Collision prevention using group communication for asynchronous cooperative mobile robots *

Rami Yared, Xavier Défago, Matthias Wiesmann†

JAIST, School of Information Science
Japan Advanced Institute of Science and Technology
Email: {r-yared, defago, wiesmann}@jaist.ac.jp

Abstract

The paper presents a fail-safe mobility management and a collision prevention platform for a group of asynchronous cooperative mobile robots. The fail-safe platform consists of a time-free collision prevention protocol, which guarantees that no collision can occur between robots, independently of timeliness properties of the system, and even in the presence of timing errors in the environment. The collision prevention protocol is based on a distributed path reservation system. Each robot in the system knows the composition of the group, and can communicate with all robots of the group.

A performance analysis of the protocol provides insights for a proper dimensioning of system parameters in order to maximize the average effective speed of the robots.

1 Introduction

Many interesting applications of mobile robotics envision groups or swarms of robots cooperating toward a common goal. Consider a distributed system composed of cooperative autonomous mobile robots cultivating a garden. Cultivating a garden requires that mobile robots move in all directions in the garden sharing the same geographical space. A robot has no prior knowledge about neither the paths of other robots, nor their speeds.

A robot uses its local sensing system to detect unknown fixed obstacles in the garden, and a robot is based on its local motion planning facility to compute a path between the current location and the goal. This path avoids the collisions with fixed known obstacles.[1]

*Work supported by MEXT Grant-in-Aid for Young Scientists (A) (Nr. 18680007).

†Swiss National Science Foundation Fellowship PA 002-104979. Now with Google Switzerland, Freigustraße 12, 8002 Zürich

[1]The robots are the only moving entities in the considered applications.

In cooperative autonomous mobile robot environments, where robots move with unpredictable speeds, sensor-based motion planning approaches cannot guarantee *safe* motion, as mobile robots may collide with each other, because of the unpredictable speeds of robots and the uncertainty of the sensory information.

Specification. The robots are not provided with vision capability. In the considered applications, there is no centralized control nor global synchronization.

Problem. The robots are moving in different directions sharing the physical space, thus collisions between mobile robots can possibly occur. It is very important to focus on the problem of *preventing* collisions between mobile robots. Collision prevention leads to a dependable system and prevents the occurrence of serious damage to the robots, which could cause failures in the system.

Requirements. It is essential to provide a fail-safe platform on which mobile robots can rely for their motion. This platform guarantees that no collision between robots can occur.

A robot knows neither the positions of other robots nor their destinations. Additionally, the speed of a robot is unknown by other robots, and there is no known upper bound on robot's speed, so a robot cannot estimate the position of another robot in the system. Therefore, robots need to cooperate in order to achieve a fail-safe motion. Cooperation is, however, difficult to obtain under the weak communication guarantees offered by wireless networks, because retransmission of messages is needed to ensure message delivery in wireless environments. The communication delays to deliver messages are difficult to anticipate. The previous arguments ensure that a time-free collision prevention protocol is indispensable.

21st International Conference on Advanced Networking and Applications(AINA'07)
0-7695-2846-5/07 $20.00 © 2007 IEEE

Contribution. In this paper, we present a fail-safe platform on which cooperative mobile robots rely for their motion. Our fail-safe platform consists of a time-free collision prevention protocol for an asynchronous system of cooperative mobile robots, and guarantees that no collision can occur between robots, independently of timeliness properties of the system, and even in the presence of timing errors in the environment. The collision prevention protocol is based on a distributed path reservation system. The paper presents proofs of correctness of the protocol and proves the deadlock freedom, and the liveness properties of the protocol. A performance analysis of the protocol provides insights for a proper dimensioning of system parameters in order to maximize the average effective speed of the robots.

Related work. Martins et al. [4] demonstrated a scenario of three cooperating cars, elaborated in the CORTEX project, which relies on the existence of Timely Computing Base TCB wormholes. The TCB concept was introduced by Veríssimo and Casimiro in [6, 7]. Martins et al. in [4] use an application's fail-safety and time-elasticity to overcome the uncertainty of the environment. The approach of Matins et al. [4] is time-elastic, while our approach is time-free.

Nett et al. [5] presented a system architecture for cooperative mobile systems in real-time applications. The approach of Nett et al. [5] aims at real-time cooperative mobile systems. The communications are synchronous, assuming the existence of a known constant upper bound on the communication delays, the infrastructure is based on wireless LAN, and the protocols use the access point as a central router. Our approach fundamentally differs in several aspects, our approach is asynchronous, and the mobile robots form naturally a mobile ad hoc network on which they rely for their communication. MANETs have no centralized control nor global synchronization, also they do not guarantee the real-time constraints to deliver messages.

Structure of the paper. The rest of the paper is organized as follows. Section. 2 describes the system model, definitions, and terminology. Section. 3 defines the collision prevention problem and its specification. In Section. 4, we present our collision prevention protocol. Section. 5 presents a performance analysis of the protocol. Section. 6 concludes the paper.

2 System model and terminology

2.1 System model

We consider a system of n mobile robots $S = \{r_1, \ldots, r_n\}$, moving in a two dimensional plane. Each robot has a unique identifier. The total composition of the system is known to each robot.

Robots have access to a global positioning device that, when queried by a robot r_i, returns r_i's position with a bounded error ε_{gps}.

The robots communicate using wireless communication such that a robot r_i can communicate with all robots of the system. Communications assume retransmissions mechanisms such that communication channels are reliable.

The system is asynchronous in the sense that there is no bound on communication delays, processing speed and on robots speed of movement.

2.2 Total Order Broadcast.

Total Order Broadcast, also called (atomic broadcast), is a group communication primitive, which ensures that all correct processes deliver all the messages in the same order. Therefore, the total order broadcast ensures that all the correct processes agree on the same total order delivery of the messages, so they can behave consistently. The total order broadcast primitive also guarantees several other properties. If a correct process broadcasts a message m then, this process eventually delivers m. The total order broadcast also ensures that if a process delivers a message m then, all correct processes eventually deliver m. Furthermore, for any message m, every process delivers m at most once, and only if m was previously broadcasted by some process. (see Défago et al. [1] for a survey).

2.3 Definitions and terminology

Paths. A *chunk* is a line segment along which a robot moves. A path of a robot is a continuous route composed of a series of contiguous chunks. A path can take an arbitrary geometric shape, but we consider only line segment based paths for simplicity.

Errors. The are three sources of geometrical incertitude concerning the position and the motion of a robot. Error related to the position information provided by the positioning system denoted ε_{gps}. In addition, the motion of a robot creates two additional sources of errors. The first error is related to the translational movement, denoted: ε_{tr}. The second error is related to the rotational movement, denoted: ε_{θ}.

Zones. A *zone* is a finite convex area of the plane. A *zone* is defined as the area needed by a robot to move safely along a chunk. This includes provisions for the shape of the robot, positioning error, and imprecision in the moving of the robot. The zone must contain the chunk the robot is following. Figure 1 shows the zone Z_i for a robot r_i located in point A and moves along the segment AB, where d represents the radius of the geometrical shape of r_i. The

21st International Conference on Advanced Networking and Applications(AINA'07)
0-7695-2846-5/07 $20.00 © 2007 IEEE

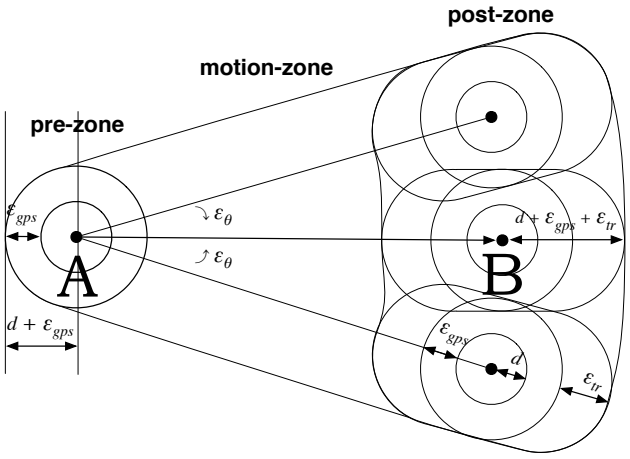

Figure 1. Reservation Zone.

zone Z_i is composed of the following three parts, illustrated in Figure 1: the first part named *pre-motion* zone and denoted $pre(Z_i)$, is the zone that robot r_i possibly occupies while waiting (before moving). The second part named *motion* zone and denoted $motion(Z_i)$, is the zone that robot r_i possibly occupies while moving. The third part named *post-motion* zone and denoted $post(Z_i)$, is the zone that robot r_i possibly reaches after the motion.

3 Collision prevention protocol

3.1 Problem Specification

Before entering a zone Z, robot r executes $reserve(r, Z)$. After leaving a zone Z, robot r executes $release(r, Z)$.

A robot r_i *releases* the zone Z_i that it has owned and keeps only a part of $post(Z_i)$ under its reservation. The part of the zone that has been released by r_i is denoted: $RelZ_i$.

Operations. We say that two reservation operations $reserve(r_i, Z_i)$, $reserve(r_j, Z_j)$ *conflict* if $r_i \neq r_j$ and $Z_i \cap Z_j \neq \emptyset$, we denote this $o_1 \bowtie o_2$. If a robot r executed $reserve(r, Z)$ but did not execute yet $release(r, Z)$, we say that *r owns* zone Z.

Schedules. We call a *schedule* an ordered sequence of operations $S = \{o_1, \ldots o_m\}$ where every operation is either $reserve(r_i, Z_i^j)$ or $release(r_i, Z_i^j)$.

The notion of schedule is closely related to the notion of histories used to model database operations [2]. The notation $o_1 \overset{S}{\succ} o_2$ is used to mark that operation o_2 happens after o_1 in schedule S. We say that a schedule is *correct*, if it enforces the following constraints.

- if a robot r executes $release(r, Z)$, then it executed $reserve(r, Z)$ before.
- if robot r owns zone Z then there is no robot $r^{'}$ that owns a zone $Z^{'}$ such that $Z \cap Z^{'} \neq \emptyset$

If in a given schedule all robots own at most k zones, we say that this is a *k-schedule*. As robots need to be able to reserve at least two zones (the one currently occupied and the next one) $k \geq 2$. We say that two schedules S_a and S_b are *compatible* if:

- All operations of a given robot are in the same order, i.e. $\forall r \mid o_i^r \overset{S_a}{\succ} o_j^r \longmapsto o_i^r \overset{S_b}{\succ} o_j^r$.
- All conflicting operations are in the same order i.e. $\forall o_i, o_j \mid o_i \bowtie o_j \; o_i \overset{S_a}{\succ} o_j \longmapsto o_i \overset{S_b}{\succ} o_j$

We say that two schedules are *equivalent* if they are compatible and contain the same set of operations:

- They contain the same set of operations, i.e. $\forall o \mid o \in S_a \longmapsto o \in S_b$

The *local schedule* S_r of robot r is the ordered subset of a schedule that only contains operations that either initiated by robot r or conflict with operations initiated by robot r.

Scheduler. A scheduler is an algorithm that takes as input a sequence of zone requests and builds as output for every robot $r \in R$ a local schedule S_r. A scheduler is *correct* if all local schedules S_r are compatibles with correct schedule S. We distinguish two types of deadlocks, the first type of deadlocks is due to a cyclic *happens after* relation, and the second type of deadlocks is due to pathological situation of intersection between two requested zones. The details of deadlock situation are explained in the extended version of the paper [8]. We say that a scheduling algorithm is *deadlock free* if it avoids deadlocks. In the rest of the paper, we concentrate on algorithms that are correct and deadlock free.

4 State Machine Scheduler

The protocol is essentially a mutual exclusion on geographical areas. It is based on the state machine approach of Lamport [3]. Briefly, the idea is as follows. Each robot maintains a copy of the reservation queue and a protocol ensures that all requests/releases are delivered in the same sequence. With all replicas starting in the same state, they evolve consistently with no need for further synchronization.

4.1 Basic idea.

The algorithm consists of a distributed path reservation system, such that a robot must reserve a zone before it moves. When a robot reserves a zone, it can move *safely* inside the zone. All robots run the same protocol. When a robot wants to move along a given chunk, it must reserve the zone that surrounds this chunk. When this zone is reserved, the robot moves along the chunk. Once the robot reaches the end of the chunk, it releases the zone except for the area that the robot occupies. When moving along a path, the robot repeats this procedure for each chunk along the path.

4.2 Detailed scheduler description

We present the variables used in the protocol.

- Z_i is the zone currently requested or owned by robot r_i.
- Dag_{wait} is a directed acyclic graph represents the wait-for relations between robots, such that the vertices represent the robots, and a directed edge from vertex(r_i) to vertex(r_j) indicates that r_i waits for r_j.

We explain briefly[2] the phases of the scheduler with respect to a robot r_i. The robot r_i is located in the *pre-motion* zone $pre(Z_i)$. When robot r_i requests a new zone Z_i, it proceeds as follows.

1. TO-Broadcast. r_i performs a total order broadcast of a message that consists of two parts. The first part is a REQUEST with the parameters of the requested zone Z_i, and the second part is a RELEASE of the previous owned zone.

2. Append-Vertex. When the robot r_i TO-delivers a new message, a new vertex is added to the wait-for graph Dag_{wait} and an existing vertex is removed from the graph. The new added vertex corresponds to the REQUEST part of the message and the removed vertex corresponds to the RELEASE part of the message. When a robot releases the previous zone, the corresponding vertex and its incoming edges are removed from the wait-for graph. When a robot r_i requests a new zone Z_i, a new vertex is added to Dag_{wait} with outgoing edges from the new vertex to all the vertices of the graph such that the corresponding zone intersects with the requested zone Z_i. When the vertex that represents the robot r_i in Dag_{wait} becomes a sink vertex (has no outgoing edges), the requested zone Z_i is reserved to r_i.

3. Request-Rejection. As the possibility of deadlock exists, the scheduler can reject some zone requests to avoid deadlock situations. The routing algorithm of the robot needs to be able to handle those rejections, either by retrying at a later time, or by planning a different route.

4. Rejection-Handler. If the request (r_i, Z_i) is rejected due to a Request-Rejection situation then, the routing algorithm of robot r_i handles the rejected request either by retrying at a later time, or by planning a different route (alternative path). If there is no available alternative path and the request is still rejected after a certain number of trials then, an exception is raised.

5. Release. When r_i reaches the *post-motion* zone $post(Z_i)$, it computes its new position and thus it computes the zone to be released which is Z_i except the place that r_i may possibly occupy (footprint in addition to the positioning system error ε_{gps}). Initially, the released zone is set to $\perp$. All the robots build the same wait-for graph Dag_{wait}.

Property 1 (Liveness) *If a robot r_i requests Z_i then eventually (r_i owns Z_i or an exception is raised).*

r_i requests $Z_i \Rightarrow \Diamond$ (r_i owns Z_i or Exception)

Property 2 (Non triviality) *Exception is raised only if there is no available alternative path and the request is rejected after a certain number of trials.*

The proofs of the correctness, the deadlock freedom and the liveness properties of the scheduler, are omitted due to space restrictions. They can be found in the extended version of the paper [8].

5 Performance analysis

We study the performance of our protocol in terms of the time needed by a robot r_i to reach a given destination when robots are active (robots do not sleep). We compute the average effective speed of robots executing our collision prevention protocol. We provide insights for a proper dimensioning of system parameters in order to maximize the average effective speed of the robots. For simplicity, we assume in this section that the physical dimensions of robots are too small such that a robot can be considered as a point in the plane. The geometrical incertitude related to the positioning system, translational and rotational movements are neglected.

[2]The algorithm, omitted due to space limitations, is presented in the extended version of the paper [8].

5.1 Time needed to reserve and move a chunk

The average physical speed of a robot is denoted by: V_{mot}. We calculate the average time required for a robot r_i to reserve and move along a chunk of length D_{ch} with a physical speed V_{mot}.

When a robot requests a zone, it releases the previously owned zone thus, a robot waits at most for $(n-1)$ robots where n is the number of robots in the system. So, the average number of robots that r_i waits for is: $n_{avg} = \frac{n-1}{2}$

Communication delays. In order to evaluate the performance of the protocol, we need to consider an average communication delays in the system, although the protocol is time-free. The average communication delays in the system is denoted: T_{com}. When all the robots are active running the protocol, then the time needed to reserve and move along a chunk denoted T_{ch} is the sum of the time needed by each of the following steps:

1. The delay to deliver a message, which depends on the performance of the total order broadcast algorithm. We consider that the delay to deliver a message is $T\ n$, where T is a constant.
2. The time to receive the release messages from n_{avg} robots each of which has owned its zone for $\frac{D_{ch}}{V_{mot}}$ time units. $n_{avg}(T_{com} + \frac{D_{ch}}{V_{mot}})$
3. The time needed by r_i to move along a chunk. $\frac{D_{ch}}{V_{mot}}$

Therefore, the time needed to reserve and move along a chunk T_{ch} is:

$$T_{ch} = Tn + \frac{n-1}{2} T_{com} + (\frac{n+1}{2}) \frac{D_{ch}}{V_{mot}} \qquad (1)$$

5.2 Average effective speed

We compute the average effective speed V of a robot r_i as a function of the chunk length D_{ch} and of the number of robots n in the system.

The average effective speed V is:

$$V = \frac{D_{ch}}{Tn + \frac{n-1}{2} T_{com} + (\frac{n+1}{2V_{mot}}) D_{ch}} \qquad (2)$$

The previous relation shows that the effective speed is a function of the chunk length and the number of robots n, also the effective speed depends on some system-based fixed parameters such as the communication delays T_{com} and the physical speed of robots V_{mot}. The effective speed depends also on the performance of the total order broadcast algorithm.

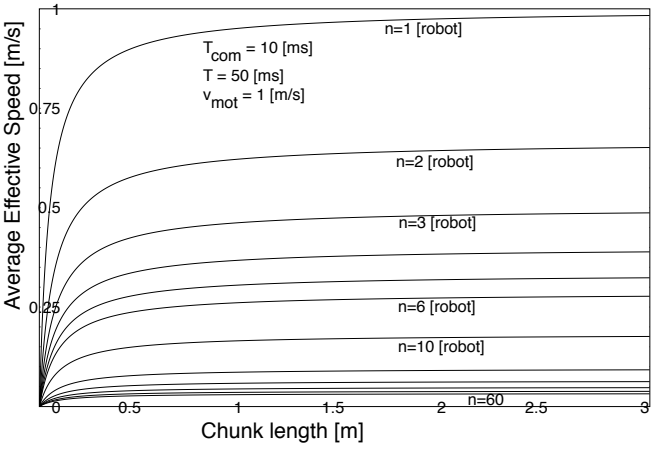

Figure 2. Effective speed vs chunk length.

5.3 Average effective speed vs chunk length

We focus on the relation between the average effective speed and the chunk length for a given number of robots n. The effective speed increases as the chunk length increases. The explanation is that a robot r_i waits at most for $n-1$ robots (in a group of n robots) to move along each chunk of its path. r_i needs to do a certain number of steps to reach the destination, and the number of steps depends on the chunk length. When the chunk length increases, the number of steps decreases. Therefore, the average effective speed V increases with the chunk length D_{ch}. Equation 2 implies that the average effective speed approaches toward the value $\frac{2V_{mot}}{n+1}$ as the chunk length tends to infinity. Figure 2 represents the relationship between the effective speed and the chunk length for different values of the number of robots. The average effective speed of robots increases as the chunk length increases for a given number of robots. There is an optimal value of the chunk length that maximizes the effective speed of the robots. The effective speed keeps this maximal value as the chunk length getting longer than the optimal value. The average effective speed has a horizontal asymptote at $\frac{2V_{mot}}{n+1}$

Numerical values. The values of the fixed system parameters are: $T_{com} = 10[ms]$, $V_{mot} = 1[m/s]$. We consider that the time required to deliver a message is $T\ n$, where $T = 50[ms]$. The number of robots varies from one robot until 60 robots, and the chunk length varies from zero to 3 meters. The average effective speed increases as the chunk length increases until it reaches a maximal value. Figure 2 shows that, in a case of a system composed of 3 robots for example, the maximal effective speed is $0.48[m/s]$ which corresponds to optimal chunk length $\approx 2[m]$.

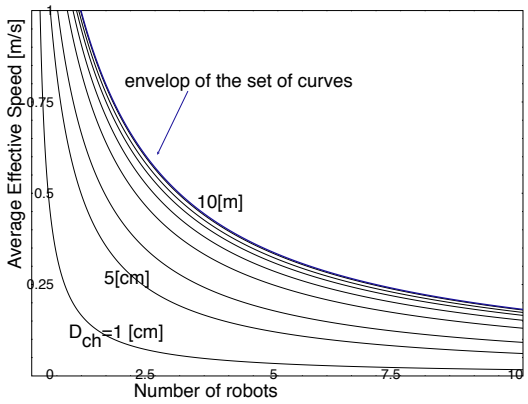

Figure 3. Effective speed vs number of robots.

5.4 Average effective speed vs number of robots

We focus on the relation between the average effective speed V and the number of robots n in the system for a given value of the chunk length. The effective speed decreases as the number of robots increases for a given chunk length, because a robot r_i must wait for more robots. Figure 3 shows the variation of the average effective speed with respect to the number of robots for different values of the chunk length.

Numerical values. The chunk length varies from $1[cm]$ until 10 meters. (Figure 3). The set of curves in Figure 3 have an envelop curve, given by the following equation: $V = \frac{2V_{mot}}{n+1} = \frac{2}{n+1}$

- The envelop curve corresponds to the average effective speed for very high values of the chunk length (tends to infinity), because the average effective speed approaches to a constant value.
- All curves in Figure 3 approaches to zero, when the number of robots tends to infinity. (horizontal asymptote at effective speed = 0).

6 Conclusion

We presented a fail-safe mobility management and achieved a collision prevention platform for a group of asynchronous cooperative mobile robots.

Our fail-safe platform consists of a time-free collision prevention protocol, which guarantees that no collision can occur between robots, independently of timeliness properties of the system, and even in the presence of timing errors in the environment. The collision prevention protocol is based on a distributed path reservation system. Each robot in the system knows the composition of the group, and can communicate with all robots of the group. We proved the correctness, the deadlock freedom, and the liveness properties of the protocol. We have analyzed the performance of the protocol and provided insights for a proper dimensioning of system parameters in order to maximize the average effective speed of the robots. We have also successfully implemented a working prototype of the platform on Pioneer 3DX robots, in Java and using the ARIA library[3].

Acknowledgments

We are grateful to Nak-Young Chong, Nikolaos Galatos, Maria Gradinariu, Yoshiaki Kakuda, Takuya Katayama, Richard D. Schlichting, Yasuo Tan, Tatsuhiro Tsuchiya.

This research was conducted for the program "Fostering Talent in Emergent Research Fields" in Special Coordination Funds for Promoting Science and Technology by the Japan Ministry of Education, Culture, Sports, Science and Technology.

References

[1] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, December 2004.

[2] J. Gray and A. Reuter. *Transaction processing: concepts and techniques. Data Management Systems*. Morgan Kaufmann Publishers, Inc., San Mateo(CA), USA, 1993.

[3] L. Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks*, 2:95–114, 1978.

[4] P. Martins, P. Sousa, A. Casimiro, and P. Veríssimo. A new programming model for dependable adaptive real-time applications. *IEEE Distributed Systems Online*, 6(5), May 2005.

[5] E. Nett and S. Schemmer. Reliable real-time communication in cooperative mobile applications. *IEEE Trans. Computers*, 52(2):166–180, 2003.

[6] P. Veríssimo. Uncertainty and predictability: Can they be reconciled? In *Future Directions in Distributed Computing*, pages 108–113, 2003.

[7] P. Veríssimo and A. Casimiro. The Timely Computing Base model and architecture. *IEEE Trans. Computers*, 51(8):916–930, 2002.

[8] R. Yared, X. Défago, and M. Wiesmann. Collision prevention using group communication for asynchronous cooperative mobile robots. Research Report IS-RR-2007-002, Japan Advanced Institute of Science and Technology (JAIST), Hokuriku, Japan, February 2007.

[3] ARIA: Advanced Robotics Interface for Applications. (http://www.activrobots.com/).