Collision prevention using group communication for asynchronous cooperative mobile robots

Rami Yared, Xavier Défago, and Matthias Wiesmann

School of Information Science,
Japan Advanced Institute of Science and Technology (JAIST)

February 22, 2007

IS-RR-2007-002

Japan Advanced Institute of Science and Technology (JAIST)

School of Information Science
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

http://www.jaist.ac.jp/

ISSN 0918-7553

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

*Work supported by MEXT Grant-in-Aid for Young Scientists (A) (Nr. 18680007).
†Swiss National Science Foundation Fellowship PA 002-104979

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]

In cooperative autonomous mobile robots environments, where robots move with unpredictable speeds, the motion planning approaches cannot guarantee a *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 a vision capability. In the considered system, 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 damages to the robots which causes 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 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 messages 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.

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. The collision prevention protocol is based on a distributed path reservation system.

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

The paper also presents proofs of correctness of the protocol and also 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. [3, 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 [8, 9]. Martins et al. in [4] use an application's fail-safety and time-elasticity to overcome the uncertainty of the environment.

The *fundamental* difference between our fail-safe platform and the approach of Matins et al. [4], is that the approach in [4] is time-elastic, while our approach is time-free.

Nett et al. [5, 6] presented a system architecture for cooperative mobile systems in real-time applications. They considered a traffic control application in which a group of mobile robots share a specified predetermined space. The approach of Nett et al. [5, 6] 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, which ensures full connectivity.

Our approach fundamentally differs in several aspects, our approach is asynchronous, and the mobile robots in our system 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 an illustrative example of the protocol. Section. 6 presents a performance analysis of the protocol. Section. 7 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 movement.

2.2 Definitions and terminology

Paths. We denote by *chunk* 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. We call a *zone* 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 imprecisions 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 desires to move along the segment AB, where d represents the radius of the geometrical shape of r_i. The 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.

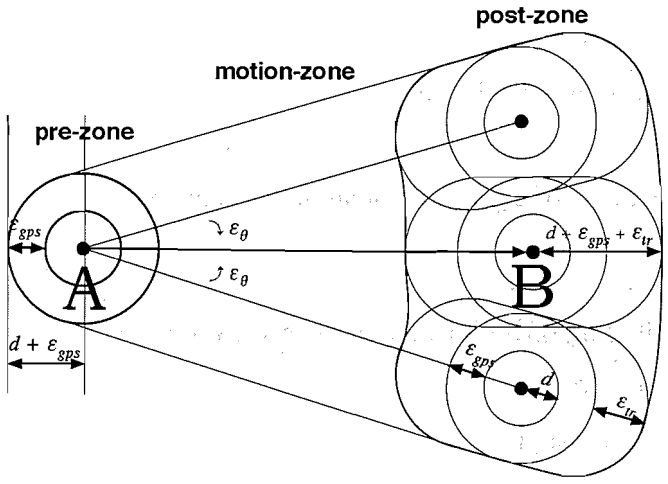

Figure 1: Reservation Zone.

3 Problem definition and 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 *locale 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.

3.1 Deadlock situation

There are pathological situations of intersection between Z_i and Z_j, such that neither r_i nor r_j can move. We say that r_i and r_j are in a deadlock situation when none of them can move. For example, a deadlock situation between two robots, occurs when each robot requests a zone that intersects with the *pre-motion* zone of the other, so none of the robots can be granted its requested zone, since each of them requests a resource owned by the other robot. Figure 2 illustrates this situation.

A deadlock situation can be expressed as follows.
$[Z_i \cap pre(Z_j) \neq \emptyset]$ and $[Z_j \cap pre(Z_i) \neq \emptyset]$

There are other pathological intersection situations between Z_i and Z_j, such that if one of the robots has granted its zone then, the other robot may never be able to own its requested zone (starvation situation).

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. 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

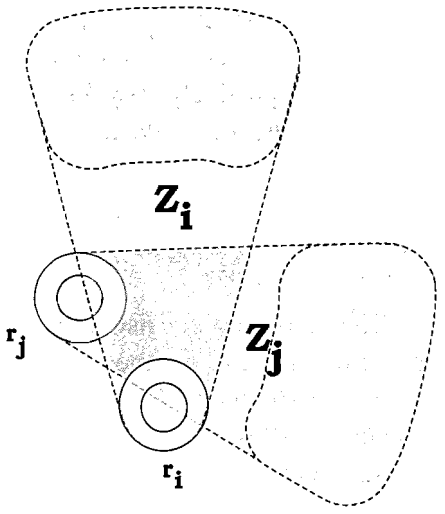

Figure 2: Deadlock situation : Z_i intersects with $pre(Z_j)$ and Z_j intersects with $pre(Z_i)$.

handle those rejections, either by retrying at a later time, or by planning a different route.

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.

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 state machine scheduler uses the state machine replication approach [7].

4.1 Total Order Broadcast

TOTAL ORDER BROADCAST also called ATOMIC BROADCAST, is a fundamental problem in distributed systems. The TOTAL ORDER BROADCAST primitive ensures that messages sent to a set of processes are, in turn, delivered by all those processes in the same total order. Informally, the problem is defined as a broadcast primitive whereby all processes deliver the same sequence of messages.

There exists a vast amount of literature about total order broadcast presented (see Défago et al. [1] for a survey).

The problem is defined in terms of two primitives, which are called *TO-broadcast(m)* and *TO-deliver(m)*, where m is some message. When a process p executes *TO-broadcast(m)* (respectively *TO-deliver(m)*), we say that p TO-broadcasts m (respectively TO-deliver m). We assume that every message m can be uniquely identified, and carries the identity of its sender, denoted by *sender(m)*. In addition, we assume that, for any given message m, and any run, *TO-broadcast(m)* is executed at most once [1].

Idea of the scheduler. 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. The path reservation is performed in a consistent manner. 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.

All robots run the same distributed algorithm. When a robot r_i requests a zone Z_i, r_i broadcasts a message indicating a request of a zone REQUEST(Z_i)and a release of the previous owned zone. RELEASE(PREVIOUS($RelZ_i$)) using a total order broadcast primitive.

A wait-for graph is generated according to the delivered requests and releases. The wait-for graph represents the wait-for relations between robots. If zone Z_i of a robot r_i intersects with zone Z_j of robot r_j, then a wait-for relation between r_i and r_j is established. When a robot r_i reaches the *post-motion* zone $post(Z_i)$, r_i releases the previous zone, and requests a new zone.

All the robots in the system deliver requests and releases in the same order, thus consistent reservations and releases of zones take place.

Variables 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.

4.2 Scheduler description

We explain briefly 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 with the parameters of the released zone PREVIOUS($RelZ_i$). The robot r_i releases the previous reserved zone and requests a new zone Z_i.

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 requests a new zone, a new vertex is added to Dag_{wait} with outgoing edges from the added vertex to all the vertices of the graph whose 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

   If the requested zone Z_i intersects with the *post-motion* zone $post(Z_j)$ of a vertex in the wait-for graph Dag_{wait} then, the request (r_i, Z_i) is rejected.

   If the requested zone Z_i intersects with the *pre-motion* zone $pre(Z_j)$ of any robot r_j of the system, then the request (r_i, Z_i) is rejected.

4. Rejection-Handler

   If the request (r_i, Z_i) is rejected due to a situation that belongs to the above mentioned (Request-Rejection) situations 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 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}.

Algorithm 1 State machine scheduler (Code for robot r_i)

```
1: Initialisation:
2:    PREVIOUS(RelZ_i) := ⊥; Dag_wait := ⊥;

3: procedure Request(Z_i)
4:    TO-broadcast [REQUEST, Z_i, RELEASE, PREVIOUS(RelZ_i)] {r_i TO-broadcasts a request
      of a new zone Z_i and a release of PREVIOUS(RelZ_i)}
                                   {Z_i is set to ⊥ if r_i does not acquire to move any more}

5:    when TO-Deliver [REQUEST, Z_j, RELEASE, PREVIOUS(RelZ_j)]
6:       Dag_wait := Dag_wait \ vertex(r_j, PREVIOUS(RelZ_j)) {remove the vertex representing r_j
         and its incoming edges}
7:       if Z_j intersects with the pre-motion zone pre(Z) of any robot of the system then
8:          Rejection Handler(r_j, Z_j)
9:       end if
10:      for all r_k ∈ Dag_wait do
11:         if Z_j intersects with Z_k then
12:            if Z_j intersects with post(Z_k) then
13:               Rejection Handler(r_j, Z_j)
14:            else
15:               Dag_wait := Dag_wait ∪ vertex(r_j, Z_j)
16:               Dag_wait := Dag_wait ∪ DirectedEdge(vertex(r_j), vertex(r_k))  {add a new vertex
                  with outgoing edges to the vertices whose zone intersects with Z_j}
17:            end if
18:         end if
19:      end for

20:      when the vertex of r_i in Dag_wait becomes a sink vertex (has no outgoing edges)
21:         return                {all robots r_j that r_i waits for, has released their zones}
22:      end when
23:   end when
24: end Request(Z_i)
```

4.3 Properties

In this subsection, we present the liveness properties of the scheduler.

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)

Algorithm 2 Rejection Handler (Code for robot r_i)

```
1:  Initialisation:
2:     number of trials (r_i, Z_i) := 0;

3:  procedure Rejection Handler (r_i, Z_i)

4:     number of trials (r_i, Z_i) := number of trials (r_i, Z_i) + 1 {wait a certain delay then retry the
       request (r_i, Z_i)}
5:     if (number of trials (r_i, Z_i) > MaxAllowedNumber) then
6:        if no possible alternative path then
7:           throw Exception
8:        end if
9:        Z_i := Z_i^alternative                                   {try an alternative path}
10:    end if
11: end Rejection Handler
```

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.*

4.4 Proof of correctness

In this subsection, we prove the correctness, the deadlock freedom, and the liveness properties of the scheduler.

Lemma 1 *The state machine scheduler is correct.*

PROOF.

- According to Algorithm 1, if a robot r_i executes release(r_i, Z_i) then, r_i executed reserve(r_i, Z_i) before.
- If zone Z_i of robot r_i intersects with zone Z_j of robot r_j then, either r_i waits for r_j or r_j waits for r_i. The wait-for relation is determined according to the order that the requests are delivered by the primitive TO-deliver of the total order broadcast.

  Let us assume that r_i waits for r_j, so r_j releases $RelZ_j$, after that r_i owns Z_i. When the robot r_i becomes the owner of Z_i, the robot r_j is deprived from the ownership of zone Z_j. The robot r_j just keeps a part of $post(Z_j)$ under its reservation. Z_i does not intersect with the part of $post(Z_j)$ that still reserved by r_j, because:

  1. $pre(Z_i) \cap post(Z_j) = \emptyset$ (Proof by contradiction).
     If $pre(Z_i) \cap post(Z_j) \neq \emptyset$ then, the request (r_j, Z_j) is rejected (Algorithm 1, line 8), which leads to a contradiction.

2. $Z_i \cap post(Z_j) = \emptyset$ (Proof by contradiction).
   If zone Z_i of r_i intersects with the *post-motion* zone of r_j then, the request (r_i, Z_i) is rejected (Algorithm 1, line 13), which leads to a contradiction.

All robots of the system deliver the same set of requests and releases in the same order due to the total order broadcast primitive. (The schedules of all the robots of the system are *equivalent*, see Section 3).

Consequently, all robots generate the same wait-for graph Dag_{wait}, and the ownership of intersecting zones satisfies the mutual exclusion. So, if robot r_i owns zone Z_i then, there is no robot r_j that owns a zone Z_j such that $Z_i \cap Z_j \neq \emptyset$.

Therefore, the state machine scheduler is correct.

□Lemma 1

Lemma 2 *The wait-for graph Dag_{wait} generated by a robot r has no cycles.*

PROOF.

If r_i requests a zone Z_i then, r_i must release the previous reserved zone. (Algorithm 1, line 4). So, if robot r_i waits for robot r_j then, it is impossible that r_j waits for r_i. We proceed the proof by contradiction. Let us assume that r_j waits for r_i. The vertex that represents r_j must be removed from Dag_{wait} before adding the new vertex of r_j to Dag_{wait}. Thus, r_i does not wait for r_j, which leads to a contradiction, since the assumption is that robot r_i waits for r_j.

Therefore, the wait-for graph Dag_{wait} generated by a robot r has no cycles.

□Lemma 2

Lemma 3 *The state machine scheduler is deadlock free.*

PROOF.

1. The total order broadcast ensures that the local schedules of all robots of the system are equivalent, since all the robots deliver the same set of requests and releases in the same order. Consequently all robots generates the same wait-for graph Dag_{wait}.

2. Lemma 2 proves that the wait-for graph Dag_{wait} generated by a robot r has no cycles.

3. If zone Z_i intersects with the *pre-motion* zone $pre(Z_j)$ of any robot r_j of the system then, the request (r_i, Z_i) is rejected. (Algorithm 1, line 8).

4. If zone Z_i of r_i intersects with the *post-motion* zone $post(Z_j)$ of r_j such that vertex(r_j) belongs to Dag_{wait} then, the request (r_i, Z_i) is rejected. (Algorithm 1, line 13).

Therefore, the state machine scheduler is deadlock free.

□Lemma 3

Lemma 4 *The state machine scheduler is correct and deadlock free.*

PROOF.

Lemma 1 and Lemma 3 prove that the state machine scheduler is correct and deadlock free.

□Lemma 4

Lemma 5 *If a robot r_i requests Z_i then eventually (r_i owns Z_i or an exception is raised).*

PROOF.

If robot r_i requests zone Z_i then:

1. If Z_i does not intersect with a zone Z_j, then r_i owns Z_i.

2. If Z_i intersects with a zone Z_j, then a directed edge is created between vertex(r_i) and vertex(r_j) in the wait-for graph Dag_{wait}. According to Lemma. 2 the graph Dag_{wait} has no cycles. Therefore, r_i eventually owns Z_i.

3. If request (r_i, Z_i) is rejected, then the Rejection Handler is called. If the Rejection Handler algorithm does not find a solution then, an exception is raised (Algorithm 2, line 7).

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

□Lemma 5

Lemma 6 *Exception is raised only if there is no available alternative path and the request is rejected after a certain number of trials.*

PROOF.

Exception is raised only by the Rejection Handler algorithm, and only if there is no available alternative path after rejecting the request a determined number of times. (Algorithm 2, line 7). □Lemma 6

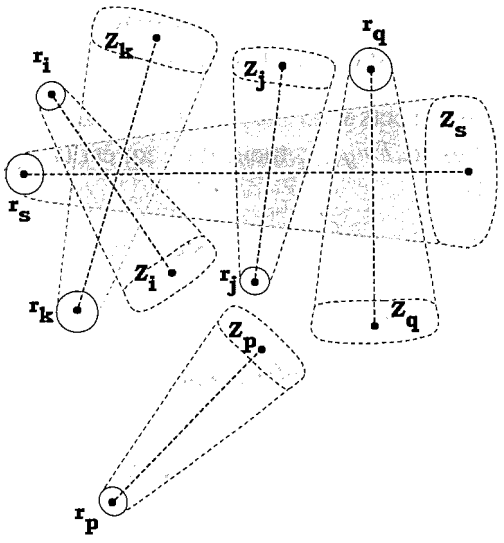

Figure 3: A group composed of six robots.

5 Example.

Consider an application composed of the following six robots (r_i, r_j, r_k, r_p, r_q, r_s).

First batch The different intersections between the requested zones are represented in Figure. 3.

Z_i intersects with (Z_k, Z_s), Z_j intersects with Z_s, Z_k intersects with (Z_i, Z_s), and Z_p does not intersect with any other requested zone.

- Each robot TO-casts a message carrying the parameters of the requested

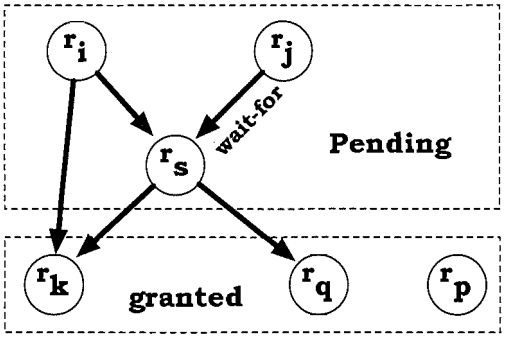

Figure 4: The wait-for graph in the first batch.

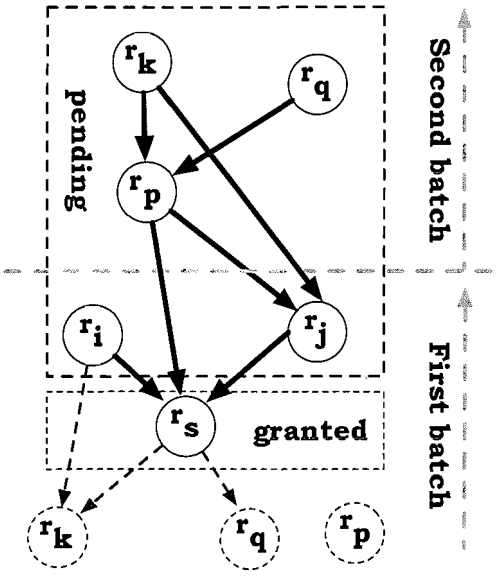

Figure 5: The wait-for graph in the second batch.

zones. The delivered messages are: [(r_k, Z_k), (r_q, Z_q), (r_s, Z_s), (r_i, Z_i), (r_p, Z_p), (r_j, Z_j)].

- We assume in this example that there is no pathological situations between the requested zones. Figure 4 represents the generated wait-for graph.
- In the wait-for graph Dag_{wait} the vertices (r_k, r_q, r_p) are *sink* vertices (has no outgoing edges), so they do not wait for any robot. Therefore, they reserve the corresponding zones, and become the owners of (Z_k, Z_q, Z_p) respectively.

Second batch Let us consider that (r_k, r_q, r_p) have reached the *post-motion* zones ($post(Z_k)$, $post(Z_q)$, $post(Z_p)$) respectively. Each of the robots r_k, r_q and r_p broadcasts a message carrying a request for a next zone Z_k and a release of the zone PREVIOUS($RelZ_k$). The intersections of the requested zones are as follows. Z_p intersects with both (Z_k, Z_q) but Z_k does not intersect with Z_q. The second batch proceeds as follows.

- Update the wait-for graph by removing the following vertices: vertex(r_p), vertex(r_q), and vertex(r_k) in addition to their incoming edges.

  We assume that each of the following zones (Z_p, Z_q, Z_k) do not intersect with a *post-motion* zone ($post(Z_s)$, $post(Z_i)$, $post(Z_j)$).

Figure 5 represents the resulting wait-for graph Dag_{wait} in the second batch.

6 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 movement are neglected.

6.1 Time needed to reserve and move along 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 of the system. So, the average number of robots that r_i waits on 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 (robots do not sleep), then the time needed to reserve and move along a chunk denoted T_{ch} is computed as the sum of the time needed by each of the following steps:

1. The delay of the total order broadcast algorithm denoted by: T_{AB}. We assume that the delay of the total order broadcast algorithm is: $T\,n$
2. The time needed for local computations by robots (to build the *wait-for* graph) is neglected.
3. 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 is: $n_{avg}(T_{com} + \frac{D_{ch}}{V_{mot}})$.
4. The time needed by r_i to move along a chunk is: $\frac{D_{ch}}{V_{mot}}$.

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

$$T_{ch} = Tn + n_{avg}(T_{com} + \frac{D_{ch}}{V_{mot}}) + \frac{D_{ch}}{V_{mot}} \quad (1)$$

So,

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

6.2 Average effective speed

In this subsection, 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. A robot r_i makes on average $\frac{D_{trip}}{D_{ch}}$ steps to move along a path of length D_{trip}. The time to progress a distance D_{trip} is:

$$T_{trip} = \frac{D_{trip}}{D_{ch}} [Tn + \frac{n-1}{2} T_{com} + (\frac{n+1}{2}) \frac{D_{ch}}{V_{mot}}] \quad (3)$$

The speed V is $\frac{D_{trip}}{T_{trip}}$. Thus, the average effective speed V is:

$$V = \frac{D_{ch}}{Tn + \frac{n-1}{2} T_{com} + (\frac{n+1}{2V_{mot}}) D_{ch}} \quad (4)$$

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.

6.3 Average effective speed vs chunk length

In this Subsection, we focus on the relation between the average effective speed and the chunk length for a given number of robots n.

The first derivative of the function effective speed with respect to the chunk length is:

$$\frac{dV}{dD_{ch}} = \frac{Tn + \frac{n-1}{2} T_{com}}{[Tn + \frac{n-1}{2} T_{com} + (\frac{n+1}{2V_{mot}}) D_{ch}]^2} \quad (5)$$

The derivative of the effective speed with respect to the chunk length is always positive. So, 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 a destination, and the number of steps is a function of the

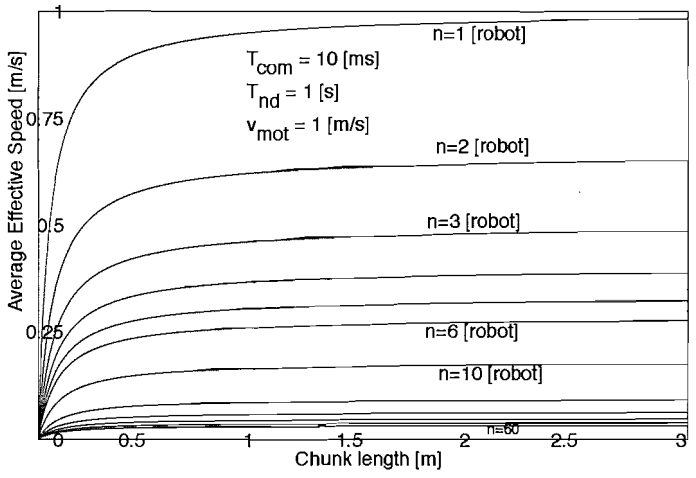

Figure 6: Average effective speed vs chunk length.

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. 4 implies that the average effective speed approaches toward the value $\frac{2V_{mot}}{n+1}$ as the chunk length tends to infinity.

$$\lim_{D_{ch}\to\infty} V = \frac{2}{n+1}V_{mot} \tag{6}$$

Figure 6 represents the relationship between the speed and the chunk length for different values of number of robots. The average effective speed of robots increases as the chunk length increases for a given number of robots, and there is an optimal value of the chunk length that maximize the average effective speed for a given number of robots. That optimal value of the average effective speed, remains constant as the chunk length getting larger than the optimal value of the chunk length. The average effective speed has a horizontal asymptote at $\frac{2V_{mot}}{n+1}$

Numerical values. The values of the fixed system parameters are: $T_{com} = 10[ms]$, the physical speed $V_{mot} = 1[m/s]$. We consider that the time required by the Total Order Broadcast algorithm is: $T\,n$ where T = 50[ms]. The values of the number of robots from one robot until 60 robots, and the chunk length varies from zero to 3 meters. The effective speed increases as the chunk length increases until it reaches a maximal value. Figure 6 shows that, in a case of a system composed of 3 robots for example, the maximal average effective speed is $0.48[m/s]$ which corresponds to optimal chunk length $\approx 2[m]$.

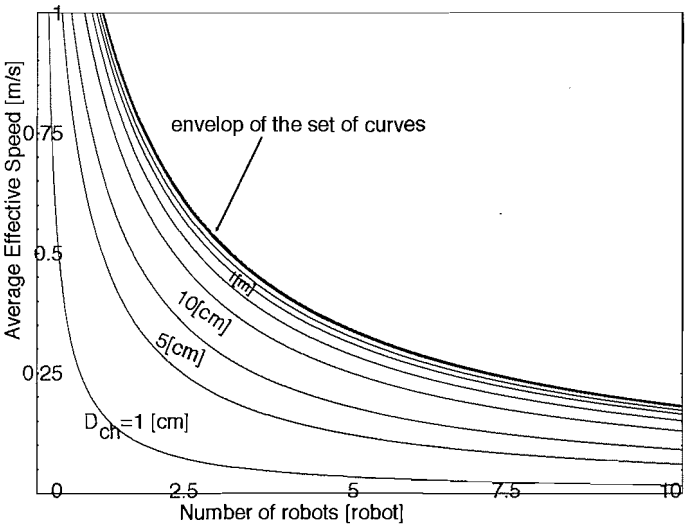

Figure 7: Average effective speed vs number of robots.

6.4 Average effective speed vs number of robots

In this Subsection, we focus on the relation between the average effective speed V with respect to the total number of robots n in the system for a given value of the chunk length. The relation average effective speed vs number of robots is presented in Equation. 4. 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.

The derivative of the effective speed with respect to the number of robots is:

$$\frac{dV}{dn} = \frac{-D_{ch}(T + \frac{T_{com}}{2} + \frac{D_{ch}}{2V_{mot}})}{[Tn + \frac{n-1}{2}T_{com} + (\frac{n+1}{2V_{mot}})D_{ch}]^2} \tag{7}$$

Figure 7 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 values of the fixed system parameters are: $T_{com} = 10[ms]$, the physical speed $V_{mot} = 1[m/s]$, $T = 50[ms]$. The values of the number of robots varies starting from a system with a single robot to a system with 10 robots, for different values of chunk length from $1[cm]$ to 10 meters. (Figure 7).

The set of curves in Figure 7 have an envelop curve, given by the following equation: $V = \frac{2V_{mot}}{n+1}$

- The envelop curve corresponds to the average effective speed for very high values of the chunk length (tends to infinity), since the average effective

speed approaches to a constant value for a given number of robots in the system.

- All curves in Figure 7 approaches to zero, when the number of robots tends to infinity. (horizontal asymptote at effective speed = 0).

7 Conclusion

In this paper, 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 in terms of average effective speed of robots as a function of the chunk length and the number of robots. The effective speed depends also on some system parameters such as the average communication delays and the physical speed of robots. The performance analysis show that the average effective speed of robots increases with the chunk length for a given number of robots, and there is an optimal value of the chunk length that maximizes the average effective speed for a given number of robots. The performance analysis show also that the maximal value of the effective speed, remains constant while the chunk length is getting larger than the optimal value. The average effective speed decreases as the number of robots increases for a given chunk length. The effective speed of robots approaches to zero as the number of robots becomes very large.

We implemented our collision prevention protocol on Pioneer 3dx robots, using Java and the library ARIA[2].

Acknowledgments

We are grateful to Nak-Young Chong, Nikolaos Galatos, Maria Gradinariu, Yoshiaki Kakuda, Takuya Katayama, Richard D. Schlichting, Yasuo Tan, Tatsuhiro Tsuchiya, and the anonymous reviewers for their insightful comments.

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

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] P. Martins, P. Sousa, A. Casimiro, and P. Veríssimo. Dependable adaptive real-time applications in wormhole-based systems. In *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'04)*, Florence, Italy, June 2004.

[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] E. Nett and S. Schemmer. An architecture to support cooperating mobile embedded systems. In *ACM Intl. Conf. on Computing Frontiers (CF'04)*, pages 40–50, Ischia, Italy, April 2004.

[7] F. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys.*, 22(4):299–319, 1990.

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

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