Locality-preserving distributed path reservation protocol for asynchronous cooperative mobile robots

Rami Yared, Julien Cartigny, Xavier Défago, and Matthias Wiesmann

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

February 13, 2006

IS-RR-2006-003

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

Locality-preserving distributed path reservation protocol for asynchronous cooperative mobile robots

Rami Yared*, Julien Cartigny*, Xavier Défago*,†, Matthias Wiesmann*

*School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

†PRESTO, Japan Science and Technology Agency (JST)

Email: {r-yared, cartigny, defago, wiesmann}@jaist.ac.jp

Abstract

Many interesting applications of mobile robotics envision groups or swarms of robots cooperating toward a common goal. Due to their inherent mobility and limited energy resources, it is only natural to consider that the robots form a mobile ad hoc network (MANET) on which they can rely for their communication. Cooperation is however difficult to obtain under the weak communication guarantees offered by MANETs. In this paper, we focus on a very fundamental cooperation problem, namely, preventing the robots from colliding against each other in a fully decentralized manner.

This paper presents a distributed path reservation system for a group of "blind" mobile robots. The protocol assumes a mobile ad hoc network formed by the robots themselves, and takes advantage of the inherent locality of the problem in order to reduce communication. In contrast with other work, our protocol requires neither initial nor complete knowledge of the composition of the group. The protocol makes only very weak timing assumptions regarding both communication and movement, and relies instead on a well-defined neighborhood discovery primitive.

1. Introduction and related work

We consider an asynchronous distributed system composed of cooperative autonomous mobile robots forming a mobile ad hoc network MANET on which they can rely for their communication. In such a system there are no upper bounds on the communication delays or on the speed of robots thus it is impossible for a robot to keep track of other robots in the system.

We aim at resolving the collision freedom problem by providing a distributed path reservation protocol for a group of blind mobile robots in this system. This protocol takes advantage of the inherent locality of the problem. Our protocol guarantees deterministically the collision freedom between robots as a safety property, so no collision between robots can occur during the execution of the protocol.

1.1. Related work

TCB wormholes Martins et al in ([6] and [7]) provide an implementation of a system based on Timely Computing Base (TCB) [14], designed for a wireless mobile system used in their practical demonstration, such that the local TCB components allow detecting time failures in a timely manner. They demonstrated, under certain conditions, the fundamental mechanisms underlying the construction of adaptable real-time applications with the help of TCB by providing an emulation of 3 balls, each of them is controlled by a *sentient* object running in an application trying to avoid collisions among the balls. In this scenario a sentient object controlling a ball disseminates

an event with the ball position, speed and direction, thus each object constructs a real-time image of the reality. These events should be timely delivered to achieve the real-time image and avoid collisions. If timing failures are being detected then the system stops or the balls speed is adjusted, also a timely change of a ball direction may avoid a possible collision.

Cooperative mobile applications Nett et al. [11] present a layered system architecture for cooperative mobile systems in real-time applications. Coordination is the key to achieve the required effective cooperation, which means that individual systems can take actions under local control based on a sufficiently consistent view of the common system environment. The system model is synchronous assuming the existence of a known constant upper bound on the communication delays. The communication hardcore of their architecture presented in [10] and [11] is based on a wireless LAN and the designed communication protocols use the access point as a central router since each station participating in the group communication protocols must be within the reach of the access point. The communication hardcore can manage real-time constraint requirements. Nett et al. [11] considered a traffic control application explained in [12] as a testbed of their system architecture. In this testbed a group of mobile robots are driving along traces that form 2 closed loops. The two loops overlap such that there is a shared space situation with the need for cooperation of mobile systems. Each robot near the shared zone triggers an event to the Cooperative Application Development Interface (CADI) layer of the system architecture which offers each robot near the shared zone a consistent view of the global state of the system, (i.e. the positions and velocities of all the robots near the shared zone with respect to the same point of time) based on the QoS provided by the communication hardcore. Therefore, robots can schedule their access to the shared zone by adjusting their velocities in order to use it efficiently and to avoid collisions.

Robotics research The problem of robots collision avoidance has been handled using different strategies which are sensor-based motion planning methods. Minguez et al. [8] compute collision-free motion for a robot operating in dynamic and unknown scenarios, also they survey the existing collision avoidance navigation approaches. Motion planning algorithms consider a model of the environment (either previously known or dynamically built), to compute a collision free path between the current robot location and the goal. In dynamic or unknown environments the trajectories generated by motion planning algorithms become inaccurate thus they can not be applied to such environments. Solving this problem involves sensing directly within the motion planning by applying a *perception-action* process that is repeated periodically at a high rate. These approaches use a local fraction of the information available (sensory information), so they can fall into trap situations. Some of these approaches apply mathematical equations to the sensory information and the solutions are transformed into motion commands. (e.g., [9]). Another group of methods compute a set of suitable motion commands to select one command based on navigation strategy (e.g., [13]), while other methods (e.g., [8]), compute a high level information description (entities near obstacles, areas of free space), from the sensory information then apply different techniques simplifying the difficulty of the navigation to obtain a motion command in complex scenarios.

2. System model

We consider dynamic system of mobile robots. The number of robots is unknown. Each robot is identified by a unique identifier. Each robot has a positioning system that returns the robot's position with a bounded error. There is no bound in the time needed to compute the robot's position. We do not consider robot failures in this model.

The robots are linked by an ad-hoc network and communicate by exchanging messages. The topology of the network is unknown. All the robots have a limited communication range, denoted C. If the distance between two robots R_i and R_j is less than C, R_i can receive messages sent by R_j and robot R_j can receive messages sent by R_i. Communications channels are reliable but non FIFO. There is no bound on the time needed to process and transmit a message

3. Neighborhood discovery (*Discover*)

Requirements. Our proximity detection primitive called *Discover* enables a robot to detect its local neighbors that exist in a proximity of one communication hop and satisfying a certain known predefined condition. The implementation of this primitive is based on geocasting a ping message in a geographical area centered on the robot at the

time of calling *Discover* with a radius equals to the communication range, then *all* the robots located in that geographical area and satisfying the condition must acknowledge the caller of *Discover*. Therefore, it can determine its requested group of robots.

Difference with Geocast protocols. Geocasting protocols in Mobile Ad hoc Networks (MANETs) have been studied and analyzed in different papers (e.g., [4], [16], [5]). In [4] a flooding-based geocasting protocols have been proposed, while routing-based geocast protocols were presented in other works (e.g. [3]), and others propose cluster-based geocast protocols (e.g. [1]). These protocols relied on strategies and techniques to improve the performance of the geocast delivery in terms of accuracy and message overhead. Geocasting protocols were aimed at different problem than that of our primitive *Discover* with different requirements.

Node presence detector. Detecting the presence of nodes in an asynchronous system where there are no bounds on communication delays, cannot be solved *deterministically.* [2] The impossibility is based on the fact that, in such systems, a very slow node can not be distinguished from an absent one. Thus, a timing property is required to implement the primitive *Discover*.

Timing property. The primitive *Discover* relies on the following timing property: there exists a known upper bounded time delay called (Δ) such that the following property holds:

Property 1 (Delay) $\forall$ *instant of time t,* $\forall$ R_i*, If* R_i *starts* $Discover_i$ *at (t), then* R_i *receives an acknowledgment from any* R_j *located in its communication range and satisfying a certain predefined condition, at (t') such that:* $t' - t \leq \Delta$

Timing characteristics of MANETs. Providing a mobile node with access to the wireless medium in a timely manner is well known subtle problem. There are some proposals to enhance MANETs with real-time capabilities, but as far as we know they are not implemented.

Timely Computing Base concept. We base on the concept of Timely Computing Base (TCB) wormhole-based systems introduced by P. Veríssimo and A. Casimiro in ([14], [15]) to build our proximity detection primitive. In TCB based systems there exist some parts which are more predictable than others. These more predictable parts can be seen as wormholes (they are comparably small parts of the system), through which it is possible to do things much faster than apparently possible in the other parts of the system. The architecture of a system with a TCB has a *payload* part, in which protocols and application processes are executed. Communications take place over a global network (payload channel). The system has also a *control* part, made of TCB modules in each node, interconnected by a medium called *control* channel. In a TCB based system the payload part can have any degree of synchronism, on the other hand the TCB subsystem (control part) provides some synchronism properties such as known upper bounds on message delivery delays. A practical demonstration of a TCB system is presented in Martins et al. [6] and [7]).

Design of the primitive Discover. Our primitive *Discover* relies on a special communication hardcore (control network) in which it uses a distinct radio channel from the channel used to exchange messages in the remainder (main) part of our protocol (payload network). (illustrated in Fig .**??**). The ping messages used by the primitive *Discover* are very light weight messages. The information carried by a ping "*Discover*" message is just the position coordinates of the robot caller, while a messages used in the protocol have generally a large size carrying information about the parameters of a complex geographical zone or carrying a long list of integer numbers representing robots identifiers. (see Section **??**) Therefore, a timely behavior of *Discover* can be ensured since it is based on broadcasting light weight ping messages to a one hop proximity area over a dedicate network. This behavior meets the requirements to achieve this primitive.[1] [2]

4. Definitions

Definition 1 (Chunk, Path) *Ch: a chunk is a line segment along which a robot moves. P: a path is a continuous route composed of a series of contiguous chunks.*[3]

1 The compensation of the draw back of the hidden terminal problem on the time-bounded access to the wireless medium is out of scope of this paper

2 *Discover* geocasts (broadcast) a ping message that is supposed to cover all the robots located in a circular area whose radius is the communication range C. We do not address in this paper, the very special case where a robot existing in the given area, is not covered by the wave of *Discover*.

3 A path can take an arbitrary geometric shape, but in this paper we consider only line segment based paths for simplicity without lack of generality.

Paths are planed and provided by an upper layer to the protocol (application layer).

Definition 2 (Geometric Incertitude) *The are 3 sources of errors concerning the position and the motion of a robot:*

1. *Error related to the position information provided by the positioning system denoted by ε_{gps}. In addition, the motion of a robot creates 2 additional sources of errors:*
2. *Error related to the translational movement, which is denoted by: ε_{tr}.*
3. *Error related to the rotational movement, which is denoted by: ε_{θ}.*

Definition 3 (Zone) *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 have the following properties:

- be a convex shape
- contain the chunk the robot is following

5. Collision freedom protocol

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 where the robot is waiting. When moving along a path, the robot repeats this procedure for each chunk along the path.

Definition 4 (Reservation Zone) *The reservation range specifies that a robot R_i can only reserve zones that are entirely within a circle centered on the robot at the time of reservation with a radius equals at most to the half the transmission range. $Zone_{.,i} \subset Cir_i(pos_i, \frac{C}{2})$*

5.1. Structure of the reservation zone

The reservation zone for robot R_i is composed of the following three sub zones, illustrated by (Fig. 1)

1. *Pre-Motion* Zone: $Pre(Zone)$ the zone that robot R_i may occupy while waiting (not moving) until it reserves $Zone$.
2. *Post-Motion* Zone: $Post(Zone)$ the zone that robot R_i may occupy after the motion, while waiting until reserves the next zone.
3. *Motion* Zone: $Mot(Zone)$ the zone that robot R_i occupy while moving.

5.2. Definitions and notations

Relation between robots and a zone. The relationship between robots and zone changes in time. A zone is said to be free if it is not owned by any robot. A robot requests a zone, and eventually it is granted this zone. We say the robot owns the zone and all the points contained in this zone. When the robot has left a zone, it releases the zone. A given point can be owned by only one robot.

We define the following relations between robots and zone:

Definition 5 (Zone (released)) *It is the zone released by a robot R_i. A robot releases its reserved zone and keeps only $Post(Zone)$ under its reservation.*

$RelZone_{.,i} = Zone_{.,i} \setminus Post(Zone_{.,i})$

Definition 6 (Zone (owned)) *R_i is the owner of $Zone_{.,i}$ if R_i reserves $Zone_{.,i}$ and does not yet release the zone $RelZone_{.,i}$*

$(R_i = Own(Zone_{.,i})) \iff Reserve(R_i, Zone_{.,i}) \wedge \neg Release(R_i, RelZone_{.,i})$

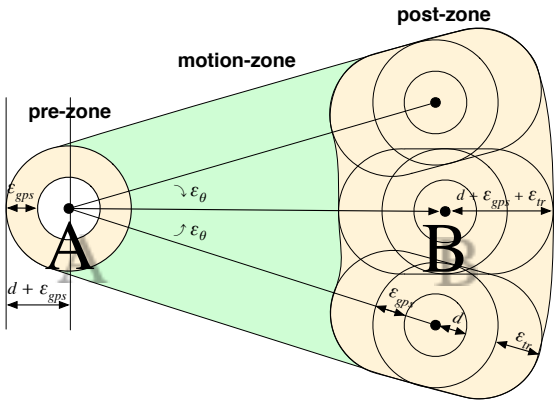

Figure 1. Reservation Zone.

Relation between robots. We introduce the following definitions related to possible relations between robots during the run of the protocol:

Definition 7 (Conflict) *R_i and R_j are in "Conflict" situation if both requested zones overlap.*
The Conflict relation is expressed as follows:
Conflict (R_i, R_j) $\iff (S_\cap \neq \emptyset) \wedge (S_\cap \cap Pre(Zone_{m,j}) \cap Post(Zone_{m,j}) \cap Pre(Zone_{k,i}) \cap Post(Zone_{k,i}) = \emptyset)$

Definition 8 (Risk) *R_i is in "Risk" situation with R_j if the requested zone by R_i overlaps either with* Pre-Motion *zone or with* Post-Motion *zone of R_j.*
In other words, R_i is in "Risk" situation with R_j $\iff$ *the requested zone by R_i overlaps with a brake zone of R_j.*
Risk(R_i, R_j) $\iff (Zone_{.,i} \cap Pre(Zone_{.,j}) \neq \emptyset) \vee (Zone_{.,i} \cap Post(Zone_{.,j}) \neq \emptyset)$

We define $Risk_{pre}$ and $Risk_{post}$ as follows:

- $Risk_{pre}(R_i, R_j) \iff Zone_{.,i} \cap Pre(Zone_{.,j}) \neq \emptyset$
- $Risk_{post}(R_i, R_j) \iff Zone_{.,i} \cap Post(Zone_{.,j}) \neq \emptyset$

Definition 9 (Turn) *The* Turn *of a robot R_i is defined as its rank order to reserve $Zone_{.,i}$, (precedence relations) with respect to a conflicting group of robots.*
If (Conflict(R_i, R_j) then R_i precedes *R_j* $\iff$
Release(R_i, $RelZone_{.,i}$) happens before *Reserve(R_j, $Zone_{.,j}$).*

In other words, *R_i precedes R_j* $\iff Turn(R_i) < Turn(R_j)$ denoted by: $T(R_i) < T(R_j)$

5.3. Variables

We present the variables used in the collision freedom protocol:

- $Zone_{.,i}$ is the current requested or the current reserved zone by R_i
- G_i represents the group of robots that replied to $Discover_i$ (i.e. the robots such that the requested zone or the reserved zone when received $Discover_i$) overlaps with the reservation range of R_i
  $G_i = \{R_j : Zone_{.,j} \cap Cir_i(pos_i, \frac{C}{2}) \neq \emptyset$ when received $Discover_i\}$
- $WLAfter_{i\rightarrow}$ is the list of robots waiting for R_i since it precedes all the members of this list.
  $WLAfter_{i\rightarrow} = \{R_j : T(R_i) < T(R_j)\}$
- $WLAfter_{\rightarrow i}$ is the list of robots that R_i is waiting for, since all the members of this list precede R_i
  $WLAfter_{\rightarrow i} = \{R_j : R_i \in WLAfter_{j\rightarrow}\}$
- Overlap_i is the list of robots that conflict with R_i.

- Depend$_i$ is the list of dependencies of R_i that contains the requests (R_j, $Zone_{.,j}$) such that $Zone_{.,j}$ overlaps with $Zone_{.,i}$ and also the dependencies of each request (R_j, $Zone_{.,j}$).

  We define the relation "depend" denoted $\sim$ as follows: $R_i \sim R_j \iff (Zone_{.,i} \cap Zone_{.,j} \neq \emptyset \vee \exists$ a finite sequence (R_k, R_{k+1}, ..., R_{k+n}) such that $Zone_{.,i} \cap Zone_{.,k} \neq \emptyset \wedge \forall\ 0 \leq p \leq n-1$, $Zone_{.,k+p} \cap Zone_{.,k+p+1} \neq \emptyset \wedge Zone_{.,k+n} \cap Zone_{.,j} \neq \emptyset$)

5.4. Protocol description

All robots run the same protocol. We explain the phases of the protocol with respect to R_i. The protocol can be decomposed in four phases:

1. *Discover*: R_i calls the neighborhood discover primitive $Discover_i$, then R_i determines the group G_i.
2. Negotiate phase: The goal of the Negotiation phase is that R_i determines the conflicting robots with it, then R_i builds the graph of the precedence relations to determine the group of robots that precede it and the group of robots that pass after it. The vertices of the graph represent the conflicting robots and the directed edges represent the precedence relations. The Negotiation phase can be summarized by the following steps:
   - R_i multicasts a message req$_i$ to all the members of G_i carrying all the parameters of $Zone_{.,i}$.
   - R_i waits until it receives either a $Discover_j$ or a message msg$_j$ from each robot $R_j \in G_i$.
   - The group G_i contains two disjoint subgroups of robots: the first subgroup denoted $G1_i$ is composed of robots R_j such that R_i does not belongs to G_j and a second subgroup denoted $G2_i$ is composed of robots R_j such that R_i belongs to G_j. In other words, the first subgroup $G1_i$ denotes the robots R_j such that $Discover_i$ happened after they determine G_j, while the second subgroup $G2_i$ denotes the robots R_j such that $Discover_i$ happened before they determine G_j. When a robot R_i calls the primitive $Discover_i$ then all robots that may collide with R_i receives $Discover_i$ and reply to it, consequently R_i can determine its group G_i. After that if R_i receives a request from a robot R_k then the requested or the reserved zone by R_k when it received $Discover_i$ was not overlapping with the reservation range of R_i but the current one overlaps.
   - R_i receives a message "AfterMe" from robots in the group $G1_i$ if the requested zone $Zone_{.,i}$ overlaps with $Zone_{.,j}$, otherwise R_i receives "OK" message from robots of $G1_i$. R_i constructs the list Overlap$_i$ by adding the R_j to the list with the dependency type ("AfterMe").
   - R_i receives a message msg$_j$ that contains the parameters of the requested zone $Zone_{.,j}$ from robots in the group $G2_i$ if $Zone_{.,j}$ and $Zone_{.,i}$ overlaps, otherwise msg$_j$ indicates "OK". R_i updates its Overlap$_i$ list by adding R_j.
   - After R_i has determined its Overlap$_i$ list, it multicasts the Overlap$_i$ list to each robot in that list, and it waits until receive the Overlap$_j$ from each robot R_j that belongs to Overlap$_i$, so it builds the list Depend$_i$ which contains the group of robots that conflict with R_i in addition to the conflicting robots with each element of that group. The motivation behind exchanging the dependency list is that all the conflicting robots can build the same Directed Acyclic Graph Dag_{res} of the precedence relations consistently.
   - R_i constructs from the list Depend$_i$ the graph $Dag_{AfterMe}$ related to "AfterMe" precedence relation, (see 5.5) and then it adds the precedence relations related to Risk constraints (see 5.6), thus, it constructs the graph Dag_{risk} composed with the graph $Dag_{AfterMe}$. If a cycle is created by adding a directed edge to Dag_{risk} then an exception is raised to the upper layer that can make a decision to resolve the conflict.

     After that, R_i constructs the final precedence relations graph Dag_{res} that contains $Dag_{AfterMe}$ and Dag_{risk}. The construction of the final graph Dag_{res} is done by adding directed edges according to a sorting criteria specified by the application layer (see 5.7). If adding a directed edge creates a cycle in Dag_{res} then the orientation of the directed edge is reversed which breaks the cycle. Dag_{res} is built starting from the vertex that represents the robot of the minimal identifier in the increasing order of the robots identifiers. The idea of scanning the vertices according to a specified predetermined order is to ensure that R_i and all the members of Depend$_i$ build the same graph Dag_{res} in a consistent manner.
   - After building the precedence relations graph Dag_{res}, R_i determines the group of robots that precede it $WLAfter_{\rightarrow i}$ and the group of robots that pass after it $WLAfter_{i\rightarrow}$.

- R_i waits until receiving the release messages from all the members of the group that precede it $WLAfter_{\rightarrow i}$.

The modular design of our protocol yields a flexibility concerning the ordering techniques and strategies depending on the application requirements. To determine the turn relation between two robots it is possible to consider their distances to the intersection zone and in case of equidistance situation then the identifiers are considered.

3. Reserve phase: This phase starts when R_i receives a release message from all its predecessors, then it reserves $Zone_{.,i}$ and becomes $Own(Zone_{.,i})$. Since R_i reserves its requested zone it moves inside it.
4. Release phase: When R_i finishes moving inside its reserved zone $Zone_{.,i}$, it multicasts a release message to all its followers (members of the group $WLAfter_{i\rightarrow}$).

Algorithm 1 Collision Avoidance Protocol

1: $Discover_i \Rightarrow G_i$
2: $\text{Overlap}_i := \emptyset$
3: $\text{Depend}_i := \emptyset$
Thread Reply
4: reply to the primitive *Discover* sent by other robots.
5: **if** Receive request($Zone_{.,k}$) from $R_k \notin G_i$ **then**
6: **if** Conflict(R_k, R_i) **then**
7: Send("AfterMe" + $Zone_{.,i}$, R_k)
8: $WLAfter_{i\rightarrow} := WLAfter_{i\rightarrow} \cup \{R_k\}$
9: **end if**
10: **end ifEnd of the thread Reply**
Negotiate phase
11: Multicast (request($Zone_{.,i}$), G_i)
12: **Wait until** Receive(msg_j) from all $R_j \in G_i$
13: **if** $R_i \notin G_j \wedge Zone_{.,j} \cap Zone_{.,i} \neq \emptyset$ **then**
14: Receive("AfterMe" + $Zone_{.,j}$) {R_j *has determined its* G_j *before* $Discover_i$}
15: $WLAfter_{\rightarrow i} := WLAfter_{\rightarrow i} \cup \{R_j\}$
16: $\text{Overlap}_i := \text{Overlap}_i \cup (R_j, Zone_{.,j}, type := "AfterMe")$
17: **end if**
18: **if** $R_i \in G_j \wedge Zone_{.,j} \cap Zone_{.,i} \neq \emptyset$ **then**
19: $\text{Overlap}_i := \text{Overlap}_i \cup (R_j, Zone_{.,j})$
20: **end if**
21: $\text{Depend}_i := \text{Overlap}_i$
22: **for** all $R_j \in \text{Overlap}_i$ **do**
23: Multicast(Depend_i, R_j) {*disseminate the dependencies*}
24: **end for**
25: **Wait until** Receive Depend_j from all $R_j \in \text{Overlap}_i$
26: $\text{Depend}_i := \text{Depend}_i \cup \text{Overlap}_j$
27: $Dag_{AfterMe} := \text{AfterMeHandler}(\text{Depend}_i)$
28: $Dag_{risk}(R_i) := \text{Risk Handler}(Dag_{AfterMe}, \text{Depend}_i)$
29: $Dag_{res}(R_i) := \text{Conflict Resolver}(Dag_{risk}(R_i), \text{Depend}_i, \text{Sorting-Criteria SC})$
30: Update($WLAfter_{\rightarrow i}$, $WLAfter_{i\rightarrow}$)
Reserve phase
31: **when** reception of release messages from all members of $WLAfter_{\rightarrow i}$
32: Reserve(R_i, $Zone_{.,i}$)
33: **end when**
34: Move()
Release phase
35: **when** Release(R_i, $RelZone_{.,i}$)
36: Multicast a release message to members of: $WLAfter_{i\rightarrow}$
37: **end when**

5.5. AfterMe Handler

This handler generates a graph of precedence relations concerning the relation "AfterMe". The input of AfterMe Handler is the list of dependencies of R_i Depend_i and the output is the Directed Acyclic Graph $Dag_{AfterMe}$ such that its vertices represent the robots related by the relation "AfterMe" and each directed edge represents a precedence relation between a couple of robots R_x and R_y. If R_y sent a message "AfterMe" to a R_x then a directed edge (R_y, R_x) is added to the graph.

If R_j sent a message "AfterMe" to R_i then R_i should be after R_j and after any R_k predecessor to R_j. Consequently, no circular precedence relationship can occur when generating $Dag_{AfterMe}$.

Algorithm 2 AfterMe Handler algorithm

```
1: procedure AfterMe Handler(Depend_i)
2:   for all (R_x, R_y) ∈ Depend_i do
3:     if DependType= "AfterMe" then
4:       Dag_AfterMe := Dag_AfterMe ∪ DirEdge(R_y ⟶ R_x)        {R_y sent "AfterMe" message to R_x}
5:     end if
6:     if DirEdge(R_x, R_y) and DirEdge(R_y, R_z) then
7:       Dag_risk := Dag_risk ∪ DirEdge(R_x ⟶ R_z)   {if R_y sent "AfterMe" message to R_z then R_z should be after any R_x
         that precedes R_y}
8:     end if
9:   end for
10: end
```

5.6. Risk Handler

The Risk Handler is responsible for handling the situations concerning the "Risk" relation (i.e. when there is an overlapping between a requested zone by a robot R_i with a *Pre-Motion* or a *Post-Motion* zone of another robot R_j).

The input of the Risk Handler of R_i is the list of dependencies of R_i and $Dag_{AfterMe}$ generated by the AfterMe Handler, since Dag_{risk} represent both Risk and AfterMe relations. Risk Handler delivers: either the graph $Dag_{risk}(R_i)$ which determines the precedence relations between robots in "Risk" situation with R_i, or raising exceptions whenever a detectable deadlock situation occurs or a circular precedence relationship is detected.

The precedence relations of Dag_{risk} is specified according to the constraints of the "Risk" situation only. A directed edge between two vertices (R_x, R_y) in Dag_{risk} means that R_x is in Risk situation with R_y and that R_x must precede R_y because of "Risk" constraints. When a circular precedence relationship is created by adding a directed edge, or a deadlock situation is detected then the appropriate exception is raised to the upper layer because Risk Handler can not modify the precedence relations due to the constraints of "Risk" situation. A circular precedence relation may occur while generating Dag_{risk} as follows: there is a directed edge (R_j, R_i) due to "AfterMe" relation, and a directed edge (R_i, R_j) is going to be added to Dag_{risk} because of a Risk situation. In a such a case an appropriate exception is raised to the upper layer.

Description of the Risk Handler

- $\text{Risk}_{pre}(R_i, R_j)$ means that $Zone_{.,i}$ overlaps with $Pre(Zone_{.,j})$. If $Zone_{.,i}$ overlaps with $Pre(Zone_{.,j})$ only, then R_j *must* precede R_i.
- $\text{Risk}_{post}(R_i, R_j)$ means that $Zone_{.,i}$ overlaps with $Post(Zone_{.,j})$. If $Zone_{.,i}$ overlaps with $Post(Zone_{.,j})$ only, then R_i *must* precede R_j.
- A detectable deadlock situation (e.g., Deadlock_3) occurs if both $\text{Risk}_{pre}(R_i, R_j)$ and $\text{Risk}_{post}(R_i, R_j)$. In this case $Zone_{.,i}$ overlaps with both $Pre(Zone_{.,j})$ and $Post(Zone_{.,j})$. Consequently, neither R_i nor R_j can move, therefore a detectable *deadlock* situation occurs. Other detectable deadlock situations (Deadlock_1, Deadlock_2, Deadlock_4) are deduced by a similar reasoning.
- Dag_{risk} is constructed by adding directed edges between vertices according to the constraints of "Risk".
- Detectable deadlock situations "direct" between two robots can occur in the system are explained in the Risk Handler algorithm (Deadlock_1, Deadlock_2, Deadlock_3, Deadlock_4) situations. They are caused by overlapping between zones in special cases.
- Other detectable deadlock situations "indirect" can happen because of circular precedence relationship in the Dag_{risk}. In such a case an exception is raised since it is not possible to change the direction of any edge in Dag_{risk}

Algorithm 3 Risk Handler algorithm

1: **procedure Risk Handler**($Dag_{AfterMe}$, Depend_i)
2: **for** all $(R_x, R_y) \in \text{Depend}_i$ **do**
3: $\text{Deadlock}_1 = \text{Risk}_{pre}(R_x, R_y) \wedge \text{Risk}_{pre}(R_y, R_x)$
4: $\text{Deadlock}_2 = \text{Risk}_{post}(R_x, R_y) \wedge \text{Risk}_{post}(R_y, R_x)$
5: $\text{Deadlock}_3 = \text{Risk}_{pre}(R_x, R_y) \wedge \text{Risk}_{post}(R_x, R_y)$
6: $\text{Deadlock}_4 = \text{Risk}_{pre}(R_y, R_x) \wedge \text{Risk}_{post}(R_y, R_x)$
7: **if** $\text{Deadlock}_1 \vee \text{Deadlock}_2 \vee \text{Deadlock}_3 \vee \text{Deadlock}_4$ **then**
8: Raise Exception "what to do?" to the upper layer.
9: **end if**
10: **if** $\text{Risk}_{post}(R_x, R_y) \vee \text{Risk}_{pre}(R_y, R_x)$ **then**
11: $Dag_{risk} := Dag_{risk} \cup \text{DirEdge}(R_x \longrightarrow R_y)$
12: **end if**
13: **if** $\text{Risk}_{pre}(R_x, R_y) \vee \text{Risk}_{post}(R_y, R_x)$ **then**
14: $Dag_{risk} := Dag_{risk} \cup \text{DirEdge}(R_y \longrightarrow R_x)$
15: **end if**
16: **if** $\text{DetectCycle}(Dag_{risk}(R_i))$ **then**
17: Raise Exception "what to do?" to the upper layer.
18: **end if**
19: **end for**
20: **end**

All robots generate the same Dag_{risk}, since all robots know the dependencies of other robots.

The possibility that a deadlock situation happens is due to different parameters, such as the geometrical characteristics of the reservation zone. In some situations, deadlock cases can be reduced or canceled by a careful path planning managed by the application layer.

The protocol raises an exception to the upper layer in order to handle such situations. The upper layer decides an action which can be an alternative paths provided to robots.

5.7. Conflict Resolver

The Conflict Resolver of R_i constructs $Dag_{res}(R_i)$ which separates the set of the robots conflicting with R_i in two subsets: the predecessors and the successors of R_i. The input of the Conflict Resolver is the following: the dependencies of R_i (Depend_i), Dag_{risk} created by the Risk Handler, and a sorting criteria (SC) provided by the application layer.

Description of the Conflict Resolver algorithm The goal is to determine the precedence relations betweenR_i and the conflicting robots with R_i. This is performed by sorting each couple of the conflicting robots (R_i and its dependencies) according to the sorting criteria SC. When a circular precedence relationship is created then the precedence relation between the couple of robots that caused the cycle is reversed to break this circular precedence situation.

Conflict Resolver permits to build $Dag_{res}(R_i)$ in a consistent manner such that R_i and the conflicting robots with R_i build the same $Dag_{res}(R_i)$. To achieve the consistency, each robot builds Dag_{res} as follows:

- R_i adds a directed edge to Dag_{res} between two vertices (R_x, R_y) according to SC, starting from the vertex that represents the robot of minimal identifier among the set of dependencies of R_i, compared with the remainder of the vertices in the increasing order of their identifiers.
- If adding the directed edge (R_x, R_y) creates a cycle in Dag_{res} then replace the directed edge (R_x, R_y) by the directed edge (R_y, R_x) to break the circular precedence relationship.

6. properties of our protocol

The properties achieved by our protocol are the following:

Algorithm 4 Conflict Resolver algorithm

```
1: procedure Conflict Resolver(Dag_risk(R_i), Depend_i, Sorting-Criteria SC)
2:   if SC == "NULL" then
3:     Run a consensus algorithm   {If there is no a specified sorting criteria, then the conflicting robots run a consensus algorithm}
4:   else
5:     Dag_res(R_i) := Dag_risk(R_i)
6:     for (x := MinID; x ≤ MaxID; R_x ∈ Depend_i) do
7:       for (y > x; y ≤ MaxID; R_y ∈ Depend_i) do
8:         if Conflict(R_x, R_y) then
9:           DirEdge(R_x, R_y) := Sort_SC(R_x, R_y)   {sort the 2 robots according to the Sorting Criteria SC}
10:          Dag_res(R_i) := Dag_res(R_i) ∪ DirEdge(R_x, R_y)
11:          if DetectCycle(Dag_res(R_i)) then
12:            DirEdge(R_x, R_y) := DirEdge(R_y, R_x)   {If a cycle is detected then inverse the direction of the edge}
13:          end if
14:        end if
15:      end for
16:    end for
17:  end if
18: end
```

6.1. Safety property

The collision freedom property is the Safety property of our protocol, so no collision between robots can occur. This property is expressed as follows:

Property 2 (Mutual exclusion) *If the requested zone of R_i overlaps with that of R_j then one and only one of them becomes "Owner" to its required zone. In other words, a zone in the plan can be owned by one and only one robot.*

$(Zone_{.,i} \cap Zone_{.,j} \neq \emptyset) \Rightarrow (R_i = Own(Zone_{.,i}))$ **XOR** $(R_\mathbf{j} = Own(Zone_{.,j}))$

Property 3 (Integrity) *A robot can release a zone if and only if it has previously reserved this zone. This property is expressed as follows:*

For any zone $Zone_{.,i}$, $Release(R_i, RelZone_{.,i})$ takes place only if previously: $Reserve(R_i, Zone_{.,i})$

$Reserve(R_i, Zone_{.,i})(happensbefore)Release(R_i, RelZone_{.,i})$

6.2. Liveness properties

Property 4 (No Circular Precedence Relationship) *Our protocol has no circular precedence relationship, consequently if R_i requests $Zone_{.,i}$ then it eventually reserves it, so no starvation occurs unless one of the detectable deadlock situations explained in the Risk Handler section (see 5.6). Dealing with detectable deadlock situations takes place in the upper layers, our protocol present the lowest layer that handles the safety property of the system.*

Resolving the detectable deadlock situations in the upper layers can be carried out by different strategies such as offering an alternative path or an alternative chunk of a certain path (motion planning).

7. Advantages and improvements

We mention some advantages of our locality preserving protocol:

1. A robot R_i can use its maximal speed to move along a reserved chunk since it is deterministically guaranteed that R_i is exclusively the only owner of the reserved zone.
2. The protocol does not require any routing techniques except for avoiding circular precedence relationships (deadlock situations).
3. It is possible to improve the performance in terms of time that a robot R_i requires to reach its destination by adjusting the reservation range according to the local density of robots around R_i returned by the neighborhood discovery primitive $Discover_i$

4. By adding a mechanism that enables a robot R_i to demand an alternative path or alternative chunk from an upper layer (path planner layer) a tangible improvement can be performed as follows:
   R_i can "hear" gratuitously the required zones by its local neighbors due to the nature of communications in Mobile Ad Hoc networks. If the requested zone $Zone_{.,i}$ conflicts with a certain number of requested zones by the local neighbors then R_i demands an alternative chunk or an alternative path where the congestion is lower.

8. Conclusion

We introduced a collision freedom protocol for asynchronous cooperative mobile robots based on a distributed path reservation system. Our protocol takes advantages of the inherent locality of the problem.

This protocol offers a dynamic scheduling mechanism for zone reservation requests coming dynamically in this system, our protocol copes with this dynamic context, relying on a dynamic mechanism to manage the order of requests in a consistent manner.

Our protocol requires neither initial nor complete knowledge of the composition of the group. It is based on a well-defined neighborhood discovery primitive and makes very weak timing assumptions.

References

[1] Chih-Yung Chang, Chao-Tsun Chang, and Shin-Chih Tu. Obstacle-free geocasting protocols for single/multi-destination short message services in ad hoc networks. *Wireless Networks*, 9(2):143–155, 2003.

[2] Fischer, Lynch, and Paterson. Impossibility of distributed consensus with one faulty process. *JACM: Journal of the ACM*, 32, 1985.

[3] Young-Bae Ko and Nitin H. Vaidya. GeoTORA: A protocol for geocasting in mobile ad hoc networks. In *ICNP*, page 240, 2000.

[4] Young-Bae Ko and Nitin H. Vaidya. Flooding-based geocasting protocols for mobile ad hoc networks. *MONET*, 7(6):471–480, 2002.

[5] Christian Maihöfer and Reinhold Eberhardt. Time-stable geocast for ad hoc networks and its application with virtual warning signs. *Computer Communications*, 27(11):1065–1075, 2004.

[6] P. Martins, P. Sousa, A. Casimiro, and P. Veríssimo. Dependable adaptive real-time applications in wormhole-based systems. In *DSN*, page 567, 2004.

[7] P. Martins, P. Sousa, A. Casimiro, and P. Verissimo. A new programming model for dependable adaptive real-time applications. *IEEE Distributed Systems Online*, 6(5):1–1, May 2005.

[8] Javier Minguez and Luis Montano. Nearness diagram (nd) navigation: Collision avoidance in troublesome scenarios. volume 20, pages 45–59, 2004.

[9] Luis Montano and J Asensio. Real-time robot navigation in unstructured environments using a 3d laser rangefinder. In *IEEE/RSJ Conf. Intelligent Robots and Systems*, pages 526–532, 1997.

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

[11] Edgar Nett and Stefan Schemmer. An architecture to support cooperating mobile embedded systems. In *Conf. Computing Frontiers*, pages 40–50, 2004.

[12] Stefan Schemmer, Edgar Nett, and Michael Mock. Reliable real-time cooperation of mobile autonomous systems. In *SRDS*, pages 238–246, 2001.

[13] R Simmons. The curvature-velocity method for local obstacle avoidance. In *IEEE/RSJ Conf. Intelligent Robots and Systems*, pages 3375–3382, 1996.

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

[15] Paulo Veríssimo and Antonio Casimiro. The timely computing base model and architecture. *IEEE Trans. Computers*, 51(8):916–930, 2002.

[16] Peiling Yao, Ed Krohne, and Tracy Camp. Performance comparison of geocast routing protocols for a MANET. In *ICCCN*, pages 213–220, 2004.