Computing Science Group

Practical Issues in Deploying Mobile Agents to Explore a Sensor-Instrumented Environment

Ettore Ferranti and Niki Trigoni

CS-RR-09-02

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD

Practical Issues in Deploying Mobile Agents to Explore a Sensor-Instrumented Environment

Ettore Ferranti and Niki Trigoni
Computing Laboratory,
University of Oxford, Oxford, UK
{Ettore.Ferranti | Niki.Trigoni}@comlab.ox.ac.uk

***Abstract*—When an emergency occurs within a building, it is safer to send autonomous mobile agents instead of human responders, to explore the area and identify hazards and victims. Existing exploration algorithms [1], [2] allow mobile agents to make distributed navigation decisions by communicating with nearby fixed sensors embedded in the environment. These algorithms are very efficient in terms of exploration time, but they have only been evaluated in simulation environments, where idealized assumptions were made regarding the ability of agents to localize sensors and move accurately towards them. The objective of this work is to investigate practical issues of building a real testbed of mobile agents and fixed sensors, and implementing exploration algorithms in such a testbed.**

In particular, we describe our experiences from building a real system consisting of a Surveyor SRV-1 robot and Tmote Sky sensors running the Contiki OS [3]. We select two existing exploration algorithms, Ants [1] and Brick&Mortar [2], and discuss challenges in trying to implement them in our testbed. To address these challenges, we propose practical solutions that allow a mobile agent to: (i) identify and localize fixed sensors deployed in its vicinity; and (ii) accurately move towards a carefully selected fixed sensor. Using our real network deployment, we derive realistic models of localization and odometry errors. We then insert these error models into a realistic simulation environment, in order to extensively compare Ants and Brick&Mortar, and measure their performance degradation as a result of introducing realistic errors.

I. Introduction

When an emergency occurs within a building, the area is typically off-limits for anyone not wearing respiratory equipment, garments or barrier materials to protect themselves from exposure to biological, chemical, and radioactive hazards. In such adverse conditions, it is safer to deploy a group of autonomous robots, hereafter referred to as *mobile agents* to explore the area as fast as possible in search for hazards and victims. Mobile agents must overcome three important limitations during the exploration process: 1) lack of location information in indoor environments where GPS is inaccurate; 2) lack of direct connectivity with each other via long-range wireless links[1]; and 3) lack of map information after emergencies that often change the building plan.

In order to address these challenges, recent work has proposed instrumenting the emergency area with tiny fixed sensors [1], [2]. By carefully reading and updating the state of the instrumented environment, mobile agents are able to explore the environment without map or location information. More importantly, they are able to communicate with each other indirectly by leaving useful information on local sensors to be picked up by other agents that happen to roam through the same part of the sensor network.

For simplicity, consider an area instrumented with fixed sensors lying in a grid topology. Wall cells, i.e. cells that are occupied by some obstacle, are the only ones without fixed sensors. We assume that a mobile agent is able to communicate with the fixed sensor on the current cell, as well as with at most eight fixed sensors in the surrounding cells. We also assume that the mobile agent has on-board sensing devices that allow it to detect hazards and victims within the current cell[2].

Exploration algorithms that use the above model [1], [2] typically follow four steps: 1) Sensor localisation: the mobile agent identifies the fixed sensors lying in the current and eight surrounding cells, and measures their relative positions with respect to itself; 2) Sensor querying: the mobile agent queries the state of the fixed sensors that were detected and localized in the previous step; 3) Sensor updating: the mobile agent updates the state of the fixed sensor in the current cell, taking into account the states of the fixed sensors in the eight surrounding cells. 4) Navigation: the mobile agent carefully selects one of the surrounding fixed sensors and navigates towards it.

Note that exploration decisions are made in a completely distributed manner, by simply relying on the local state of the instrumented environment. The weakness of existing exploration algorithms is that they have only focused on the sensor tasking and sensor marking steps, and have largely ignored the practical issues pertaining to sensor localization and navigation. They make unrealistic assumptions about the ability of an agent to accurately localize sensors in its vicinity, and move towards a selected sensor without odometry errors. Although these assumptions are convenient for simulation purposes, they are inadequate when it comes to evaluating exploration algorithms in real testbeds.

The objective of this paper is to investigate practical issues arising from applying distributed exploration algorithms in a real environment. Our contributions are as follows:

- We provide a detailed description of our testbed, includ-

[1]Radio wave propagation inside buildings with smooth metal surfaces can be so bad that radio "dead spot" can exist where the signal is virtually non-existent.

[2]The size of a cell is thus determined by the sensing range and by the communication range of the mobile agent. Assuming that sensors are accurately positioned in the middle of cells, and the agent could be located anywhere in its current cell, the agent's sensing range must be at least the size of the cell diagonal and its communication range at least 1.5 times the size of the cell diagonal.

ing the hardware and software architecture of mobile agents and fixed sensors.

- We propose practical mechanisms for the localization and navigation steps, and test them in a real testbed to derive realistic models of localization and navigation errors.
- We discuss pathological cases that illustrate how existing exploration algorithms, namely Ants and Brick&Mortar, are affected by localization and navigation errors.
- We insert our error models into a simulation environment, and assess how the performance of Ants and Brick&Mortar degrades as a result of introducing realistic errors.

The remainder of this paper is organized as follows: Section II provides an overview of related work on distributed exploration algorithms and technologies for sensor localization. Section III provides a detailed description of the hardware and software architecture of our testbed. Section IV proposes practical mechanisms to localize sensors and navigate towards them. Section V-A evaluates the performance of these mechanisms in a real testbed and provides illustrative examples that show the impact of localization errors on the behavior of exploration algorithms. Section V-B quantitatively evaluates exploration algorithms in a simulation environment with and without realistic localization and navigation errors.

II. Background

A. Existing exploration algorithms

There is a plethora of distributed exploration algorithms in which agents query a sensor-instrumented environment and make navigation decisions based on the state of local sensors. The purpose of this section is not to provide a complete review of existing algorithms (we refer the interested reader to [2]), but to provide a brief description of two representative examples, Ants [4], [1] and Brick&Mortar [2], the first one being a very simple and robust algorithm, and the second being very efficient in terms of exploration time. Later in the paper, we will examine issues pertaining to the implementation of these algorithms in a real testbed.

Agents in both algorithms operate by reading and updating the state of fixed sensors located in the current or surrounding cells. A cell can be in one of the following states:

- ***Wall***: The cell cannot be traversed by an agent because it is blocked by an obstacle.
- ***Unexplored***: No agent has been in the cell yet.
- ***Explored***: The cell has been traversed at least once, but the agents might need to go through it again in order to reach other *unexplored* cells.
- ***Visited***: The agents have already explored the cell, and they do not need to go through it again to reach other cells.

Both Ants and Brick&Mortar are guaranteed to achieve the *Exploration Objective*, i.e. agents eventually traverse all cells in the area at least once. This means that no cell is left in the *unexplored* state. When this objective is achieved, cells can be in any of the *explored*, *visited* or *wall* states. In addition, Brick&Mortar is shown to achieve the *termination objective*, which means that the agents not only traverse the entire area, but are also able to determine when the task is completed. When this happens, all cells in the area are either *walls* or *visited*. No cell is left in the *unexplored* or *explored* state. By definition, the *Exploration Objective* is always achieved earlier (or at the same time as) than the *Termination Objective*. Both objectives should be achieved in the minimum amount of time, because in an emergency scenario as the one we are considering, speed is essential. The faster the *Exploration Objective* is achieved, the faster victims and hazards are identified. The quicker the *Termination Objective* is achieved, the earlier human responders can enter the area with the certainty that there are no hidden hazards.

Ants: we first discuss the Ants algorithm proposed by Svennebring and Koenig in [4], [1]. This is a distributed algorithm that simulates a colony of ants leaving pheromone traces as they move in their environment. Initially, all cells are marked with value 0 to denote that they are *unexplored*. At each step, an agent reads the values of the four adjacent cells in the North, East, South and West directions and chooses to step onto the least traversed cell (the one with the minimum value, Fig.1A). Each time it steps into a cell, it updates its value, for example by incrementing its value by one (Fig.1B). The authors provide a proof that the agents will eventually cover the entire terrain (provided that it is not disconnected by *wall* cells), and thus that the *Exploration Objective* is always achieved. The first advantage of the algorithm is its simplicity: agents do not require memory or radio communication, but only one-cell lookahead. Secondly, there is no map stored inside the agents: if one of them is relocated (accidentally or on purpose) it will not even realise it and it will continue to do its work as if nothing happened. At the storage device of each cell, we only need to store an integer counting the number of times that agents have visited the cell. The main limitation of the Ants algorithm is that the *visited* state is not used during the process of marking cells. Therefore, the *termination objective* is never achieved, and the agents continue the exploration phase until they run out of energy.

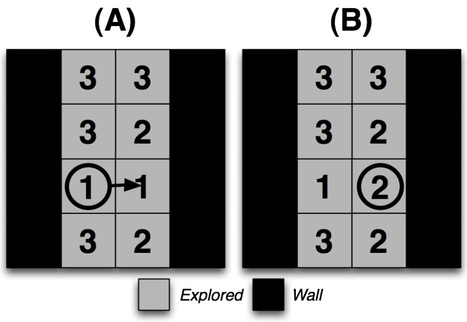

Fig. 1. An agent running the Ants algorithm always chooses to go toward the adjacent cell that was explored the least amount of times (A). Each time the agent steps into a cell, it updates its current value, for example incrementing it by one (B).

Brick&Mortar: the second algorithm [2] we examine is designed to address the weaknesses of Ants algorithm described above. Unlike Ants however, agents using Brick&Mortar know when the exploration task is completed and they do not spend much time revisiting the same cells. The driving idea is that of thickening the existing walls by progressively marking the cells that surround them as *visited*. Once again, *visited* cells are equivalent to *wall* cells in that they can no longer be accessed. In the description of the algorithm, we refer to *wall* and *visited* cells as inaccessible cells, and to *unexplored* or *explored* cells as accessible cells. The algorithm aims to progressively thicken the blocks of inaccessible cells, whilst always keeping accessible cells connected.

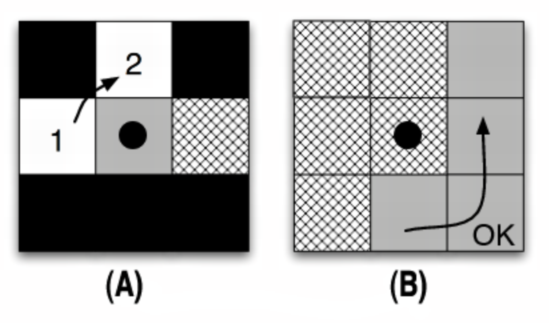

Fig. 2. The cell in the center of (A) cannot be marked as *visited* because it would block the way between cells 1 and 2, while the cell in the center of (B) is not blocking the way between two accessible cells and is thus marked as *visited*.

In the marking step the agent updates the state of the current cell, choosing between the *explored* and *visited* states. The cell is marked as *visited* only if it is not blocking the way between two accessible cells located in the North, East, South or West directions. Two cases of applying this local rule are illustrated in Figure 2: one where the current cell is marked as *explored* (map a), and one where it is marked as *visited* (map b). In addition to this local rule, Brick&Mortar uses a loop closure mechanism to ensure that the *Exploration* and *Termination Objectives* are achieved even in the presence of obstacles (clusters of wall cells in the middle of the exploration area). Details of this mechanism are beyond the scope of this paper, and are omitted for space reasons.

Both Ants and Brick&Mortar abstract away the problems of detecting fixed sensors in the vicinity of an agent, localizing them (i.e. detecting in which cells they lie with respect to the cell where the agent is located), and identifying them one by one. However, in practice these tasks are far from trivial, and failures to accurately detect, localize and identify sensors could cause a significant degradation of the performance of the two algorithms. Agents that do not accurately read local sensor state are likely to make suboptimal navigation decisions, slowing down the exploration process, and in the case of Brick&Mortar even trapping and immobilizing agents in inaccessible areas before the entire area has been explored. In the next subsection, we overview several techniques of localizing sensors with respect to an agent, using radio signal strength, infrared, ultrasound and camera technologies.

B. Technologies for sensor localization

Radio Signals: Radio signal strength is a not reliable way of identifying the robot relative position with respect to tags deployed in an environment. In fact, it heavily depends on factors like the relative orientation of the deployed motes, their height from the floor, the material of the floor, and the obstacles in the environment (the line of sight). Moreover, in the literature (e.g. [5], [6], [7], [8], [9], [10], and [11]) it is widely accepted that radio propagation is (i) non-isotropic (i.e. the received signal, at a given distance from the sender, is not the same in all directions), it has (ii) non-monotonic distance decay (i.e. lower distance does not mean better link quality), and (iii) the communication is based on asymmetrical links (i.e. if A hears B, it cannot be assumed that B hears A).

However, several pervious studies used radio frequency to cope with localisation issues. For example, to solve the problem of determining if the robot is in the neighbourhood of a sensor, Batalin et al. [12] create an algorithm called Adaptive Delta Percent, which takes into account the signal strength of the messages received from the various tags while the robot is moving in order to guide it toward one of them. A strong limitation of this approach is that the authors consider an experiment to be successful if the robot is able to reach a tag in the environment within a distance of 3m, an accuracy which is unreasonable for our scenario. In their work, Bhattacharya et al. in [13] assume that pre-deployed nodes are location-aware and both robot and nodes have a limited radio communication range, so that a robot can establish both if it is close to a certain node and if it is able to communicate with it. The authors do not provide results regarding the effectiveness of this method, and the approach is not suitable if communication ranges from different nodes overlap.

Infrared Signals: Several systems have been created to define mobile robot localisation in indoor environments. Some of them use ultrasonic and infrared technologies simultaneously [14], others radio frequency (RF) and infrared together [15], and some just infrared techniques [16]. However, infrared signals are not completely suitable for our scenario because they have a particularly limited transmission range (i.e. ∼20-30cm), thus the robot risks not being able to identify the deployed tag if the dimension of the cell is bigger than the allowed range. Moreover, interference from the IR component of other light sources could compromise the localisation process [17].

Ultrasonic Signals: Ultrasonic sensors [18] alone could be used to avoid obstacles, but not to identify specific tags in the environment due to the poor resolution of their readings. Therefore, we argue that IR or sonar could not represent suitable technologies neither to localise the agent with respect to the tags surrounding it (avoiding *localisation errors*), or to guide the agent during the navigation process, thus correcting *movement errors*.

Cameras and image processing: Since the previous approaches are not suitable for our scenario, we decided to explore agent localisation using camera technologies. Several approaches investigated this area adopting feature cluster recognition [19], [20]. In particular, some of them use image processing techniques to recognise landmarks in the environment [21], [22]. However, most of the approaches present very sophisticated techniques to recognise specific features and landmarks within the environment to autonomously guide the agent in the exploration (e.g. in SLAM). On the contrary, we plan to use cameras in a simpler way.

In particular, cameras allow us not only to extract the infrared component of a LED light source, but also to identify the position of the LED within the taken picture. Moreover, by progressively processing all images that the agent takes of the environment while moving and extracting their differences, it is possible to keep the LED light in the centre of the image so that *odometry errors* can be corrected if the relative position of the LED light within the image changes. In this way, the agent can be driven by more accurate movements and it is able to correct its position autonomously. The camera range is limited only by its resolution, thus we do not have the IR short-range constraints. In fact, the chosen robot (described in Section III-A) has not only a high-resolution camera, but also a high computational power allowing it both to precisely identify the LED light within the image and to progressively process multiple images directly on the robot platform.

III. Our System

In the previous section, we provided a high level description of two exploration algorithms, Ants and Brick&Mortar, which have been extensively tested in simulation environments, but not in a testbed of real robots and fixed sensors. Their main shortcoming is that they ignore the issues of how a robot localizes surrounding sensors, and moves accurately towards a selected sensor. We overviewed alternative approaches for sensor localization, and proposed cameras as the preferred technology that enables agents to identify sensors and localize them accurately.

We are now in a position to start describing our experiences from building a real system of agents and fixed sensors, and implementing exploration algorithms in the presence of realistic localization and navigation errors. In this section, we provide an overview of the system architecture, including the hardware characteristics and the software modules of both the agent and sensor platforms. As we will discuss in Section IV, the hardware and software capabilities of our system affect the design of practical localization and navigation techniques.

A. Hardware architecture

Our system consists of three different platforms: 1) mobile agent: Surveyor SRV-1 robot connected with a Tmote Sky mote; 2) fixed sensor: Tmote Sky mote with external bright LED and 3) gateway: laptop connected with a Tmote Sky mote (via its USB interface) used primarily for visualisation of experiment results. Figure 3 shows the used robot together with the Tmote Sky sensors.

The Surveyor SRV-1 robot has a 1000mips 500MHz processor, 32 MB SDRAM and 4 MB flash. It has an omnivision (OV9655) 1.3 megapixel camera with variable resolution from 160x128 to 1280x1024. It is also equipped with a Lantronix Matchport 802.11b/g WiFi radio which it uses to communicate results to the gateway for visualisation. It has tank-style treads with differential drive via four precision DC gearmotors (100:1 gear reduction) and its speed ranges from 20cm to 40cm per second. Its size is 120mm long, 100mm wide and 80mm tall, and it weighs 350g. It has open source firmware written in C programming language and provided by Surveyor. The firmware has been custom edited to suit the needs of the algorithms, and compiled using the GNU Toolchain for the Blackfin processor. The robot is connected with a Tmote Sky mote (described below) so that it can communicate with the fixed sensors in its vicinity, using the IEEE 802.15.4 standard.

Fixed sensors are Tmote Sky motes with 8MHz TI MSP430 microcontrollers, 10KB RAM, 48 KB flash, and 1MB external flash memory. They are equipped with 2.4 GHz IEEE 802.15.4 compliant radios that feature data rates of up to 250kbps. They are approximately 70mm long, 30mm wide and 20mm tall, and they weigh 23g excluding batteries. In order to increase the visibility of fixed sensors by the mobile agents, we attached to each mote a bright red LED (Maplin 5mm, 2.5V, 25mA) via the GPIO line. The motes are able to switch the attached LED on or off by a small driver which operates directly on pin 23 of the MSP430 microcontroller.

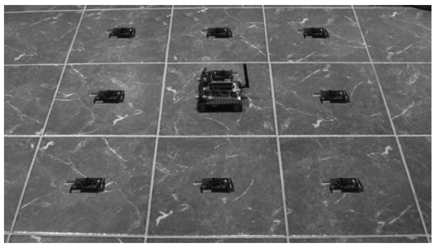
Fig. 3. Hardware architecture of our system: Mobile agent (Surveyor SRV-1 robot with Tmote Sky mote on top) tasked to explore fixed sensors (Tmote Sky motes) deployed on the floor.

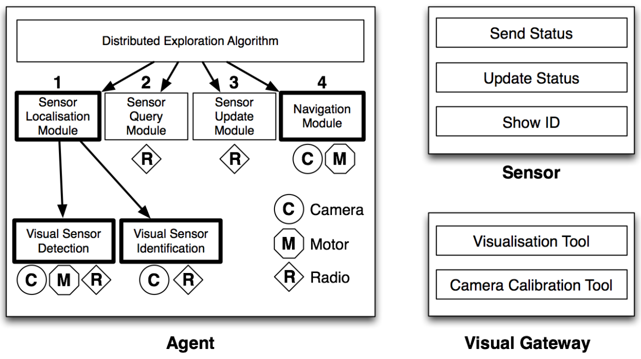

Fig. 4. Software architecture of our system.

B. Software architecture

A graphical representation of the software architecture of the system can be found in Figure 4. The software running on each mobile agent consists of four distinct modules: 1) the sensor localization module; 2) the sensor query module; 3) the sensor update module; and 4) the navigation module. The role of the sensor localization module is to localize the fixed sensors currently surrounding the mobile agent, which consists of the subtasks of detecting these sensors by camera, finding their relative positions with respect to the agent, and identifying the unique ID associated with each sensor. This module utilizes the camera for detection and identification purposes, the motor to rotate the agent around itself to take multiple pictures of its vicinity and the radio to communicate with the fixed sensors within communication range. Details of the sensor localization module will be provided in Section IV. The sensor query module is dedicated to the retrieval of information from nearby sensors that were localized in the previous step. Query messages are sent to fixed sensors, which, in response, send a report of their state. The type of state information reported depends on the exploration algorithm; for example, in Ants, sensors report how many times they have been traversed by agents, whereas in Brick&Mortar they report if they are in the *unexplored*, *explored* or *visited* state. The sensor update module processes all the information received from the sensors in the previous step, and updates the current (closest) sensor accordingly. It also determines which sensor to approach next. Finally, the navigation module ensures that the mobile agent can accurately approach the carefully selected fixed sensor. In Figure 4, it is possible to see which module is using which capability of the robot, namely camera, motor and radio, in order to accomplish its assigned task.

Since the robot in fact lacks any kind of visual output, the debugging of the system would be almost impossible without monitoring its status via network messages sent from the mobile agent to the visual gateway. It must be noted however that the image processing and all the computationally challenging tasks are executed by the CPU of the mobile agent, while the external computer is only relied upon to visualise the progress of exploration. This is very important because we wanted to test whether it is possible to execute exploration algorithms on a small and inexpensive platform such as the robot we are using, without any additional help from an external processing unit or central server. The gateway also hosts an additional console written in Python, which is used to remotely control the mobile agent, in order to test all its functions and calibrate the camera with the color of the external LED used by the fixed sensors. The user is able to move the robot to the desired position, facing a LED which is currently switched on, and take a picture which is then visualised on the screen by the console. After that, the user can select a small sample of the depicted LED by dragging a rectangle on the screen, and that sample will be used by the robot as future reference of the LED color.

IV. Techniques

Previous work [2] focused on the sensor query and sensor update modules, and made convenient assumptions about the ability of agents to accurately localize local sensors, and to navigate toward one of them without incurring odometry errors. In this work, on the contrary, the main effort is focused on the relaxation of these assumptions, and in particular on building a working prototype of the entire system described above, in order to prove that the exploration algorithms described in Section II-A can be effectively ported from theory to practice, and implemented in a real testbed.

In this section, we provide a detailed description of the steps involved in localizing sensors by a mobile agent, and moving towards one of them. Consider the example of a testbed layout, in which the mobile agent is placed at the centre of a grid of sensors, as depicted in Figure 5. In what follows, we discuss practical techniques for 1) detecting sensor LEDs in a picture taken by a mobile agent; 2) discovering the identifier of a detected sensor; 3) establishing the relative position of the sensor with respect to the mobile agent; and 4) moving towards a selected sensor within view. All of these techniques are performed by the mobile agent, and have been added to its firmware.

Detecting a sensor LED: The color space used in the robot to process images is YUV, which separates the luminance component (Y) of a pixel from its colors components (U and V). For this reason, by using this color space it is possible to distinguish a color that is emitted from a light source (like a red LED) from that of a less bright object (like a red chair) by using the Y component. In order to correctly detect LEDS in a captured frame, we first calibrate the mobile agent, with the aid of the gateway. The mobile agent takes a picture of the surrounding area, and sends it to the gateway, and the user draws mini-rectangles around the LEDs of any sensors in the frame. This process is used to sample the YUV color range of LEDs mounted on the sensors in order to detect them correctly during the exploration. The calibration script running on the gateway scans every pixel within the given rectangle to find the minimum and maximum value of each one of the Y,U and V components of the color space. When sampling the picture of an LED, the range of the Y component is expected to be high and narrow, because an LED is always very bright, while the U and V components may change according to the light conditions.

Once the calibration phase is completed, the mobile agent is ready to detect LEDs in a given frame as follows. In order to identify the sensors within the current field of view, the mobile agent sends a broadcast message to ask the motes to switch their LEDs on. When the message has been sent, a picture is taken, to identify any blobs within a given YUV color range in the frame and return the coordinates of their barycenter. Two different thresholds are given as parameters, which are derived from the calibration phase: the first one is the minimum dimension of a blob (in pixels), while the other is the maximum gap (in pixels) between two blobs for them to be considered the same blob. The first threshold is very effective in fixing a maximum range for the vision of the robot. Since the dimension of LEDs is in fact the same, the size of a blob in the frame corresponds roughly to a sensor which is at a given distance from the camera (and thus from the mobile agent). Even if this information is not precise enough to compute an accurate distance of the mote from the robot, because of the different orientations in which the LED can be at a given moment, it can be used to discard sensors which are too far from the mobile agent, and thus not in an adjacent cell. The function scans every pixel of the frame in horizontal lines, keeping track of the ones within the given YUV color range. For each line, pixels are clustered in different sets according to their relative distance i.e. if the pixel currently examined is further than the specified threshold from the previous set, than a new set is created. Each time a new line of the frame is processed, the sets in it are matched against the sets in the previous lines (their number depending again on the specified distance threshold) to see if they overlap. The overlapping sets are clustered together in blobs, while the ones that do not overlap are used to form new blobs. Finally, a list with only the blobs that contain more than the given amount of pixels are returned.

Discovering the IDs of detected sensors: In order to identify the sensors within the current field of view, the mobile agent broadcasts a message to all sensors in communication range asking them to switch their LEDs on and off according to their identifier. That is, if the identifier of a sensor is 101 in binary base, the sensor will switch its LED on in the first time interval, off in the second, and on again in the third. The mobile agent takes consecutive pictures at each time interval and processes them in order to derive the identifiers of all sensors detected in the previous step. A list of every sensor with its corresponding ID and coordinates within the frame is finally returned.

Locating sensors with respect to mobile agent: The goal of this step is to establish the relative position of detected and identified sensors with respect to the mobile agent. The mobile agent starts turning the camera around in small steps, so as to cover the whole area surrounding its current position. In each step, it detects and identifies all sensors currently in its field of view, and stores them, together with their coordinates within the frame, and the number of steps which have been performed, starting from the point when the first sensor was detected. When the first identified sensor is detected once again, the mobile agent assumes that the entire local area has been scanned, and stops. At this point, the coordinates of the motes are converted from cartesian frame coordinates to polar, with the axis centered on the agent, and a map is built with the positions of the sensors with respect to the agent. The relative angle is computed by considering how many steps the agent has performed to cover the area, the number of steps after which a given sensor was identified, and the horizontal position of the mote within the considered frame. The distance of the sensor from the robot is instead computed using the vertical coordinate of the sensor in the frame. A fitting equation was computed for this purpose, using a measurement tape in front of the camera and matching real distances in the tape with vertical coordinates in the frame (in pixels) and obtaining $estDist = \frac{y^2}{420.0} - 0.931y + 65.0$.

Moving toward a selected sensor within view: Once all sensors in close proximity of an agent have been detected, identified and localized, the exploration algorithm queries their state, updates the state of the closest one, and carefully selects which sensor to approach in the next step. The exact process of sensor selection depends on the exploration algorithm and is outside the scope of this paper. Instead, we are interested in the practical issue of moving the agent from its current position to its new position, which must be as close as possible to the selected sensor. To do so, the agent must first face the selected sensor, by performing a number of turning steps that depends on the polar coordinates of the selected sensor. To ensure that it faces the correct sensor, it tells all sensors to switch off their LEDs except for the selected sensor which has its LED on. As soon as the mobile agent detects the sensor's LED in its frame, it makes minor corrections to its orientation, so that the x coordinate of the LED is at the center of the captured frame. Once it is facing the sensor, the agent moves toward it, regularly taking pictures to adjust its trajectory according to the current position of the blob in the current frame. Once the sensor is so close that the LED cannot be seen in the frame any more, the agent stops and declares its mission accomplished.

V. EXPERIMENTS

We are now in a position to test the feasibility of our proposed techniques in a real setting in order to derive realistic localization and navigation errors. We will then feed the derived error models into a simulation environment, to test exploration algorithms extensively in realistic conditions.

A. *Real Experiments*

In this section, we describe our results from implementing Ants (together with our localization and navigation techniques) on a testbed of real agents and real sensors. The sensors are deployed on the ground in a 5x5 grid, at the centers of square cells of size 48 cm. The agent is placed in the middle of the cell in the center of the grid, as shown in Figure 5. The purpose of this experiment is four-fold: 1) to measure errors in localizing detected sensors; 2) to measure the percentage of sensors that are not detected at all; 3) to measure navigation errors, i.e. inaccuracies in approaching a selected target sensor; and 4) to count the times taken by an agent to localize surrounding sensors and navigate towards one of them.

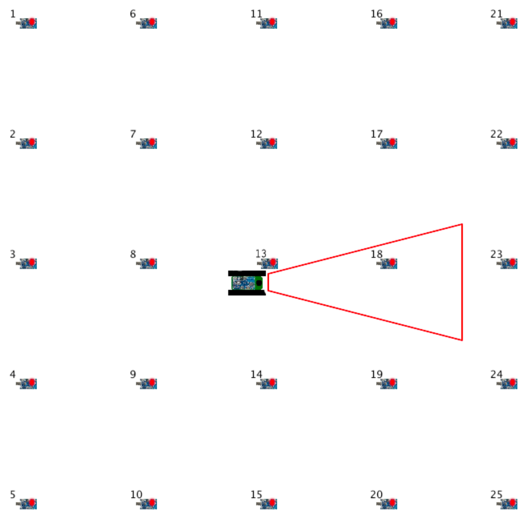

Fig. 5. Example of a testbed layout, with the mobile agent placed at the centre of a grid of sensors.

Figure 6 shows estimated (circles) and real (squares) positions of sensors surrounding a given agent. In this case one can notice how, even if the sensors were not always correctly localised, the errors are always small enough, so that a sensor can not be thought to be in another cell from its own. Figure 7 is the projection of an error model built from these data, with

10000 simulated sensor positions. These results imply that the localization errors have a minimal impact on the performance of exploration algorithms, since agents make decisions based on the relative position of the cells where sensors are located, and not based on their actual relative coordinates.

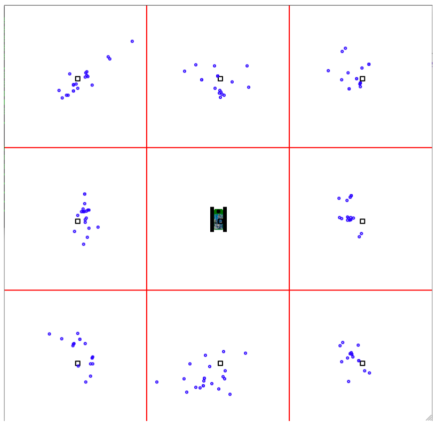
Fig. 6. Localisation of sensors around the agent. The positions of sensors as estimated by the agent are represented as circles, while the real sensor positions are represented as squares.

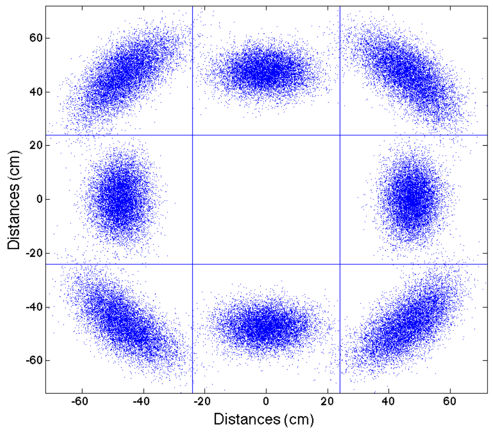

Fig. 7. Distribution of the localisation error model, as generated by a projection of 10000 sensor positions.

Unlike localization errors, errors in actually detecting the presence or absence of sensor nodes in the local neighboorhood can have a much more detrimental effect. Real experiments showed that the percentage of undetected sensors, due to adverse light conditions, is not negligible and amounts to 5.56% of all sensors. Let us consider the impact of undetected sensors on the functionality of Ants and Brick&Mortar. Recall that in Ants, an agent reads the state of surrounding sensors and moves to the least explored one. If an agent misses a sensor, it simply assumes that the cell has not been equipped with a sensor before and it has not been explored yet. Therefore it moves into the cell and places a new sensor on the floor and marks it with the state 1 (meaning that it has been explored once). Hence, in the worst case, a cell that has already been traversed by an agent is traversed yet one more time. However, since agents running Ants never stop the exploration process, they are eventually guaranteed to explore the entire area, which means that Ants is robust to undetected sensors.

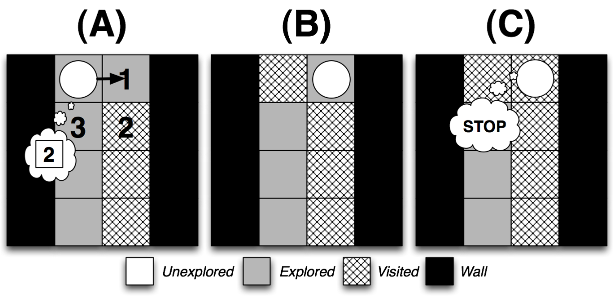

Fig. 8. A pathological case in which the problem of undetected sensors adversely impacts the performance of Brick&Mortar.

On the other hand, undetected sensors can significantly damage the performance of Brick&Mortar, by trapping agents within *visited* cells, and not letting them participate in the exploration process. Figure 8 shows such a pathological case, while an agent is exploring a small area. In Stage A, the agent does not detect the sensor on the bottom-right cell (cell 2), and assumes that there is free path between cell 1 and 3. For this reason, it marks the current cell as *visited*, in fact disconnecting two *explored* areas while doing so. In Stage B, the cell occupied by the agent does not block the path between any two adjacent cells in the North, East, South or West directions, because all such adjacent cells are already *visited* (or *walls*). Hence, in Stage C, the agent marks the current cell as *visited* and erroneously assumes that the exploration task is finished. Since the adjacent cell on the east is *wall*, and those on the West and South are *visited*, the agent cannot step onto any of these inaccessible cells, and is effectively trapped. After a while, all agents can potentially be trapped in this manner long before the exploration process is completed. Hence, undetected sensors can have a detrimental effect not only in the efficiency, but also in the ability of Brick&Mortar agents to cover the entire area. An empirical evaluation of the performance degradation of Brick&Mortar as a result of undetected sensors is provided in Section V-B.

Figure 9 represents the positions (circles) at which the robot stopped when trying to reach the sensors on the ground (squares): in this case the errors are almost unnoticeable, because the robot only stopped when the LEDs on the sensors were below the camera horizon, hence very close. For this reason, we can assume that navigation errors are negligible and non-cumulative during the exploration process.

Our final objective was to measure the time taken by each agent to detect, identify and localize local sensors and move towards one of them. The average number of turns required by the agent to detect all the sensors around it is 18.28 (with st. dev. 2.13). In each turn, if a sensor is detected, the agent takes on average 9.55 sec. (st. dev. 0.34) to localize it, and, if not, 1.81 sec. (st. dev. 0.20) to move on. We also observed that it takes an agent on average 55.42 sec. (st. dev. 3.62) to navigate from its current position to the next selected sensor, because it needs to stop several times and correct its orientation to be

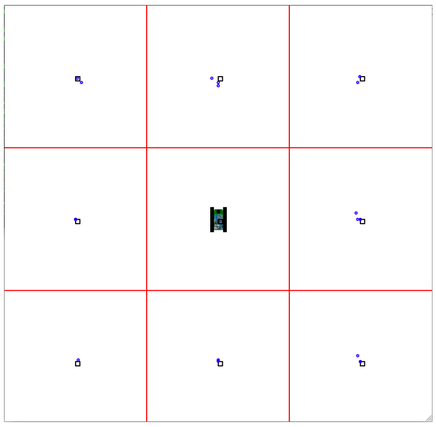

Fig. 9. Points of arrival of the robot when trying to reach the sensors. The points at which the robot arrived are circles, while the real sensors positions are squares.

able to approach the target sensor accurately. These times are of course very high and show that the system is not ready for a deployment in a real emergency scenario. We plan to address this issue in future work.

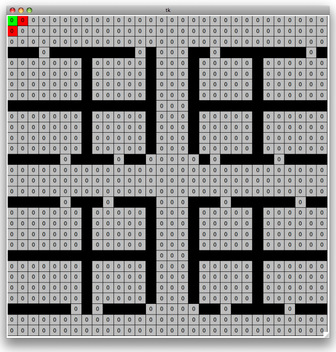

Fig. 10. Example of an office-like scenario with 4x4 rooms as used in the simulations.

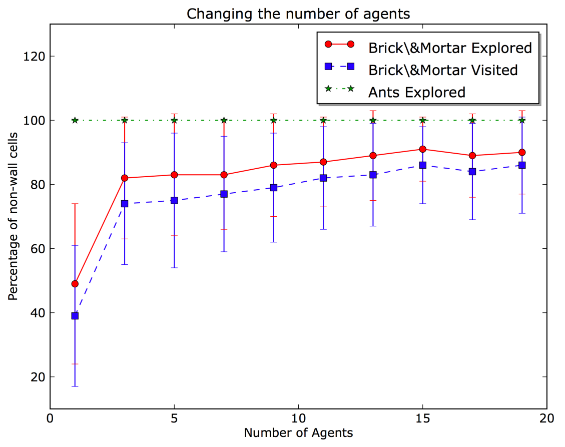

Fig. 11. Effect of changing the number of agents.

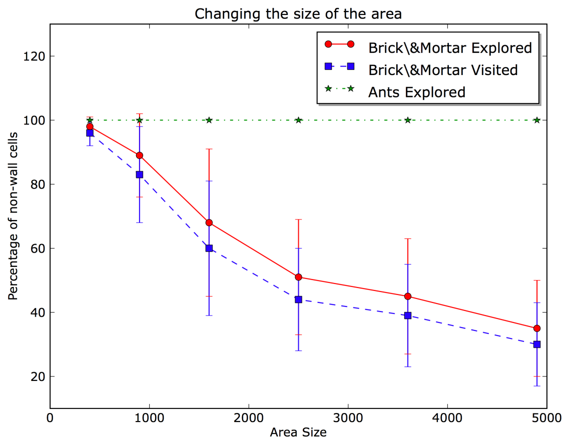

Fig. 12. Effect of changing the area size.

B. *Simulations with realistic errors*

Having derived realistic localization, detection and navigation errors, we are now in a position to insert them in a simulation environment and test Ants and Brick&Mortar in realistic error conditions. Previous work [2] reported that in idealized conditions, Brick&Mortar significantly outperforms Ants in terms of exploration time, i.e. the time taken by agents to explore all cells at least once. They also reported that agents running Ants have no way of knowing when the entire area is covered, whereas agents running Brick&Mortar detect termination when they are surrounded only by *visited* or *wall* cells. In this section, we show that although Ants is less efficient than Brick&Mortar in idealized conditions, it is significantly more robust in realistic error conditions.

In order to investigate the extent to which sensor detection errors affect exploration, we ran several simulations varying the number of agents and the size of the area. The simulations were performed on automatically generated environments representing office-like scenarios, like the one in Figure 10, with a default area size of 30x30 cells and 4x4 rooms in them. The positions of doors and walls was changed randomly during the experiments, while the default number of agents was 20. Figure 11 shows that one agent running Brick&Mortar is able to explore only ∼50% of the area (and mark ∼40% of it as *visited*) before stopping. Adding more agents allows us to achieve better results, without however reaching the 100% mark. In the same plot, we observe that agents running Ants always manage to explore 100% of the area's cells. Figure 12 shows the percentage of cells being *explored* or *visited* by the two algorithms, as we vary the number of cells in the area. In relatively small areas, agents running Brick&Mortar manage to terminate before all of them are blocked, but as we increase the area size, the agents have more opportunities to be trapped before the *Exploration* and *Termination Objectives* are achieved. Hence, it becomes obvious that a small percentage of undetected sensors (5.56%) can greatly compromise the performance of Brick&Mortar, whereas it has minor effects on Ants.

VI. Conclusions and Future Work

In this paper, we described our experiences from building a real system consisting of a Surveyor SRV-1 robot and Tmote Sky sensors running the Contiki OS [3]. We selected two existing exploration algorithms - Ants [1] and Brick&Mortar [2], and discussed challenges in trying to implement them in our testbed. To address these challenges, we proposed practical solutions that allow a mobile agent to: (i) detect, identify and localize fixed sensors deployed in its vicinity; and (ii) accurately move towards a carefully selected fixed sensor. By testing these techniques in a real testbed, we derived realistic models of detection, localization and navigation errors, and investigated how they affect the performance of two existing exploration algorithms, namely Ants and Brick&Mortar. We concluded that 1) localization errors, although non negligible, have almost no impact on the performance of the two algorithms; 2) detection errors significantly compromise the performance of Brick&Mortar, whereas they hardly affect the simpler and more robust Ants algorithm; and 3) navigation errors are negligible and can be completely ignored. Hence, although Brick&Mortar was shown to outperform Ants in idealized conditions, its performance is severely compromised in the presence of sensor detection errors.

In the future, we plan to address a number of algorithmic and practical issues that stemmed from this research. The algorithmic issues involve improving the Brick&Mortar algorithm to make it robust to sensor detection errors. One possibility would be to let Brick&Mortar agents adopt the Ants approach whenever they are trapped, and to operate as usual in normal conditions. In this way, we could potentially combine the benefits of Brick&Mortar in terms of exploration time, and the benefits of Ants in terms of robustness. The practical issues involve reducing the time taken by agents to detect, identify and localize local sensors, and move towards a selected sensor. We also intend to study the case in which sensors are not deployed in a perfect grid, but they are placed by agents roaming through the area in an imperfect manner.

Acknowledgments

The authors would like to thank Dimitrios Coussios for his feedback during the deployment of the testbed. Effort sponsored by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-06-1-3003. The U.S. Government is authorized to reproduce and distribute reprints for Government purpose notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

References

[1] J. Svennebring and S. Koenig, "Building Terrain-Covering Ant Robots: A Feasibility Study," *Autonomous Robots*, vol. 16, no. 3, pp. 313–332, May 2004.

[2] E. Ferranti, N. Trigoni, and M. Levene, "Brick&Mortar: An On-Line Multi-Agent Exploration Algorithm," in *ICRA07: Proceedings of the 2007 IEEE International Conference on Robotics and Automation.* IEEE Press, April 2007, pp. 761–767.

[3] A. Dunkels, B. Groñvall, and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the* 1^{st} *annual IEEE International Workshop on Embedded Networked Sensors*, November 2004, pp. 455–462.

[4] S. Koenig and Y. Liu, "Terrain Coverage with Ant Robots: a Simulation Study," in *AGENTS01: Proceedings of the 2001 ACM International Conference on Autonomous Agents.* ACM Press, May 2001, pp. 600–607.

[5] D. Kotz, C. Newport, R. S. Gray, J. Liu, Y. Yuan, and C. Elliott, "Experimental Evaluation of Wireless Simulation Assumptions," Tech. Rep. Technical Report TR2004-507, Dartmouth College Computer Science, June 2004.

[6] D. Aguayo, J. Bricket, S. Biswas, G. Judd, and R. Morris, "Link-level Measurements from an 802.11b Mesh Network," in *SIGCOMM04: Proceedings of the 2004 ACM Conference of the Special Interest Group on Data Communication.* ACM Press, August 2004, pp. 121–132.

[7] J. Park, S. Park, D. Kim, P. Cho, and K. Cho, "Experiments on radio interference between wireless LAN and other radio devices on a 2.4 GHz ISM band," in *VTC03: Proceedings of the 2003 IEEE Semiannual Vehicular Technology Conference.* IEEE Press, April 2003, pp. 1798–1801.

[8] D. Kotz, C. Newport, and E. C., "The mistaken axioms of wireless-network research," Tech. Rep. Technical Report TR2003-467, Dartmouth College Computer Science, July 2003.

[9] J. Zhao and R. Govindan, "Understanding Packet Delivery Performance In Dense Wireless Sensor Networks," in *SenSys03: Proceedings of the 2003 ACM Conference on Embedded Networked Sensor Systems.* ACM Press, November 2003, pp. 1–13.

[10] A. Cerpa, N. Busek, and D. Estrin, "SCALE: A tool for Simple Connectivity Assessment in Lossy Environments," Tech. Rep. Technical Report CENS-21, UCLA, September 2003.

[11] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker, "Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks," Tech. Rep. Technical Report CSD-TR 02-0013, UCLA, February 2002.

[12] M. Batalin, G. Sukhatme, and M. Hattig, "Mobile Robot Navigation using a Sensor Network," in *ICRA04: Proceedings of the 2004 IEEE International Conference on Robotics and Automation.* IEEE Press, April 2004, pp. 636–642.

[13] S. Bhattacharya, N. Atay, G. Alankus, C. Lu, B. Bayazit, and G.-C. Roman, "Roadmap Query for Sensor Network Assisted Navigation in Dynamic Environments," Tech. Rep. Technical Report WUCSE-2005-41, Washington University in St. Louis, Department of Computer Science and Engineering, November 2005.

[14] S. S. Ghidary, T. Tani, T. Takamori, and M. Hattori, "A new Home Robot Positioning System (HRPS) using IR switched multi ultrasonic sensors," in *Proceedings of the IEEE SMC Conference.* IEEE Press, October 1999.

[15] I. Kelly and A. Martinoli, "A scalable, on-board localisation and communication system for indoor multi-robot experiments," *Sensor Review*, vol. 24, no. 2, pp. 167–179, January 2004.

[16] N. Kirchner and T. Furukawa, "Infrared Localisation for Indoor UAVs," in *ICST05: Proceedings of the 2005 International Conference on Sensing Technology*, November 2005, pp. 60–65.

[17] S. Kataoka and K. Atagi, "Preventing IR interference between infrared waves emitted by high-frequency fluorescent lighting systems and infrared remote controls," *IEEE transactions on industry applications*, vol. 33, no. 1, pp. 239–245, January/February 1997.

[18] J. H. Lim and J. J. Leonard, "Mobile Robot Relocation from Echolocation Constraints," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 9, pp. 1035–1041, September 2000.

[19] A. Djekoune and K. Achour, "Visual Guidance Control Based on the Hough Transform," in *Proceedings IEEE Intelligent Vehicles Symposium 2000.* IEEE Press, October 2000, pp. 614–619.

[20] R. O. Castle, D. J. Gawley, G. Klein, and D. W. Murray, "Video-rate recognition and localization for wearable cameras," in *BMVC07: Proceedings of the 2007 British Machine Vision Conference*, September 2007, pp. 1100–1109.

[21] A. J. Davison, Y. G. Cid, and N. Kita, "Real-Time 3D SLAM with Wide-Angle Vision," in *IAV04: Proceedings of the 2004 IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, July 2004.

[22] A. C. Rice, "Dependable Systems for Sentient Computing," *PhD Thesis, Digital Technology Group, Computer Laboratory, University of Cambridge*, 2007.