On Maintenance of Connectivity of Mobile Robot Network Using a Decentralized Scheme

Term Project Report for ESE 680 (Distributed Systems & Networks)

submitted by
Subhrajit Bhattacharya

1 Introduction:

Maintaining communication connectivity among mobile robots is an important issue in a wide class of problems. Problems like formation control, rendezvous, consensus attainment or even the problem of goal attainment requites that there exists a communication path between the agents. The primary challenge is in achieving the goal that the agents are supposed to pursue and at the same time maintaining the connectivity of the network. However often it may turn out that one of these objectives compromises the other. Thus the final goal in studying the problem may be to reach an optimum such that the required connectivity is maintained and the goals are reached as much as possible.

2 Some literature survey and the scope of present work:

In [2] the problem of maintaining the graph connectivity in a network of mobile robots has been addressed in an involved and rigorous way. The main idea behind the particular approach is to modify the topology of the graph in such a way that the second eigenvalue of the graph Laplacian increases. Since the second eigenvalue is a measure of the connectivity of the graph, an increase in its value will imply increase in connectivity of the graph. Hence this is basically an optimization problem where the objective is the second eigenvalue of graph Laplacian and the search space is the space of all possible topologies of the graph. However this approach may turn out to be more conservative than desired. There may be situations where we may not desire to actually increase the graph connectivity, and rather just wish that the graph doesn't get disconnected. That means we may want that the second eigenvalue of the graph Laplacian will not become zero, but can assume any small value.

The present work deals with development of an algorithm that will ensure the connectivity of communication graph among mobile robots. The algorithm will act as a controller adjacent to the primary controllers for the system. The basic aim behind the design of this controller is that it will interfere very little with the system till the connectivity graph is not on the verge of being sacrificed. It lets the graph topology to change naturally under such situation. But when the controller detects that the connectivity of the graph is close to its breaking it puts hard restrains on the velocity inputs so as to ensure that the graph remains connected.

3 Overview of the problem:

The problem consists of N mobile robots moving on a flat surface. Each robot can communicate with its neighbors within a radius of r_c. Each agent starts from an initial location and navigates towards their respective goals. The primary controller for an agent that calculates the default velocity command consists of the computation of gradient of a simple potential field at the current location of the robot. In our present simulations we assume a potential field with an unique minima at the goal and we don't perform any collision avoidance.

Our goal is to now devise a secondary controller which will modify the velocity command that was issued by the primary controller so as to ensure that the graph connectivity is maintained. In our approach we'll achieve this in a rather conservative fashion. That means we will devise the algorithm such that the graph connectivity is never broken even if that means some or all the agents will not be able to reach their respective goals. But sacrifice of graph connectivity will not be allowed.

4 Assumptions, hypothesis and solution approach:

4.1 Assumptions and hypothesis:

This particular problem relies on the following hypothesis:

i. Each agent is a point on the flat surface and has absolute maneuverability (i.e. can change the direction of motion instantaneously). There is no obstacle in the workspace and being points, the agents don't collide with each other.

ii. Each agent issues new velocity commands at a regular interval of time Δt. Within a particular interval of Δt between the successive issue of velocity commands by a particular agent, the velocity of the agent remains constant.

iii. There is an upper bound on the magnitude of the velocity of the agents. We represent this maximum speed by v_m.

iv. The velocity limits (v_m) and the time intervals between the successive issue of velocity commands (Δt) need to be the same for all the agents.

v. However the agents need not be synchronized.

vi. The agents are always ready to communicate with each other whenever they are within the *critical radius* r_c. The information that can be obtained by an agent from a neighbor are:

   a. The ID of the neighbor

   b. The relative position of the neighbor from the agent in the agent's local coordinate system (possibly measured by a range sensor)

vii. If the time required for communication between two agents is τ, then we assume that $N\tau < \Delta t$. This is a very important condition that needs to be true for our present approach of solution.

4.2 Solution approach:

The solution approach primarily consists of the following stages as performed by each agent at each time-step:

i. Creating an initial estimate for the adjacency matrix depending on which other agents it can communicate with.

ii. Sharing this estimate with its neighbors and propagation of the information over the network in a distributed fashion. This will enable an agents to have an estimate of the net adjacency matrix of the particular connected component of the graph of which it is a part.

iii. Once the 'global' adjacency matrix is known, the agents detect the *critical edges* that are connected to itself.

iv. Knowing the *critical edges*, the agents now modify their velocities accordingly so that none of the critical edges get broken over the time till the issue of the next velocity command. This velocity modification is performed in two stages. The first stage is basically a heuristic one, whereas the second stage will ensure connectivity in a more rigorous fashion.

Note that here by 'time-step' we meant the time instants at which new velocity commands are issued by the individual robots. In the following sections we'll investigate each of these above mentioned stages one by one.

5 Estimating and propagating the Graph adjacency:

The main idea in this stage is each agent has an estimate of the graph adjacency which they update by communicating with its neighbors. During this update process it is assumed that the graph topology doesn't change. Although this assumption is rather controversial, the assumption will work fairly well as long as the graph remains connected during this process, even if its topology changes. And in the later section we'll see that we can actually put bounds on the velocities so as to ensure that the graph remains connected.

Thus, if A^k is the estimate of the adjacency matrix made by agent k, we will refer to the $(i,j)^{th}$ element of the matrix at the p^{th} step of updating it by ${}^pA^k_{i,j}$. There are two processes now:

a) *Initiation:* ${}^0A^k_{i,j}$ is initiated for each agent k by the following rule:

$$ {}^0A^k_{i,j} = \begin{cases} 1 & \text{, if either } i \text{ or } j \text{ is connected to agent } k \\ 0 & \text{, otherwise} \end{cases} $$

b) *Propagation:* The elements of adjacency matrix are updated by talking to the neighbors according to the following rule:

$$ {}^{p+1}A^k_{i,j} = \max\{{}^pA^k_{i,j}, \max_{l \in N_k} {}^pA^l_{i,j}\} $$

where, N_K is the set of neighbors of agent k.

It can be proved [1] that the number of steps that the process of propagation will take to be completed (i.e. for an agent to obtain the whole adjacency matrix of the connected component of the graph) is not more than the length of the longest path in the particular connected component of the graph. Since the agents don't have an estimate of the length of the longest path, we run the propagation for N times, which can be the maximum length of any path in a grap with N nodes.
Moreover it is to be noted that the information about the adjacency matrix that gets propagated is the one that was initiated at the *initiation* step. Thus even if the graph topology gets changed during the *propagation* process, if the graph remains connected along this time, the final estimate that each agent will have about the graph adjacency is the one that was existent at the time of *initiation* step.

6 Finding the *critical edges* connected to a particular agent:

Once each agent has an idea about the adjacency of the particular connected component of the graph, they should determine which ones are the *critical edges* that are connected to itself. So here we define a *critical edge* of a connected component of a graph to be *such an edge, the removal of which will break the particular connected component into smaller disconnected components.*

The above definition for *critical edge* is extremely limited and it is expected that a modified definition will help in better performance of the algorithm. As we will see at the end of this paper, there can be cases where this definition for *critical edge* will result in catastrophic failure of our present algorithm. In future there will be attempts to modify this definition so as to make the algorithm robust.

As for now, with the present definition for a *critical edge*, we can tell if an edge is critical or not in two trivial ways:

i. Remove the particular edge in question and compute the Laplacian of the newly formed graph. Compute the second eigenvalue of this Laplacian. If this eigenvalue is zero then the edge is a critical edge.

ii. Remove the particular edge in question and compute the adjacency matrix $\tilde{A}$ of the newly formed graph. Compute $\tilde{A}^s = \tilde{A} \cdot \tilde{A} \cdots \tilde{A}$ (s times), for all $s \in \{1, 2, \ldots, N\}$. If at least one of these powers of $\tilde{A}$ is a positive matrix (i.e. a matrix with all the elements being positive), then the particular edge is not a critical edge, else it is.

Since the second method is computationally less expensive, we implemented it for our simulations.
We define $C_i \subseteq N_i$ the subset of neighbors to agent i, the connections to which form critical edges for the particular component of the graph. Agent i will be responsible in ensuring maintenance of only the critical edge that connect i and $j \in C_i$.

7 Stage-I in modifying the velocity of an agent:

Once the *critical edges* have been identified, now the agents need to modify their velocities such that the critical edges attached to them are not broken. The *Stage-I* is a rather a heuristic one where the agent i basically reduce the components of their velocities parallel to the critical edges $(i,j), j \in C_i$, thus slowing down the rate at which it was getting separated from the agents connected to it through the *critical edges*.
If agent i is the one that is under consideration, and C_i is the set of all agents with which it is connected via critical edges, we first define the following:

i. $\mathbf{V}_i = \left\{ \begin{array}{c} V_i^1 \\ V_i^2 \end{array} \right\}$ be the default velocity command that agent i receives from the primary controller.

ii. $\mathbf{r}_{ij}$ be the vector from agent i to agent $j \in C_i$. Thus if $\mathbf{x}_i$ and $\mathbf{x}_j$ are the position vectors of agent i and agent j respectively, $\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i$.

iii. $\mathbf{n}_{ij} = \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} = \left\{ \begin{array}{c} n_{ij}^1 \\ n_{ij}^2 \end{array} \right\}$ be the unit vector along $\mathbf{r}_{ij}$ for all $j \in C_i$.

iv. We define the *parallel component matrix* and *perpendicular component matrix* respectively as follows:

$$\mathbf{M}_{ij}^{\parallel} = \left[\begin{array}{cc} (n_{ij}^1)^2 & n_{ij}^1 \cdot n_{ij}^2 \\ n_{ij}^1 \cdot n_{ij}^2 & (n_{ij}^2)^2 \end{array} \right]$$

$$\text{and, } \mathbf{M}_{ij}^{\perp} = \mathbf{I} - \mathbf{M}_{ij}^{\parallel}$$

where, $\mathbf{I}$ is the identity matrix. Note that with these definitions, $\mathbf{M}_{ij}^{\parallel}\mathbf{V}_i$ and $\mathbf{M}_{ij}^{\perp}\mathbf{V}_i$ are the components of $\mathbf{V}_i$ parallel and perpendicular to $\mathbf{n}_{ij}$ respectively.

v. Thus we define a *moderation matrix* as follows,

$$\widehat{\mathbf{M}}_{ij} = \left\{ \begin{array}{ll} \mathbf{I} & \text{, if } \mathbf{M}_{ij}^{\parallel}\mathbf{V}_i \cdot \mathbf{n}_{ij} > 0 \\ \mathbf{M}_{ij}^{\perp} + f(\mid \mathbf{r}_{ij} \mid)\mathbf{M}_{ij}^{\parallel} & \text{, if } \mathbf{M}_{ij}^{\parallel}\mathbf{V}_i \cdot \mathbf{n}_{ij} \leq 0 \end{array} \right.$$

where, $f(r)$ is a monotonically decreasing function in $r \in [0, r_c]$ such that $f(0) = 1$ and $f(r_c) = 0$. For our simulations we chose $f(r) = \cos \frac{\pi r}{2r_c}$.
Now if we consider $\widehat{\mathbf{M}}_{ij}\mathbf{V}_i$, such a definition of $\widehat{\mathbf{M}}_{ij}$ and f will ensure that the effect of the secondary controller on the velocity will be none when $\mathbf{V}_{ij}$ tends to reduce the distance between agents i and j, and will be very little when $\mid \mathbf{r}_{ij} \mid \ll r_c$. But it will tend to decrease the component of $\mathbf{V}_i$ parallel to $\mathbf{n}_{ij}$ as $\mid \mathbf{r}_{ij} \mid$ approaches r_c. The following figures makes it more clear:

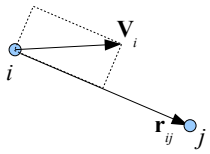

Fig 1a: $\mathbf{M}_{ij}^{\parallel}\mathbf{V}_i \cdot \mathbf{n}_{ij} > 0$

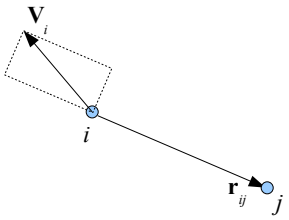

Fig 1b: $\mathbf{M}_{ij}^{\parallel}\mathbf{V}_i \cdot \mathbf{n}_{ij} < 0$

Thus we define the modified velocity after this *Stage-I* of velocity modification as,

$$\mathbf{u}_i = \Big(\prod_{j \in C_i} \widehat{\mathbf{M}}_{ij} \Big) \mathbf{V}_i$$

We note that since matrix multiplication is not commutative, the order in which we choose the $j \in C_i$ will make difference. As for now we choose any arbitrary order.

8 Stage-II in modifying the velocity of an agent:

As evident from the procedure for velocity modification in *Stage-I*, it will not guarantee that a critical edge doesn't get broken. Hence in this *Stage-II* of velocity modification we'll impose more strict bounds on the velocities.

It is to be understood that all the above mentioned processes, starting from the estimation of graph adjacency to the modification of velocities, require finite amount of time. Thus there is a time interval between the issue of two consecutive velocity commands. In out present problem we'll assume that this time interval to be less than an upper bound. We denote this upper bound by Δt. Thus we'll need to figure out an upper bound on the velocity issued at a particular time step such that by the time the next velocity command is issued (i.e. after a time interval $\leq \Delta t$), the critical edges don't get broken. The following proposition gives a loose bound on the magnitude of the velocities. This bound is a sufficient condition for ensuring connectivity of critical edges, but not a necessary condition.

Proposition: *If Δt is the upper limit of time elapsed between the issue of k^{th} velocity command and the $(k+1)^{th}$ velocity command, and if (i,j) was an estimated critical edge at the time of issue of k^{th} velocity command, then $\mid \mathbf{v}_i^k \mid < \frac{r_c - |\mathbf{r}_{ij}^k|}{2\Delta t}$ and $\mid \mathbf{v}_j^k \mid < \frac{r_c - |\mathbf{r}_{ij}^k|}{2\Delta t}$ will ensure that the distance between agents i and j will remain less than r_c till the $(k+1)^{th}$ velocity command is issued (i.e. $\mid \mathbf{r}_{ij}^{(k+1)} \mid < r_c$). [Note: Here $\mathbf{r}_{ij}^k$ refers to the estimate of the separation vector that is used for calculation of the k^{th} velocity command, $\mathbf{v}_i^k$]*

Proof: (i,j) is a critical edge at the k^{th} step implies that $\mid \mathbf{r}_{ij}^k \mid < r_c$. Now

with the given hypotheses we proceed to prove that $\mid \mathbf{r}_{ij}^{(k+1)} \mid < r_c$.
We note,

$$\begin{aligned} &\mid \mathbf{v}_i^k \mid < \tfrac{r_c - \mid \mathbf{r}_{ij}^k \mid}{2\Delta t} \text{ and } \mid \mathbf{v}_j^k \mid < \tfrac{r_c - \mid \mathbf{r}_{ij}^k \mid}{2\Delta t} \\ \Rightarrow &(\mid \mathbf{v}_i^k \mid + \mid \mathbf{v}_j^k \mid)\Delta t \ < \ r_c - \mid \mathbf{r}_{ij}^k \mid \end{aligned}$$

Again, by using triangular inequality,

$$\mid \mathbf{v}_j^k - \mathbf{v}_i^k \mid \leq \mid \mathbf{v}_i^k \mid + \mid \mathbf{v}_j^k \mid$$

Thus,

$$\begin{aligned} &\mid \mathbf{v}_j^k - \mathbf{v}_i^k \mid \Delta t \ < \ r_c - \mid \mathbf{r}_{ij}^k \mid \\ \Rightarrow &\mid \mathbf{r}_{ij}^k \mid + \mid \mathbf{v}_j^k - \mathbf{v}_i^k \mid \Delta t \ < \ r_c \\ \Rightarrow &\mid \mathbf{r}_{ij}^k + (\mathbf{v}_j^k - \mathbf{v}_i^k)\Delta t \mid < \ r_c \qquad \text{[Using triangular inequality]} \end{aligned}$$

Now we note that,

$$\begin{aligned} \mathbf{r}_{ij}^{(k+1)} &= \mathbf{x}_j^{(k+1)} - \mathbf{x}_i^{(k+1)} \\ &= (\mathbf{x}_j^k + \mathbf{v}_j^k \Delta t) - (\mathbf{x}_i^k + \mathbf{v}_i^k \Delta t) \qquad \text{[Worst case scenario]} \\ &= \mathbf{r}_{ij}^k + (\mathbf{v}_j^k - \mathbf{v}_i^k)\Delta t \end{aligned}$$

Thus we have proved, $\mid \mathbf{r}_{ij}^{(k+1)} \mid < r_c$.

Hence now that we have determined the upper bounds on the velocity magnitudes that need to be imposed, we define appropriate scale factors corresponding to each of the critical edges, and hence scale the velocity obtained from *Stage-I* with the smallest scaling factor.

Thus, for agent i, if $\mathbf{u}_i$ is the velocity after it has been modified by the *Stage-I* algorithm, we define scale factors for every $j \in C_i$ as follows,

$$f_{ij}^k = \frac{r_c - \mid \mathbf{r}_{ij}^k \mid}{2\Delta t \mid \mathbf{u}_i^k \mid}$$

And the final velocity command is thus

$$\mathbf{v}_i^k = \min\left\{1, \min_{j \in C_i} f_{ij}^k\right\} \mathbf{u}_i^k$$

Such a velocity command will ensure that the velocity commands remain within the bunds mentioned in the foresaid *Proposition.*

9 Failure situation - Limitations on the definition of *Critical Edge*:

As mentioned previously, our definition for *critical edge* adjacent to a particular agent was rather limited. Here we'll present an example when this definition will fail to detect edges as critical when the maintenance of those edges would have been extremely important in maintaining connectivity of the graph. Consider the situation illustrated by the following figure:

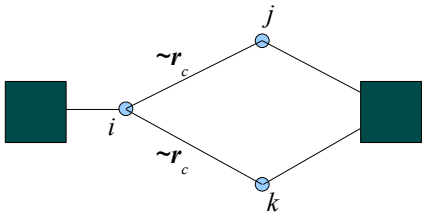

Fig 2: Situation when the algorithm will possibly fail to maintain graph connectivity

The above figure illustrates a case where the agent i is connecting two components of the graph by the edges (i, j) and (i, k). However it doesn't recognize either of those two edges to be critical since removing any one of them will not cause the graph to get disconnected. However here the fact that the lengths of each of these edges are close to r_c makes the situation more demanding. It may very well happen that by the end of the next time step both the edges get broken. None of the agents would have tried to maintain these two edges and hence the graph becomes disconnected.

The obvious possible solution of getting around this problem is to revise the definition of *critical edge* so that in situations like these, the edges (i, j) and (i, k) are also incorporated in the list of critical edges of agents i, j and k. We may mark all the edges connected to agent i whose lengths are close to r_c as *Subcritical edges* and denote the set of neighbors making subcritical edges by S_i. Hopefully we can determine an ϵ such that if $\mid \mathbf{r}_{ij} \mid > r_c - \epsilon$ then we incorporate the agent j into S_i. Then find all possible combinations of edges in S_i whose simultaneous breaking will result in the graph being disconnected. Hence we can include all such edge groups into the list of *critical edges* of i, C_i.

However at present we haven't included the above mentioned scheme in our algorithm for the simulation purpose.

10 Simulation and Results:

We simulated the whole algorithm in MATLAB. The potential field used by the primary controller is just a simple monotonically increasing function of the distance of an agent from its goal. We don't do any collision checking between the agents, and there is no obstacle in the workspace. The following figures demonstrate some of the simulation results. Brief description of each of the results follow.

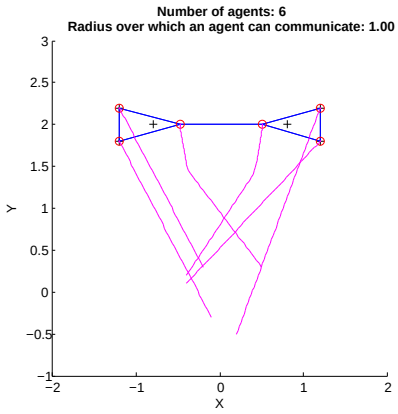

Fig 3a: Simulation-A

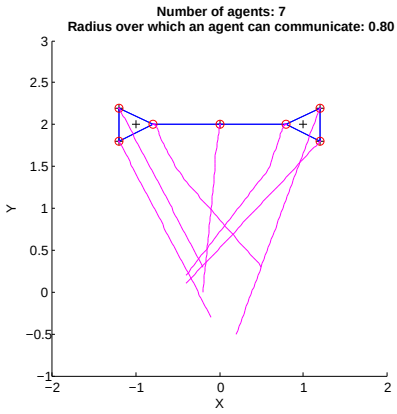

Fig 3b: Simulation-B

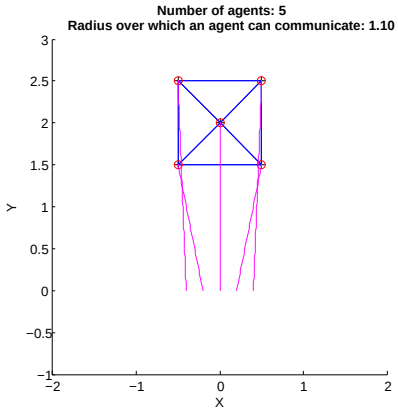

Fig 3c: Simulation-C_1

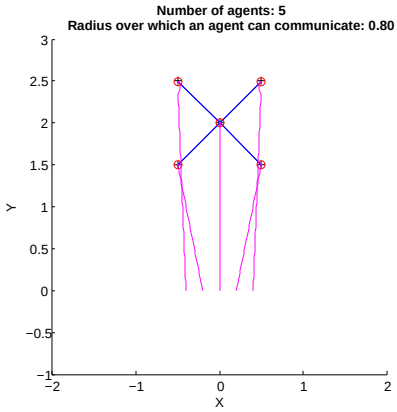

Fig 3d: Simulation-C_2

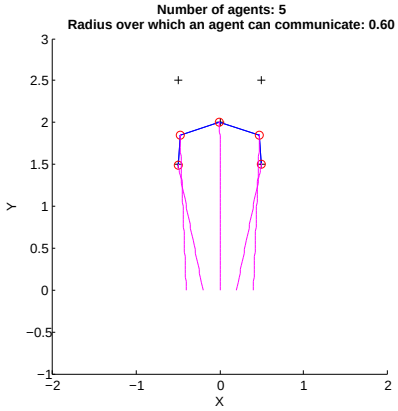

Fig 3e: Simulation-C_3

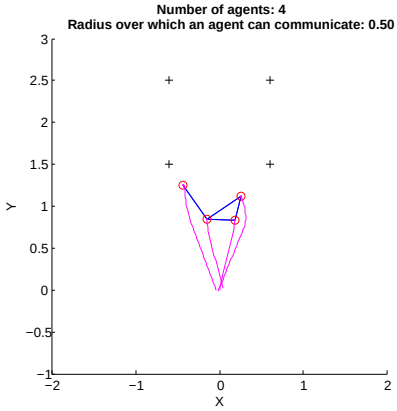

Fig 3f: Simulation-D

In each of the above figures, the black '+' denote the goals for the agents. The red circles mark the final steady-state position of the agents. The curves in magenta are the trajectories of individual agents. The title of each figure gives the number of agents and the value of r_c for the particular simulation.

Simulation-A shows a case where the two agents at the middle have their goals so located that if they reach their respective goals, the graph would have become disconnected. So they started off heading straight towards their goal, but at one point where the distance of separation between the two middle agents got close to $r_c = 1.0$, then they started modifying their velocities. Hence they changed their trajectories and finally reached positions such that the graph remains connected. As we can observe, in this particular simulation the agents have reached somewhat close to their optimum positions near their respective goals that were possible to be reached without sacrificing the graph connectivity.

Simulation-B shows a similar case, but now we have seven agents, and the agent at the center is connected by two critical edges on either sides. This simulation illustrates the ability of the algorithm to handle more than one critical edge for a particular agent.

Simulation-C_1 is a simple case where the goals are so located that every agent can reach their respective goals without sacrificing graph connectivity. Here we used $r_c = 1.1$.
In *Simulation-C_2* we reduce the value of r_c to 0.8. Here too the agents can reach their respective goals, but we note that the value of r_c is small enough to break direct connection between the agents at the corners of the square. They use the agent at the center of the square as an anchor for maintaining connectivity.
In *Simulation-C_3* we have further reduced the value of r_c to 0.6. Now we notice that not all the agents can any more reach their respective goals without sacrificing graph connectivity. Hence two of the agents stop at points quite far from their respective goals.

Simulation-D shows a worst case scenario. Here the value of $r_c = 0.5$ is very small compared to the separation between the goals. Hence at one point when the agents start moving far from each other rather rapidly, their velocities get modified abruptly and they finally come to a halt at positions very far from their respective goals. This particular simulation clearly demonstrates that our algorithm may not give the near-optimum solution under many such circumstances.

11 Conclusion and possibility for future works:

In the present work we have been able to develop a framework for a very elementary algorithm that will be able to attach itself as a secondary controller alongside the primary controllers for mobile agents in order to ensure that the connectivity of the network does not get broken. Although the present algorithm is not quite robust or optimum, we have already figured out the drawbacks and are on our way towards finding solutions for overcoming them. As already mentioned within the paper, there are several places in the present algorithm that requires immediate attention and improvement.

The most important one being the definition of *critical edges*. We will try

to follow the steps mentioned under *Section 9* in order to broaden the scope of critical edges and hence make the algorithm more robust.

The next important and demanding modification is to ensure that the algorithm returns a more optimum result. The illustration in *simulation-D* shows that in particular cases where the distance of separation between the goals are much greater than r_c, our algorithm performs in a rather inefficient manner.

12 Acknowledgement:

I would like to thank Professor Ali Jadbabaie, Department of Electrical and Systems Engineering, University of Pennsylvania, for his kind and valuable advices which made this work possible.

13 References:

[1] A. Tahbaz Salehi and A. Jadbabaie, *A one-parameter family of distributed consensus algorithms with boundary: from shortest paths to mean hitting times*, Proceedings of the IEEE Conference on Decision and Control, San Diego, CA, December 2006.

[2] M. C. De Gennaro and A. Jadbabaie, *Decentralized Control of Connectivity for Multi-Agent Systems*, IEEE Conference on Decision and Control, San Diego, CA. December 2006.

[3] Ali Jadbabaie, Jie Lin, and A. Stephen Morse, *Coordination of Groups of Mobile Autonomous Agents Using Nearest Neighbor Rules*, IEEE Transaction on Automatic Control, Vol. 48, No. 6, 2003.