2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Deployment of Swarms of Micro-Aerial Vehicles: from Theory to Practice

Aveek Purohit, Pei Zhang, Brian M. Sadler, Stefano Carpin

***Abstract*— We study the problem of deploying a high number of low-cost, low-complexity robots inside a known environment with the objective that at least one robotic platform reaches each of N preassigned goal locations. Our study is inspired by *SensorFly*, a micro-aerial vehicle successfully used for mobile sensor network applications. SensorFly nodes feature limited on-board sensors, so one has to rely on simple navigation strategies and increase performance through redundance in the team. We introduce a simple, fully scalable deployment algorithm exploiting the limited capabilities offered by the SensorFly platform, and we explore its performance by feeding the simulation system with parameters extracted from the real SensorFly platform.**

I. Introduction

In this paper we are interested in the following variant of the *deployment* problem. Given an environment with N target locations (e.g., N rooms inside a large building), the goal is to deploy a team of K robots so that eventually at least one robot reaches each of the N locations. After one robot reaches one of the N locations it may stop there or proceed further. The task is successfully completed as soon as each location as been visited by at least one robot. Problems of this type find applications in urban search and rescue, surveillance, intelligence, and related fields. For example, each of the robots might carry a sensor to detect dangerous chemicals and the objective is to collect samples in a set of critical locations. Instances of this problem have frequently been studied under the assumption that N and K are not too different and often K is smaller than N. In this case robots need to coordinate their plans to ensure each location will be reached. In many scenarios, the integrity of the robot team is a prime concern because individual robot failures may hinder the ability to successfully complete the deployment.

In our research we tackle the problem from a different perspective. Our focus is on deployment problems where robots can be considered expendable assets, i.e., we consider robot swarms. This assumption is legitimate when robots are simple devices with limited cost and capabilities. In this case we may assume K is much larger than N and we can tolerate that some robots fail to accomplish their task. Moreover, we not only focus on the ability to successfully complete the mission, but also on the time taken to complete the mission. Therefore, deployment algorithms that will solve the problem with probability 1 "as time tends to infinity" are not of interest. We recently studied a slightly different instance of the deployment problem from a theoretical standpoint [2]. Therein, we modeled the environment as a graph with one vertex per relevant location. We assumed each edge was characterized by a success function S returning the probability that a robot could traverse an edge as a function of the time spent trying to make the transition. For a given environment (or equivalent graph), our focus was to understand the tradeoff between the number of robots, the probability of successfully completing the deployment, and the time spent. Our initial investigation has been useful to better understand the problem at hand and to identify a relevant space of design parameters when building multi-robot systems to solve this class of deployment problems. However, the theoretical setup made some strong assumptions. For example, it assumed that robots were capable of self localizing in the graph. It also assumed all robots were equipped with a primitive to move along an edge with a fixed speed.

In an effort to link theory and practice, and with the eventual goal of eventually deploying a fully functioning system, in this paper we relax some of the previous assumptions and tie our investigation to a specific robotic platform that can be considered an expendable asset. In particular, we relax the hypothesis that robots can reliably self localize themselves in the environment (or graph). The minimalist platform we consider is the *SensorFly* node that has been previously developed to deploy mobile sensor networks.[1] This robot is inexpensive, lightweight (30g) and offers minimal onboard processing power. Nevertheless, teams of SensorFly robots can collaborate and communicate with each other. The objective of this paper is to explore in simulation a simple *open loop* deployment strategy and determine how communication can be used to overcome the inherent limitations of this approach. Indeed, because of the limited sensor payload associated with the SensorFly

A. Purhoit and P. Zhang are with Carnegie Mellon University, Silicon Valley Campus, Moffett Field, CA, USA.
B.M. Sadler is with the Army Research Lab, Adelphi, MD, USA.
S. Carpin is with the School of Engineering, University of California, Merced, CA, USA.

Stefano Carpin is partially supported by the Army Research Lab under contract SUPP-13-6-CNC. Pei Zhang are partially supported by National Science foundation under awards CNS-1135874 and CNS-1149611, and by DARPA under grant 1080247-D11AP00265-DOI-ZHANG. Any opinions, findings, and conclusions or recommendations expressed in these materials are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the funding agencies of the U.S. Government.

[1]The term *node* was coined when the SensorFly platform was introduced. In the following, the terms *node* and *robot* will be considered synonyms.

platform, one cannot assume to implement state-of-the-art, sophisticated localization algorithms. Therefore an open loop approach (with all its limitations) is the best one can hope for, and communication between agents is used to improve the performance.

The rest of the paper is organized as follows. Related work is shortly discussed in Section II, and in Section III we describe the SensorFly node that inspired our model. Section IV defines the deployment problem based on the characteristics of the SensorFly platform. The deployment algorithm is then described in Section V. Simulations are offered in Section VI and future work and conclusions are sketched in Section VII.

II. Related Work

Deployment problems are related to coverage because in both problems robots are required to reach certain locations to gather data. Coverage is a very active research area and was greatly influenced by the work of Bullo and co-authors [1], [4]. However, coverage problems are very often studied in obstacle-free environments and focus on metrics different than the one we consider in our research, i.e., time to completion. Deployment problems are also related to robotic dispersion, like [8], [10]. Many approaches to robotic dispersion embrace a behavior-based approach and often aim at obtaining *asymptotic* properties, like e.g., guaranteeing that eventually all areas of a given environment are reached, or maintaining communication. The deployment problem studied in this paper instead is concerned with the transitory stage, i.e., we are interested in the process through which robots reach an assigned set of final destinations, and we aim at minimizing the time spent to complete the task. Large teams of simple, expendable robots have been designed and fielded for various applications. Notable examples include the SCOUT platform [10], Millibots [9], and the Kilobot [14], just to name a few. The SensorFly node we consider in this paper can be assimilated to these platforms, although it was developed within the sensor networks community and as such it is designed with a greater attention to communication and energy efficiency. Finally, there have been studies where robots collaborate and self deploy an infrastructure for localization [5], [7]. However, in our study we assume robots are not equipped with sensors capable of returning mutual information, so this class of approaches is not applicable. In our former paper [2] we studied a variant of the deployment problem where robots were required to reach each location and then stop, while here we allow robots to move forward after having visited one of the target places. Although the definition of the problem is slightly different in this paper, most of the lessons learned here can be transferred to the other scenario.

III. The SensorFly Platform

The SensorFly [11], shown in Figure 1, is a low-cost, low-weight, micro-aerial vehicle (MAV) swarm platform for facilitating research in emerging indoor sensor-networking applications such as disaster response, urban surveillance, and toxic plume monitoring. These applications scenarios are hazardous and rapidly-changing, requiring resilience, adaptability, and speed from the monitoring solution. The SensorFly swarm comprises of a relatively large number of miniature aerial sensor nodes capable of autonomous movement. Due to cost and size constraints individual nodes are limited in their sensing, computing, and communications capabilities. The swarm seeks to utilize collaboration and numbers to provide greater resilience, adaptability and speed of sensing compared to monolithic robots.

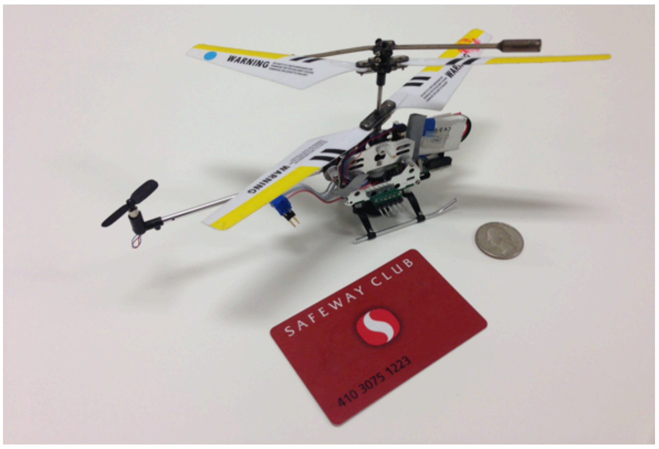
Fig. 1: The figure shows a SensorFly node with a US quarter dollar coin.

Each aerial SensorFly node is equipped with an 8-bit AVR AtMega128rfa1 micro-controller, inertial sensors (a 3-axis accelerometer and 3-axis gyroscope,) an ultrasonic ranger for altitude estimation, a 3-d magnetometer, an optical flow sensor for velocity estimation, and an 802.15.4a compatible radio with Round-trip time-of-flight (RToF) measurement capability. The entire platform weighs less than $30g$. Each node has a 130mAh LiPo battery with a continuous flight time of 6-8 minutes on a single charge. The design of the platform seeks to strike a careful balance between weight and individual sensing capability. The SensorFly node flight mechanism consists of a co-axial counter-rotating dual rotor that is passively stable for hover and forward flight. The node has 3 motors, 1 for each of the 2 rotors, and 1 for the tail rotor. Altitude control is attained by controlling the speed of the two main rotors of the node. Yaw control is achieved by increasing the speed of one rotor and reducing the speed of the other rotor by the same amount. Fixed-velocity forward flight of the node is attained by turning on the tail motor, which provides a forward tilt resulting in forward momentum. The SensorFly nodes are capable of receiving high-level motion commands, e.g., “Turn X degrees” and “Move forward X seconds”. The nodes execute motion commands via on-board PID controllers utilizing angular and translational velocity feedback from the gyroscope and optical-flow sensors, respectively. Assumptions made in the following algorithm and simulations mirror the capabilities offered by the SensorFly node and are based on extensive field experience with this platform [11].

IV. Problem Formulation and Modeling

We consider the following deployment task. We are given an indoor environment with N target locations, and the goal is to deploy a team of K robots so that eventually at least one robot reaches each of the N locations. We assume that a map of the environment is known and that all robots are initially deployed in a known area (or room), although their precise position within the area is not necessarily known. The assumption that the map is known is admittedly strong, but there is growing availability of indoor maps for public places (e.g., Google indoor maps) and this hypothesis is useful to start studying the problem in a simplified scenario.

In our previous work [2] we assumed that robots were capable of self localizing in the map without errors, and this allowed us to reduce the problem to an abstract formulation based on undirected graphs. Here we remove the hypothesis that robots can reliably localize themselves. SensorFly nodes can estimate their own location using exclusively on board sensors. Since the SensorFly platform does not include exteroceptive sensors, this estimation is necessarily based on proprioceptive sensors only. The problem we consider in this paper is then more difficult because imprecise localization implies that it may take a long time to reach a given target area. Moreover, because of imprecise localization robots may end up repeatedly bumping into walls and fail to complete a desired motion between two locations.

The problem we consider can be modeled using a graph. Let $G = (V, E)$ be a graph modeling the environment, where V is the set of vertices and E is the set of edges. The set of target locations is $T \subseteq V$. The existence of an edge e_k between vertices v_i and v_j indicates that there exists a path from v_i to v_j. One vertex in V represents the location where the SensoFly nodes are initially deployed and is indicated with the letter d.

V. Deployment algorithm

The role of the deployment algorithm is to increase the probability that each of the N relevant locations is reached by at least one robot. We associate to the graph G a spanning tree $\mathcal{T}$ rooted at the deployment vertex d.

We hypothesize that the swarm nodes are introduced into the operating environment at a known location. For example, a firefighter introduces nodes into a collapsed building through an accessible opening. In our case this is the area associated with the deployment vertex d.

Nodes estimate their location (vertex in graph) with respect to their initial location through dead-reckoning. Dead-reckoning is known to be an error-prone technique because its cumulative error grows over time. However, given the minimalistic platform we embrace, one cannot run more sophisticated localization techniques based on on-board sensors or assume the availability of external infrastructure offering localization. The SensorFly platform is equipped with the magnetometer and gyroscope sensors that together measure pose, as well as, an optical flow sensor that measures velocity. The challenge is then to exploit this reduced sensor payload to successfully complete the deployment task. The pseudo code for the deployment strategy is given in Algorithm 1. The same algorithm is supposed to be executed by each of the K robots in the team.

Initially, all nodes mark all vertices (except d) as non-visited. At each iteration the robot estimates its own location and maps it to a vertex in the graph. Then, if there are adjacent vertices that are unvisited the robot randomly selects one of them and tries to reach it. Otherwise (all neighbors are marked as visited), one random neighbor is selected.

The error in dead-reckoning location estimates accumulates with time. To account for the inaccurate location, the algorithm incorporates a strategy of backing-off in a randomly chosen direction on contact with obstacles. The length of the back-off segments is computed as an exponential function of the number of collisions in a specified time window.

```
1  while battery lasts do
2      estimate location $v_i \in \mathcal{T}$ from dead-reckoning sensors;
3      mark $v_i$ as visited and propagate info to other nodes;
4      find set of vertices $\mathcal{A}$ adjacent to $v_i$;
5      find set of unvisited vertices $\mathcal{U} \in \mathcal{A}$;
6      if $U$ is non-empty then
7          randomly choose a vertex $v_j \in \mathcal{U}$
8      end
9      else
10         randomly choose a vertex $v_j \in \mathcal{A}$
11     end
12     Move along edge $(v_i, v_j)$ to vertex $v_j$;
13     if collsion with obstacle then
14         increment collision counter $col\_cnt$;
15         compute back-off time $t_b$ as a exp function of $col\_cnt$;
16         choose random direction $p$;
17         back-off in direction $p$ for time $t_b$;
18     end
19     else
20         decrement collision counter $col\_cnt$;
21     end
22 end
```

Algorithm 1: Deployment algorithm

This deployment algorithm utilizes the ability of nodes to communicate with each other to propagate knowledge of visited nodes. With communication, the resource constrained nodes collaboratively improve the speed of coverage. All assumptions made in the deployment algorithm are inspired and consistent with the abilities offered by the real SensorFly platform, as further explained in the next section.

VI. Simulations

In this section we illustrate how the model we formulated can be used to simulate the deployment performance of the robot team. In particular, in numerous operative scenarios one is interested in deciding the number of robots to deploy in order to complete the deployment task within a given time. When the environment is complex it may be useful to run a preliminary simulation to get a sense about an appropriate size for the team. The goal of this section is not to present a comprehensive evaluation over a diverse set of environments, but rather to show how informed decisions can be made based on the assumptions we made and the simulations we present. Two environments offering different challenges are used.

A. Simulation Setup

We utilize a simulation environment for the SensorFly MAV indoor sensor swarm [12], [13][2] to evaluate our deployment algorithm in a realistic scenario. The software is written in Phython and optimized to use GPUs to allow an high accuracy simulation of systems including tens of SensorFly nodes. The simulator provides the ability to specify a realistic physical arena. It supports various Micro-Aerial Vehicle (MAV) mobility models, sensors and sensor noise models, wireless communication model, and application-specific sensing models. These models are based on data collected from MAV nodes during a large number of indoor environments. We use an empirically determined actuation noise model for the SensorFly platform [12] to inform our simulations. The SensorFly platform uses PID control with feedback from an optical flow sensor (velocity) and a gyro (turn) to execute the commanded motion. From [12], the pose and velocity error is determined to be within $+/-20\%$ of commanded value.

B. Experiments

Figure 2 shows the map of a simple indoor environment built by an autonomous robot running a Simultaneous Localization and Mapping (SLAM) algorithm while exploring part of the Science and Engineering building at the University of California, Merced. The figure also shows the corresponding graph with $N = 7$ vertices. In this case the deployment vertex is v_1. The objective of the swarm is to deploy at least one robot in each of the vertices. Figure 3 shows instead a more complex environment with 22 rooms. This environment is also obtained running a SLAM algorithm using a publicly available dataset.[3] Note that given a map produced by a SLAM algorithm, graphs like those displayed in the figures can be automatically extracted and do not have to be produced by hand [6].

[2]The SensorFly simulator is available for download at `https://bitbucket.org/imaveek/sensorflysim`.

[3]This map is obtained from the sdr40 dataset available on `http://radish.sourceforge.net`.

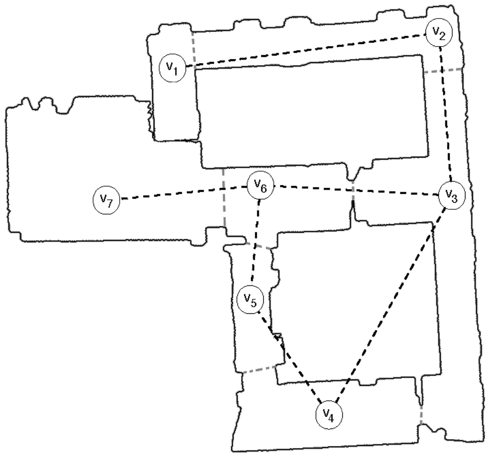

Fig. 2: Test environment with 7 rooms used to illustrate the presented framework. Nodes are initially deployed in the area corresponding to vertex v_1.

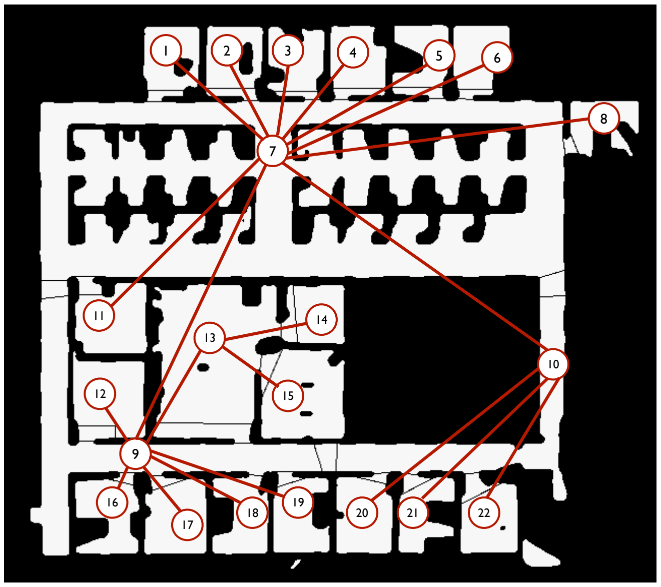

Fig. 3: Test environment with 22 rooms and associated tree. Nodes are initially deployed in the area corresponding to vertex 7.

Figure 4 shows the trend for completion time as a function of the number of robots in the 7 room map. Figure 6 shows the trend for completion time as a function of the number of robots in a larger 22 room map. Here, completion time is defined as the moment when each target vertex has been reached by at least one robot. For comparison purposes figure 4 shows also the completion time as a function of K in case the swarm just follows a random walk strategy.

Evidently, the expected time to completion is heavily influenced by the amount of error affecting the dead-reckoning estimation. Figure 5 and 7 then show the trend for completion time, for 10 nodes, as a function of the dead-reckoning error in the 7-room and 22-room map

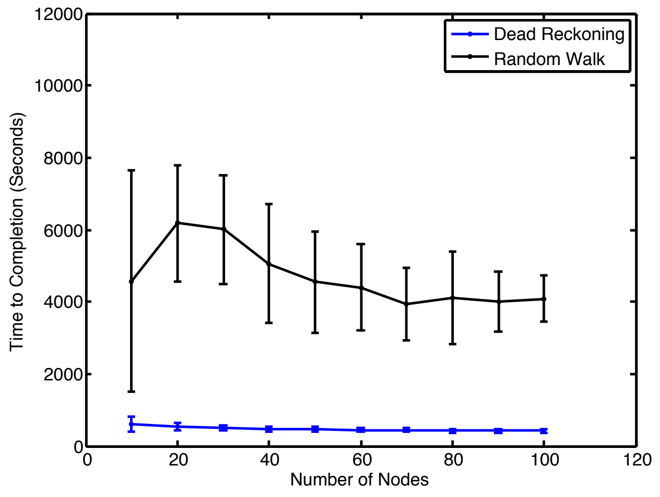

Fig. 4: Smulated time to complete a deployment as a function of the number of robots (7 room map). Error bars show standard deviation over 100 simulation runs. Dead-reckoning error is up to $+/-20\%$.

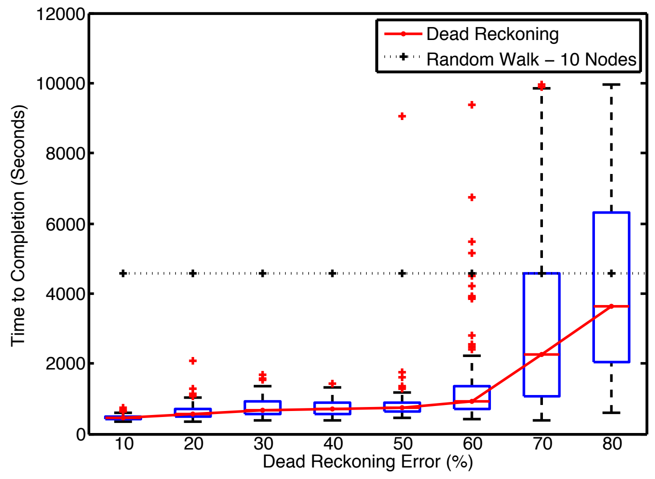

Fig. 5: Time to complete a deployment of 10 robots as a function of dead reckoning noise (7 room map). The box-plot shows the median, 25th and 75th percentile from 100 simulation runs.

respectively. The time to completion is a function of the robots' ability to localize themselves and correctly move towards designated map regions. In absence of external infrastructure and limited available on-board sensors, MAV swarm platforms must rely on innacurate localization estimates from techniques such as dead-reckoning. In addition, dead-reckoning error accumulates with time. The plots show the impact of location sensor noise on the time to complete deployment. As a consequence of inaccurate localization, robots may mark vertices as *visited* even when they have not reached them. This information is passed to other team members and has a negative impact that is eventually remedied when robots randomly pick the next location to visit among all their neighbors. Figure 8 shows the rate of misclassification as a function of the dead reckoning error. The take home message from this set of simulations is that even though dead-reckoning is an error prone technique, teams of SensodFly MAVs can successfully complete the deployment task by relying on communication and increased robustness through redundancy.

The simulation system we used is tuned to match the performance observed on SensorFly nodes in real world scenarios. While of course one should be careful in extrapolating results observed in simulation, the performance curves depicted are useful to make decisions about the approximate size of the team needed to meet the temporal constraints of the deployment task.

VII. Conclusions and Future work

In this paper we have extended our recent work on multi-robot deployment [2] tying our former high-level theoretical model with the SensorFly platform, an experimental MAV. In particular, we have relaxed the hypothesis that robots can reliably self localize in the map, and we have instead assumed that only rough localization can be obtained through dead-reckoning. This assumption is justified by the limited sensor payload currently available on the SensorFly nodes.

The deployment algorithm tries to overcome the limitations imposed by dead-reckoning through communication, one of the capabilities offered by the SensorFly node. In addition, heuristic maneuvers (backing off) to evade problematic situations emerging during the deployment have been implemented and have shown to be effective.

Numerous directions can be explored to extend this work,

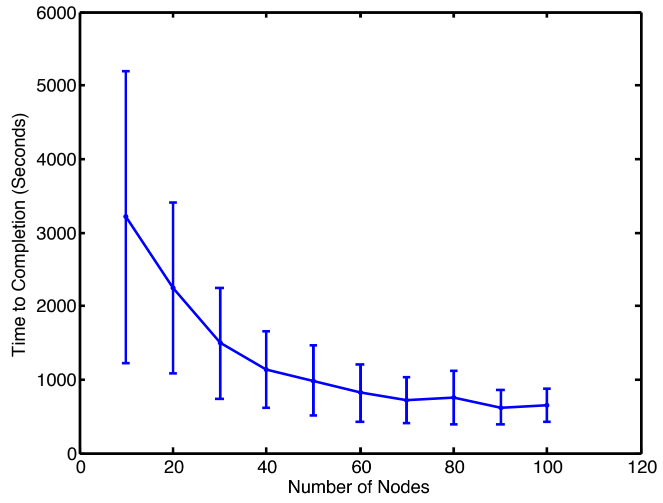

Fig. 6: Time to complete a deployment as a function of the number of robots (22 room map). Error bars show standard deviation over 100 simulation runs. Dead-reckoning error is up to $+/-20\%$.

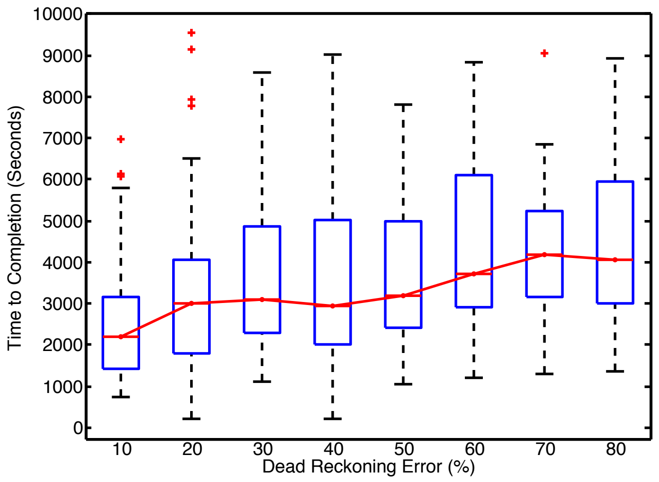

Fig. 7: Time to complete a deployment of 10 robots as a function of dead reckoning noise (22 room map). The box-plot shows the median, 25th and 75th percentile from 100 simulation runs

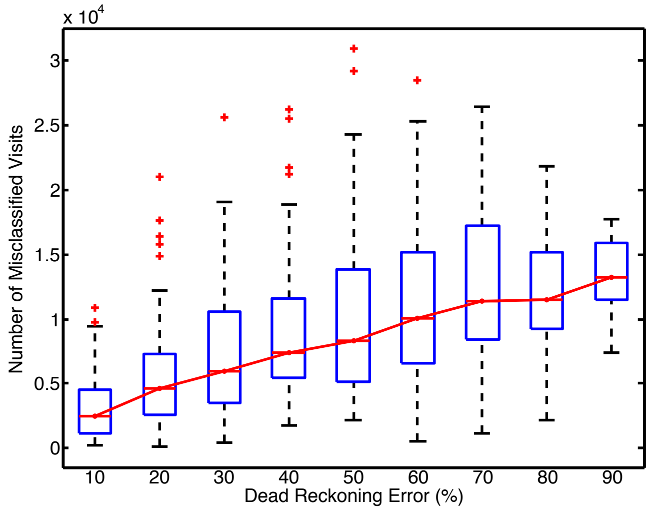

Fig. 8: Number of misclassifed vertex visits (10 robots) as a function of dead reckoning noise (22 room map). The box-plot shows the median, 25th and 75th percentile from 100 simulation runs

and we here sketch just a few of them. First, when navigating from vertex v_i to vertex v_j one can tune the velocity of the robot to maximize the chance of successfully completing the move. To do so, one needs a function mapping velocity to probability of success, as we did in [2]. Future work aims at predicting these parameters from simulation before the system is actually deployed.

Then one could consider more sophisticated navigation strategies explicitly considering temporal deadlines during the planning process. For example, deployment strategies based on constrained Markov decision processes are being developed for the deployment problem [3] and could be used with the SensorFly too. Of course, it will also be important to consider the case when the map is not known a priori, and to develop deployment strategies for this case. Our eventual goal is to migrate the developed algorithms from simulation to the real world, taking advantage of the preliminary experience gained while developing and perfecting the SensorFly MAV.

One of the limitations associated with the presented deployment algorithm is the lack of a formal performance analysis, as we did in [2]. From a practical point of view it would be useful to derive analytic relationships between the size of the team, the complexity of the environment, the time needed to complete the deployment, and the probability to successfully complete the mission. This analysis is part of our current and future work.

References

[1] F. Bullo, J. Cortés, and S. Martínez. *Distributed Control of Robotic Networks*. Princeton, 2009.

[2] S. Carpin, T.H. Chung, and B. Sadler. Theoretical foundations of high-speed robot team deployment. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2025–2032, 2013.

[3] S. Carpin and B.M. Sadler. Fast Multirobot Deployment Algorithms Using Constrained Markov Decision Processes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2013 (submitted).

[4] J. Cortés, S. Martínez, T. Karatas, and F. Bullo. Coverage control for mobile sensing netorks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, 2004.

[5] G. Erinc, G. Pillonetto, and S. Carpin. Online estimation of variance parameters: experimental results with applications to localization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1890–1895, 2008.

[6] A. Kolling and S. Carpin. Extracting surveillance graphs from robot maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2323–2328, 2008.

[7] R. Kurazume and S. Hirose. An experimental study of cooperative positioning system. *Autonomous Robots*, 8(1):43–52, 2000.

[8] J. McLurkin and J. Smith. Distributed algorithms for dispersion in indoor environments using a swarm of autonomous robots. In *Distributed Autonomous Robotic Systems (DARS)*, 2004.

[9] L.E. Navarro-Serment, R. Grabowski, C.J.J. Paredis, and P.K. Khosla. Millibots – the development of a framework and algorithms for a distributed heterogeneous robot team. *IEEE Robotics and Automation Magazine*, 9(4):31–40, 2002.

[10] J.L. Pearce, P.E. Rybski, S.S. Stoeter, and N.P. Papanilolopoulos. Dispersion behaviors for a team of multiple miniature robots. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1158–1163, 2003.

[11] A. Purohit, Z. Sun, F. Mokaya, and P. Zhang. SensorFly: Controlled-mobile sensing platform for indoor emergency response applications. In *In Proceeding of the 10th International Conference on Information Processing in Sensor Networks (IPSN)*, pages 223–234, 2011.

[12] A. Purohit, Z. Sun, and P. Zhang. SugarMap: Location-less Coverage for Micro-Aerial Sensing Swarms. In *In Proceeding of the 12th International Conference on Information Processing in Sensor Networks (IPSN)*, 2013.

[13] A Purohit and Pei Zhang. Controlled-mobile sensing simulator for indoor fire monitoring. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1124–1129, 2011.

[14] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: a low cost scalable robot system for collective behaviors. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3293–3298, 2012.