COLLISION PREVENTION PROTOCOL FOR A DYNAMIC GROUP OF ASYNCHRONOUS COOPERATIVE ROBOTS

Mariana FRATU, Aurel FRATU
Transilvania University of Brasov, Romania

Abstract. The protocol is based on a fully distributed path reservation system. It assumes an ad hoc network formed by the mobile grippers of the robots, and takes advantage of the wireless communication. The protocol requires initial complete knowledge of the composition of the robots group. A performance analysis of the protocol provides insights for a correct dimensioning of system parameters in order to maximize the average effective speed of the robots' grippers.

Keywords: Collision prevention protocol, autonomous cooperative robots' grippers, distributed algorithms

1. Introduction

This paper extends work that was presented in the previous article "*Collision Prevention Method for a Dynamic Group of Cooperative Robots who Communicate Wirelessly*", recently proposed for **RECENT** journal. The main additions are that is presented a more rigorous specification for the collision prevention problem, and include proofs of correctness of the protocol and its properties.

The paper presents a collision prevention protocol for a dynamic group of cooperative robots' grippers with asynchronous communications. The protocol is based on a Neighbourhood Discovery Service (NDS), which is the only synchronous part of the system. Furthermore, although the communication range may be limited, no routing is needed as robots only communicate with their direct neighbourhood.

2. Locality-Preserving Protocol

A dynamic distributed system of robot-grippers, $S = \{R_i\}$ is considered, in which each robot-gripper has a unique identifier where i, $(1 \leq i \leq p)$ denotes the index of dynamical systems comprising a network, and p is the total number of the individual elements.

The total composition of the system, of which the cooperative grippers group has only a partial knowledge, can change dynamically [1].

All cooperative grippers of the group run the same distributed algorithm. The algorithm consists of a distributed path reservation system, such that a gripper must reserve a zone before it moves. When a robot R_i requests a zone Z_i for her gripper, R_i must determine all the robots R_j that conflict with R_i. The robots R_j are located within one communication loop with respect to R_i, because the reservation range of the robots must be within half of the transmission range [2].

A neighborhood discovery primitive (*NDiscover*) returns the set of neighbors $Neighbor_i$ within one communication step 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 prudent negotiation with each of the robots that R_i conflicts with.

Therefore, R_i and each robot R_j decide consistently about the arrangement of their requests. So, a dynamic arrangement for the conflicting requests takes place. When R_i receives a release message from all the robots that R_i waits for, it reserves zone Z_i and becomes the owner of zone Z_i. After R_i has reached the post-motion zone, R_i releases zone Z_i except for the area occupied by R_i.

3. Protocol variables. Definitions and Terminology

Further the variables used by the collision prevention protocol are presented.

- *Paths*: Corridor denotes a bordering zone along which gripper of the robot moves. A path of a robot' gripper is a continuous route composed of a series of contiguous corridors.
- *Errors*: There are three sources of geometrical incertitude concerning the position and the motion of a robot' gripper. An error related to the position information provided by the positioning system is denoted e_{ps}. In addition, the motion of a robot creates two additional sources of errors. The first error is related to the translational movement, denoted e_{tr}. The second error is related to the rotational movement, denoted e_{ro}.

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

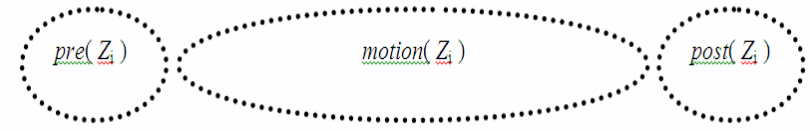

Fig. 1. Parts of the zone Z_i

A moving zone is defined as the region needed by a robot' gripper to move safely along a corridor. This includes requirements for the shape of the robot, positioning error, and imprecision in the moving of the robot' gripper. The zone must have a convex shape and contain the corridor that robot' gripper rides.

- *zone* 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 the zone 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 the zone 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_{wait} is a wait-for delay such that R_i waits for R_j to release Z_j.
- *mssg* is a message exchanged during the run of the protocol.

Each *mssg* 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 variable indicates that robot R_i has a requested zone and that R_i has determined its wait-for graph Dag_{wait}.

4. Protocol phases

The phases of the protocol are explained with respect to a robot R_i. The robot R_i is located in the pre-motion zone $pre(Z_i)$. When robot the robot R_i requests a new zone Z_i, it proceeds as follows.

1) *Discovery phase*. The robot 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 the robot R_i for the zone Z_i, since Z_j intersects with the circle centered on R_i with radius equals to the reservation range.

Neighborhood discovery primitive is a function that enables a robot to detect its local neighbors. These neighbors are within one communication loop and satisfy a certain known predefined condition. *NDiscover* primitive can be implemented as the traditional neighborhood discovery primitive of mobile ad hoc networks [3].

A contact message in a geographical region is centered on one gripper 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*.

2) *Negotiation phase*. The negotiation phase of the robot R_i starts by the determination of the set G_i which consists of the robots of $Neighbor_i$ that conflict with the robot 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 [4]. If the robot R_i receives a request from a robot R_k (Z_k intersects with the zone Z_i) and R_k does not belong to G_i, then R_k must wait for the robot R_i.

The negotiation phase proceeds as follows.

- R_i multicasts $mssg_i$ = (*REQUEST, i, Z_i*) indicating that the robot 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 step with respect to the robot R_i.

- the robot R_i waits until it receives a response message $mssg_j$ from each member of $Neighbor_i$.
- after the robot R_i has received the messages $mssg_j$, it determines the set of robots G_i that conflict with R_i. The set G_i is obtained from the received messages $mssg_j$ after discarding the release messages $mssg_j$ = (*RELEASE, j, Z_j*), and discarding the request messages $mssg_j$ = (*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 $(G_1)_i$ 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 $mssg_j$ = (*WAITFORME, j, Z_j*)). The second is the complementary subset denoted $(G_2)_i$, which is composed of robots R_j such that R_i belongs to G_j. Thus the robot R_i must wait for all the robots of $(G_1)_i$, in addition to a subset of $(G_2)_i$. This subset would be determined by the Conflict Resolver.

- the robot R_i determines the dependency set $Depend_i$ by applying an Echo algorithm.

Echo algorithm

The Echo algorithm is explained as follows: the robot R_i multicasts a token message to each robot that belongs to G_i. Upon receipt of the first message of the robot R_i by a robot R_k from R_j (R_j is called the father of R_k), it multicasts the message of the robot 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 the robot 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 the robot R_i has received the token message from all the robots of G_i, it obtains the dependency set. The dependency set is associated with the messages of type *WAITFORME*. The robot R_i computes the dependency set when it does not receive any *WAITFORME* message.

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.

- the robot 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 the robot R_i to R_j means that the robot R_i waits for R_j.

The build the wait-for graph Dag_{wait}

Dag_{wait} is built as follows. The robot R_i starts by establishing the imposed wait-for relations and then it breaks links for the remainder of the conflicting robots by the Conflict Resolver.

- At first, the robot R_i builds the *WAITFORME* graph, denoted Dag_{wm}. This graph corresponds to the relation between the robot R_i and R_j from the set $(G_1)_i$.
- The next step, the robot R_i builds Dag_{pg} by adding the directed boundaries imposed by the confliction intersection situations.
- After having established the imposed wait for relations, the robot R_i adds the directed boundaries 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}, the robot R_i determines $Before_i$ the set of robots that the robot R_i waits for. The robot 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. (the robot R_i sends to R_k the message $mssg_i$ = (*WAITFORME, i, Z_i*)). The robot R_i keeps updating the set $After_i$ until the robot R_i releases Z_i.
- The robot R_i waits until it receives a release message from each robot in the set $Before_i$.

3) *Reservation phase*. When the robot R_i has received a release message from all the robots of the set $Before_i$, or (when the set $Before_i$ is empty), the robot R_i reserves Z_i and becomes the owner of Z_i.

4) *Release phase*. When the robot 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 loop 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.

5. Conflict Resolution Algorithm

The Conflict-Resolver 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 [5].

A conflict resolution policy can be as follows: the robot 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 [6].

The Conflict Resolver generates the graph Dag_{wait} by breaking secure 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.

6. Conclusions

This 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. It guarantees that no two robots ever collide, regardless of the respective activities of the robots. When moving along a path, the robot' gripper repeats this procedure for each corridor along the path.

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 robots, situated in a limited transmission range.

The collision prevention protocol was developed to be used in a managed network with a centralized controller that coordinates the communication schedule for the network. A managed network has the prime benefit that each channel in the range can be used without concern for collisions of robots' grippers. Standards specify packet formats and layered transaction models for the wirelessly communication and how multi routes are maintained to a management application. It remains the duty of the network designer to assign data flows based on service requests and reports on how successful the wireless paths have been. The protocol is intended for industrial automation where applications are extremely performance-sensitive; the adoption in this regime will come only after extensive evaluation.

References

1. Yared, R. et al.: *Locality-preserving distributed path reservation protocol for asynchronous cooperative mobile robots*. Proceeding of the 7th IEEE International Symposium on Autonomous Decentralized Systems (ISADS'07), ISBN: 0-7695-2804-X, 2007, p. 188-195
2. Yared, R. et al.: *Collision prevention using group communication for asynchronous cooperative mobile robots*. Proceeding of the 21st IEEE International Conference on Advanced Information Networking and Applications (AINA'07), ISBN: 0-7695-2846-5, 2007, p. 244-249
3. Mourikis, A.I., Roumeliotis, S.I.: *Performance Analysis of Multi-robot Cooperative Localization*. Journal IEEE Transactions on Robotics, 22(4), 2006, p. 666-681
4. Mirzaei, F.M. et al.: *On the Performance of Multi-robot Target Tracking*. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), ISBN: 1-4244-0601-3, 2007, Rome, Italy, p. 3482-3489
5. Clark, C. et al.: *Motion planning for multiple mobile robots using dynamic networks*. Proceeding of the IEEE International Conference on Robotics and Automation, Taipei, Taiwan, ISBN: 0-7803-7736-2, 2003, p. 4222-4227
6. Hidaka, Y.S. et al.: *Optimal Formations for Cooperative Localization of Mobile Robots*. Proceedings of the IEEE International Conference on Robotics and Automation, 2005, Barcelona, Spain, p. 4137-4142

Received in June 2011