Adaptive Embedded Roadmaps For Sensor Networks

Gazihan Alankus, Nuzhet Atay, Chenyang Lu, O. Burchan Bayazit

Washington University in St. Louis
{gazihan,atay,lu,bayazit}@cse.wustl.edu

Abstract— In this paper, we propose a new approach to wireless sensor network assisted navigation while avoiding moving dangers. Our approach relies on an embedded roadmap in the sensor network that always contains safe paths. The roadmap is adaptive, i.e., it adapts its topology to changing dangers. Mobile robots in the environment use the roadmap to reach their destinations. We evaluated the performance of embedded roadmap both in simulations using realistic conditions and with real hardware. Our results show that the proposed navigation algorithm is better suited for sensor networks than traditional navigation field based algorithms. Our observations suggest that there are two drawbacks of traditional navigation field based algorithms, (i) increased power consumption, (ii) message congestion that can prevent important danger avoidance messages to be received by the robots. In contrast, our approach significantly reduces the number of messages on the network (up to 160 times in some scenarios) while increasing the navigation performance.

I. Introduction

Traditionally, mobile robots rely on on-board sensors to collect environmental information. However, as the technical challenges of wireless sensor networks are being solved, a new interest is raised to employ them in the robot navigation task. It has been shown that the assistance of a sensor network could significantly improve the navigation task when there are dynamically moving dangers to avoid in the environment [1]. However, any navigation algorithm utilizing sensor networks must consider the limited resources provided by the network, mainly low bandwidth, small battery and limited processing power. This requires intelligent approaches that can utilize the hardware efficiently.

In this paper, we propose Adaptive Embedded Roadmaps (AER) as a new approach to sensor network assisted navigation. The problem is to navigate safely in a danger field, i.e., to reach a goal from a starting point while avoiding dynamic danger regions. This requires routes to be updated continuously to avoid the dangers. Our solution is to embed a roadmap inside the sensor network that will maintain a collection of possible paths. This roadmap is built in the sensor network using a distributed fashion similar to the probabilistic roadmap method (PRM) [2]. First, some motes probabilistically become roadmap nodes, i.e., milestone motes. Through message passing, these motes connect themselves to the nearby milestone motes. The optimal message route between two milestones becomes an edge of the roadmap. When a goal is specified, the embedded roadmap creates a spanning tree from all milestone motes to the goal in a distributed fashion, which is in turn followed by one or more robots. Since the environment is dynamic, the network is adaptive. For example, if a mote on an edge is in danger, the edge is disconnected and an alternative edge is built. Similarly, if a milestone mote is in danger, the roadmap node

it hosts is migrated to a nearby non-milestone mote and the connections are rebuilt. We also address the physical obstacle problem by using a lazy approach [3]. While following the embedded roadmap, if a robot discovers a physical obstacle, the robot informs the milestone motes that have the edge over the obstacle. Those motes then disconnect themselves and the robot is directed to an alternative path.

Remember that wireless sensor network assistance to robot navigation can be classified into two groups: (i) on-board processing, (ii) in-network processing. In the first approach, the sensor network transfers the spatio-temporal information to the robot and the robot makes navigation decisions. The number of messages are proportional to the number of sensor motes involved in data collection, which can be too large. Certain techniques are suggested to target interesting locations [1], but they ultimately have to deal with a certain vicinity of the robot. The second approach to sensor network assisted navigation aims to find the path in the sensor network using the limited computing power of sensor motes in a distributed fashion [4], [5]. The straightforward technique is to build a navigation field over the sensor motes to reach the goal. While this approach is sufficient in the presence of static danger regions (i.e., dangers that are not spreading), the field needs continuous update in the presence of moving dangers. This may cause high network congestion, as shown in [6].

Our approach has several advantages over the traditional approaches. It targets the embedded network similar to target based strategy of on-board processing, yet it is not restricted to the local regions. It uses the global spatio-temporal information similar to in-network processing, but it does not constantly update a global navigation field. The routes include only a subset of the motes and are updated only when safety of the embedded roadmap changes. This way, the reduced message traffic increases both sensor life and improves navigation safety by avoiding network congestion. Additionally, since only the motes that are part of the roadmap need to be active, the other motes can sleep to reduce power consumption. With the help of the robots that use the sensor network, our approach can also address obstacles that are invisible to the sensors, an important point that the traditional in-network algorithms fail to consider. Finally, the embedded roadmap could be utilized to coordinate the movements of the multiple robots.

We have experimented with several scenarios including multiple robots, multiple goals, dynamic obstacles, static obstacles, hardware failures etc. Our simulations show that under realistic conditions our algorithm performed far better than traditional in-network processing algorithms. We have also showed that it is feasible to implement our algorithm

on real hardware.

Our results demonstrate that (1) an embedded adaptive roadmap can be used to represent spatio-temporal information of an environment, (2) such a roadmap can safely guide a mobile robot towards its destination at a small fraction of communication cost compared to basic sensor network assisted navigation algorithms.

In the next section, we give a summary of related work. We present a formal definition of the problem in Section III. We briefly describe our system in Section IV. Section V discusses how to build, maintain and utilize an adaptive roadmap on sensor networks. We present our experimental results in Section VI and Section VII concludes our paper.

II. RELATED WORK

The most commonly used algorithms for sensor network assisted navigation use a global navigation field over the sensors. In [4], the goal generates an attractive potential field that pulls the robot towards the goal, while an obstacle generates a repulsive potential field that pushes the robot away from the obstacle. The method in [7] locally iterates to compute utilities that guide the robot to the goal. This approach is further used in [8], [9] and [10] for robot coverage and exploration of space and for multi-robot task allocation. In [11], a similar method is analyzed. The approach presented in [12] proposes navigation of mobile sensor nodes by forming initial paths with a global flooding. Since initial path stays constant, this approach cannot handle dynamic obstacles. The approach used in [13] assumes that a path already exists in the network, and uses controlled flooding to guide the robot to the start of the path. Another global navigation field approach is suggested in [5] for sensorless communication platforms using GNATs. In that approach, the passing mobile robots communicated the attraction information to GNATs which in return updated the navigation field. Generally, the navigation field based algorithms can be very costly in large networks with dynamic obstacles since any update on the field requires a global flooding. In order to reduce the communication cost, the targeted querying protocols were suggested [1], [6]. In these approaches, the sensors send the spatio-temporal information to the robot and robot makes the navigation decisions.

Generally, navigation field based techniques increase the power consumption and require large bandwidths. Both of which are decisive factors in sensor network performance. The targeted querying algorithms do not suffer from these constraints but they usually target nearby locations, i.e., only collect information from robots' vicinity which may affect navigation performance. In our work, we address the shortcomings of both approaches. Through embedded roadmap, we have a targeted navigation field that can safely move the robots to their destinations in the presence of dynamic obstacles.

Recently, Buragphain et al. proposed navigation algorithms based on the skeleton graph of a sensor network [14]. Similar to a roadmap, a skeleton graph is a sparse subset of the original network, which helps reduce the communication overhead for navigation. However, a distinguishing feature of our algorithm is that it dynamically maintains and adapts the roadmap in response to the movement of danger fields to

order to enhance the robustness of navigation approach in dynamic environments. In contrast the algorithms presented in [14] did not present algorithms for maintaining the skeleton graphs when the danger field moves. Furthermore, we have implemented and demonstrated our algorithm on a physical sensor network testbed, while their algorithms are evaluated through simulations.

III. PROBLEM FORMULATION

The navigation problem that we address in this paper is to find *safe paths* for mobile robots through a sensor field. We define a *safe path* as a path that is clear of *dynamic obstacles*, i.e., obstacles whose location or shape changes with time (e.g. car, fire).

In this paper, we consider fire as the representative example for a dynamic obstacle. Thus, the temperature of the region traversed by a robot is a function of time and is affected by the location and movement of fire. In this case, the problem can be restated as that of finding safe paths for mobile robots, from start to goal, without the robots getting burned. The temperature values of a region is discretized to different danger levels using a number of thresholds Δ_i. The danger level δ between Δ_i and Δ_{i+1} is considered to be i. The number and value of thresholds are application specific design choices. In this paper we used four thresholds as follows. $\delta = 0 (cool)$ (if T, i.e., temperature of the region, is less than Δ_{cool}), $\delta = 1 (normal)$ (if $\Delta_{cool} < T \leq \Delta_{normal}$), $\delta = 2 (warm)$ (if $\Delta_{normal} < T \leq \Delta_{warm}$), $\delta = 3 (hot)$ (if $\Delta_{warm} < T \leq \Delta_{hot}$), $\delta = 4 (burn)$ (if $\Delta_{warm} < T \leq \Delta_{burn}$) and $\delta = \infty$ above Δ_{burn}, where the sensor hardware no longer functions. This discretization is useful since it avoids messages generated by slight changes in temperature.

A sensor or robot is assumed to get burned if the temperature at its location is higher than the threshold Δ_{burn}. A safe path is now redefined as one where the maximum temperature along the path taken by the robot remains below the threshold Δ_{hot}, while the robot is on the path. Cooler paths are thus considered safer than hotter paths.

The goodness of path $\mathbb{P}$ that passes through motes (i.e., wireless sensor nodes) $m_{i \in 1..n}$ is defined by following function:

$$goodness(\mathbb{P}) = c_1 (\sum_{i \in 2..n} |m_i - m_{i-1}|) + c_2 \max_{i \in 1..n} \delta_i \quad (1)$$

In other words, the goodness of path is the sum of the normalized path length and scaled maximum danger level on the path. By changing the variables c_1 and c_2, the path can be weighted for the length or safety.

We make the following assumptions in the paper: (i) Motes are location aware. (ii) The robot communicates with the sensor network through an on-board gateway device. (iii) Motes have a limited sensing range R_S. (iv) Wireless communication between motes takes place in a fixed-radius cookie-cutter radio model with message congestion.

The sensing range R_S is defined by the continuous behavior of danger. It is chosen such that if the temperature sensed by a node is below the threshold Δ_i, then the temperature at any point within the sensing range is below Δ_{i+1}. Therefore, paths with sensed temperature above Δ_{hot} may have points above Δ_{burn}, which explains our choice of safe paths being below Δ_{hot}.

Note that, if the motes do not have location sensors (e.g., GPS or crickets [15]), they may find their locations based on network connectivity or radio signal strength using existing localization algorithms [16], [17]. Alternatively, our navigation algorithm can be modified to use Adaptive Delta Percent [7] that utilizes sensor signal strength, in which case we would not need location awareness for the motes.

Even though we consider the specific scenario where the dynamic obstacle is fire, our solution can be generalized to other types of dynamic environments where safety is defined by changing sensory values (e.g., chemical spills, hazardous gas and air pollution).
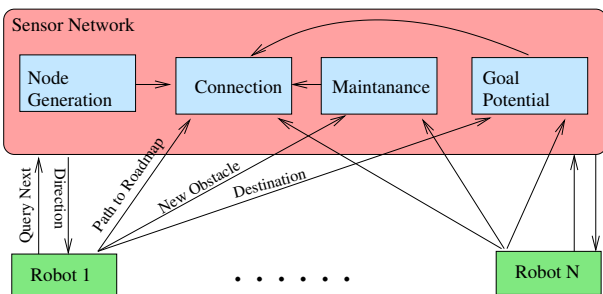

Fig. 1. System overview.

IV. SYSTEM OVERVIEW

In order to assist the robot navigation, the sensor network must have some abstract mechanisms. For example, there must be some mechanisms for *node generation* and *node connection* to build an embedded roadmap. After it is built, the sensor network also needs a *maintenance* mechanism to keep the embedded roadmap up-to-date. Finally, when a robot asks for a path, the sensor network needs to find an optimal route to the goal through *Goal Potential* mechanism. Figure 1 summarizes the interaction within the network and with the robots. After *node generation*, *connection* builds the roadmap. *Maintenance* may revoke *connection* to disconnect some edges that are in danger, or find alternative edges. When the robot arrives, it finds a path to the embedded roadmap through *connection* mechanism. *Goal Potential* mechanism is responsible for finding the best path to the goal. The embedded roadmap then directs the robot towards the goal. While following the path, if the robot discovers an obstacle that is invisible to sensor network's sensors, it informs the embedded roadmap and an alternative route is found.

V. SENSOR NETWORK ASSISTED NAVIGATION

After the initial deployment of the sensor network, the embedded roadmap is built in a distributed fashion. Node generation is handled by turning some motes to milestone motes (i.e., motes that contain a roadmap node). The connection phase is a local planning operation where the milestones broadcast connection request to their vicinity. This request is further propagated by receiving motes. The propagation continues until requests from two milestones intersect. In which case, the mote at the intersection sends connection messages to both originators. Among several possible connections between two milestones, the best path (according to Eqn. 1) is selected as the edge.

Once the roadmap is built, a robot can utilize it to navigate. Since the robot is not aware of the topology of the roadmap, the sensor network must find the best path. For this purpose, we use an NF2-like [18] wavefront expansion on the roadmap. First, through geographic routing [19] , the robot asks the mote closest to its goal (i.e., goal mote) to connect itself to the roadmap. Once the goal mote is connected to the embedded roadmap, it originates a potential wave on the roadmap where the potential value represents the goodness of the path. When the wave reaches a milestone mote, the milestone sets the best direction towards the goal. At the same time, the robot requests a connection to the roadmap. After receiving several responses from the nearby milestone motes, then the robot selects the best route and follows it. When the robot reaches a milestone mote, the mote directs it towards the next mote in the path. This process is repeated until the robot reaches its goal.

Our embedded roadmap is adaptive and changes based on the spatio-temporal information. The topology and the edge weights are altered if the danger spreads towards the roadmap, hence the roadmap always contains safe paths. If the robot on its path recognizes an obstacle unknown to the sensors, it informs the embedded roadmap to remove edges on the obstacle.
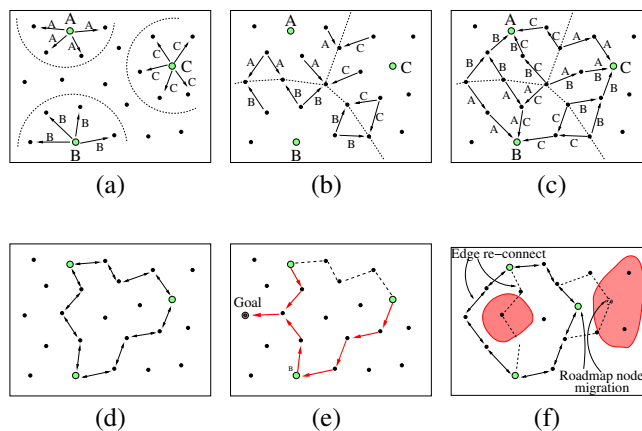

Fig. 2. Building embedded roadmap. (a) Milestones motes, (A, B,C), start neighbor discovery. (b) NEIGHBOR-DISCOVER is propagated by the receiving motes. (c) Neighbors are found and propagated back to the milestones. (d) According to the goodness metric, the best routes to the neighbors become the edges. (e) Goal connects to the roadmap and the best routes through a navigation field on the roadmap are set. (f) An edge on danger disconnects then re-connects and a roadmap node migrates (dashed lines are previous edges, shaded areas are danger regions).

A. Building Embedded Roadmap

The building process of embedded roadmap is similar to traditional PRMs [2]. The roadmap nodes are now the milestone motes selected according to some criteria, e.g., probabilistically or based on mote capabilities. Once the milestone motes are selected then they are connected through message passing. In this process, an edge between two milestone represents the best sequence (according Eq. 1) of motes to reach one milestone from the other.

Node Generation. Each mote decides to host a roadmap node with a probability p. If they host a roadmap node, then they become milestone motes. Please note that it is also possible to utilize alternative decision criteria such as mote capabilities or sensor inputs.

Node Connection. The first step in node connection is to discover the closest milestone motes that are possibly out of each other's communication range. This is done by sending NEIGHBOR-DISCOVER messages to one-hop neighbors (Figure 2(a)). A NEIGHBOR-DISCOVER message has four fields, $[m_s, \mathbb{M}_o, l_r, \delta_r]$ i.e., the one-hop neighbor that sent this message, the originator (the discovering milestone), the length of the route to originator, and the maximum danger level on the route. Since a mote m receives the NEIGHBOR-DISCOVER only from its one-hop neighbors, and it knows their positions, the goodness of the route to $\mathbb{M}_o$ is $goodness = c_1(|m - m_s| + l_r) + c_2 \max(\delta_r, \delta_m)$, where δ_m is the danger level of mote m.

Each mote on the network has a *Vicinity* table which contains the list of the milestone motes from which that mote received NEIGHBOR-DISCOVER messages. The table also stores the goodness of the route and one-hop neighbor that sent the message. Upon receiving a new NEIGHBOR-DISCOVER message, a mote checks its *Vicinity* table to see if the new route is better than any existing route to $\mathbb{M}_o$ (if there is one). If the old route was better, the message is discarded and nothing further is done. Otherwise, *Vicinity* table is updated according to NEIGHBOR-DISCOVER message to represent a better route.

Next, the mote checks the *Vicinity* table to see if other NEIGHBOR-DISCOVER messages were received from different milestone motes. If no message was received, then NEIGHBOR-DISCOVER is propagated further after updating the fields (Figure 2(b)). If the mote has received a NEIGHBOR-DISCOVER message from another mote $\mathbb{M}_n$, both $\mathbb{M}_o$ and $\mathbb{M}_n$ needs to be informed about their neighborhood. This is achieved by sending NEIGHBOR-FOUND messages to both. This message has five fields, $[m_s, \mathbb{M}_o, \mathbb{M}_n, l_n, \delta_n]$, i.e., sender of the message, the destination milestone (originator of discovery), the neighbor milestone, the length of route to the neighbor and maximum danger level on route the neighbor. Since the mote needs to inform both neighbors, it sends two NEIGHBOR-FOUND messages, one for each site. l_n and δ_n of the messages are found from *Vicinity* table and the messages are sent to the one-hop neighbors stored in *Vicinity* (the best route). Upon receiving a NEIGHBOR-FOUND message, each mote checks its *Vicinity* table to see if there was a better route from $\mathbb{M}_n$. If not, it adds the neighbor to its *Vicinity* table, updates m_s, l_n and δ_n and propagates the NEIGHBOR-FOUND message to the one-hop neighbor on the best route towards the originator. This process is repeated until NEIGHBOR-FOUND message is received by $\mathbb{M}_o$ (Figure 2(c)). This milestone mote checks its *Vicinity* table to see if it knows a better route to neighbor $\mathbb{M}_n$. If not it adds this route to the *Vicinity* table. Otherwise, NEIGHBOR-FOUND message is discarded.

Once the neighboring milestones are discovered, each milestone mote creates an edge between itself and the lower-id neighbor milestones by sending CREATE-EDGE message in their directions. The direction is selected from *Vicinity* table entry. Each intermediate mote receiving this message

recognizes itself as an edge-mote and propagates the message to the next mote in the direction of the destination milestone mote. Once CREATE-EDGE message is received by the destination milestone mote, an acknowledgment is sent back to the originator. At the end of this process, all milestone motes know their milestone neighbors on the roadmap and the weight (goodness) of the edge between them. Similarly, all of the edge motes are aware that they are part of the embedded roadmap. Figure 2(d) shows an example of an embedded roadmap after the connections and Algorithms 1 and 2 summarize these processes.

Algorithm 1 Node Connection: Milestone Mote

1: Broadcast NEIGHBOR-DISCOVER message
2: **while** not time-out **do**
3:   **if** NEIGHBOR-FOUND message is received **then**
4:     check *Vicinity* table for neighbor
5:     **if** Path to neighboring milestone is better than current route **then**
6:       Add neighboring milestone, direction to it and path goodness to *Vicinity*
7:     **end if**
8:   **end if**
9: **end while**

Algorithm 2 Node Connection: Non-Milestone Mote

1: **while** waiting for messages **do**
2:   **if** NEIGHBOR-DISCOVER received **then**
3:     Compute goodness of route to originator
4:     **if** Previous route to originator in *Vicinity* is better **then**
5:       Go back to mesg. waiting state
6:     **end if**
7:     **if** Another milestone mote in *Vicinity* **then**
8:       Unicast NEIGHBOR-FOUND messages in direction of milestones
9:       Goto back to mesg. waiting state
10:     **end if**
11:     Update NEIGHBOR-DISCOVER message and propagate to neighbors
12:   **end if**
13:   **if** NEIGHBOR-FOUND received **then**
14:     Compute goodness of route to neighbor
15:     **if** Previous route to neighbor in *Vicinity* is better **then**
16:       Go back to mesg. waiting state
17:     **end if**
18:     Update NEIGHBOR-FOUND message and unicast to neighbor in route to originator
19:   **end if**
20:   **if** EDGE-CREATE received **then**
21:     set state to Edge-Mote
22:     Update EDGE-CREATE message and unicast to neighbor in route to destination milestone
23:   **end if**
24: **end while**

B. Goal Potential

Goal motes represent the robot destinations. The decision to become a goal mote can be initiated by environmental factors or by a robot. In order to utilize the robustness provided by embedded roadmap functions, the goal mote becomes a milestone mote if it is not already one. If the goal is on an edge mote, the edge is broken. After the goal mote becomes a milestone, it connects to the nearby milestone motes. The connection is done through the same mechanisms used in roadmap connection phase (i.e., Algorithms 1 and 2). The next step is generating the navigation field on the roadmap, using a GOAL message originating from the goal mote. A GOAL message has four fields $[m_s, G_o, l_g, \delta_g]$, i.e., the sender of the message, goal id, length of the path to the goal and maximum danger level on the way to the goal. Once initialized by the goal mote, this message is forwarded to all motes on the roadmap, aggregating the information about best paths to the goal. Every mote m on the roadmap

---

Algorithm 3 Goal Dissemination: All Roadmap Motes

1: **if** GOAL message is received **then**
2:    check *Goals* table for this goal
3:    **if** Incoming message is better than the one in the table **then**
4:       G = aggregated goal message with values for the path including this mote
5:       Set the entry in the *Goals* table according to G
6:       Send G to other neighboring edge motes
7:    **end if**
8: **end if**

Algorithm 4 Maintenance: Milestone and Edge Motes

1: δ_t = current temperature level
2: **for** Each edge neighbor **do**
3:    **if** $\delta_t = \delta_{hot}$ **then**
4:       break the edge, send BREAK-EDGE
5:       **if** This is a milestone mote **then**
6:          Call milestone migration
7:       **end if**
8:    **else**
9:       δ_l = last temperature level sent to this neighbor
10:       **if** $\delta_t \neq \delta_l$ **then**
11:          send UPDATE-EDGE with δ_t
12:       **end if**
13:    **end if**
14: **end for**

Algorithm 5 Milestone Migration

1: Ask the one-hop neighbors for their temperature readings
2: Wait until they answer or timeout ends
3: Target = one-hop neighbor with the best temperature
4: Send MIGRATE-MILESTONE message to Target
5: Cancel being milestone for this node

---

maintains a *Goal-Potentials* table that has one entry per goal that keeps goodness of the path and its one-hop neighbor. The contents of the record is propagated to other nodes on changes, similar to NEIGHBOR-DISCOVER aggregation, i.e., the outgoing GOAL message would contain the updated path length as $l_g + |m - m_s|$, the danger level as $\max(\delta_g, \delta_m)$ etc. This way, a distributed minimum spanning tree is maintained on the roadmap for each goal mote. Algorithm 3 summarizes the goal dissemination and Figure 2(e) shows an example. Note that if multiple robots try to reach the same goal, the goal potential of that goal needs to be computed only once.

C. Roadmap Maintenance

Roadmap Edge Maintenance. In order to direct the robots to the safe paths, the embedded roadmap always needs to be aware of goodness of the routes to the goal. To provide an up-to-date information of goodness, an edge mote on the roadmap sends UPDATE-TEMPERATURE messages to its edge neighbors whenever its danger level changes (i.e., there is a significant change in the temperature). This message is propagated to the milestones at each end of the edge. Upon receiving this message a milestone updates its *Vicinity* and *Goal-Potentials* tables accordingly. If the new temperature is greater than Δ_{hot}, the edge is broken. After an edge is broken, the milestone mote with the higher id tries to reconnect the edge after $t_{reconnect}$ seconds. If a change in the edge (either goodness or topology) affects the best route, the milestone mote initiates the aggregation of GOAL messages to its neighbors to maintain the best paths to the goal. Algorithm 4 summarizes this process.

Roadmap Node Migration. Milestone motes are the most important motes in this algorithm. The roadmap may become highly disconnected if they die or sense Δ_{hot}. Therefore it is important for the milestone motes to stay alive and be in low temperature areas. In time, milestone motes may inevitably get in fire. This leads to broken edges and a possibly disconnected roadmap. To overcome this problem, we introduce maintenance of milestone motes by milestone migration.

The purpose of milestone migration is to make milestone motes transfer the roadmap node to one of their neighboring motes. When the milestone mote senses Δ_{hot}, it asks its one-hop neighbors for their temperature readings to see which neighbor is the safest. Then, it sends a MIGRATE message to the safest neighbor including the current state of the milestone mote. It also sends BREAK-GOAL messages to its neighboring milestone motes to break the edges and connect to the new milestone mote. The edge connections to the new milestone mote are done similar to basic edge creation. Pseudo code for migration can be seen in Algorithm 5. Figure 2(f) shows an example maintenance scenario.

Robustness and Node Failure. If a mote on the roadmap dies before it informs other motes, the roadmap may become disconnected. In order to avoid such cases, a heart beat message could be sent over the roadmap motes to check their health. Instead of continuously checking the health of all the motes in the roadmap (which is a costly operation), we check the motes only when the robot is about to move on their edges. When a robot arrives to a milestone mote, the milestone sends a SENTINEL message towards the goal. This message is propagated until it reaches the milestone at the other end of the edge. If any edge mote propagating this message to its one-hop neighbor could not get an acknowledgment, it becomes a milestone mote, breaks the edge and connects to the milestone motes in the vicinity. *Goal-Potentials* are updated accordingly. As an additional precaution, while following the roadmap, if a robot could not get an reply from the next mote on the path, the robot informs the last mote. The last mote then behaves as if SENTINEL message failed, and updates the connections.

D. Navigation

Reaching Roadmap. In order to find the roadmap, the robot makes a local query to the sensor network with a FIND-ROADMAP message. The motes that receive this query forward it along the entries in their *Vicinity* table. A local query tree is formed as a result. When this tree hits a mote in the roadmap, a ROADMAP-FOUND message is sent to the robot along with the distance and temperature information of the path. The robot selects the best one among these messages and starts following it until it reaches the roadmap.

Following Roadmap. Once the robot is on the roadmap, it sends FOLLOW-QUERY messages and gets the goal information from its one hop neighbors using ROADMAP-FOLLOW messages. This is repeated until the robot reaches the goal. If the roadmap edge that the robot was following is broken, the robot sends a FIND-ROADMAP message to the network and tries to reach the roadmap again. If the robot discovers a static obstacle, an OBSTACLE-HIT message is sent to the current edge, which in turn breaks the edge and disables its recreation. If the robot senses a temperature level of Δ_{hot}, it returns back to the last mote on its path and informs the mote. The mote then breaks the edge and the robot is directed to an alternative path.

VI. EXPERIMENTS

In our experiments we would like to answer following questions: (i) how successful an embedded roadmap is in

preventing the robot moving into the danger, (ii) how well our approach is working with respect to existing navigation field based algorithms (iii) how feasible it is to use our algorithm on real hardware.

In order to answer those questions, we ran our experiments both in the simulated sensor network and on real hardware. Videos of the experiments can be viewed at http://www.cse.wustl.edu/~bayazit/sn.

Simulations. We first compare our adaptive embedded roadmap (AER) algorithm's performance to traditional navigation field algorithms. For this purpose, we compare the efficiency and navigation safety of each algorithm. We are also interested in evaluating our algorithm's robustness and its behavior when the physical obstacles exist in the environment. In our experiments, we have run our algorithm on several fire scenarios. As a representative of global navigation field algorithms in sensor networks, we have selected [4] which we will refer as GNF. In our implementation of GNF, we included a number of improvements such as maximum hops proposed in [4] instead of periodically flooding the entire network from each mote. In our experiments, we have used NS-2 Network Simulator [20]. NS-2 is one of the most commonly used simulators in the networking research and can accurately simulate network behavior such as wireless message transfer, message transmission and network congestion, etc. In order to have a realistic fire simulation, we have used NIST Fire Dynamics Simulator (FDS) [21]. We used 10 different realistic fire scenarios. In all the scenarios, the fire starts in different locations scattered over the region and then spreads over the region in time. The size of the environment is $450m \times 450m$. The simulated robot can detect the temperature and nearby static obstacles. Experiments with non-holonomic constraints, localization and motion uncertainties are left for future research. Unless otherwise stated, there are no static obstacles in the scenario. During the experiments, the default values for the AER parameters were as the following: $\Delta_{cold} = 50, \Delta_{warm} = 70, \Delta_{hot} = 90, \Delta_{burn} = 120, c_1 = 0.01, c_2 = 100$ (see Section III, note that since safety is more important for us, the danger constant is very high). The random mote failure ratio we used was less than 1%. We experimented with the roadmap node generation probability and found 0.1 to be a useful constant. The values below had problems with coverage and values above introduced redundancy in the graph. The quantitative analysis of roadmap node generation probability, as well as adaptation of number of nodes in the roadmap are left for future research.

Efficiency. The purpose of our first experiment is to compare the efficiency of AER and GNF, which is measured using the number of messages. Note that messages are sent very quickly in the network unless they are dropped, but sending messages is the main cost in a sensor network because of high battery usage. To compare the scalability of each approach, we created three networks with (i) 25 motes, (ii) 100 motes, and (iii) 225 motes. All the networks have uniform distribution of motes (i.e., grid topology). We ran both algorithms with different networks with the same fire scenario, with the same start and goal points. While GNF generated and updated the navigation field as described in [4], AER built and maintained the embedded roadmap. We used the average of 10 runs for AER since
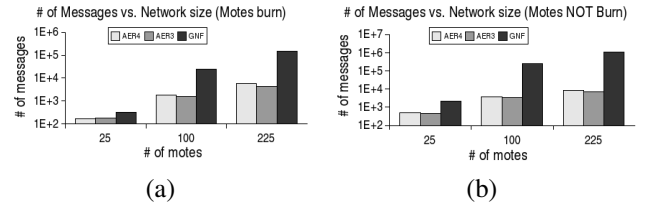

Fig. 3. Number of Messages. x-axis represents number of motes in the environment, logarithmic y-axis is number of messages. (a) When the motes fail in high temperature . (b) When the motes are not effected by fire.

it is probabilistic. To see how a change in the number of danger levels would effect our algorithm (see Section III), we experimented running AER without the danger level δ_{warm} (i.e., a total of three danger levels as opposed to original four). Additionally we evaluated how the survivability of a mote effects the performance. Normally, a mote burns when its temperature reaches δ_{hot}. However, there are several examples of danger scenarios where the danger may not harm the sensor (e.g., chemical spill or radiation fall-out). Hence we have ran each algorithm with and without sensor burning. Figure 3 shows our experimental results. In the figure, AER4 refers to original algorithm and AER3 refers to AER with three danger levels. The x-axis represents number of motes in the sensor network and logarithmic y-axis represents the number of messages generated in the network. In Figure 3, (a) is when motes burn, (b) is when they do not. As the number of sensors increases the number of messages generated by GNF significantly increases. Also, if the motes are not effected by the danger, the efficiency of GNF further decreases. In all cases, AER algorithm performed better than GNF. For example, in experiment with 225 motes and no mote burning, AER generated 160 times fewer messages. Both AER variants performed similarly. This suggests that number of danger levels do not significantly effect the total number of messages in the network hence the maintenance cost is actually very low. We were expecting to see more messages from GNF so we further investigated the number of dropped messages. We found that a large portion of the messages generated by GNF were actually dropped because of network congestion. This is the main drawback of GNF algorithm since the motes close to the robot may fail to propagate messages to the robot, jeopardizing its safety.

Navigation Safety and Quality. Our first experiment showed that AER is more efficient than GNF. Next we wanted to see if AER can generate paths as safe as GNF. In order to compare the path safety, we considered three different cases: (i) single robot reaching single goal, (ii) two robots reaching two goals, (iii) four robots reaching four goals. The sensor field has 225 motes placed uniformly. In this experiment, we considered 10 fire scenarios. Our comparison is based on the average safety of each algorithm. Figure 4(a) shows that AER performed better than GNF, because in some scenarios with GNF, it was too late for the robot to escape the fire when the danger information arrived. Figure 4(b) shows the distances that the robots traveled for the same experiments. This figure shows that the paths generated by AER is not very different in length than paths
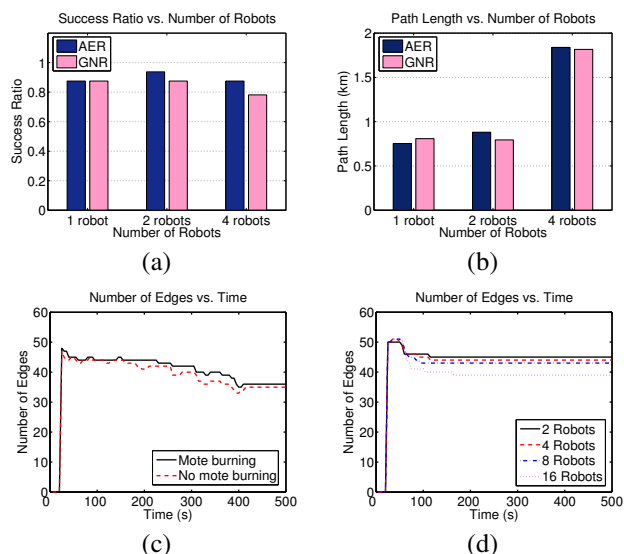
Fig. 4. Simulation results. (a) Navigation safety for different scenarios. (b) Comparison of path lengths. Path lengths do not scale linearly with number of robots since different scenarios include different length routes. (c) Number of edges on the roadmap as the danger spreads. (d) Number of edges on the roadmap as the robots discover static obstacles.

of GNR.

Robustness and Node Failure. In this experiment, we evaluated the robustness of our algorithm by increasing the mote failure ratio ($10\%, 20\%$ and 30%). We ran the same scenario with a single robot for each failure ratio. During the simulation we randomly killed some motes until the failure ratio is reached. These experiments are repeated 10 times. We found that AER is very robust and the robot could reach the goal every time. We observed that SENTINEL messages detected the disconnected motes and helped find alternative paths, as long as the connectivity of the roadmap is satisfied. We are also interested in observing the behavior of AER when fire is spreading. Our maintenance functions should keep the network connected and find alternative edges as the danger moves. Figure 4(c) shows the average number of edges in the roadmap as the fire spreads. In the figure, experimental results for both mote-burning and non-mote-burning cases are shown. It can be seen that there is a slight decline in the number of edges. We found that as the fire spreads, only few motes were left to maintain the roadmap (in mote-burning, the motes failed after Δ_{burn}, in mote-non-burning, there were fewer motes with temperature less than Δ_{safe}). However, an interesting observation was that, our network was behaving similarly when motes failed or danger reached them.

Obstacle Detection. In our previous experiments, since we did not want the presence of static obstacles to effect the evaluation of AER, we assumed the only obstacle in the environment was fire. In this experiment, we investigated the behavior of the embedded roadmap when static obstacles exist. For this purpose, we designed an environment without fire but with static obstacles. There are four cases, 2,4,8 and 16 robots trying to reach 2,4,8 and 16 goals respectively. The embedded roadmap is the same for all cases. We would

like to observe how the number of edges in the environment changes as the robots discover edges. In order to avoid additional edges during the goal connection, the robots start on roadmap nodes and try to reach other roadmap nodes. Figure 4(d) shows the change in the number of edges as the robots move. The y-axis represents the number of edges and x-axis is the simulation time. Initially, the embedded roadmap is built. With 16 robots, the number of edges drops sharply and then converges. In contrast, with 2 robots the drop is very slow and the remaining number of edges is very high. In fact, as the number of robots increases, the number of edges reduces with a speed proportional to the number of robots. This shows that during the deployment of embedded roadmap, as the number of robots using it increases, the roadmap's topology will converge to satisfy environmental constraints.
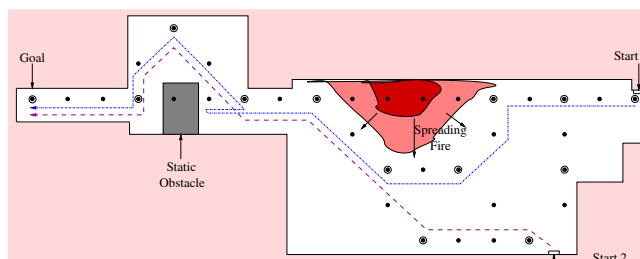

Fig. 5. Environment for the experiment with real hardware. There are two starting locations. Fire starts as the first robot moves (path shown in blue, i.e., dotted line). Through sensor network, first robot avoids the fire. When it discovers the static obstacle, it updates the embedded roadmap edges. The second robot (path shown in red, i.e., dashed line) then moves around the static obstacle. Movie of the experiment is available at [22].

Experiment on the real hardware. In order to evaluate the feasibility of our algorithm on the real hardware, we have designed an experiment where two robots were trying to reach an exit while fire was spreading. Figure 5 shows the experimental environment. The width of the environment is 20 meters, and the height is 6 meters max. There are 34 motes in the environment. Their locations are shown as the black dots in the figure. The milestone motes also have circles around them. Motes are restricted to communicate only with the immediate neighbors in a grid fashion. Two robots are simulated by a single Pioneer 3-DX robot [23]. After reaching the goal, the robot is relocated and simulated the second robot. For the sensor network we have used Tmote sky motes [24]. The embedded roadmap is implemented on Agilla [25], a mobile agent middleware for sensor motes. The mobile agents are software programs that can migrate from one mote to another. In addition to being able to clone themselves, they can also communicate with other agents.

In our implementation there are two agents, roadmap agent and edge agent. Initially, roadmap agent is injected to the network. Through migration, this agent visits every mote on the network. At each mote, it probabilistically decides to become a milestone mote. If it becomes one, then it leaves a copy on the mote and jumps to the nearby motes. If it does not become a milestone, then it communicates with the base to send an edge agent to the mote. Once all the motes of the environment have the agents on them, then the connection phase starts. The connection algorithm in Section V-A is

implemented using inter-agent communication. The maintenance can be done through agent migration. The goal is another agent that is injected to the network by the robot. The aggregation of the best path from the goal (Section V-B) is done by inter-agent communication. Fire is simulated with another agent.

Since $\epsilon-goodness$ of this mote space is very low, we have manually selected the roadmap modes to retain connectivity. The movies of other embedded roadmaps with probabilistic selection is available at [22]. Figure 5 shows the paths of each robot. The robots have no a priori knowledge of the environment and the network directs the robots to the goal. The robot sensors are restricted to detect the obstacles that are only 3 feet away. First robot starts at the upper right corner of the environment. Initially, the sensor network directs it through shortest path, but when the fire starts, the direction of the robot changes. After avoiding fire, the robot follows the shortest path again, as the network directs. However, there is an obstacle on the path about which sensor network has no information. The robot detects obstacle, sends a message to the network to delete that edge. Then, it returns to the last roadmap mote that it visited, and starts following an alternative path, and eventually reaches the goal. The path of this robot is shown with the blue (dotted) line. In the experiment, the robot reached goal in around 180 seconds. Second robot starts from the lower right corner. At this time, fire spreads farther, so instead of following the shortest path on the diagonal, the robot follows a path that is closer to the wall to stay away from fire. When this robot comes to the obstacle that first robot detected, sensor network takes the robot around it. The path of this robot is shown with red (dashed) line. In the experiment, the second robot reached the goal in around 110 seconds. The movie of this experiment is available at [22]. This experiment shows that our algorithm can be implemented on real hardware and can safely navigate the robots in the presence of moving dangers.

VII. CONCLUSION

In this paper, we have presented a new approach to sensor network assisted navigation. At the heart of our approach is an adaptive embedded roadmap that always contains safe paths of the environment. Our experiments show that this approach reduces the workload of the sensor network and improves navigation safety. The embedded roadmap enabled us to use global information at a reduced network usage. Our future work includes using the embedded roadmap to reach mobile goals as well as implementing embedded roadmap on mobile sensors. The mobility of the target and sensors brings challenging problems to be solved, at the same time it creates new opportunities for the network to react against danger for a safer navigation.

Although we used the embedded roadmap mainly for navigation purposes, such a roadmap can have many other uses once it is built. Examples for such uses are using the network for message passing between robots and coordinating the robot movements through embedded roadmap.

REFERENCES

[1] G. Alankus, N. Atay, C. Lu, and B. Bayazit, "Spatiotemporal query strategies for navigation in dynamic sensor network environments,"

in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, August 2005, pp. 3718–3725.

[2] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, August 1996.

[3] R. Bohlin and L. E. Kavraki, "A randomized algorithm for robot path planning based on lazy evaluation," in *Handbook on Randomized Computing*, P. Pardalos, S. Rajasekaran, and J. Rolim, Eds., pp. 221–249. Kluwer Academic Publishers, 2001.

[4] Q. Li, M. De Rosa, and D. Rus, "Distributed algorithms for guiding navigation across a sensor network," in *Proceedings of the 9th annual international conference on Mobile computing and networking*. 2003, pp. 313–325, ACM Press.

[5] K. J. OHara and T. J. Balch, "Distributed path planning for robots in dynamic environments using a pervasive embedded network," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent System, AAMAS 2004*, July 2004, pp. 1538–1539.

[6] S. Bhattacharya, N. Atay, G. Alankus, C. Lu, B. Bayazit, and G.-C. Roman, "Roadmap query for sensor network assisted navigation in dynamic environments," in *International Conference on Distributed Computing in Sensor Systems (DCOSS'06)*, June 2006.

[7] M. A. Batalin, G. S. Sukhatme, and M. Hattig, "Mobile robot navigation using a sensor network," in *IEEE International Conference on Robotics and Automation*, New Orleans, Louisiana, Apr 2004, pp. 636–642.

[8] M. A. Batalin and G. S. Sukhatme, "Coverage, exploration and deployment by a mobile robot and communication network," *Telecommunication Systems*, vol. 26, no. 2-4, pp. 181–196, August 2004.

[9] M. A. Batalin and G. S. Sukhatme, "Sensor network-based multi-robot task allocation," in *Proceedings of IEEE/RSJ International Conference On Intelligent Robots and Systems*, October 2003, vol. 2, pp. 1939–1944.

[10] M. A. Batalin and G. S. Sukhatme, "Using a sensor network for distributed multi-robot task allocation," in *Proceedings of IEEE/RSJ International Conference On Intelligent Robots and Systems*, April 2004, vol. 1, pp. 158–164.

[11] M. A. Batalin and G. S. Sukhatme, "The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment," in *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, April 2005, pp. 3478–3485.

[12] A. Verma, H. Sawant, and J. Tan, "Selection and navigation of mobile sensor nodes using a sensor network," in *Third IEEE International Conference on Pervasive Computing and Communications (PerCom'05)*, 2005, pp. 41–50.

[13] P. Corke, R. Peterson, and D. Rus, "Coordinating aerial robots and sensor networks for localization and navigation," in *Proceedings of the Seventh International Symposium on Distributed Autonomous Robotic Systems*. June 2004, Distributed Autonomous Robotic Systems 6, Springer-Verlag.

[14] C. Buragohain, D. Agrawal, and S. Suri, "Distributed navigation algorithms for sensor networks," in *Proceedings of IEEE INFOCOM*, April 2006.

[15] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system," in *Proc. 6th ACM MOBICOM*, Boston, MA, August 2000.

[16] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher, "Range-free localization schemes for large scale sensor networks," in *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, New York, NY, USA, 2003, pp. 81–95, ACM Press.

[17] K. Lorincz and M. Welsh, "Motetrack: A robust, decentralized approach to rf-based location tracking," in *Proceedings of the International Workshop on Location and Context-Awareness (LoCA 2005) at Pervasive 2005*, May 2005, pp. 63–82.

[18] J. C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[19] B. Karp and H. T. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *MobiCom'00*, 2000, pp. 243–254.

[20] "NS-2 Network Simulator," http://www.isi.edu/nsnam/ns/.

[21] Kevin McGrattan et. al., *Fire dynamics simulator (version 4) technical reference guide*, National Institute of Standards and Technology, 2004.

[22] "Adaptive Embedded Roadmaps For Sensor Networks, companion site for ICRA'07," http://www.cse.wustl.edu/~bayazit/sn.

[23] "Mobilerobots inc.," http://www.mobilerobots.com/.

[24] "Moteiv corporation," http://www.moteiv.com/.

[25] C.-L. Fok, G.-C. Roman, and C. Lu, "Rapid development and flexible deployment of adaptive wireless sensor network applications," in *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'05)*. June 2005, pp. 653–662, IEEE.