5th National Conference on Control Architecture for Robots (CAR) Douai, France 18-19 May 2010

Maintaining Connectivity in Multi-Robot Systems Through Connectivity Awareness

Le Van Tuan[1,2,3,4], Noury Bouraqadi[1,2], Stinckwich[3,4], Victor Moraru[4], and Arnaud Doniec[1,2]

1: Université de Lille Nord de France
2: Ecole des Mines de Douai
3: GREYC–Université de Caen
4: IRD UMMISCO, Hanoï, Vietenam

contact: noury.bouraqadi@mines-douai.fr

Abstract

Maintaining the network connectivity in mobile Multi-Robot Systems (MRSs) is a key issue in many robotic applications. In our view, the solution to this problem consists of two main steps: *(i) making robots aware of the network connectivity; and (ii), making use of this knowledge in order to plan robots tasks without compromising connectivity*. In this paper, we present an application-independent distributed algorithm executed on individual robots to build the connectivity awareness. So, robots can plan their motions while keeping connected.

Keywords: Multi-Robot Systems - Coordination - Connectivity

1 Connectivity Awareness Basic idea

Our basic idea in maintaining the network connectivity in MRSs is the following: each robot R_i in the MRS, while performing its task, has to keep in touch with *at least* one neighboring robot R_j from which a communication path to a reference robot can be established. Concretely, in the system shown in figure 1, R_5 has to maintain the connection with R_1, and R_6 has to stay in touch with R_5. Similarly, R_2 is responsible for maintaining the communication link with R_1. R_3 and R_4 should move around in such a way that the links between them and R_2 will not be broken. For R_7, there are two different paths to R_1, it needs to maintain *at least* one link with either R_3 or R_4. So R_7 has more choice to move while taking the connectivity into account. If the robots are all successful as such, then the connectivity of the whole system will be ensured.

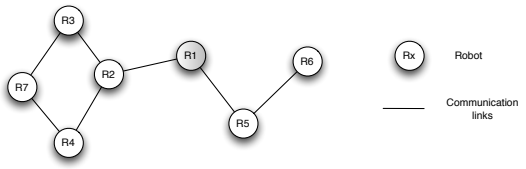

Figure 1: A Networked Robotics System

(a) Robot 3

Access Robots	Access Paths
R_2	(R_1)

(b) Robot 7

Access Robots	Access Paths
R_3	(R_2, R_1)
R_4	(R_2, R_1)

Table 1: Example of access robots and access paths for some robots of figure 1

Definition 1 (Access Robot and Access Path) *Given reference robot R_r, and two different robots R_i and $R_j \in V$ ($i \neq j$), R_j is called an access robot for R_i iff there exist an edge $\{R_i, R_j\} \in E$ (i.e. R_i and R_j are neighbors) and there exist a path $p(R_j, R_r)$ such that $R_i \notin p$. We call p an* access path.

Based on definition 1, the maintenance of connectivity can be interpreted as following: given a reference robot, for preserving the network connectivity, robots need to maintain the communication links with one of their access robots while performing their tasks.

2 Connectivity Table

A connectivity-awareness for a given robot is materialized as a *connectivity table* containing a set of access paths. These paths represent a partial view of the network connectivity. For example, the connectivity table of the robot R_7 in the network in figure 1 might looks like the table 1. The table has two access paths corresponding to two access robots R_3 and R_4. Based on this knowledge, robots know which neighbors they should depend on for maintaining the connectivity with the whole network. This section presents our algorithm to build the connectivity table, and to maintain its consistence in presence of the mobility.

We assume that at the beginning of a mission, robots are close to each other and form a network; therefore, they can communicate and rely on a Mobile Ad Hoc Network (MANET) [Per01] routing protocol for message transmission. We also assume that a message sent by a node is received correctly within a finite period of time (a *step*) by *all* its neighbors, and that every robots has a unique ID. The main concerns in making robots network connectivity-aware now turn out to be selecting the reference robot, and building the connectivity tables.

3 Building the Connectivity Table

The choice of a reference node can be application-dependent and might involve multiple criteria such as the energy level, the number of neighbors, hardware requirements, etc. A generalized approach is to employ a market-like bidding mechanism to select the node that will be the reference.

Algorithm 1: `Algorithm performed by every robot on the reception of newAccessPath message`

Input: The `New-Access-Path` Message M
Output: The connectivity table T of the robot is updated

```
1  begin
2      p ← M.accessPath();
3      if this.id() ∉ p then
4          s ← M.getSender();
5          p.addFirst(s.id());
6          T.add(p);
7          if T.isInternallyDisjointWith(p) then
8              M.setSender(this);
9              M.setNewPath(p);
10             this.oneHopBroadcast(M);
11         end
12     end
13 end
```

Consider the robot network of the figure 1. Suppose that R_1 is chosen to be the reference robot, it will then broadcast to all its one-hop neighbors a `New-Access-Path` Message. Such a message encodes the *sender's ID*, and the new access path. Since the sender here is also the reference node, the ID is R_1 and the access path is empty. R_2 and R_5 will receive the message and process it according to algorithm 1. Since they have not received this message before and their connectivity tables are still empty, R_2 and R_5 modify the path by adding R_1'ID in the path's head, and then add the path into their respective tables. Last, the modified messages are forwarded. Note that, the algorithm allows forwarding only paths that are *internally disjoint* to those already in the table. A path p_1 is said to be internally disjoint to another path p_2, if they share no robot, except may be the first and the last robot. For example, paths (R_3, R_2, R_1) and (R_6, R_5, R_1) are internally disjoint. Also, (R_4, R_5, R_1) and (R_4, R_6, R_1) are internally disjoint. Conversely, paths (R_3, R_2, R_1) and (R_4, R_2, R_1) are **not** internally disjoint, since they do share R_2.

At step 3, R_6 receives the message broadcasted by R_5. At the same step, R_3 and R_4 receive the message sent by R_2, they both modify the message, add the modified path to their table, and then forward the updated messages. Note that R_1 receives also messages from R_2 and R_5, but it ignores them because R_1's ID is already in the path. Loops are thus filtered out.

At step 4, R_6 continues to forward the message, but R_5 ignores it because R_5's ID is already in the path (filtering out loops). Similarly, message sent by R_3 reaches R_7 and R_2, but only R_7 proceeds and add the path into its table. At the last step (step 5), there is only R_7 which attempts to forward a message. Consider the message that reaches R_7 after passing through R_2 and R_3.

Robot	Connectivity Table
R_1	()
R_2	(R_1)
R_3	(R_2, R_1)
R_4	(R_2, R_1), (R_7, R_3, R_2, R_1)
R_5	(R_1)
R_6	(R_5, R_1)
R_7	(R_3, R_2, R_1), (R_4, R_2, R_1)

Table 2: Connectivity tables for robots of figure 1 built by algorithm 1

After updating this message and storing the modified path, R_7 will broadcast it. So, R_4 and R_3 will receive this updated version. R_3 will ignore it because R_3's ID is already in the path. R_4 will add it to its connectivity table, but it does not forward it because the message path is not *internally disjoint* with the other path already stored in R_4. Indeed, the path in the message just received by R_4 is (R_7, R_3, R_2, R_1), while the path already in the connectivity table of R_4 is (R_2, R_1). These two access paths are **not** internally disjoint, since they share robot R_2.

At step 4 also, a message is broadcasted by R_4 with path (R_4, R_2, R_1). R_2 ignores it because it is already in the access path. R_7 does add it to its access table. However, since R_7 already has a path going through R_2[1], it does not forward the message. The process finisheses, as there is no message sent over the network.

This process of building the connectivity table is loop-free, since robots store and propagate only access paths that are internally disjoints i.e. paths that do not share any robot except the access robot. For the same reason, this construction terminates within l steps, where l is the length in term of hop-count of the longest access path in the network.

Regarding the performance of the algorithm, let $\bar{d}$ be the average number of neighbors per robot. The memory complexity for saving a connectivity table is $O(2\bar{d})$ entries. The total messages complexity is $O(2n\bar{d})$. A full proof is given in [LBS+09].

4 Updating the Connectivity Table

Since the environment is subject to change, and that the robots move during their mission, the network topology can change over time. This poses a problem of ensuring the coherence of the connectivity table with the actual situation of the network. There are two situations that might make the information in the connectivity table obsolete: a robot "meets" new neighbors or it is out of reach of an access robot.

When a robot detects a breakage of a link to a neighbor, it will remove all the access paths going through this link in its connectivity table. Then it broadcasts a `Link-Broken` message to its neighbors. The message contains the edge composing the id of the sender and of disconnected neighbor robot. Any robot receiving a `Link-Broken` message will remove from

[1]We make here the assumption that R_7 processes the message from R_3 before the one sent by R_4.

its connectivity table any access paths through the broken link. If a path is deleted, the robot forwards the `Link-Broken` message to its neighbors. Therefore, all robots that might use the broken link be notified, and then update their connectivity tables.

When robots meet new neighbors, they will exchange their connectivity table to each other. Algorithm 1 will be executed on each robots to detect, store new access paths and notify neighbors about updates.

5 Conclusion

We presented in this paper a distributed algorithm that makes mobile robots in a multi-robot system aware of the network connectivity. This awareness is a separated concern that can be reused in various robotic applications with different application-dependent strategies for connectivity maintenance. This work was presented in detail at conference ICRA2009 [LBS$^+$09]. We showed that our solution allows checking the robotic network biconnectivity, i.e. that each robot is connected to the network through at least two different access paths with a complexity, in the worst case, of $O(4n^2)$ messages, whereas the state of the art [AS06a, AS06b] requires $O(2n!)$ messages.

We also developped a more efficient variant of our connectivity table construction algorithm. The main idea of this result we presented in the RIVF2009 conference [LBSM09], is that a robot stores all access paths it receives but forwards only two of them which significantly reduces the amount of messages broadcasted over the network. Indeed, the message complexity for constructing connectivity tables drops to $O(2n)$ for n robots.

References

[AS06a] Mazda Ahmadi and Peter Stone. A distributed biconnectivity check. In *proceeding of the 8th International Symposium on Distributed Autonomous Robotic Systems (DARS'06)*, 2006.

[AS06b] Mazda Ahmadi and Peter Stone. Keeping in touch: Maintaining biconnected structure by homogeneous robots. In *proceeding of the 21st National Conference on Artificial Intelligence (AAAI)*, 2006.

[LBS$^+$09] Van Tuan Le, Noury Bouraqadi, Serge Stinckwich, Victor Moraru, and Arnaud Doniec. Making networked robot connectivity-aware. In *Proceedings of ICRA (International Conference on Robotics and Automation)*, Kobe, Japan, May 2009.

[LBSM09] Van Tuan Le, Noury Bouraqadi, Serge Stinckwich, and Victor Moraru. Connectivity awareness in networked robotic systems. In *Proceedings of IEEE-RIVF International Conference on Computing and Communication Technologies*, Da Nang City, Vietnam, July 2009.

[Per01] C. Perkins. *Ad Hoc Networking*. Addison-Wesley, 2001.