanging of intentions between robots recognizing themselves as partners (figure 3). This way enables each robot to calculate the impact of the other robots' plans on its own one. Intentions of other robots can be partially extracted from observation if the robots' behavioral models are known [Par93]. But the information will certainly be partial and ambiguous. This impediment is overcome if we use *active communication* between robots.

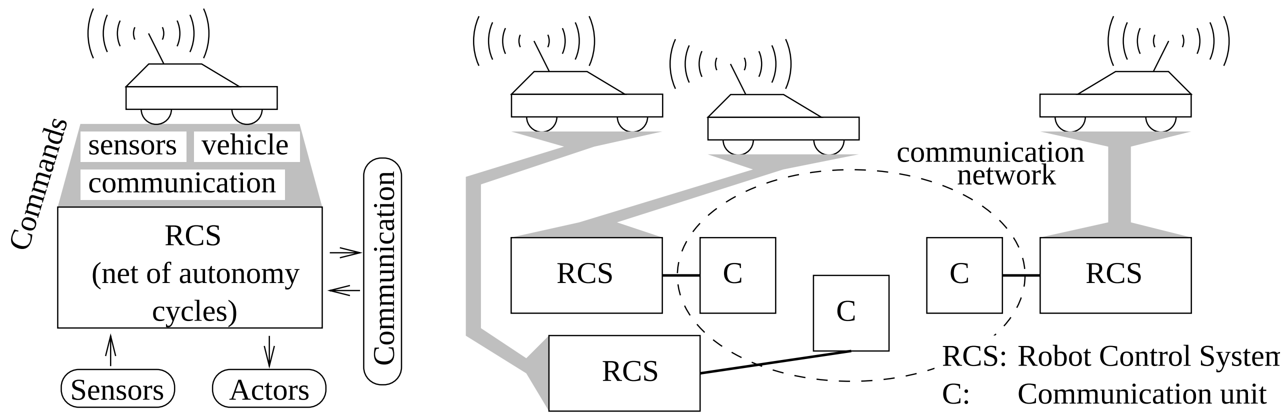


Figure 3: Communicating robots together with their interlinked robot control systems

We distinguish two different forms of cooperation in a distributed robot system. In this paper we will concentrate on all effects that may arise when a robot tries to pass an intersection:

1. *Asynchronous cooperation* arises if several robots fulfill individual tasks in the same environment and interfere with one another due to common space for movements. The interaction between the robots is not known a priori and may occur at any time. Mutual exclusion to shared resources must be guaranteed without any global concept (figure 4).

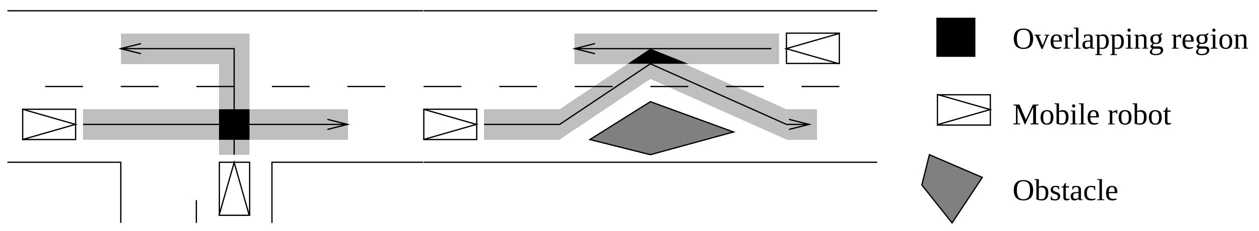


Figure 4: Asynchronous cooperation in traffic (temporal and spatial trajectory modification)

2. *Synchronous cooperation* implies that several robots are in duty of a common task. They must coordinate their actions in time and space. This form of cooperation incorporates a preplanning phase for the robots to prepare the concerted action. Thereafter the robots use their sensors and communication devices for the maintenance of the union as well as for their united appearance to the "outer world". The synchronized robots constitute an abstract vehicle (principle of recursion) where each robot fulfills its part of the overall task. We demonstrate sychronous cooperation by deadlock detection and resolution as illustrated in figure 5. The robots are facing free intersections but cannot pass because of the destination roads being congested. Additionally they are blocking themselves mutually. The robots contribute to deadlock resolution by establishing a vehicle formation with a certain command structure between the robots. A single master robot is tightly coordinating the movements of all other robots.

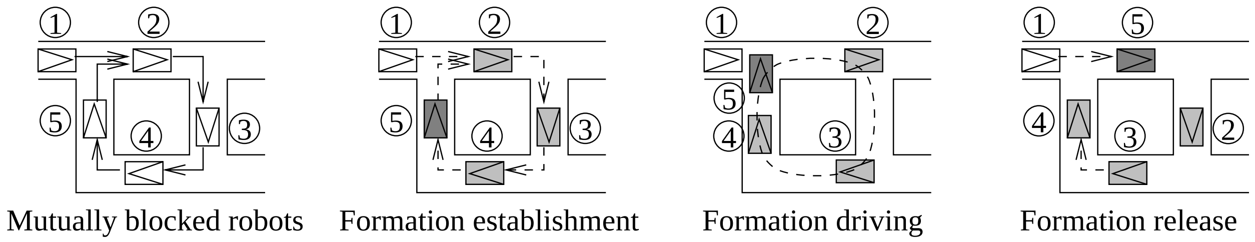


Figure 5: Synchronous cooperation of robots (deadlock handling)

2 Basics for Cooperative Navigation

2.1 World model

Robots navigating in the same environment must have a common understanding of their operation area. At least they must agree upon a common coordinate system to know their mutual spatial relationship. We supply each robot with a graph representation of its navigation space if it is in duty of transportation tasks. For instance, factory layouts show working areas separated by hallways for transportation of raw material and part products. In addition we see load areas, parking lots, intersections between different hallways and so on. The transportation area can adequately be modeled by a traffic net consisting of nodes and edges (figure 6). The edges constitute the traffic lines where robots are not allowed to stay for an indefinite period of time. The nodes form either the intersections between traffic lines or the linkage between a traffic line and a ramp to the net. The dead end of a ramp symbolizes a parking area or a loading bay, thus places where a robot may stay for an infinite period of time. Our modeling easily supports an hierarchical structure. Each node of a higher level may itself contain a traffic net with entrances corresponding to the edges incident to the node.

Usually we distinguish different types of roads like highways, local roads, one-way roads and so on. We have restricted ourselves to three types of roads: 2-way roads, static 1-way roads and dynamic 1-way roads. A dynamic 1-way road may only be passed if the robot has succeeded in reserving the desired direction. We are modeling the transition between a 2-way and a 1-way road by introducing two "2-way intersections".

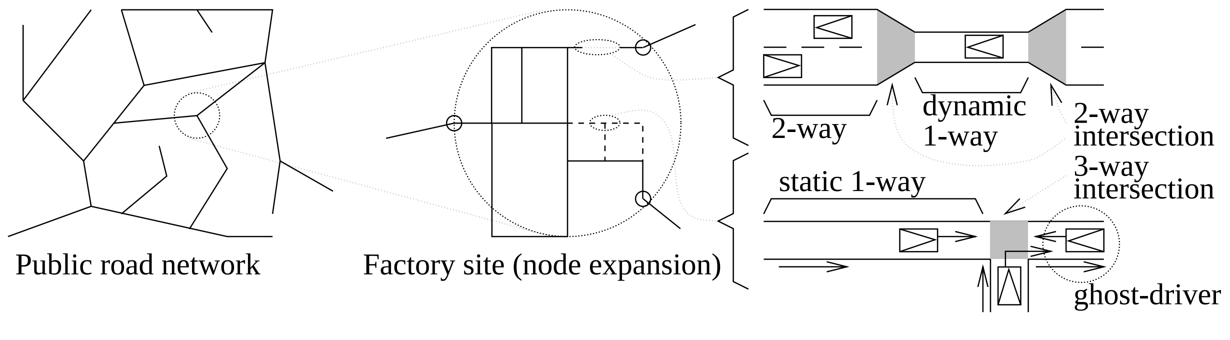


Figure 6: Hierarchically structured traffic net

Navigation in our environment means graph searching and driving along the lines of the traffic net. We use the A^* algorithm for planning a path from the robot's starting point to its destination. The possible directions of each road must be regarded in order to avoid a deadlock situation (figure 6 right). Cooperation necessities arise while traveling along the traffic lines and during passing an intersection. The latter issue is obvious. Concerning cooperation along the lines we must consider manoeuvres like lane switching because of obstacles.

2.2 Communication primitives

Cooperation between multiple robots requires them to exchange data and plans. We concentrate on message passing instead of interpreting complex sensor data based on behavioral models:

- Periodical data exchange (location and velocity) to support sensory detection of robots
- Event-based data exchange for temporal and spatial coordination of driving manoeuvres

We see two different methods for data exchange:

1. The communication unit of each robot incorporates a so-called *signboard* that is acting as a mirror for the state of the robot [WP94]. The robot itself can read and write on its signboard, all other robots may only read from it.
2. The use of *directed message exchange* enables a sending robot to deliver a message immediately to the addressees.

In principle message flow across several instances can be achieved if the instances are periodically examining their signboards. But the more instances a message has to pass the worse transfer time becomes as compared to directed message exchange with immediate response.

2.3 Technical requirements

- In manufacturing driving along marked lines is widely used for robot navigation. This resembles the situation we see in traffic. Our modeling of the world allows an easy integration of available guiding equipment.
- In traffic cars may not enter an intersection if they cannot pass it because of a congested destination road. We use this constraint for supplying buffer space for deadlock resolution.
- Radio broadcasting is limited due to technical properties. This does not affect data transfer between neighboring robots using the signboard metaphor. For wider distances we propose the use of radio telephones.

3 Asynchronous Cooperation

A typical example for asynchronous cooperation is intersection passing. In this section we assume that there is no congestion on the destination road (see section 4 for congestion handling).

3.1 Periodical message passing

To avoid collisions between themselves all robots are periodically broadcasting odometrical and topological data (figure 7). Due to the limited range of radio transmission only neighboring robots will get knowledge about one another. This is sufficient because only those robots are potential competitors for the right to pass the intersection.

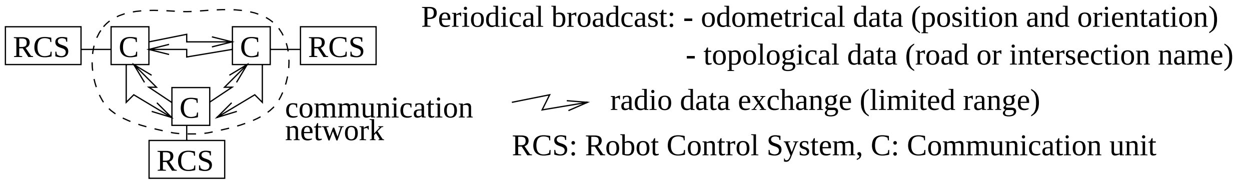

RCS
C
C
RCS
C
RCS
communication network
Periodical broadcast: - odometrical data (position and orientation)
- topological data (road or intersection name)
radio data exchange (limited range)
RCS: Robot Control System, C: Communication unit

Figure 7: Periodic broadcasting between communication units

3.2 Mutual exclusive access

Mutual exclusive access to a shared resource like an intersection is achieved by using the signboard incorporated in the communication unit (figure 8):

1. Initially the signboard signals state NONE. If the robot tries to access a shared resource its state changes to REQUEST.

2. The signboards of all neighboring robots are consulted. If all signboards are in state NONE the robot can change its state to TAKE and use the resource.

3. Finally the robot sets its state to NONE.

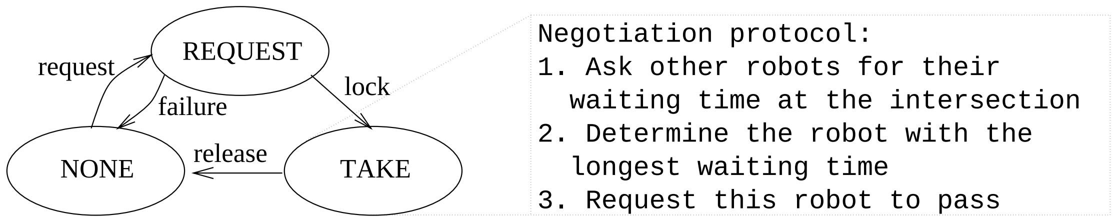

REQUEST
request
lock
failure
NONE
release
TAKE
Negotiation protocol:
1. Ask other robots for their waiting time at the intersection
2. Determine the robot with the longest waiting time
3. Request this robot to pass

Figure 8: State-transition diagram for mutual exclusion and subsequent negotiation phase

If each robot recognizes all robots in its neighborhood the algorithm guarantees that no more than one robot gets the right of way. However the choice of the robot is purely random. If we want to support application specific protocols we can determine a random robot that is in charge of selecting a negotiation protocol and conducting the subsequent negotiation between all robots as a *moderator*. As a first step towards more sophisticated negotiation protocols ([LMB95]) we compare the arrival times and choose the robot that is waiting longest.

4 Synchronous Cooperation

In chapter 3 we have already indicated that a robot may only pass an intersection if its destination road has enough space to completely hold the robot. If this precondition is not fulfilled the robot has got stuck in a congestion. If the robot is followed by another one waiting for it to go on we might be confronted with a deadlock. In the following we discuss possible deadlock dependency graphs, deadlock detection and deadlock resolution.

4.1 Deadlock dependency graphs

Each robot has a single driving intention at a time indicated by an arrow in figure 9a. If robot a is waiting in front of an intersection and cannot completely pass it because another robot b is currently standing at the entrance of the destination road robot a is *dependent* on robot b (indicated by an arrow in figure 9b). A deadlock situation has occured if the dependency graph contains a cycle. The deadlock dependency graph never branches in forward direction because each robot is blocked by the single robot in its intended driving direction. This has two results:

1. The graph shows exactly one deadlock cycle called *kernel*. Every robot not being a kernel member is a part of an attached dependency tree.
2. Several general deadlock dependency graphs are never interconnected. But they may go across the same traffic intersections.

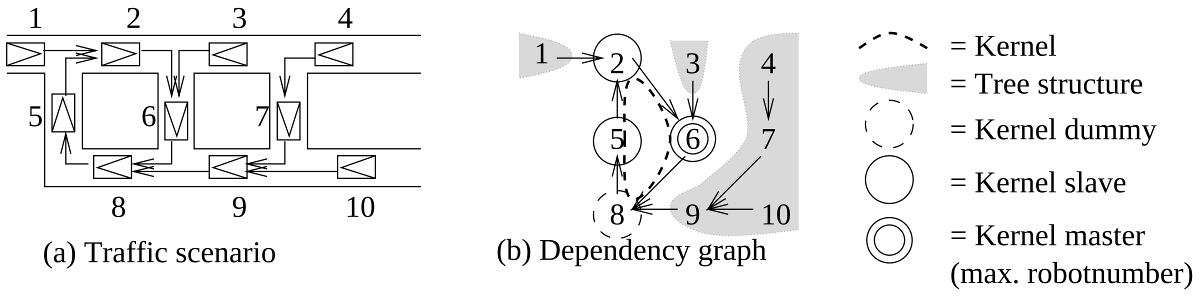


Figure 9: General dependency graph in a deadlock situation

We say that a robot waiting ahead of an intersection inside the kernel is a *representative* for the whole road where it is currently staying. In addition we mark an unique representative to be the *master* of all others. Consequently the other representatives are called *slaves*. The robots neither being master nor slaves but participating in the deadlock kernel are called *dummys*. Without loss of generality the representative with the highest robot number in a deadlock kernel takes the master's role.

4.2 Deadlock detection

Possible candidates for deadlock detection are all robots that want to pass an intersection ahead of themselves. If they cannot enter the destination road and they are immediately followed by other robots they are possibly members of a deadlock dependency graph. The deadlock is detected by flooding the dependency graph with messages in *backward* direction (figure 10):

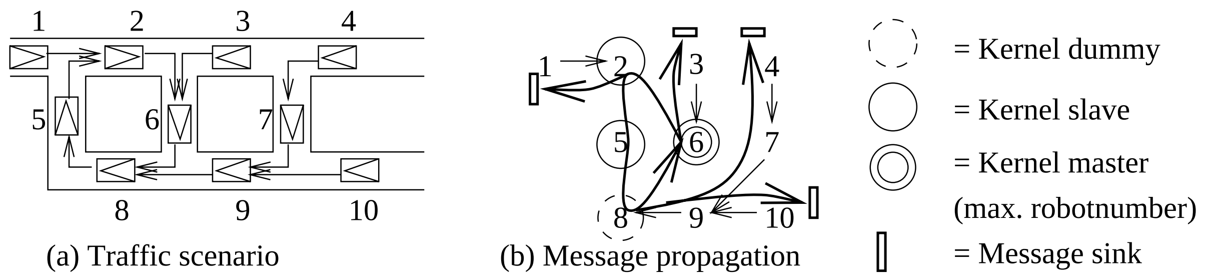


Figure 10: Message propagation in the general dependency graph

1. If a robot is waiting ahead of an intersection it creates a DEADLOCK REQUEST message and sends it to its followers (backward direction). If there is a follower on the same road the message is directed to it. Otherwise the message is directed to all robots waiting in front of the previously passed intersection.

2. All robots receiving the message check if they are blocked by the sending robot. If they are not blocked by it the message is destroyed. Otherwise they propagate the received messages to their respective followers if the message hasn't been created by themselves. If a robot stays in front of an intersection it adds its identification to the forwarded message.

3. If the robot is a member of the deadlock kernel the message finally arrives at its creator. Now this robot has detected the deadlock cycle (Note: If the robot is a member of an attached tree structure the message finally reaches the "leaves" and is destroyed).

Obviously each robot waiting in front of an intersection finally detects the deadlock kernel and becomes a representative either a *master* or a *slave*. Furthermore this robot gets knowledge of all robots waiting in front of other intersections. *Dummy* robots do not remember the existence of a deadlock. They are enclosed by robots forward and backward and are just following the robots ahead of them. Therefore we don't care about them during deadlock handling.

4.3 2-phase-commit protocol for deadlock resolution

Up to now each robot has pursued its individual task and message transfer has been limited to neighboring robots. For deadlock resolution the representatives must form a union commanded by the master robot. Due to technical limitations message exchange between representatives is no longer possible locally. Instead we must use point-to-point connections, e. g. radio telephone. Furthermore we must consider robot and communication failure during the concerted action of master and slave robots. We propose the following *error model*:

- Each robot shows a fail-stop-behavior. It is either working or it is doing nothing. But it is not in a state between working or failure.

- Point-to-point connection between robots can break down at any time.

We assume that each failure is only temporary (error recovery). Errors are detected by timeout mechanisms and in general robot and communication failure cannot be distinguished. Without taking care of transient failures we might use a one-phase protocol (figure 11). Because of a missing synchronization phase between master and slaves the master does not know which slaves are ready for executing the command. Consequently the termination may be inconsistent between the slaves. That means some slaves will fulfill the master's order while others won't care about it. In case of a concerted action of all robots this results in an inconsistent system state. We tackle this problem by using a 2-phase approach consisting of a synchronization phase where an hierarchical command structure between master and slaves is established followed by a command phase (figure 12):

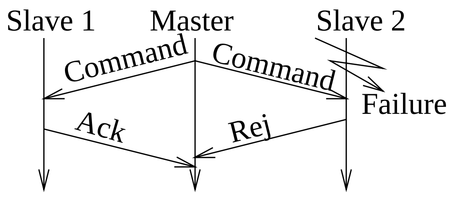


Figure 11: 1-phase protocol

Synchronization phase:

1. The master asks all slaves if they are prepared for subsequent command execution.
2. If a slave is ready for execution it sends an acknowledgement otherwise a rejection.
3. If all slaves are ready the master starts command execution otherwise aborts.

Command phase:

1. The master propagates its decision to all slaves that have previously answered with an acknowledgement. If no slave has rejected during the preparation phase the master commands EXECUTION otherwise ABORT.
2. All slaves are terminating according to the master's decision.
3. The master is gathering the acknowledgements.

Figure 12 illustrates two variants of the 2-phase-commit protocol. If the distinction is necessary we call them *2-phase-parallel-commit* respective *2-phase-serial-commit*.

Communication or robot failures:

Figure 13 shows recovery from different failures. It is important that master and slaves must not leave the second phase until all slaves have confirmed command execution.

4.4 Deadlock resolution

Our deadlock resolution strategy bases upon the idea of forming a convoy of all robots that are members of the deadlock kernel. Thereafter each robot of the convoy goes ahead until all vehicles previously waiting in front of an intersection have entered their destination road (figure 5). It is very important to ensure that no robot from the attached tree structures (figure 9) enters the kernel. Otherwise there would be no space for applying any useful deadlock handling strategy. At first we assume that the deadlock kernel chain does not overlap itself

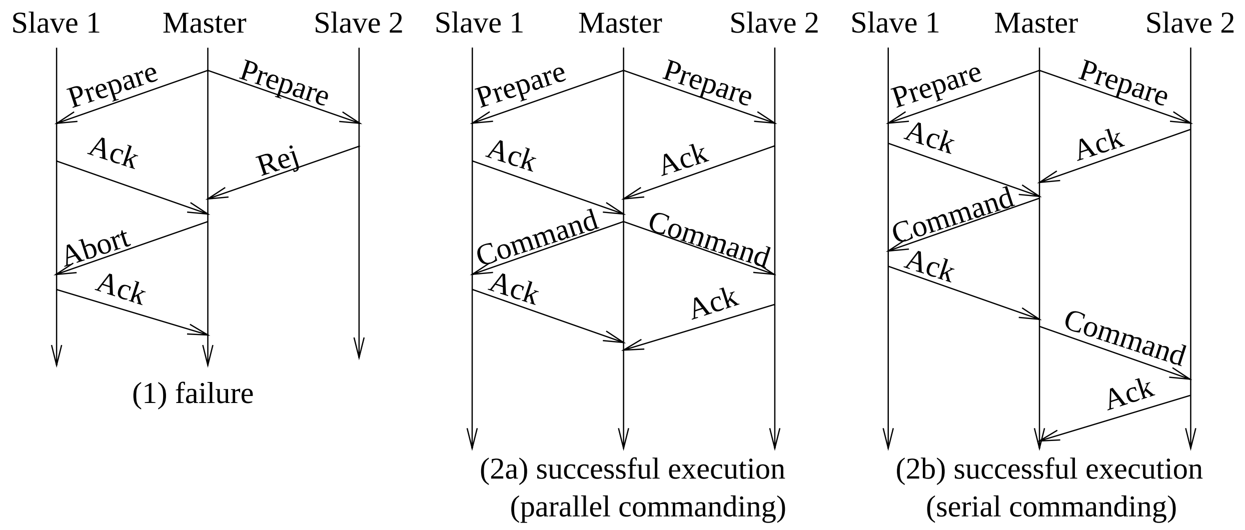
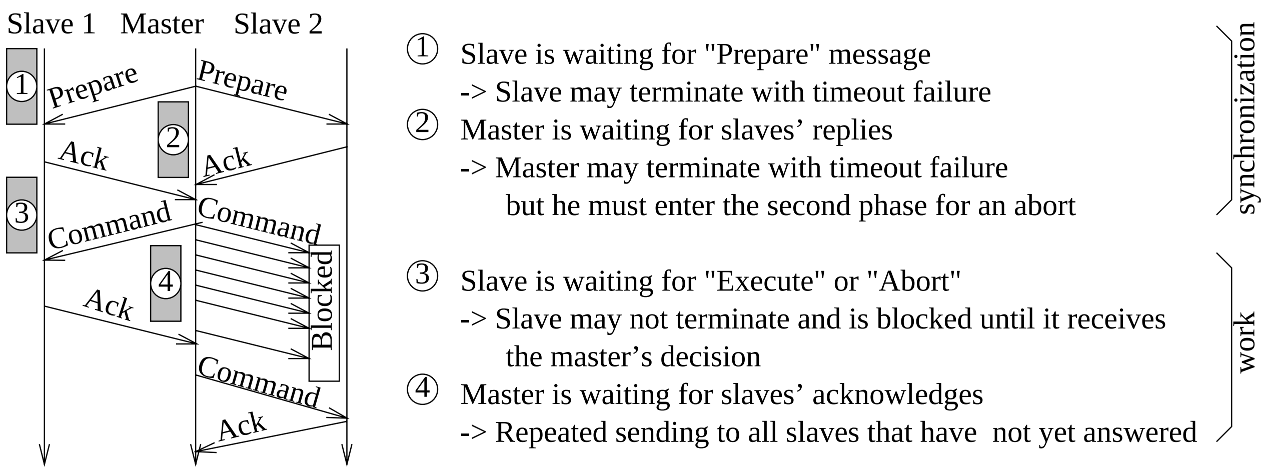


Figure 12: 2-phase-commit protocol (success and failure)



① Slave is waiting for "Prepare" message
-> Slave may terminate with timeout failure
② Master is waiting for slaves' replies
-> Master may terminate with timeout failure
but he must enter the second phase for an abort

③ Slave is waiting for "Execute" or "Abort"
-> Slave may not terminate and is blocked until it receives
the master's decision
④ Master is waiting for slaves' acknowledges
-> Repeated sending to all slaves that have not yet answered

Figure 13: 2-phase-commit protocol (timeout)

due to representatives waiting in front of common intersections (this restriction is eliminated in subsection 4.5). Overlaps with other deadlock dependency graphs are permitted.

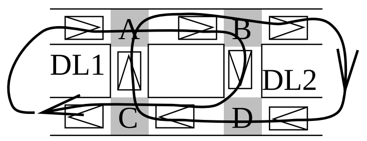
As already mentioned deadlock resolution requires all robots to participate in a concerted action. After deadlock detection all representatives are either master or slaves and each representative is aware of its role (subsection 4.2). Thus a convoy of robots with different roles has been established. Before the robots of the deadlock kernel may proceed all representatives must reserve their respective intersections. To minimize reservation time the master robot commands reservation after a preceding synchronization phase. The subsequent simultaneous entering of the intersections is also prepared by previous synchronization. In short we use the 2-phase-commit protocol for intersection reservation as well as for intersection passing.

4.4.1 Intersection reservation

As already mentioned each representative detects a deadlock situation on its own. Furthermore each robot knows about its role during deadlock resolution. We use a 2-phase-serial-commit protocol as shown in figure 12.

1. At first the master and the slaves must synchronize and all slaves must submit to the command of the master. In case of timeout the situation for the respective robot has not changed and the robot may perhaps calculate another route. If all slaves have sent an acknowledgement a command hierarchy between the unique master and the slaves is established. It is very important that the slaves are no longer allowed to terminate on their own. They have to respect their master's decision until the protocol has finished.

2. After synchronizing with all slaves the master calculates an occupation plan for all intersections along the deadlock chain in the sequence of increasing intersection identifiers. The master arranges for reservation of the intersections by commanding the corresponding slave representatives to compete for the respective intersections one after the other. The master itself competes for its intersection if it is its turn. Thus several deadlock chains sharing common intersections will not block mutually (second level deadlock, figure 14).



	Spatial Sequence:	Allocation Sequence:
DL-Chain 1: Intersections	A, B, C, D	A, B, C, D
DL-Chain 2: Intersections	D, C, A, B	A, B, C, D

DL = Deadlock

Figure 14: Avoiding mutually blocked deadlock chains during intersection allocation

4.4.2 Intersection Passing

If all intersections have successfully been reserved the master robot can command the slave robots to enter the intersections. If the deadlock kernel chain does not overlap itself all robots may proceed in parallel and we can use a 2-phase-parallel-commit protocol (figure 12). Otherwise the master has to calculate a driving sequence and the 2-phase-serial-commit protocol must be applied (subsection 4.5). In general it is sufficient if a single robot in the kernel chain can proceed. This robot uses the intersection in front of itself as a buffering zone. The robots following this robot can proceed thereafter and finally the robot itself can enter its destination road. Thus all robots have made progress concerning their individual goal. After turning the deadlock kernel chain for a "vehicle length" all robots are individual drivers again:

1. The synchronization phase is completely identical to synchronization in subsection 4.4.1.

2. After synchronizing the master checks the deadlock kernel for overlaps. If there is no overlap all robots are commanded to drive in parallel. Otherwise the master calculates a driving sequence and the slaves are commanded accordingly. It is important that each robot or communication failure must be transient. Otherwise there is no chance for the deadlock kernel chain to make progress.

Please note that a single robot does not change its role during deadlock resolution. But deadlock resolution may result in another deadlock situation that again requires deadlock detection and subsequent resolution. Always the robots in the kernel make progress towards their individual destination and no robot must go back at any time. This guarantees that there will never be an oscillation over a series of deadlock situations.

4.5 Specialities

4.5.1 Self-overlapping deadlock kernel chain

In general a robot must pass an intersection completely because it is a no stopping zone. We have used this area as a buffering space during deadlock resolution. Now we want to show theoretical restrictions of this approach. Figure 15(a) shows a deadlock chain overlapping itself. The master robot cannot command its slave robot to enter the intersection because it is blocked thereafter. The other way around the master robot itself cannot enter the intersection because the slave is blocked thereafter. In general this problem can be avoided if buffering space is reserved on the roads themselves or on a special region of the intersection (figure 15(b)). If the overlapping deadlock chain contains a single 3-way intersection the just mentioned problem does not exist. Now there is buffering space inside the deadlock chain for at least the robot waiting ahead of the 3-way intersection. The master robot can easily determine the robots that are able to proceed and in consequence the whole chain makes progress. Deadlock kernel chains will nearly always cover some intersections that the chain does not visit a second time.

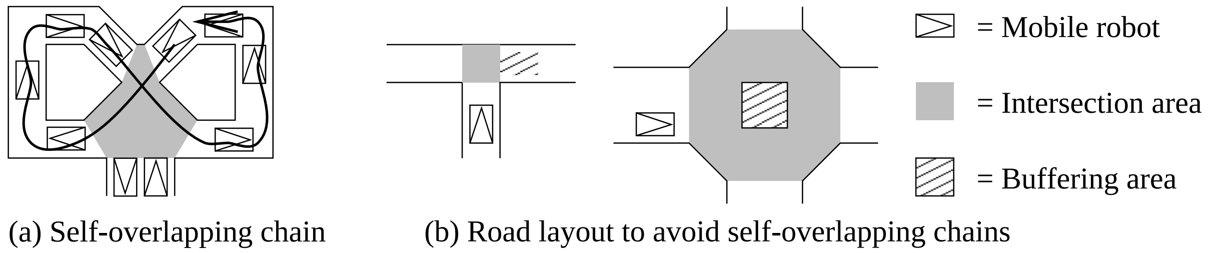

= Mobile robot
= Intersection area
= Buffering area
(a) Self-overlapping chain
(b) Road layout to avoid self-overlapping chains

Figure 15: A self-overlapping deadlock chain and methods to avoid blocking

4.5.2 Error model of the 2-phase-commit protocol

Recovery may be impossible in case of robot failure. The main problem consists in the second phase of the protocol if some robots have already executed their commands while others are faulty. In general this problem is insolvable but let us assume that the total duration time of the protocol can be estimated. Now we can use a timeout mechanism that throws each robot back into its idle state if the timer expires:

- A breakdown of a single robot during intersection reservation is not critical because no robot has moved yet. If a single robot shows no further activity the timers of all other robots will terminate and finally each robot reaches its idle state.

- If a breakdown of a robot happens during intersection passing there is no general solution. The defective robot must enter the shoulder before the timers of all other robots expire. After leaving the traffic net the other robots can go on.

5 Implementation

In our laboratory we use three mobile vehicles each of which is equipped with wireless ethernet. Thus the robots can communicate with one another and with the stationary workstation cluster. For each robot the control system is running on both a stationary workstation as well as the robot's onboard computer. We use PVM [Sun90] for message exchange between different workstations. In the near future we intend to exploit the ethernet connection between the robots' onboard computers. For prototyping program development is mainly done on the stationary workstation cluster [BBR+95]. Shifting the control cycles to the robots' onboard computers will accelerate their speed but does not affect our conceptual considerations. For implementation of our concepts we have decided to experiment with real vehicles prior to developing a simulation system:

1. By using real vehicles we have made valuable experience with the shortcomings of radio transmission. Wireless connection is essential for cooperating robots and must cope with these shortcomings. Furthermore practical experience with sonar sensors and actors of our robots is necessary for developing simulation models. We have already tested mutual exclusion strategies with two robots running on a "8"-shaped course [ROL95].

2. For testing of deadlock detection and deadlock resolution strategies at least three vehicles are required. To speed up program development we have developed a parallel simulator. The snapshot of our exemplary world shows a triangle-shaped obstacle named "Statue" surrounded by roads in its center (figure 16). By placing a robot on each of these roads we encounter a deadlock situation if the robots try to drive around the obstacle.

The parallel simulator consists of several UNIX-processes. PVM allows the use of a workstation cluster for instantiating the UNIX-processes [Sun90]. An animation process is in duty of preparing the graphical output as shown in figure 16. For each robot its movement and propagation of sensor cones are calculated by a dedicated process called "emulator". Thus the parallel simulator consists of a system of $n + 1$ UNIX-processes if n robots are running (figure 17). By using a separate workstation for each of the animation process and the emulator processes we can exploit the power of $n + 1$ computers connected by ethernet.

The animation component of our parallel simulator is periodically (step 1) requesting the robot emulation processes for the robots' actual states and sensor propagations (step 2). Each emulator stores received robot control commands in a queue. Upon request each emulator scans its queue for the actual command and calculates the robot state for the actual time. Then each emulator sends the actual state back to the animation process (step 3). The robot control systems are feeding the parallel simulator with commands in an asynchronous manner. By connecting input and output lines of the distributed robot system to device drivers of real vehicles we are able to verify simulated scenarios in our laboratory.

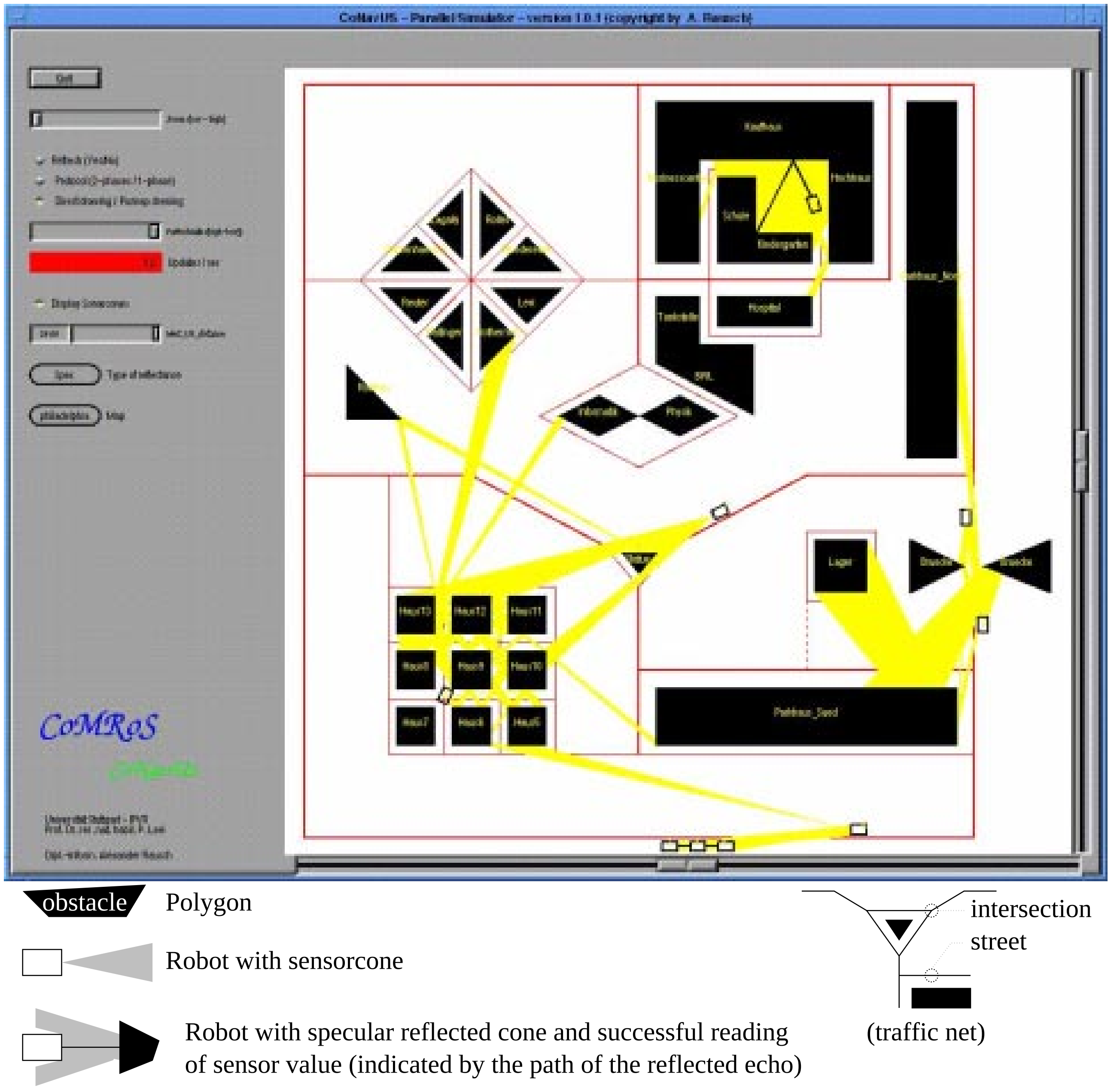


Figure 16: Parallel simulator

All emulation processes are operating in parallel. Thus we can achieve a theoretical speed-up of n if we use a separate workstation for simulating each of n robots. In addition our time-based simulation approach guarantees an overall timely consistent simulation progress if we neglect message transfer times with respect to the refresh rate of the animation. As opposed to our approach event-based simulation systems must take much more effort to synchronize simulation progress of multiple robots.

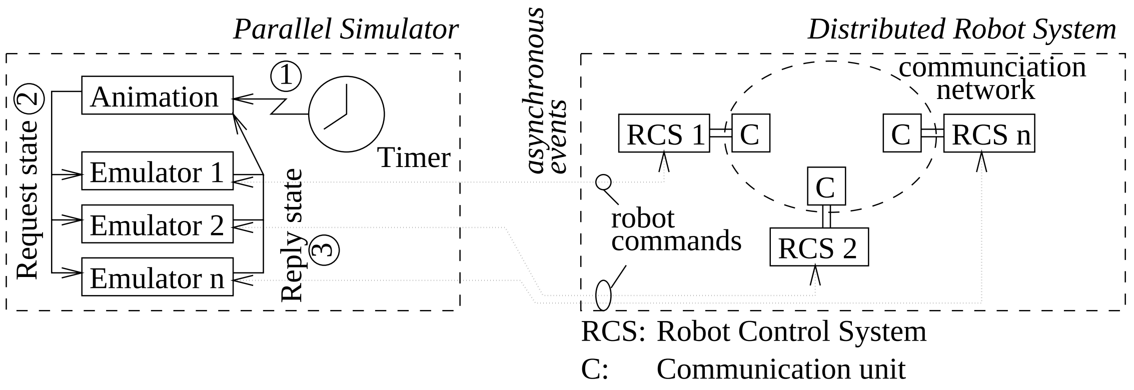


Figure 17: Process structure of the parallel simulator

For technical realization of our concepts we propose short range broadcasting (neighboring robots) via special-purpose hardware supporting the ALOHA principle [Tan89]. Radio telephone provides long range communication between master and slaves. We realize the differences between the proposed technical realization and the conditions in our laboratory. The production use of our concepts will suffer from many lost messages. We have simulated the loss of messages by deliberately destroying messages and by applying timeout-methods to detect this fact.

6 Discussion and Outlook

After studying cooperation aspects using real robots we have developed a parallel simulation tool for examining the behavior of multiple robots. Up to now we have got the following results:

- The algorithm for mutual exclusion has been implemented and thoroughly tested.
- Synchronous behavior between several robots must be controlled by a robot that is acting as a moderator. We have shown this important fact during deadlock resolution. The master robot coordinates the activity of all other slave and dummy robots.
- By using our simulation tool we have noticed severe sensor interferences between different robots. We want to combine cooperation concerning navigation and the use of active sensors. We are currently implementing cooperation strategies for firing active sensors.

7 Acknowledgement

Our research is funded by the German Research Foundation (DFG). We thank Mr. H.-G. Filipp for thoroughly studying our concepts and for doing much of the implementation work.

References

[BBR+95] H. Bayer, Th. Bräunl, A. Rausch, M. Sommerau, and P. Levi. Autonomous vehicle control by remote computer systems. In *Intelligent Autonomous Systems IAS-4*, pages 158–165, Amsterdam - Oxford - Washington DC, March 1995. IOS Press.

[Gro88] David D. Grossman. Traffic control of multiple robot vehicles. *IEEE Journal of Robotics and Automation*, 4(5):491–497, October 1988.

[Lev89] P. Levi. Architectures of individual and distributed autonomous agents. In *Intelligent Autonomous Systems IAS-2*, pages 315–324, 1989.

[LMB95] P. Levi, M. Muscholl, and Th. Bräunl. Cooperative mobile robots stuttgart: Architecture and tasks. In *Intelligent Autonomous Systems IAS-4*, pages 310–317, Amsterdam - Oxford - Washington DC, March 1995. IOS Press.

[Par93] Lynne E. Parker. Designing control laws for cooperative agent teams. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 582–587, Atlanta, Georgia, May 1993.

[ROL95] W.A. Rausch, N. Oswald, and P. Levi. Cooperative crossing of traffic intersections in a distributed robot system. In *SPIE Sensor Fusion and Networked Robotics VIII*, volume 2589, pages 218–229. SPIE, October 1995.

[Sun90] V. Sunderam. PVM: A framework for parallel distributed computing. In *Concurrency: Practice & Experience*, volume 2, pages 315–339, December 1990.

[Tan89] A. S. Tanenbaum. *Computer Networks*, chapter 3. Prentice-Hall International, Inc., 2nd edition, 1989.

[WP94] J. Wang and S. Premvuti. Fully distributed traffic regulation and control for multiple autonomous mobile robots operating in discrete space. In H. Asama, T. Fukuda, T. Arai, and I. Endo, editors, *Distributed Autonomous Robotic Systems*, pages 144–161. Springer Verlag, 1994.