Collision Prevention Platform for a Dynamic Group of Asynchronous Cooperative Mobile Robots

Rami Yared, Xavier Défago, Julien Iguchi-Cartigny*, Matthias Wiesmann†
JAIST, Graduate School of Information Science
Japan Advanced Institute of Science and Technology
Email: {r-yared, defago}@jaist.ac.jp, julien.cartigny@unilim.fr, wiesmann@google.com

***Abstract*— This paper presents a fail-safe platform on which cooperative mobile robots rely for their motion. The platform consists of a collision prevention protocol for a dynamic group of cooperative mobile robots with asynchronous communications. The collision prevention protocol is time-free, in the sense that it never relies on physical time, which makes it extremely robust for timing uncertainty common in wireless networks. It guarantees that no two robots ever collide, regardless of the respective activities of the robots. The protocol is based on a fully distributed path reservation system.**

It 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. The protocol requires neither initial nor complete knowledge of the composition 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.

***Index Terms*— collision prevention, mobility, fail-safe, autonomous cooperative robots, wireless ad hoc networks, distributed algorithms.**

I. Introduction

There is a marked trend in distributed systems research toward studying problems in which hosts are mobile and their physical location can no longer be abstracted out. While most efforts are still aimed at mobile ad hoc networks and sensor networks, there is also a gradual realization that cooperative robotics raises many interesting new challenges with respect to distributed systems, and particularly in relation to mobility. Indeed, unlike traditional distributed systems and even more so than ad hoc or sensor networks, mobility becomes an essential part of the problems to address.

Many interesting applications are envisioned that rely on groups of cooperating mobile robots. Tasks may be inherently too complex (or impossible) for a single robot to accomplish, or performance benefits can be gained from using multiple robots [2].

As a simple and peaceful illustration, consider the following example. A decentralized team of tiny autonomous mobile robots cooperate to maintain a small oriental garden. Based on the needs of the garden, the team must carry on with many tasks concurrently, such as looking after the moss, picking undesired weeds, nourishing flowers, trimming the trees, bringing dead leaves to the compost, distributing water, etc. These tasks require the robots to almost constantly roam the same limited space, namely the garden, while moving along according to their respectively assigned tasks. Due to the nature of the system, the robots cannot share exact knowledge of each other's location, speed, or even current intention. Nevertheless, an important challenge is to ensure that robots will not collide against each other, regardless of their respective activities.

Context and problem statement: We consider that the robots have the ability to communicate wirelessly and also that they can query their own position according a common referential, as given by a positioning system (e.g., GPS, landmarks). However, the robots do not have the ability to detect each other's position in the environment, and they are not synchronized. In addition, communication delays are unpredictable, and actual robot motion speed is unknown.

In that context, our goal is to ensure *safe* motion, in the sense that, regardless of the respective activities of the robots, no two robots ever collide. The *safety* of the system must never be compromised, regardless of the uncertainty of the underlying system. However, the *performance* of the system may possibly degrade as the result of badly unstable network characteristics or erratic robot speed.

Related work: There are several approaches to address the problem of avoiding collisions between robots.

First, a widespread approach consists in using proximity sensors (e.g., infrared, sonar, laser) in the same way fixed obstacles are detected [3]–[5]. This approach is, however, sensitive to the robots respecting planned speeds, and normally requires an unbroken line-of-sight.

A second approach consists in relying on global motion planning, specifying the respective timing of robots as

A preliminary version of this paper appeared in *Proc. 8th IEEE Intl. Symp. on Autonomous Decentralized Systems ISADS'07* (IEEE CS press, pp. 188-195) [1].

This work was supported by MEXT Grant-in-Aid for Young Scientists (A) (Nr. 18680007).

*Julien Iguchi-Cartigny is currently at Limoges University, France.

†Matthias Wiesmann was supported by the Swiss National Science Foundation Fellowship PA 002-104979. Currently he is with Google Switzerland, Freigustraße 12, 8002 Zürich.

well as the path to follow [6], [7]. This approach is even more sensitive to the speed of the robots, and normally requires much synchronization between the robots.

A third approach uses wireless communication as a means to synchronize the robots and their motion [8], [9]. To do so, communication is extended to satisfy strict real-time guarantees, or at least probabilistic ones [8].

With all three approaches, protocols must rely on explicit time and the speed of the robots. This is fine as long as both robots and communication meet their timing assumptions. However, if a robot happens to move too slowly or too fast, or a few messages are delayed for too long, then there is a risk of collision. In contrast, our aim is to rely as little as possible on the respect of timing assumptions from the underlying system.

Contribution: We present a fail-safe platform on which cooperative mobile robots rely for their motion, thus ensuring that no (physical) collision ever occurs between robots. Its core consists of a collision prevention protocol for a dynamic group of cooperative mobile robots with asynchronous communications. The protocol is time-free, in the sense that it never relies on physical time (as given by a clock), thus making it extremely robust for timing uncertainty common in wireless networks. Furthermore, although the communication range may be limited, no routing is needed as robots only communicate with their direct neighborhood.

This paper extends work that we presented recently at an international conference [1]. The main additions are that we present a more rigorous specification for the collision prevention problem, and include proofs of correctness of the protocol and its properties.

Structure of the paper: The remainder of the paper is structured as follows. Section II describes the system model, definitions, and terminology. Section III defines the collision prevention problem and specification. In Section IV, we present the collision prevention protocol. Section V explains in detail the deadlock and starvation problem. Section VI provides the proofs that the protocol achieves the safety and the liveness properties. Section VII presents a performance analysis and provides insights to maximize the average effective speed of the robots. Section VIII discusses other related work, and Section IX concludes the paper.

II. System Model and Definitions

A. Model

We consider a dynamic distributed system of mobile robots S = $\{R_i\}$ in which each robot has a unique identifier. The total composition of the system, of which robots have only a partial knowledge, can change dynamically. 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 with a limited range D_{tr}. If the distance between two robots R_i and R_j is less than D_{tr}, then the two robots can communicate with each other. Communications assume retransmission 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. Each robot has access to a neighborhood discovery primitive[1] named $NDiscover$.

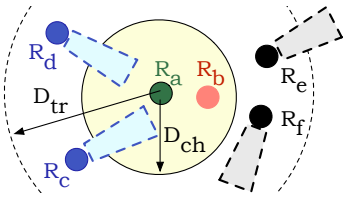


Figure 1. The neighborhood discovery primitive. D_{tr} is the transmission range and D_{ch} is the reservation range. The neighbors of R_a are: $\{R_b, R_c, R_d\}$.

Neighborhood discovery ($NDiscover$):

Characteristics : The neighborhood discovery primitive called $NDiscover$ is a function that enables a robot to detect its local neighbors. These neighbors are within one communication hop and satisfy a certain known predefined condition.

Implementation : $NDiscover$ can be implemented as the traditional neighborhood discovery primitive of mobile ad hoc networks. An implementation of $NDiscover$ primitive can be performed by Geocasting[2] a ping message in a geographical region centered on the robot at the time of calling $NDiscover$ with a radius within the transmission range. All the robots that receive the message and satisfy the predefined condition acknowledge the caller of $NDiscover$.[3]

In wireless environments, the delays in delivering messages are very difficult to anticipate. There are several reasons for the asynchrony of communications in wireless environments, such as the delays required to access the shared medium, due to competition with other nodes. The competition to access the wireless medium causes message loss due to interference, collisions between messages, and fading. Therefore, a retransmission mechanism is needed to ensure message delivery in wireless environments.

Figure 1 illustrates the $NDiscover$ primitive. The robot R_a starts $NDiscover$, the set of robots returned by $NDiscover$ is the set $\{R_b, R_c, R_d\}$. The robot R_b is located within the reservation range D_{ch}, the robots R_c and R_d request zones that intersect with the reservation circle which is the circle centered on R_a with radius D_{ch}. The robots R_e and R_f request zones that do not intersect with the reservation circle of R_a.

Node presence detector: Detecting the presence of nodes in an asynchronous system where there are

[1]The wormhole of our platform is encapsulated in the primitive Neighborhood Discovery which is available through most wireless communication devices.

[2]Geocast is defined by the transmission of a message to a predefined geographical region.

[3]An implementation of $NDiscover$ requires timing property for transmitting and processing the ping messages. $NDiscover$ relies on very lightweight ping messages carrying only the position coordinates of the caller.

no bounds on communication delays, cannot be solved *deterministically* [10]. 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 $NDiscover$.

Timing property: The primitive $NDiscover$ relies on the following timing property: there exists a known upper bounded time delay called T_{nd} such that the following property holds:
For any robot R_i, if R_i starts $NDiscover$ at some time t, then R_i receives an acknowledgment from every robot R_j located in its communication range and satisfying a certain predefined condition, at a time t' such that: $t'-t \leq T_{nd}$.

The protocol is based on a Neighborhood Discovery service, which is the only synchronous part of the system. The delay T_{nd} is specified large enough to cover the necessary retransmissions and hence ensure the delivery of messages related to the Neighborhood Discovery service.

B. 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.

Errors : The are three sources of geometrical incertitude concerning the position and the motion of a robot. An error related to the position information provided by the positioning system is 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 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 be a convex shape and contain the chunk the robot is following. Figure 2 shows the zone Z_i for a robot R_i moving along a chunk 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 2: 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.

III. Collision prevention: specification

The basic idea is essentially a mutual exclusion on geographical zones. 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

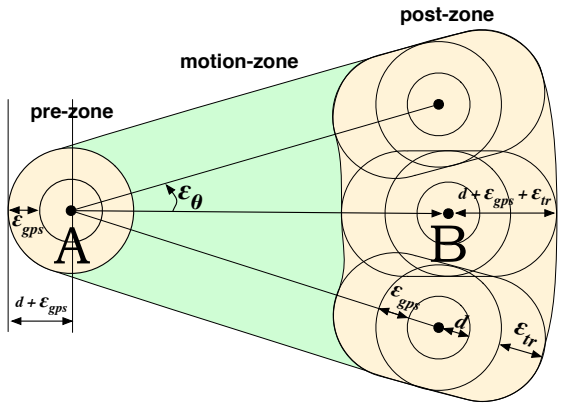


Figure 2. Reservation Zone.

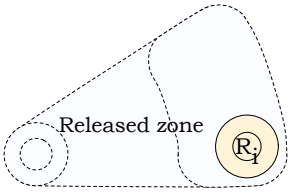


Figure 3. A robot R_i releases the previous zone and keeps only the place that may occupy $pre(Z_i)$

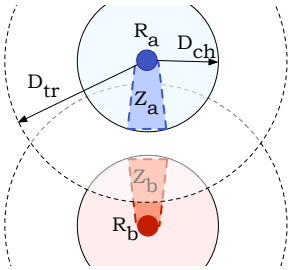


Figure 4. D_{tr}: transmission range. D_{ch}: reservation range. $D_{ch} \leq \frac{D_{tr}}{2}$. if R_a and R_b can not communicate, then they would not collide.

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.

We say that a robot R_i is the *owner* of a zone Z_i (Z_i is *granted* to R_i), if R_i reserves Z_i and did not release it yet. 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: $RelZone_i$. Figure 3 shows that the *pre-motion* zone $pre(Z_i)$ is entirely included within the previous *post-motion* zone, and presents also the current and the previous positions of R_i.

$RelZone_i$ = $pre(Z_i) \cup motion(Z_i) \cup$ SubPost(Z_i), where: SubPost(Z_i) $\subset post(Z_i)$

$pre(Z_i) \subset$ PREVIOUS($post(Z_i)$)

The relationship between robots and zones changes in time. A zone is said to be free if it is not owned by any robot. In order to resolve the collision prevention problem, and to keep the system of mobile robots always in progress towards its final goal, certain properties of *safety* and *liveness* must hold. If a robot requests a zone, then eventually it owns this zone or receives an exception. We say that the robot owns the zone and all the points contained in this zone. A given point can be owned by only one robot. If a robot owns a zone, it eventually releases that zone.

A. Reservation range property

Robots have a limited wireless transmission range. It follows that a reserved zone by a robot must be entirely within a circle centered on the robot with a radius within half of the transmission range. The motivation behind this maximal value is that each robot can communicate with all the robots that it might collide with. Figure 4 illustrates the reservation range property.

The collision prevention protocol provides a parameter named *reservation range* and denoted D_{ch}, that is within half of the transmission range ($D_{ch} \leq \frac{D_{tr}}{2}$), such that a reserved zone by a robot is entirely within a circle centered on the robot with a radius equals to the *reservation range*.

B. Properties

*Property 1 (**Mutual exclusion**):* If a zone Z_i of a robot R_i intersects with a zone Z_j of a robot R_j, then either R_i or R_j (but not both) is the owner of its zone.
Consequently, a point in the plane can be owned by only one robot.

*Property 2 (**Liveness**):* If a robot R_i requests a zone Z_i, then eventually (R_i owns Z_i or an exception is raised).

Exception is potentially raised by the protocol only if a deadlock or a starvation situation occurs.

The following property must hold to ensure the *integrity* of the system. If a robot owns a zone, then eventually it leaves that zone. If a robot leaves a zone, then it releases that zone.

IV. Collision Prevention: Locality-Preserving Protocol

All robots run the same distributed algorithm. When a robot R_i requests a zone Z_i, R_i must determine all the robots R_j that conflict with R_i. The robots R_j are located within one communication hop with respect to R_i, because the reservation range of the robots must be within half of the transmission range. The Neighborhood Discovery primitive returns the set of neighbors $Neighbor_i$ within one communication hop with respect to R_i. Therefore, R_i can determine the set of robots R_j that conflict with R_i. R_i multicasts Z_i to the list of neighbors $Neighbor_i$, then R_i waits until it receives the response messages. Consequently, R_i determines the set of robots that it conflicts with. Intuitively, R_i performs a pair-wise negotiation with each of the robots that R_i conflicts with. Therefore, R_i and each robot R_j decide consistently about the scheduling of their requests. So, a dynamic scheduling for the conflicting requests takes place. When R_i receives a release message from all the robots that R_i waits for, it reserves Z_i and becomes the owner of Z_i. After R_i has reached the *post-motion* zone, R_i releases Z_i except for the area occupied by R_i.

A. Protocol variables

We present the variables used by the collision prevention protocol.

- Z_i is the zone currently requested or owned by robot R_i.
- $Neighbor_i$ represents the set of robots that may possibly conflict with robot R_i (i.e., the output of the neighborhood discovery primitive $NDiscover$).
- G_i is a set of $\{(R_j, Z_j)\}$ such that R_j belongs to $Neighbor_i$, and Z_j is the requested or the owned zone of R_j. Z_j intersects with Z_i.
- $After_i$ is the list of robots waiting for R_i until it releases its zone.
- $Before_i$ is the list of robots that R_i waits for.
- $Depend_i$ is the *dependency* set. If a robot R_i requests Z_i then it conflicts with a set of robots each of which conflicts with another set of robots and so on. The *dependency* set is the union of G_k for each robot R_k related to R_i by the *transitive closure* of the relation *conflict*.
- Dag is a *wait-for* graph such that the vertices represent robots and a directed edge from R_i to R_j represent that R_i waits for R_j to release Z_j.
- msg is a message exchanged during the run of the protocol. Each msg message consists of three fields, the first is the type of the message which belongs to the set {REQUEST, RELEASE, WAITFORME, ACK, PROHIBITED}, the second field is the identifier of the robot sending the message, and the third field is the body of the message which consists of the specifications and the parameters of the requested (or owned) zone. The type REQUEST denotes a request message, RELEASE denotes a release message, and the type WAITFORME means that the receiver of the message must wait for the sender. The type ACK indicates the sender and receiver do not conflict. The type PROHIBITED indicates that the receiver requests a zone that intersects with the pre-motion zone of the stationary sender, which does not move any more.
- $Request\_pending_i$ is a boolean indicates that robot R_i has a requested zone and that R_i has determined its wait-for graph Dag_{wait}.

B. Protocol phases

We explain the phases of the protocol 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) Discovery phase: R_i calls the neighborhood discovery primitive $NDiscover$, to determine the set $Neighbor_i$. This set consists of robots R_j, that *may* possibly come in conflict with R_i for Z_i, since Z_j intersects with the circle centered on R_i with radius equals to the *reservation range*.
2) Negotiation phase: The Negotiation phase of R_i starts by the determination of the set G_i which consists of the robots of $Neighbor_i$ that *conflict* with R_i. The output of the Negotiation phase is the *wait-for* graph, Dag_{wait}. Thus, R_i determines the set of robots that it waits for. If R_i receives a request from a robot R_k (Z_k intersects with Z_i) and

R_k does not belong to G_i, then R_k must wait for R_i. The Negotiation phase proceeds as follows.

- R_i multicasts $msg_i = (\text{REQUEST}, i, Z_i)$ indicating that R_i requests Z_i to all the members of $Neighbor_i$ carrying the parameters of Z_i. This multicast does not require any routing because the neighbors are located within one communication hop with respect to R_i.
- R_i waits until it receives a response message msg_j from each member $R_j \in Neighbor_i$.
- After R_i has received the messages msg_j, R_i determines the set of robots G_i that *conflict* with R_i. (G_i is obtained from the received messages msg_j after discarding the release messages $msg_j = (\text{RELEASE}, j, Z_j)$, and discarding also the request messages $msg_j = (\text{REQUEST}, j, Z_j)$ such that Z_j does not intersect with Z_i). The set G_i contains two disjoint subsets of robots: the first subset denoted $(G1)_i$ is composed of robots R_j such that R_i does not belong to G_j (i.e., R_i must wait for R_j, R_j has sent the message $msg_j = (\text{WAITFORME}, j, Z_j)$). The second is the complementary subset denoted $(G2)_i$, which is composed of robots R_j such that R_i belongs to G_j. Thus R_i must wait for all the robots of $(G1)_i$, in addition to a subset of $(G2)_i$. (This subset would be determined by the Conflict_Resolver and the Pathologic_Detector).
- R_i determines the *dependency* set $Depend_i$ by applying an *Echo* algorithm inspired from [11]. The *Echo* algorithm is explained as follows. R_i multicasts a token message to each robot that belongs to G_i. Upon receipt of the first message of R_i by a robot R_k from R_j (R_j is called the *father* of R_k), it multicasts the message of R_i to all the robots of G_k except its *father* R_j. When a robot R_k has received the token message of R_i from all the robots of G_k, R_k adds the contents of G_k to the token message and sends it (*echo*) to the *father* R_j. When R_i has received the token message from all the robots of G_i, it obtains the *dependency* set[4]. The motivation for building the dependency set is to enable the conflicting robots to build the *wait-for* graph Dag_{wait} in a consistent manner and so to avoid cyclic wait-for relations.
- R_i uses the dependency set $Depend_i$ to construct Dag_{wait}. The vertices represent the robots of the set $Depend_i$ and a directed edge from R_i to R_j means that R_i waits for R_j. Dag_{wait} is built as follows. R_i starts by establishing the *imposed wait-for* relations (Subsection. IV-C), and then it breaks ties for the remainder of the conflicting robots by the Conflict_Resolver. (Subsection IV-E) At first, R_i builds the WAITFORME graph, denoted Dag_{wm}. This graph corresponds to the relation between R_i and R_j from the set $(G1)_i$. The next step, R_i builds Dag_{pg} by adding the directed edges imposed by the *pathological* intersection situations, explained in section V. After having established the *imposed wait-for* relations, R_i adds the directed edges that result from resolving the conflicts according to a specified policy. The conflicting robots build the directed acyclic graph Dag_{wait} in a consistent manner.
  - According to the graph Dag_{wait}, R_i determines $Before_i$ the set of robots that R_i waits for. $Before_i$ = $(G1)_i \cup$ subset($(G2)_i$). R_i dynamically updates the set $After_i$ by adding robots R_k that does not belong to G_i and whose requested zone Z_k intersects with Z_i. (R_i sends to R_k the message $msg_i = (\text{WAITFORME}, i, Z_i)$). R_i keeps updating the set $After_i$ until R_i releases Z_i.
  - R_i waits until it receives a release message from each robot in the set $Before_i$.
3) Reservation phase: When R_i has received a release message from all the robots of the set $Before_i$, or (when the set $Before_i$ is empty), R_i reserves Z_i and becomes the owner of Z_i.
4) Release phase: When R_i reaches the *post-motion* zone $post(Z_i)$, it releases Z_i except the place that R_i occupies. R_i multicasts a release message to all the robots that belong to the set $After_i$. These robots are within one communication hop with respect to R_i, due to the reservation range property. Therefore, the robots of the set $After_i$ can receive the release message of R_i.

C. Imposed wait-for relations

The imposed wait-for relations are the WAITFORME relations in addition to the wait-for relations imposed by the pathological intersection situations.

WAITFORME_*Handler:* The input of the WAITFORME Handler is the dependency set $Depend_i$, and the output is the directed acyclic graph Dag_{wm}. This handler generates Dag_{wm} by establishing the imposed *wait-for* directed edges that correspond to the situation where R_i must wait for R_j because R_j is a member of the set $(G1)_i$. The relation WAITFORME is transitive, so if a robot R_i must wait for R_j and the robot R_j must wait for R_k, then R_i must wait for R_k. Therefore, no cycles can be created in the graph Dag_{wm}.

D. Pathologic_Detector

The pathological intersection situations discussed in section V lead to a deadlock situation or potentially to a starvation situation. Consequently, there exist pathological intersection situations between two zones Z_i and Z_j that

[4]The *dependency* set is piggybacked with the messages of type WAITFORME. R_i computes the *dependency* set when it does not receive any WAITFORME message.

Algorithm 1 Collision prevention protocol (Code for robot R_i)

Initialization: $G_i := \emptyset$; $Before_i := \emptyset$; $After_i := \emptyset$; $Neighbor_i := \emptyset$; $Requested\_zone_i := \perp$; $Request\_pending_i := false$;

when receive (REQUEST, k, Z_k) from R_k
  if $Request\_pending_i$ **then**
    if $R_k \notin Neighbor_i$ and $Z_k \cap Z_i \neq \emptyset$ **then**
      Send(WAITFORME, i, Z_i) to R_k {R_i *piggybacks the wait-for graph* Dag_{wait}.}
      $After_i = After_i \cup \{R_k\}$ {R_i *keeps updating the set* $After_i$ *until* R_i *releases* $RelZone_i$}
    end if
    if $Z_k \cap Z_i = \emptyset$ **then**
      Send(ACK, i) to R_k
    end if {*If* $R_k \in Neighbor_i$ *and* $Z_k \cap Z_i \neq \emptyset$, *then the Pathologic_Detector or the Conflict_Resolver handles it.*}
  else
    if $Requested\_zone_i = \perp$ **then**
      if $Z_k \cap pre(Z_i) \neq \emptyset$ **then**
        Send(PROHIBITED, i, $pre(Z_i)$) to R_k
      else
        Send(ACK, i, $\perp$) to R_k
      end if
    end if
  end if
end when

procedure Request(Z_i)
  $Requested\_zone_i := Z_i$
  Phase 1:
    $Neighbor_i := NDiscover()$ {*Neighborhood Discovery*}

  Phase 2:
    multicast (REQUEST, i, Z_i) to $Neighbor_i$ {*Negotiation*}
    wait until receive response from all $R_j \in Neighbor_i$
    build the set $G_i := (R_j, Z_j)$ such that $R_j \in Neighbor_i$ and Z_j intersects with Z_i.
    $Depend_i$:= *Dependency* set {$Depend_i$ *is received with a* WAITFORME *message.*}
      {*If* R_i *does not receive any* WAITFORME *message, then* $Depend_i$ *is computed using the echo algorithm.*}

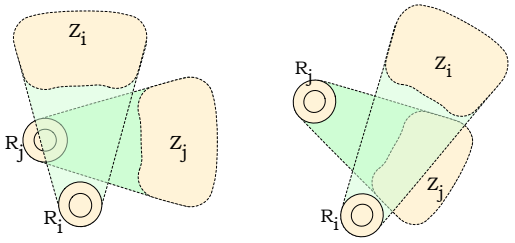
    Dag_{wm} := WAITFORME_Handler($Depend_i$)
    Dag_{pg} := Pathologic_Detector(Dag_{wm}, $Depend_i$)
    Dag_{wait} := Conflict_Resolver(Dag_{pg}, $Depend_i$, policy)
    $Request\_pending_i$:= true
    build the set $Before_i$ and update the set $After_i$ according to the directed acyclic graph Dag_{wait}
    if $Before_i \neq \emptyset$ **then**
      when receive (RELEASE, j, Z_j) from $R_j \in Before_i$
        $G_i := G_i \setminus \{R_j, Z_j\}$ {R_i *removes the entry of* R_j *from the set* G_i}
      end when {*receive the release message from all* R_j *of the set* $Before_i$}
    end if
end Request

Phase 3:
  reserve(Z_i) {R_i *reserves the zone* Z_i}

procedure Release(Z_i)
  Phase 4:
    when R_i reaches the *post-motion* zone $post(Z_i)$
      if $After_i \neq \emptyset$ **then**
        multicast($RelZone_i$) to $After_i$ {*release*(R_i, $RelZone_i$)}
          {R_i *multicasts a release message to all* R_j *of the set* $After_i$}
      end if
    end when
end Release

Algorithm 2 WAITFORME_Handler algorithm

function WAITFORME_Handler ($Depend_i$)
  for all $(R_x, R_y) \in Depend_i$ **do**
    if R_x must wait for R_y (WAITFORME) **then**
      $Dag_{wm} := Dag_{wm} \cup$ DirEdge(R_x, R_y) {R_x *must wait for* R_y, *because* $R_y \in G_x$ *but* $R_x \notin G_y$}
    end if
    if DirEdge(R_x, R_y) and DirEdge(R_y, R_z) **then**
      $Dag_{wm} := Dag_{wm} \cup$ DirEdge(R_x, R_z) {*The relation* WAITFORME *is transitive*}
    end if
  end for
  return Dag_{wm}
end



(a) Z_i intersects with $pre(Z_j)$, so R_i must wait for R_j.

(b) Z_i intersects with $post(Z_j)$, so R_j must wait for R_i.

Figure 5. Pathological intersection situations impose wait-for relations between R_i and R_j.

impose certain wait-for relations between the requesting robots. The imposed wait-for relations are presented as follows.

1) $[Z_i \cap pre(Z_j) \neq \emptyset]$ and $[post(Z_i) \cap post(Z_j) = \emptyset] \Rightarrow$ R_i *must* wait-for R_j.
2) $[Z_i \cap post(Z_j) \neq \emptyset]$ and $[post(Z_i) \cap post(Z_j) = \emptyset] \Rightarrow$ R_j *must* wait-for R_i.

Figure 5 illustrates the imposed wait-for relations due to pathological intersection situations. Figure 5(a) illustrates that R_i must wait for R_j, and Figure 5(b) illustrates that R_j must wait for R_i.

The input of the Pathologic_Detector is the dependency set $Depend_i$ and the graph Dag_{wm}. It outputs the graph Dag_{pg} by adding the directed edges according to the imposed wait-for relations due to the two previous pathological intersection situations.

If a cycle is created by adding a directed edge, then the Pathologic_Detector calls the Deadlock_Resolution policy. The cycle is created because of the two possible pathological situations.

- A pathological intersection case which leads to a deadlock situation between n robots ($n > 2$). In case of three robots, for example, R_a, R_b, and R_c. Z_a intersects with $pre(Z_b)$, Z_b intersects with $pre(Z_c)$, and Z_c intersects with $pre(Z_a)$. It is the general form of the deadlock situation.
- Z_i intersects with $post(Z_j)$ and R_i *must* wait for R_j because of an imposed wait-for relation.

When the Pathologic_Detector detects a deadlock or a starvation situation, then it calls the Deadlock_Resolution policy, which handles the deadlock or the starvation situation. If the Deadlock_Resolution policy does not find

a solution then an exception is raised by the protocol. The Deadlock_Resolution policy is discussed in subsection IV-F. Therefore, the graph Dag_{pg} has no cycles.

E. Conflict_Resolver

The *Conflict_Resolver* breaks ties and determines the *wait-for* relation between two conflicting robots according to a conflict resolution *policy*, if there is no imposed *wait-for* relation between the two robots. A conflict resolution policy can be as follows. R_i *waits-for* R_j if the number of the previous requested zones by R_i is higher than that of R_j. The conflict resolution policy is specified by the robotic application. For example, the robot farther to the intersection zone *waits-for* the closer one, and in case of an equidistance situation, their identifiers are used to break the symmetry. The Conflict_Resolver generates the graph Dag_{wait} by breaking ties between each pair of the robots of the dependency set $Depend_i$. The graph Dag_{wait} is generated in a consistent manner, such that each robot of the set $Depend_i$ generates the *same* graph Dag_{wait} starting from the graph Dag_{pg} by adding the directed edges representing the *wait-for* relations after resolving the conflict between each pair of the conflicting robots. The dependency set is scanned according to the increasing order of the identifiers of robots and the conflict resolution policy is applied. If adding a directed edge creates a cycle, then the new directed edge is reversed to break the cycle.

Algorithm 3 Conflict_Resolver algorithm

```
1: function Conflict_Resolver (Dag_pg, Depend_i, policy)
2:   Dag_wait := Dag_pg
3:   for each robot's identifier x from MINID to MAXID such that
     R_x ∈ Depend_i do
4:     for each robot's identifier y > x to MAXID such that R_y
       ∈ Depend_i do
5:       if Conflict(R_x, R_y) and no edge (R_x, R_y) then
6:         DirEdge(R_x, R_y) := policy(R_x, R_y) {apply the conflict
           resolution policy}
7:         Dag_wait := Dag_wait ∪ DirEdge(R_x, R_y)
8:         if DetectCycle then
9:           DirEdge(R_x, R_y) := DirEdge(R_y, R_x)  {If a cycle is
             detected, then inverse the direction of the edge}
10:        end if
11:      end if
12:    end for
13:  end for
14:  return Dag_wait
15: end
```

F. Deadlock_Resolution policy

The Deadlock_Resolution policy handles deadlock and starvation situations detected by the Pathologic_Detector. The policy used to resolve a deadlock or a starvation situation is based on a *Request Preemption* strategy. For a deadlock situation between two requests (R_i, Z_i) and (R_j, Z_j), the Deadlock_Resolution policy preempts one of the two requests in a deterministic manner. If the request (R_i, Z_i) is preempted, then R_i cancels its request of the zone Z_i.

R_i retries its request (R_i, Z_i) at a later time, by restarting the protocol for the same zone Z_i. If the request is still preempted after a certain number of retrials, then the protocol raises an exception.

The upper layer (e.g., motion planning layer) may then catch the exception and applies some alternative strategy. For instance, in the case of motion planning, the strategy may simply consist in retrying with an alternate route. Of course, if no alternative strategy is available, then that upper layer may itself propagate the exception to higher layers.

G. Optimizations and discussion

Pipelining of requests. Actually, in the collision prevention protocol, a robot R_i starts to request the next zone when it reaches the post-motion zone $post(Z_i)$. A possible optimization of the protocol performance can be achieved as follows. R_i starts to request the next zone when it reserves Z_i. So, R_i performs the negotiation for the next zone in parallel while moving inside the reserved zone Z_i.

The Deadlock_Resolution policy applies an application-based strategy in order to resolve deadlock and starvation situations. The performance of the protocol depends on the number of retrials of the same zone and also on the time durations between the retrials.

A robot can know, without any additional cost, the parameters of the requested zones in its local neighborhood region due to the multicast and antenna properties in wireless networks. So, if possible, the motion planning layer can plan an alternative chunk of a robot's path that avoids highly contended areas.

V. Deadlock and Starvation Situations

There are pathological intersection cases between two zones Z_i and Z_j, such that neither R_i nor R_j can be granted its requested zone (deadlock situation). If a robot R_i could not be granted its requested zone Z_i, we say that robot R_i starves. (starvation situation)

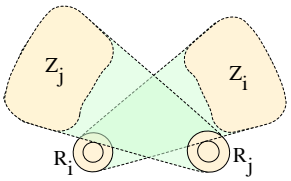
Certain pathological intersection cases constitute necessary conditions but not sufficient for a potential starvation situation.

Definition 1 (Deadlock situation): We say that robot R_i and robot R_j are in a deadlock situation when neither R_i can be granted Z_i nor R_j can be granted Z_j, due to a pathological intersection between the two zones Z_i and Z_j. This pathological intersection occurs if the requested zone Z_i intersects with the *pre-motion zone* $pre(Z_j)$ *and the requested zone* Z_j *intersects with the* pre-motion zone $pre(Z_i)$.

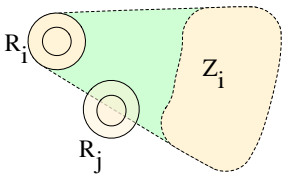
The pathological intersection case which leads to a deadlock situation between two robots R_i and R_j is the following.

Pathological situation 1 (Z_i, Z_j): $[Z_i \cap pre(Z_j) \neq \emptyset]$ and $[Z_j \cap pre(Z_i) \neq \emptyset]$.

The deadlock situation is illustrated in Figure 6(a).

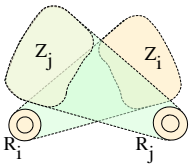


(a) Pathological situation 1, so a deadlock situation between robots R_i and R_j. $Z_i \cap pre(Z_j) \neq \emptyset$ and $Z_j \cap pre(Z_i) \neq \emptyset$.

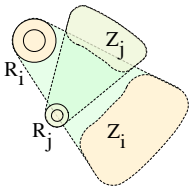


(b) Pathological situation 5, $Z_i \cap pre(Z_j) \neq \emptyset$ and R_j does not request a zone, so R_i starves.

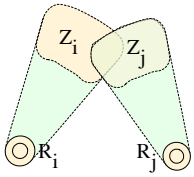
Figure 6. Deadlock and starvation situations.



(a) Pathological situation 2 : Each zone intersects with the post-motion of the other zone.



(b) Pathological situation 3 : Z_i intersects with both $pre(Z_j)$ and $post(Z_j)$.



(c) Pathological situation 4 : The post-motion zones intersect.

Figure 7. Pathological intersection situations. The necessary conditions that potentially lead to a starvation.

Definition 2 (Starvation situation): We say that robot R_i starves when Z_i cannot be granted to R_i. The starvation of R_i is due to a pathological intersection case. This pathological intersection occurs if Z_i intersects with the pre-motion zone $pre(Z_j)$ and R_j does not request a zone.

Figure 6(b) illustrates the starvation situation for robot R_i.

The pathological intersection situations that *potentially* lead to a starvation situation are the following.
Pathological situation 2 (Z_i, Z_j):
$[Z_i \cap post(Z_j) \neq \emptyset]$ and $[Z_j \cap post(Z_i) \neq \emptyset]$ and $[post(Z_i) \cap post(Z_j) = \emptyset]$
Pathological situation 3 (Z_i, Z_j):
$[Z_i \cap pre(Z_j) \neq \emptyset]$ and $[Z_i \cap post(Z_j) \neq \emptyset]$ and $[post(Z_i) \cap post(Z_j) = \emptyset]$
Pathological situation 4 (Z_i, Z_j):
$post(Z_i) \cap post(Z_j) \neq \emptyset$

The pathological situation that leads to a starvation situation for R_i is the following.
Pathological situation 5: $Z_i \cap pre(Z_j) \neq \emptyset$ and R_j does not request a zone.

VI. Proof of Correctness

We prove that the collision prevention protocol satisfies the *Safety* and the *Liveness* properties.

Lemma 1: The wait-for graph Dag_{wait} has no cycles.

Proof: The wait-for graph Dag_{wait} is based on Dag_{wm} and Dag_{pg}.

- The graph Dag_{wm} is a directed acyclic graph, since the WaitForMe relation is transitive. if a robot R_x must wait for R_y and the robot R_y must wait for R_z, then R_x must wait for R_z. (Algorithm 2, Line 7). Therefore, the graph Dag_{wm} has no cycles.
- The graph Dag_{pg} has no cycles, because if a cycle is created, then the Deadlock_Resolution policy is called to break the cycle. So, the graph Dag_{pg} has no cycles.
- We prove that the graph Dag_{wait} is a directed acyclic graph. Dag_{wait} is generated starting from Dag_{pg} which is a directed acyclic graph. If adding a directed edge to Dag_{wait} creates a cycle, then the direction is reversed. We prove that reversing the direction of the edge does not create a cycle and hence the wait-for graph Dag_{wait} is a directed acyclic graph.
  For a graph with no cycles that consists of three vertices $\{R_a, R_b, R_c\}$, if adding a directed edge creates a cycle, then it is obvious that reversing the direction of the edge does not create a cycle. For a graph with no cycles that consists of more than three vertices, the proof proceeds by contradiction.
  Figure 8 shows that when the directed edge (R_c, R_a) is added to Dag_{wait}, it creates the cycle (R_a, R_b, R_c, R_a). So, the graph has already the directed edges: (R_a, R_b) and (R_b, R_c). Figure 9 shows that the edge (R_c, R_a) is replaced by (R_a, R_c).
  Let us assume that the directed edge (R_a, R_c) creates a cycle (R_a, R_c, R_d, R_a). So, the graph has already the directed edges: (R_c, R_d) and (R_d, R_a). Consequently, the graph has already the cycle (R_a, R_b, R_c, R_d, R_a) (the graph has already a cycle), which leads to a contradiction. Therefore, if adding a directed edge creates a cycle, then reversing the direction would not create a cycle in Dag_{wait}.

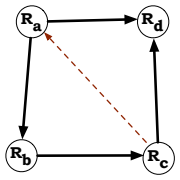


Figure 8. A directed edge (R_c, R_a) is added.

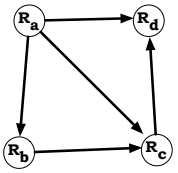


Figure 9. The direction is reversed, so the directed edge is replaced by (R_a, R_c).

Therefore, the wait-for graph Dag_{wait} has no cycles.

Lemma 2: The wait-for relations between the robots related by the transitive closure of the relation conflict, are generated consistently, so the robots build the same wait-for graph Dag_{wait}.

Proof: The set $Depend_i$ consists of the union of G_k for each robot R_k related to R_i by the transitive closure of the relation *conflict*, so $Depend_i$ equals to $Depend_k$.

The robots that R_i conflicts with, belong to G_i or to $After_i$. We prove that the set G_i is sufficient to build the wait-for graph Dag_{wait} consistently.

Let us consider three conflicting robots R_a, R_b and R_c, such that each zone intersects with the two other zones. Let us assume that the set G_a contains R_b, but does not contain R_c, ($R_c \in After_a$). Assume that the set G_b contains both R_a and R_c. When the dependency

set $Depend_b$ is determined, then R_b deduces the wait-for relation between R_a and R_c and that R_c waits for R_a, since the Z_a intersects with Z_c and $R_c \notin G_a$.

If R_a receives the set G_b (due to the dependency set $Depend_a$) before R_a receives the request message of R_c, then R_a deduces that a request message of R_c eventually arrives, and that R_c belongs to the set $After_a$, since Z_a intersects with Z_c.

If $R_c \notin G_a$ and $R_c \notin G_b$, then $R_c \in After_a$ and $R_c \in After_b$, so R_c waits for both R_a and R_b.

The wait-for graph Dag_{wait} is generated based on the set $Depend_i$, by applying a sequence of deterministic functions. The graph Dag_{wm} is generated according to the imposed wait-for relation WAITFORME. The graph Dag_{pg} is generated starting from the graph Dag_{wm} according to the imposed wait-for relations of the pathological situations.

The Conflict_Resolver defines a deterministic function (*policy*) to break ties between two conflicting robots, starting from the graph Dag_{pg} and the set $Depend_i$ which is scanned according to the increasing order of the robots identifiers. Therefore, the wait-for graph Dag_{wait} is generated consistently, and the robots that are related by the transitive closure of the relation conflict build the same wait-for graph.

*Theorem 1 (**Mutual Exclusion**):* If a zone Z_i of a robot R_i intersects with a zone Z_j of a robot R_j, then either R_i or R_j (but not both) is the owner of its zone.

Proof: If Z_i intersects with Z_j, then R_i and R_j are within the transmission range of each other, (reservation range property), thus R_i and R_j can communicate.

Let us assume that $R_j \in G_i$.

- If $R_i \notin G_j$, then R_i must wait for R_j (WAITFORME relation). If Z_i intersects with $post(Z_j)$, then this situation is detected by the Pathologic_Detector and the request (R_i, Z_i) is preempted.
- If $R_i \in G_j$ and there is no deadlock situation between R_i and R_j, then the wait-for relation is determined either by the Pathologic_Detector if there is an imposed wait-for relation due to pathological situations, or by the Conflict_Resolver. If there is a deadlock situation between Z_i and Z_j, then one of the requests is deterministically selected and preempted.

Consequently, there is a wait-for relation between R_i and R_j. According to Lemma 2 the wait-for relations between conflicting robots are generated consistently, so R_i and R_j establish the same wait-for relation and either R_i waits for R_j or R_j waits for R_i.

Let us assume that R_i waits for R_j. When R_j releases $RelZone_j$, then R_i owns Z_i. When the robot R_i is the owner of Z_i, the robot R_j is deprived from the ownership of 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 is still owned by R_j, because:

1) $pre(Z_i) \cap post(Z_j) = \emptyset$ (Proof by contradiction). If $pre(Z_i)$ intersects with $post(Z_j)$, then R_j had to wait for R_i according to the imposed wait-for relations. This leads to a contradiction, since we assume that R_i has wait for R_j.
2) $motion(Z_i) \cap post(Z_j) = \emptyset$ (Proof by contradiction). If the *motion* zone of R_i intersects with the *post-motion* zone of R_j, then R_j had to wait for R_i.
3) $post(Z_i) \cap post(Z_j) = \emptyset$ (Proof by contradiction). If the *post-motion* zones intersect, then the situation is the pathological situation 4. This leads to a contradiction.

Consequently, the *Safety* property holds.

*Theorem 2 (**Liveness**):* If a robot R_i requests a zone Z_i, then eventually (R_i owns Z_i or an exception is raised).

Proof: If a robot R_i requests a 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 wait-for relation is established between R_i and R_j and a directed edge is added to the wait-for graph Dag_{wait}. According to Lemma. 1 the graph Dag_{wait} has no cycles. Therefore, R_i eventually owns Z_i.
3) If a deadlock or a starvation situation is detected, then the Deadlock_Resolution policy is called. If the Deadlock_Resolution policy fails to resolve the situation, then an exception is raised.
4) Robot $R_j \in After_i$ eventually receives the release message of R_i when R_i reaches $post(Z_i)$. Because, R_i multicasts a release message to all the robots that belong to the set $After_i$ and to robots R_a such that $R_i \in Neighbor_a$ and the request message msg_a = (REQUEST, a, Z_a) of R_a has not yet been received by R_i. When R_i reaches the *post-motion* zone, the robots of the set $After_i$ and the robots R_a are within one communication hop with respect to R_i (due to the reservation range property). Hence the robots of the set $After_i$ and the robots R_a can receive the release message of R_i.

Consequently, the *liveness* property holds.

VII. 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 the robots are very 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. Consider a set of robots, each one moving along a chunk (line segment) of length equal to the reservation range D_{ch}.

A. Time needed to reserve and move along a chunk

The average physical speed of a robot is denoted: V_{mot}. We calculate[5] 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}.

Number of robots to wait for: The total number of robots n_{avg} that R_i waits for to reserve a chunk is:

$$n_{avg} = \frac{1}{1 - \frac{n_{reg}}{\pi(\pi+2)}} - 1, \qquad n_{reg} < \pi(\pi+2) \approx 16 \quad (1)$$

where, n_{reg} is the number of robots in the region reg, which is the region of possible collisions for a robot R_i that moves along a line segment of length D_{ch}.

Communication delays: In order to evaluate the performance of the protocol, we need to consider the average of communication delays in the system, although the protocol is time-free. The average communication delays is denoted: T_{com}. The delay of the neighborhood discovery primitive $NDiscover$ is denoted: T_{nd}. The time T_{ch} needed to reserve and move along a chunk is the following:

$$T_{ch} = T_{nd} + 2n_{avg}T_{com} + n_{avg}(T_{com} + \frac{D_{ch}}{V_{mot}}) + \frac{D_{ch}}{V_{mot}} \quad (2)$$

The optimal time T_{ch} for a robot R_i is when it is alone, so there are no robots in the region reg. In this case, the time T_{ch} is: $T_{ch}(alone) = T_{nd} + \frac{D_{ch}}{V_{mot}}$.

B. Optimal reservation range

We compute the average effective speed V of a robot R_i as a function of the reservation range and the density of robots in the system (the density is denoted: s), then we determine an optimal value of the reservation range that maximizes the average effective speed of R_i for a given value of the density of robots. In our protocol the reservation range is a constant parameter given by the system.

$$V = \frac{-sD_{ch}^3 + \pi D_{ch}}{(3T_{com} - T_{nd})sD_{ch}^2 + \frac{\pi}{V_{mot}}D_{ch} + \pi T_{nd}}, D_{ch} < \frac{\sqrt{\pi}}{\sqrt{s}} \quad (3)$$

The previous relation shows that the effective speed is a function of the reservation range and the density of robots, and also that the average effective speed depends on some system-based fixed parameters such as the average communication delays and the physical speed of robots. Figure 10(a) presents the relationship between the speed and the reservation range for different densities. The values of density extend from zero (R_i is alone) to $3[robots/m^2]$. Figure 10(a) shows the optimal reservation range for a given density. The value of the optimal reservation range maximizes the average effective speed of the robots. The curve that corresponds to the density zero (when robot is alone), in Figure 10(a), shows that the effective speed always increases as the reservation range increases, until the effective speed V approaches the physical speed V_{mot} when the value of the reservation range becomes very large. The curve has a horizontal asymptote at $V = V_{mot} = 1[m/s]$. The effective speed of R_i depends on the reservation range, even in the case when R_i is alone, because it needs to perform a certain number of steps to reach a destination, and the number of steps is a function of the reservation range. When the reservation range increases, the number of steps decreases. The relation between the effective speed and the reservation range (when R_i is alone), is the following: $V = \frac{D_{ch}}{\frac{D_{ch}}{V_{mot}} + T_{nd}}$. In each step, R_i needs T_{nd} time units to discover that it is alone. If D_{ch} approaches infinity, then V approaches V_{mot}.

Numerical values.: The values of the fixed system parameters are: $T_{com} = 10[ms]$, $T_{nd} = 1[s]$, the physical speed $V_{mot} = 1[m/s]$. For a density $s = 0.3[robot/m^2]$, the optimal reservation range is $\approx 1.53\ [m]$, which gives a maximal speed $\approx 0.51\ [m/s]$.

C. Speed vs density of robots

The average effective speed always decreases when the density of robots increases for a given reservation range. Figure 10(b) presents the relationship between the average effective speed and the density for different values of the reservation range, (from $0.7[m]$ to $2[m]$).

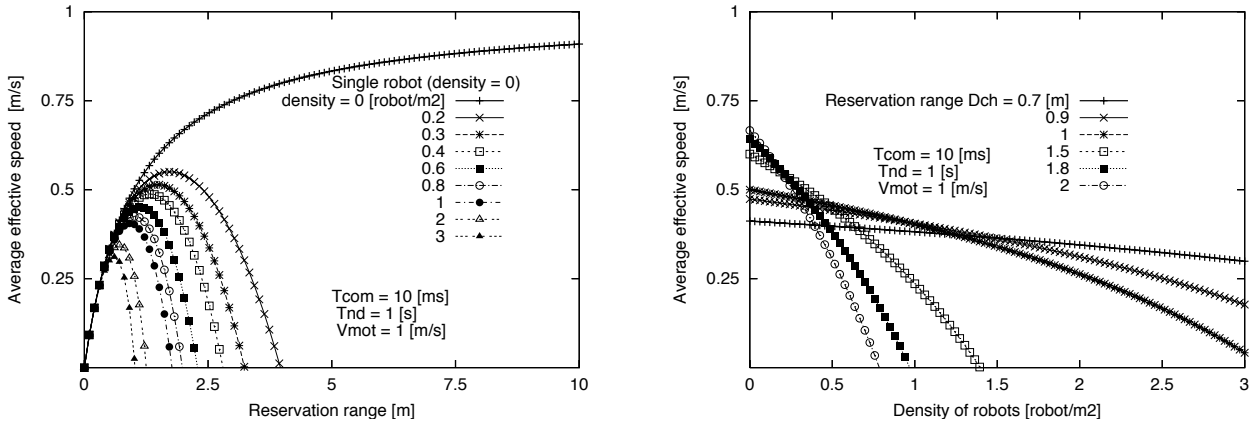
VIII. Related work

Martins et al. [8] demonstrated the avoidance of collisions between three cars, elaborated in the CORTEX project. They rely on the coexistence of two networks, as defined in the Timely Computing Base of Veríssimo and Casimiro [13]. One network, the payload network, is asynchronous and carries the information payload of the application. The second network, the control network or wormhole, enforces strict real-time guarantees and is used sparingly by the protocol. Although the system allows for deadlines to be adapted dynamically (called time-elastic), their approach differs from ours because the use of time remains explicit in their protocol. A second difference is that we assume that robots do not know the existence of other robots that are not in their local neighborhood. This said, our neighborhood discovery primitive has a role similar to the wormhole of Martins et al. in that it encapsulates the synchrony required by the protocol.

Nett et al. [9] presented a protocol 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. Communication is done through WiFi (802.11) with a base station. All robots can communicate directly with each other, and the system assumes the existence of a known upper bound on communication delays. Needless to say that the protocol relies on the strict enforcement of timing assumptions.

Based on the ad hoc protocol presented in this paper, we have recently developed a simpler version [14] aimed at small groups of mobile robots, the composition of

[5]The details and the proofs are not presented in the paper due to space limitations; see our research report [12].



(a) Speed vs reservation range.

(b) Speed vs density of robots.

Figure 10. Average effective speed as a function of the reservation range and the density of robots.

which is static and known to all. In that simpler variant, all robots can communicate directly. In contrast, the protocol presented in this paper is more challenging as it relies on *ad hoc* communication and supports *dynamic* groups of robots.

Clark et al. [15] presented a collision avoidance based on a motion planning framework by combining centralized with decentralized motion planning techniques. When robots become within communication range of each other, they dynamically establish a network. Their protocol ensures that at any time, robots in each network share a common world model by accessing sensing information of all other robots in the same network. Robots avoid collisions by re-planning their paths. Their approach relies on proper timing of communications and robots speed.

Jager et al. [7] presented a decentralized collision avoidance mechanism based on motion coordination between robots. When the distance between two robots goes below a certain threshold, they exchange information about their respective planned paths and determine whether there is a risk of collision. If a collision is possible, then they monitor each other's movements and may change their speed to avoid the collision. The approach is highly dependent on the proper timing of communication and, to some extent, on the proper control of robots' speed. Besides, the composition of the system is static and known to all robots.

Similarly, Azarm et al. [6] presented an on line distributed motion planning. When a conflict is detected between two robots, they exchange their information and determine their respective priorities. The robot with the highest priority keeps its original path while other robots must re-plan their motion.

The problem of robots' collision avoidance has also been handled using different strategies which are sensor-based motion planning methods. The detailed information about motion planning strategies is inspired from [3].

Minguez et al. [3] compute collision-free motion for a robot operating in dynamic and unknown scenarios. Motion planning algorithms compute a collision-free path between a robot's location and its destination. Robots involve sensing directly within the motion planning by sensing periodically at a high rate.

Some of these approaches (e.g., [4]) apply mathematical equations to the sensory information and the solutions are transformed into motion commands. Another group of methods (e.g., [5]) computes a set of suitable motion commands to select one command based on navigation strategy. Finally, other methods (e.g., [3]) compute a high-level information description (e.g., entities near obstacles, areas of free space) from the sensory information, and then apply different techniques simplifying the difficulty of the navigation to obtain a motion command in complex scenarios.

Sensor-based approaches depend on real-time guarantees for processing the sensory information. Furthermore, the information provided by proximity sensors is unreliable and much more limited in range than most wireless network interfaces.

IX. Conclusion

We have presented a fail-safe platform on which cooperative mobile robots rely for their motion. This platform consists of a collision prevention protocol for a dynamic group of cooperative mobile robots with asynchronous communications. The platform ensures that no (physical) collision ever occurs between robots regardless of the respective activities of the robots.

The protocol is time-free, in the sense that it never relies on physical time, which makes it extremely robust with regard to timing uncertainty common in wireless networks. It requires neither initial nor complete knowledge of the composition of the group, and it relies on a neighborhood discovery primitive which is readily available through most wireless communication devices. Furthermore, although the transmission range may be limited, no routing is needed as robots only communicate with their local neighborhood.

We have also presented a performance analysis of the protocol, which provided insights for a proper dimensioning of system parameters in order to maximize the average

effective speed of the robots. The design of the protocol yields scalability due to its locality-preserving property. Therefore, the protocol can handle a large-sized dynamic group of cooperative mobile robots, provided with limited energy resources and limited transmission range.

If a robot crashes, then the local neighbors that are located within one communication hop with respect to the crashed robot at the time of the crash are blocked waiting for the crashed robot. The two-hop and farther neighbors that do not *conflict* with any of the blocked robots, are not affected at the time of the crash. Therefore, the impact of a crash is limited in space and affects only a part of the system for a period of time, however, the *Snowball* effect takes place with the progress of time.

In the future, we intend to further investigate and optimize the performance of the protocol, and consider the problem of a crash of a certain number of robots. In particular, we will analyze the performance of the resulting protocols using simulations and experimentation.

ACKNOWLEDGMENTS

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

REFERENCES

[1] R. Yared, J. Cartigny, X. Défago, and M. Wiesmann, "Locality-preserving distributed path reservation protocol for asynchronous cooperative mobile robots," in *8th IEEE Intl. Symp. on Autonomous Decentralized Systems (ISADS'07)*, 2007, pp. 188–195.

[2] Y. Cao, A. Fukunaga, and A. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–27, 1997.

[3] J. Minguez and L. Montano, "Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios," *IEEE Trans. on Robotics and Automation*, vol. 20, no. 1, pp. 45–59, 2004.

[4] L. Montano and J. Asensio, "Real-time robot navigation in unstructured environments using a 3D laser rangefinder," in *IEEE/RSJ Conf. on Intelligent Robots and Systems (IROS'97)*, 1997, pp. 526–532.

[5] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *IEEE/RSJ Conf. on Intelligent Robots and Systems (IROS'96)*, 1996, pp. 3375–3382.

[6] K. Azarm and G. Schmidt, "Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation," in *Proc. IEEE Int'l Conf. Robotic and Automation (ICRA'97)*, 1997.

[7] M. Jager and B. Nebel, "Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots," in *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS'01)*, 2001.

[8] P. Martins, P. Sousa, A. Casimiro, and P. Veríssimo, "A new programming model for dependable adaptive real-time applications," *IEEE Distributed Systems Online*, vol. 6, no. 5, May 2005.

[9] E. Nett and S. Schemmer, "Reliable real-time communication in cooperative mobile applications," *IEEE Trans. Computers*, vol. 52, no. 2, pp. 166–180, 2003.

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

[11] A. Segall, "Distributed network protocols." *IEEE Trans. on Inf. Theory (IT-29)*, pp. 23–35, 1983.

[12] R. Yared, J. Cartigny, X. Défago, and M. Wiesmann, "Locality-preserving distributed collision prevention protocol for asynchronous cooperative mobile robots," JAIST, Hokuriku, Japan, Research Report IS-RR-2006-003, Feb. 2006.

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

[14] R. Yared, X. Défago, and M. Wiesmann, "Collision prevention using group communication for asynchronous cooperative mobile robots," in *21st IEEE Intl. Conf. on Advanced Information Networking and Applications (AINA'07)*, 2007, pp. 244–249.

[15] C. Clark, S. Rock, and J.-C. Latombe, "Motion planning for multiple mobile robots using dynamic networks," in *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA'03)*, Taipei, Taiwan, Sept. 2003.

Rami Yared is a researcher at the Japan Advanced Institute of Science and Technology (JAIST; Japan) since 2006. He obtained his Ph.D. in computer science from the Japan Advanced Institute of Science and Technology in 2006, his MS degree in computer science in 2001 from the Laboratory of Analysis and Systems Architecture LAAS-CNRS in Toulouse, France. His research interests include distributed algorithms, fault-tolerance, mobile ad hoc networks, and cooperative autonomous mobile robots.

Xavier Défago is an associate professor at the Japan Advanced Institute of Science and Technology (JAIST; Japan) since 2003. He obtained his Ph.D. in computer science from the Swiss Federal Institute of Technology (EPFL; Switzerland) in 2000, and joined JAIST as a research associate shortly thereafter. From 1995 to 1996 he also worked at the NEC C&C Research Labs in Kawasaki (Japan). His research interests include distributed algorithms, fault tolerance, large-scale distributed systems, group communication, and cooperative autonomous mobile systems.

Julien Iguchi-Cartigny is an assistant professor at the Laboratory XLIM, Limoges, France since 2005. He received a MSc and PhD degrees in computer science from the University of Lille, France, in 2000 and 2003 respectively. He joined the Japan Advanced Institute of Science and Technology (JAIST; Japan) as a JSPS postdoctoral researcher in 2004. His research interests include routing and security in dense and/or large ad-hoc networks.

Matthias Wiesmann is a software engineer for Google Switzerland in Zürich. He received his PhD in computer science from the Swiss Federal Institute of Technology in Lausanne (EPFL; Switzerland) in 2002. He graduated in computer science at the Geneva University in 1997. He worked as a research fellow at the European Organization for Nuclear Research on the Atlas project. He then did a post-doc at the Japan Advanced Institute of Science and Technology (JAIST; Japan).