2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

A Fast, Accurate, and Scalable Probabilistic Sample-Based Approach for Counting Swarm Size

Hanlin Wang[1] and Michael Rubenstein[2]

***Abstract*— This paper describes a distributed algorithm for computing the number of robots in a swarm, only requiring communication with neighboring robots. The algorithm can adjust the estimated count when the number of robots in the swarm changes, such as the addition or removal of robots. Probabilistic guarantees are given, which show the accuracy of this method, and the trade-off between accuracy, speed, and adaptability to changing numbers. The proposed approach is demonstrated in simulation as well as a real swarm of robots.**

I. Introduction

The number of robots in a robotic swarm is used in a wide variety of applications; from self-assembling a shape at a scale proportional to the number of robots in a swarm, to optimizing behaviors based on swarm size. While some approaches make use of the swarm size given to it a priori [1], having the swarm learn its size on the fly offers more flexibility to behaviors and make it more tolerant to failures that may inadvertently change the swarm size.

One application in swarm robotics that explicitly or implicitly uses the swarm size is shape formation. In [1] the shape is sized to fit the number of robots in the swarm. For [2][3] the number of robots is implicitly learned by building the shape at larger and larger scales until the shape can no longer be built completely with the swarm. Building these intermediate sizes increases assembly time, and in some cases [2], it limits the types of shapes that can be formed. Quickly knowing the size of the swarm would greatly increase assembly speed.

Other application to swarm robotics could take advantage of knowing the swarm size to optimize various behaviors of robots. For example in task allocation [4] when assigning tasks to individuals, inefficiencies can occur for larger swarms [5] due to interference and motion traffic, if the number of robots was known, the allocation of tasks could be adjusted for optimality. Other tasks, such as computing a distributed consensus value [6] require waiting for messages to reach all individuals in the swarm. To ensure this, implementations of these algorithms assume a maximum number of robots in the swarm and set the wait time to allow the message to reach all robots in a swarm that size. If the swarm size was known, a less conservative wait time could be used, speeding up the distributed computation.

Additionally, knowing the swarm size could allow for new approaches that change behavior based on swarm size. For example, a swarm could use more conservative behaviors for small numbers, but could use riskier behaviours for larger sized swarms, where the loss of individuals would have less of an impact on outcome.

Some previous approaches for counting in robot swarms and adhoc networks include electing a leader then using distributed token passing [9] and counting the number of times a token is passed, or building a network tree [10] and propagating counting information from the leaves to the root. These types of approaches are difficult to implement in a moving robotic swarm where the communication topology can be dynamic.

Other approaches are based on a distributed gossip algorithm [11], however there is an implicit maximum number allowed, based on the time allowed for each gossip round. The approach by [12] and extended by [13] can measure swarm size in dynamic systems, but either requires long range communication or an additional messaging overhead to propagate local messages globally. Similar approaches first elect a single leader and then run distributed consensus between the leader's value of 1 and all others with a value of 0 [14]. The average here will be $\frac{1}{n}$ where n is the swarm size. This approach requires synchronized communication and has an implicit maximum size based on the time the distributed consensus algorithm is run.

II. Algorithm Description

We propose an algorithm that can estimate the number of robots in the swarm, which is scalable, can adapt to changing numbers of robots, works in the dynamic communication environments of swarms and does not have an implicit maximum it can count.

The proposed algorithm is inspired by the following simple idea: Let $s_1, ...s_n$ be n independent samples that are taken from a uniform probability distribution between 0 and 1, we compute the max value of all samples, $\max_{1\leq i\leq n} s_i$. The more samples that are taken, i.e. the larger n, the closer $\max_{1\leq i\leq n} s_i$ is expected to be to 1. Furthermore, the value of $\max_{1\leq i\leq n} s_i$ can be used to estimate the number of samples, n. This can be repeated multiple rounds, where in each round, j, a new set of n samples are generated and the maximum is computed just for the samples in that round $\max^j_{1\leq i\leq n} s_i$. The average of these max samples found in all m rounds, $k = \frac{\sum_{j=1}^{m} \max^j_{1\leq i\leq n} s_i}{m}$, can be used to provide an estimate of n with less variance when compared to any single rounds $\max_{1\leq i\leq n} s_i$. See Algorithm 1 for the detailed pseudo code.

[1]H. Wang is with the Department of Computer Science, Northwestern University, Evanston, IL, USA. e-mail: h.w@u.northwestern.edu

[2]M. Rubenstein is with the Department of Computer Science and the Department of Mechanical Engineering, Northwestern University, Evanston, IL, USA. e-mail: rubenstein@northwestern.edu

Algorithm 1: Pseudo code for swarm size estimation

```
maxs ← {} // the variable to record sample maximums in all m rounds
for j ← 0 to m do
    max ← 0 // the variable to find the sample maximum in current round
    for i ← 0 to n do
        s_i ← X_i ~ U(0,1)
        if s_i > max then
            max ← s_i
    maxs ← {max} ∪ maxs
k← average of all the elements in maxs
n* = k/(1−k) // calculate the estimation of the swarm size
```

This idea is mapped to a robotic system by having each robot generate a random number sampled from a uniform probability distribution between 0 and 1. The swarm then communicates amongst themselves to compute the largest number generated by any robot, and uses that number to estimate the number of robots in the swarm. As before, this process can be repeated for multiple rounds to give better estimates of n.

III. Theoretical Results

In this section, we study the correctness and efficacy of the algorithm that is proposed in Section II. We first use lemma 1 to show that the algorithm is correct, and then use the theorem 1 to show that the error of estimation will linearly converge to 0 in probability.

Definition 1. *Let n be the actual swarm size, n^* be the estimation obtained from Algorithm 1, we define the **RE** (relative error) of the estimation n^* as:*

$$RE(n^*) = \frac{|n^* - n|}{n^* + 1}$$

Lemma 1. *The sample maximum of n i.i.d. $\mathcal{U}(0,1)$ random variables is an unbiased estimator of $\frac{n}{n+1}$.*

Proof: Suppose we have n i.i.d. uniform random variables:

$$\mathcal{X}_i \sim \mathcal{U}(0,1), i = 1, 2, .., n$$

the *CDF* (cumulative density function) of their sample maximum $\mathcal{Y} = \max_{1 \le i \le n} \mathcal{X}_i$ is:

$$Pr(\mathcal{Y} \le x) = \prod_{i=1}^{n} Pr(\mathcal{X}_i \le x) = \begin{cases} 1, & x > 1 \\ x^n, & 0 \le x \le 1 \\ 0, & x < 0 \end{cases}$$

and the *PDF* (probability density function) of $\mathcal{Y}$ is:

$$p(\mathcal{Y} = x) = \frac{\mathrm{d}Pr(\mathcal{Y} \le x)}{\mathrm{d}x} = \begin{cases} 0, & x > 1 \\ nx^{n-1}, & 0 \le x \le 1 \\ 0, & x < 0 \end{cases}$$

From this, it is straight forward to develop the $\mathcal{Y}$'s expected value $\mathbb{E}(\mathcal{Y})$ and variance $\mathbb{V}ar(\mathcal{Y})$:

$$\mathbb{E}(\mathcal{Y}) = \int_0^1 nx^n \mathrm{d}x = \frac{n}{n+1}$$

$$\mathbb{V}ar(\mathcal{Y}) = \int_0^1 nx^{n-1}(x - \frac{n}{n+1})^2 \mathrm{d}x = \frac{1}{(n+1)^2}\frac{n}{n+2}$$

■

Theorem 1. *Given an error margin ϵ, the probability that the RE(n^*) exceeds ϵ will linearly decay over number of trials m. Specifically:*

$$Pr\{RE(n^*) \ge \epsilon\} < \frac{1}{m\epsilon^2}$$

Proof: It is well known that for m i.i.d random variables $\mathcal{Y}_1, ..., \mathcal{Y}_n$ that have a expected value μ and a variance σ^2, it hold that:

$$\mathbb{E}(\frac{\sum_{i=1}^m \mathcal{Y}_i}{m}) = \mu, \quad \mathbb{V}ar(\frac{\sum_{i=1}^m \mathcal{Y}_i}{m}) = \frac{\sigma^2}{m}$$

In our case, specifically, let $k = \frac{\sum_{i=1}^m \mathcal{Y}_i}{m}$ and $\mathcal{Y}_i$ be the sample maximum of n $i.i.d.$ $\mathcal{U}(0,1)$ random variables, using the result we obtained in lemma 1, we have:

$$\mathbb{E}(k) = \frac{n}{n+1}, \quad \mathbb{V}ar(k) = \frac{1}{m}\frac{1}{(n+1)^2}\frac{n}{n+2}$$

By *Chebyshev* inequality, we have:

$$Pr(|k - \mathbb{E}(k)| \ge \delta) \le \frac{\mathbb{V}ar(k)}{\delta^2}$$

On the other hand, let $n^* = \frac{k}{1-k}$, it is straight forward to examine that:

$$RE(n^*) = \frac{|n^* - n|}{n^* + 1} = (n+1)|k - \mathbb{E}(k)|$$

that is:

$$Pr(\frac{RE(n^*)}{n+1} \ge \delta) \le \frac{1}{m\delta^2}\frac{1}{(n+1)^2}\frac{n}{n+2}$$

Note that $n+1$ is a constant. Let $\epsilon = \delta(n+1)$, by this variable change we have:

$$Pr(RE(n^*) \ge \epsilon) \le \frac{1}{m\epsilon^2}\frac{n}{n+2} < \frac{1}{m\epsilon^2}$$

■

Remark: Surprisingly, the result we obtained in Theorem 1 suggests that the convergence rate of *RE(n^*)* is independent of number of samples n, which suffices to show that the algorithm is **scalable** from a theoretical perspective.

IV. Algorithm Implementation

In practice, when implementing the algorithm proposed in Section II, each agent needs to wait "long enough" between trials so as to give the swarm sufficient time to find the largest number in each trial, i.e, enable the random number generated by itself to propagate through the entire swarm. However, this waiting time is essentially a function of swarm size, which incurs the "chicken and egg" paradox, as the our task is to estimate the swarm size.

To tackle this problem, we developed an error-driven method with which each agent a_i can use the local-only observation to maintain an estimation of the necessary waiting time. The idea is shown as following: We assume that each

agent a_i transmits the messages at the same constant frequency f. Through the experiment, each agent will maintain an estimation of swarm's communication diameter d_i, i.e. the longest distance (in term of communication hop) between one agent to another in swarm, and it use this d_i to calculate the waiting time. To be specific, between two adjacent trials, a_i waits $\frac{4d_i}{f}$ amount of time.

To estimate d_i, during each trial j, each agent uses the hop-count algorithm [7] to estimate the distance (in term of communication hop) hop_i^j between itself and the agent that generates the largest number of the trial. If the agent **underestimates** the communication diameter, then two error events may occur, where a_i will update d_i:

- a_i sees a neighbor a_j such that $d_j > d_i$: This implies some other agent has seen a larger pairwise communication distance. In this case, a_i set $d_i = d_j$;
- d_i is less than the hop_i^j: This suggests the agent's diameter estimation is less than the communication distance between itself and some other agent. This is a more serious error because hop_i^j is already a lower bound of the communication diameter, hence in this case we punish d_i by setting $d_i = 2\ hop_i^j$.

It is straight forward to examine that the waiting time $\frac{4d_i}{f}$ is sufficient to enable the agent to detect these two errors before it starts the next trial.

The other challenge we have when developing a decentralized implementation of the algorithm is that: In reality, agent's clocks are not perfectly synchronized. To solve this problem, we embedded a sequence number in each trial, and enforce the agent to only use the information coming from the message that has the same sequence number. See Algorithm 2 for implementation details.

The algorithm is implemented in a "listen-think-talk" manner. Specifically, the algorithm consists of three modules: main module, broadcast module, and message handler module. The broadcast module constantly transmits messages to neighbors at a fixed frequency f; the main module handles computation and memory management, and message handler modifies the local variables according to the incoming messages. These three modules can be implemented using three separate threads that communicate through shared memory. The sketches of these three modules are shown in Algorithm 2, 3, and 4. Note that all the variables are thread-public.

A. Main module

The *Main module* takes two inputs: f and m where f is the communication frequency and m is the number of trials that will be used for estimation. The algorithm first claim and initialize the variables that will be used in the later calculation, to be specific (Algorithm 2 *Line 1-10*):

- *buff*: a ring buffer whose length is m, it is used to store the most recent m trials' results in the history;
- *index*: the variable that helps to operate *buff*;
- *seq_#*: each trial's sequence number;
- *sample_max*: the largest number of current trial;
- n^*: agent's estimation of swarm size;

Algorithm 2: Main Module

```
   Input: m, f
1  buff ← {0}
2  index ← 0
3  seq_#← 0
4  sample_origin ← rand(0, 1)
5  sample_max ← sample_origin
6  n* ← 0
7  hop ← 0
8  max_hop_seen ← 1
9  diameter ← max_hop_seen
10 last_check ← clock()
11 while agent is active do
12     msg ← {seq_#, sample_max, hop, max_hop_seen}
13     if clock() - last_checked > 4 diameter / f then
14         if hop > max_hop_seen then
15             max_hop_seen ← hop
16         if diameter < max_hop_seen then
17             diameter ← max_hop_seen
18             continue
19         buffer[index] ← sample_max
20         index = (index + 1) mod m
21         k ← average(buff)
22         n* ← k / (1−k)
23         seq_#← seq_# + 1
24         sample_origin ← rand(0, 1)
25         sample_max ← sample_origin
```

Algorithm 3: Broadcast Module

```
1 while agent is active do
2     transmit msg
3     sleep 1/f
```

Algorithm 4: Message Handler

```
1  while agent is active do
2      if receive a msg then
3          if msg.seq_# - 1 > seq_# then
4              seq_# ← msg.seq_# - 1
5          if msg.seq_# < seq_# then
6              last_check ← clock()
7          if msg.seq_# == seq_# then
8              if msg.sample_max > sample_max then
9                  sample_max ← msg.sample_max
10             if msg.hop < hop then
11                 hop ← msg.hop + 1
12             if sample_max == sample_origin then
13                 hop ← 0
14             if max_hop_seen < msg.max_hop_seen then
15                 max_hop_seen ← msg.max_hop_seen
```

- *diameter*: agent's estimation of the communication diameter of swarm;
- *hop* and *max_hop_seen*: the variables that help to update *diameter*;
- *last_check*: the beginning time of current trial.

After the initialization phase, the agent enters the main loop. At the beginning of each iteration, the agent first forges the message that will be transmitted by Broadcast Module (Algorithm 2 *Line 12*), then checks whether it has

already waited long enough to start the next trial (Algorithm 2 *Line 13*). If the agent has waited long enough already, it first checks whether the current estimation of swarm's communication diameter needs to be updated (Algorithm 2 *Line 14-18*), and if the current diameter estimation is correct, the agent then adds the result to *buff* (Algorithm 2 *Line 19*), updates the estimation n^* (Algorithm 2 *Line 21-22*), and initiates another trial (Algorithm 2 *Line 13-25*).

B. Message Handler

When receiving a message, the Message Handler first compares *msg.seq_#* with the local *seq_#*:

If *msg.seq_# - 1* $>$ *seq_#*, it implies that the local clock is slower than the neighbor by more than one trial, the agent then sets *seq_#* = *msg.seq_# - 1* so as to catch up with the neighbor (Algorithm 4 *Line 3-4*).

If *msg.seq_#* $<$ *seq_#* , it suggests that there is a neighbor that is slower than the agent, then the agent will delay the next trial so as to give the neighbor time to catch up.

If the neighbor and the agent have the same sequence number, then the agent will use the information in the message to update its local variables (Algorithm 4 *Line 8-15*): Algorithm 4 *Line 8-9* is for finding the largest random number in the current trial; Algorithm 4 *Line 10-13* is for finding the distance (in term of communication hop) between itself and the agent that generates the largest number of the trial; Algorithm 4 *Line 14-15* is for finding the largest pairwise communication distance in swarm.

C. Complexity

First, we study the algorithm's memory complexity. One can easily examine that the algorithm's memory complexity is dominated by the size of buffer to store the trials' results in Algorithm 2, In the other words, the algorithm's memory complexity is $\mathcal{O}(m)$.

Next, we investigate the algorithm's computation complexity. We here assume that the computation complexity of querying a random number generator (RNG) is $\mathcal{O}(1)$. Then the computational cost of each iteration of the algorithm is dominated by the calculation to find the average over *buff*, as a result, the algorithm's computation complexity is $\mathcal{O}(m)$.

Last, it is straight forward to examine that the algorithm's communication complexity, i.e., the length of each message, is $\mathcal{O}(1)$.

Remark: The results we obtained in this section suggests that the algorithm's cost is independent of swarm's size n with respect to computation complexity, memory complexity, and communication complexity, which suffices to show that the algorithm is **scalable** from a engineering perspective.

V. Experimental Results

A. Basic Simulations

To simulate the estimation algorithm on a large number of robots, and on a large number of experiments, we first implemented the behavior on a simplified simulation in Matlab. In this simplified simulation, we assume robots are all synchronized and have global communication. Here the non-ideal effects of a robotic implementation are ignored to allow for fast and efficient experiments on up to 100,000 simulated robots. These first simulations were used to validate predicted properties of the algorithm such as scalability and the effect the number of rounds has on precision of estimates.

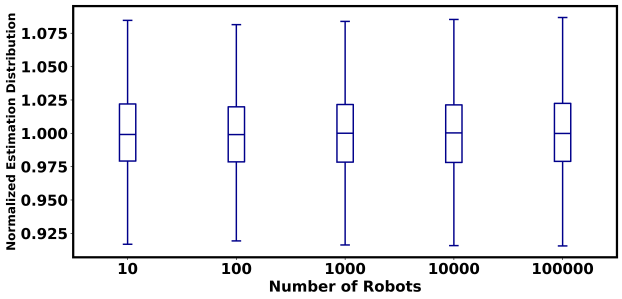

Fig. 1: The distribution of estimates for the number of robots for varying swarm size. The distribution is normalized by the true number of robots. Each boxplot represents the distribution of 2000 experiments in which the swarm used 1000 rounds to estimate the number of robots.

One property of this algorithm predicted in our analysis is that the normalized error (i.e. the percentage of error) is only dependent on the number of rounds, not the swarm size. To show this, we looked at swarms sized from 10 to 100,000 and ran the estimation for 1000 rounds. This was repeated for 2000 experiments. Figure 1 shows that the normalized accuracy of swarm size estimates are indeed independent of swarm size.

Next, the simplified simulation was used to show how the number of rounds used to estimate swarm size has an effect on the accuracy of the estimate. Fig. 2 shows the estimation of 10,000 experiments for varying numbers of rounds, showing that increasing the number of rounds produces less variance in the estimated swarm size. Fig. 3 shows how on average the estimated number is improved as more rounds are included in the estimate.

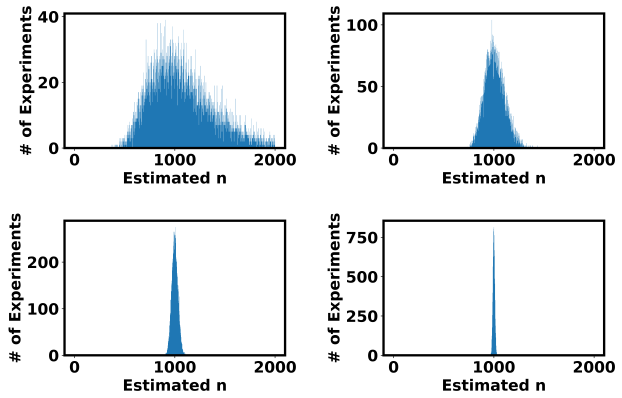

Fig. 2: The histograms of estimates of the swarm size for a 1000 robot swarm after 10 (top left), 100 (top right), 1000 (bottom left), or 10000 (bottom right) rounds. Each histogram represents the range of estimates of 10,000 experiments.

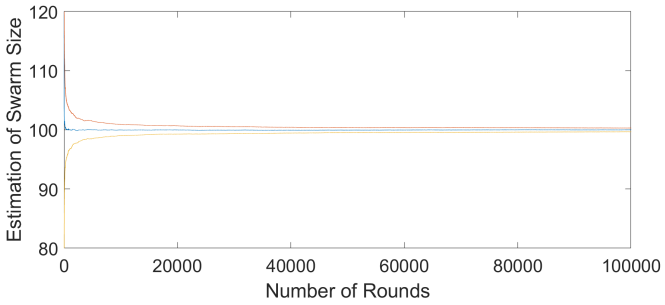

Fig. 3: The average, and 1 standard deviations above and below average of 1000 experiments with 100 simulated robots as increasing number of rounds are included in the size estimate.

B. Kilobot Simulations

To validate our method's performance on a large-scale swarm system, we tested our algorithm using an agent-based simulator that is originally developed for [8]. The simulator implements both the Kilobot's motion and communication in a very realistic way. Moreover, the user interface of the simulator is exactly the same as the one that is on actual Kilobot. In the simulation, each agent communicates with neighbors at a frequency of 15Hz and the maximum transmission unit (MTU) is 9 bytes.

First, we use simulation to investigate the effect of the buffer size m on the algorithm's accuracy and adaptability to changes. In simulation, up to 1000 agents execute our algorithm to estimate the swarm size using varying of trials in the past. The initial swarm size is 1000, and after 1000 seconds, we remove half of agents from the swarm. Fig. 4 shows the each agent's estimation about swarm size over time.

In the second test, we use simulation to investigate the convergence rate of the method. In these experiments, swarms of size 10 to 1000 agents use all the trials in the history to estimate the swarm size. The result is shown in Fig. 5.

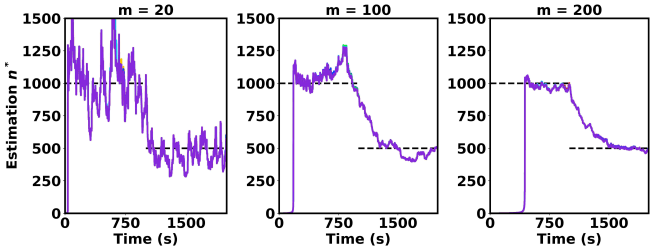

Fig. 4: The results for the test where up to 1000 agents estimate the swarm size using fixed number of trials. From left to right: $m = 20, 100, 200$. The black dotted line is the actual swarm size n and the solid colored lines are agents' estimations over time. The different colors indicates the results from different agents, plots are overlapping as many have similar estimates.

A third simulation is given to demonstrate our algorithm's performance on the swarm with extremely large size. In this test, 5000 simulated agents perform two experiments: In first experiment, agents use 200 trials in the past to estimate the swarm size, and the swarm size is initialized to be 5000 then drops to 2500 after 3000 seconds; in the second experiment, 5000 simulated agents use all the trials in the history to do

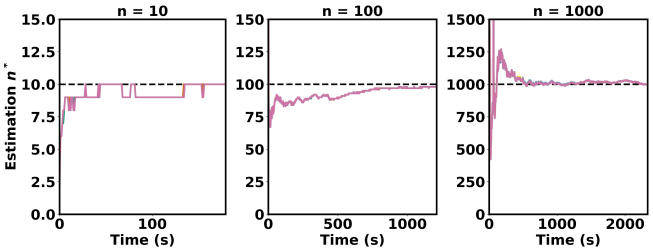

Fig. 5: The results for the tests where agents estimate the swarm size using all the trials in the history. The black dotted line is the actual swarm size n and the solid colored lines are agents' estimations over time, plots are overlapping as many have similar estimates.

the estimation, and the swarm size does not change in this experiment. The results are shown in Fig. 6.

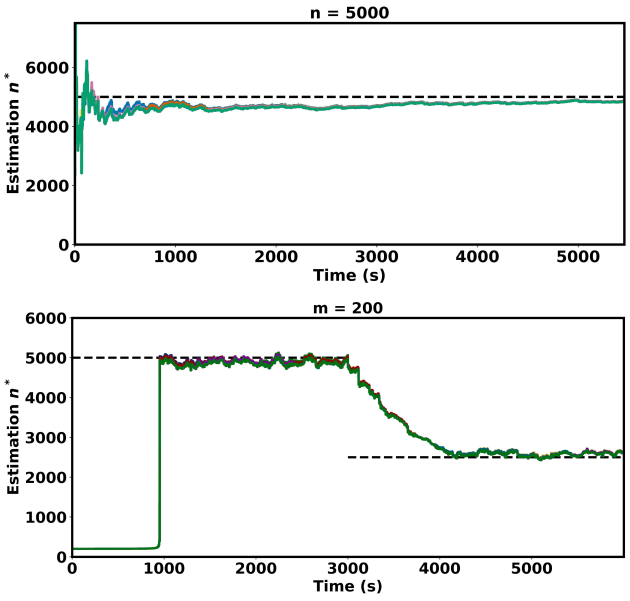

Fig. 6: The experiments to demonstrate the algorithm's performance on a swarm of 5000 agents. The black dotted line is the actual swarm size n and the solid colored lines are agents' estimations over time. The different colors indicates the results from different agents, plots are overlapping as many have similar estimates.

C. Real Kilobot implementation

To validate our algorithm's performance beyond simulation, we implemented our algorithm on a swarm of 35 Kilobots [15]. In reality, it is hard to collect the data from each individual in real-time. As a result, instead of paying attention to each individual's estimation, we use a "probe message" to collect the minimal and maximal estimation amongst the swarm.

We performed two tests on the physical Kilobot swarm: The first test is given to demonstrate the algorithm's convergence rate, and the second second test is given to show the algorithms accuracy and adaptability to changes. In the first test, 35 Kilobots are tasked to use all the trials in their history to estimate the swarm size, the result for this test is shown in Fig. 7 (top); in the second test, Kilobots use

100 most recent trials to estimate the swarm size, moreover, the swarm size is initialized to be 35 then drops to 25 after 2600 seconds. The result for the second test is shown in Fig. 7 (bottom).

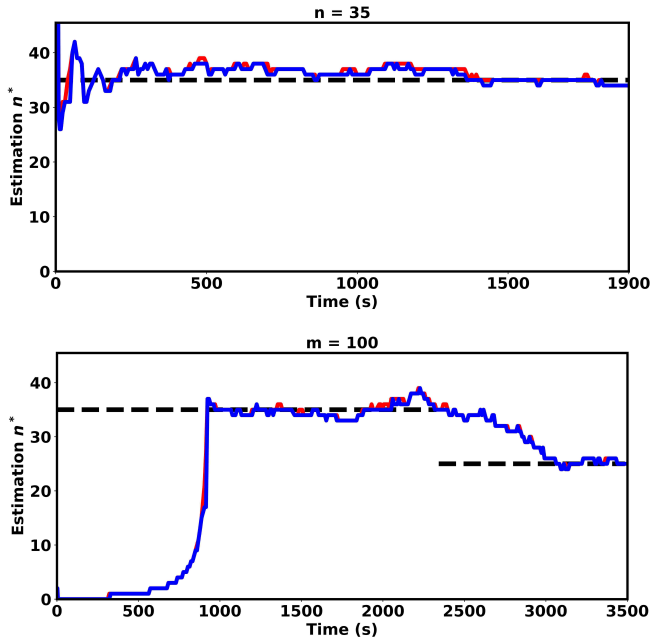

Fig. 7: The results from physical experiments. The black dotted line is the actual swarm size n, the red and blue solid lines are largest and smallest estimation amongst the swarm, respectively.

VI. Conclusion

We present an algorithm for counting the number of robot in a swarm, which is scalable, and capable of adapting to changing robot numbers during runtime. We provided theoretical results that show the convergence rate and the effects of sample rounds. The algorithm is shown to work in simulations as well as a real swarm of robots.

References

[1] Rubenstein, Michael, Alejandro Cornejo, and Radhika Nagpal. "Programmable self-assembly in a thousand-robot swarm." Science 345, no. 6198 (2014): 795-799.

[2] Stoy, Kasper, and Radhika Nagpal. "Self-repair through scale independent self-reconfiguration." In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 2, pp. 2062-2067. IEEE, 2004.

[3] Rubenstein, Michael, and Wei-Min Shen. "Automatic scalable size selection for the shape of a distributed robotic collective." In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 508-513. IEEE, 2010.

[4] Labella, Thomas Halva, Marco Dorigo, and Jean-Louis Deneubourg. "Self-organised task allocation in a group of robots." In Distributed Autonomous Robotic Systems 6, pp. 389-398. Springer, Tokyo, 2007.

[5] Lerman, Kristina, Alcherio Martinoli, and Aram Galstyan. "A review of probabilistic macroscopic models for swarm robotic systems." In International Workshop on Swarm Robotics, pp. 143-152. Springer, Berlin, Heidelberg, 2004.

[6] Yu, Chih-Han, and Radhika Nagpal. "Self-adapting modular robotics: A generalized distributed consensus framework." In 2009 IEEE International Conference on Robotics and Automation, pp. 1881-1888. IEEE, 2009.

[7] Gauci, Melvin, Monica E. Ortiz, Michael Rubenstein, and Radhika Nagpal. "Error cascades in collective behavior: a case study of the gradient algorithm on 1000 physical agents." In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, pp. 1404-1412. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[8] Ebert, Julia T., Melvin Gauci, and Radhika Nagpal. "Multi-feature collective decision making in robot swarms." In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, pp. 1711-1719. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[9] Malpani, Navneet, Yu Chen, Nitin H. Vaidya, and Jennifer L. Welch. "Distributed token circulation in mobile ad hoc networks." IEEE Transactions on Mobile Computing 4, no. 2 (2005): 154-165.

[10] Mathy, Laurent, Roberto Canonico, and David Hutchison. "An overlay tree building control protocol." In International Workshop on Networked Group Communication, pp. 76-87. Springer, Berlin, Heidelberg, 2001.

[11] Kostoulas, Dionysios, Dimitrios Psaltoulis, Indranil Gupta, Ken Birman, and Alan Demers. "Decentralized schemes for size estimation in large and dynamic groups." In Fourth IEEE International Symposium on Network Computing and Applications, pp. 41-48. IEEE, 2005.

[12] Melhuish, Chris, Owen Holland, and Steve Hoddell. "Convoying: using chorusing to form travelling groups of minimal agents." Robotics and Autonomous Systems 28, no. 2-3 (1999): 207-216.

[13] Manuele Brambilla, "Group Size Estimation" Thesis, 2009. in Swarm Robotics

[14] Elwin, Matthew L., Randy A. Freeman, and Kevin M. Lynch. "A systematic design process for internal model average consensus estimators." In 52nd IEEE Conference on Decision and Control, pp. 5878-5883. IEEE, 2013.

[15] Rubenstein, Michael, Christian Ahler, Nick Hoff, Adrian Cabrera, and Radhika Nagpal. "Kilobot: A low cost robot with scalable operations designed for collective behaviors." Robotics and Autonomous Systems 62, no. 7 (2014): 966-975.