A study of approximation in a collaborative multi-agent system

Hrishav Bakul Barua, Himadri Sekhar Paul, Chayan Sarkar
TCS Research & Innovation, India

Ansuman Banerjee
Indian Statistical Institute
Kolkata, India

ABSTRACT

The effect of approximation has been well studied in stand-alone systems. However, the problem of approximation in a collaborative system has not been studied, to the best of our knowledge. The basic goal of this article is to study the applicability of approximation in collaborative SLAM (simultaneous localization and mapping). Our experiments suggest that it is not trivial to combine multiple stand-alone approximate results to achieve a collaborative approximation, i.e., the resultant error can not be bounded without special effort. Thus, we present a model of the problem and empirically show that such a model can be used to explain the error variance in a collaborative system.

CCS CONCEPTS

• **Computer systems organization → Robotics**.

KEYWORDS

collaborative slam, multi-agent system, approximate computing

ACM Reference Format:
Hrishav Bakul Barua, Himadri Sekhar Paul, Chayan Sarkar and Ansuman Banerjee. 2018. A study of approximation in a collaborative multi-agent system. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Modern generalized computing systems are over-provisioned for accuracy, so they spend more than the necessary time and energy to achieve a "correct" output. In recent years, *inexact* or *approximate* computing has emerged as an alternative technique to address resource constraints in embedded systems. The underlying philosophy is to sacrifice a little accuracy from computation and gain in terms of time or energy or both [3]. There are a variety of applications that are inherently tolerant of approximation, e.g., computer vision, media processing, machine learning, etc. A comprehensive study on approximate computing, in both the fields of hardware and software, can be found in [8, 14].

Simultaneous Localization and Mapping (SLAM) is an application of immense importance in the area of robotics. Researchers have explored the idea of collaborative SLAM using a team of low-cost mobile robots to reduce complexity of the task. On the other hand, several research efforts have focused on efficient single-agent SLAM using approximate computing for map creation, wherein a close but not-so-accurate map of the terrain is created for localization efficiently, without draining much energy of the robot. Although, collaborative robotic systems are becoming increasingly common, to the best of our knowledge, the effect of approximate computing in a collaborative SLAM setup has not been investigated in the literature.

In this work, we propose to approximate the Multi-Robot SLAM (MR-SLAM) algorithm [7] for a distributed multi-agent scenario. In this case, multiple robots explore unknown space. Each robot senses its surroundings using a laser scanner and builds up a partial map around it. Although the robots work independently, whenever two of them come within their communication range, they exchange their partially built maps to generate a more elaborate private view of the space. This way, the robots build a complete map of the entire area without exploring it in entirety.

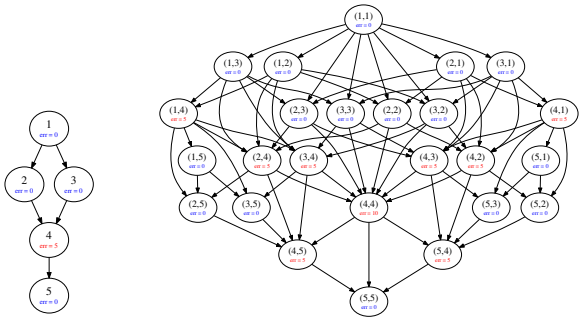
Our main objective is to study the behavior of approximation on collaborative SLAM and propose techniques for improving the execution time of collaboration using approximation. We show that the problem is not as trivial as replicating the single-agent approximate SLAM routines in a multi-agent setup. One of the main motivations of this paper is to show that independently created approximation routines executed inside single robots with promised benefits, fail to deliver expected performance gains when composed with similar approximate routines running in their peers. Since the approximation strategies are computed independently, without any consideration for collaboration, it is often the case that the composed map fails to meet the required accuracy. This is the main motivation and observation of this paper. In general, we observe that it is often the case that the performance-accuracy trade-off for a single system may not be directly applicable in a collaborative system.

2 LITERATURE REVIEW

Robotics system usually operates in an inexactly defined environment with inexactly calibrated hardware. Therefore majority of the robotics related algorithms are probabilistic in nature, and therefore inherently approximated [20]. *Approximate* or *in-exact computing* addresses approximation at the implementation level exploiting hardware as well as software. In-exact hardware, which saves energy but guarantees bounded error of output, have been proposed in [3]. Compilers, which enable a programmer to exploit the underlying inexact hardware, are also available [10, 15, 16]. Even the Internet of Things (IoT) paradigm has proposed to used approximation [11, 17]. Systematic surveys on this subject can be read in [8] and [2] that describe various tools and techniques for applying approximate computing paradigm in various domains and platforms.

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456



(a) a CDFG (C) (b) composed CDFG ($C \circ C$)

Figure 1: Control and Data Flow Graph (CDFG) abstraction of a program in stand-alone and collaborative system.

Multi-robot collaborative SLAM has gained a lot of research interest due to its potential to reduce the map building time significantly [18, 21]. It also reduces energy consumption on individual robots since each one of them does not require to survey the entire area. Foster *et al.* [5] proposed a collaborative monocular SLAM that is done using multiple resource constrained micro aerial vehicle. Similarly, Deutsch *et al.* [4] proposed a real-time SLAM framework, which is feasible due to collaboration among multiple robots.

Though collaborative SLAM brings overall speedup in the process, combined energy consumption by all the robots is not reduced by collaboration. Thus techniques such as approximate computing is also explored in this domain. Recently, Oh *et al.* [9] presented an approximate implementation of SLAM with loop perforations applied to two most computationally costly parts of the code and demonstrated significant computation and energy gain against negligible change in quality. In a more recent work, Reiter *et al.* [13] presented techniques for approximately computing trajectories for robotics system in real-time; however, their technique is not based on approximate computing paradigm.

3 MODELING FORMALISMS

Before discussing the approximate collaborative SLAM, first, we present the underlying model of a SLAM program and how it is customized to an approximate version. We use the *Control and Data Flow Graph* (CDFG) abstraction of a program to model the underlying computation of a SLAM process. Any program p can be modeled as a CDFG $G = < B, T, b_0 >$, where B is the set of basic blocks derived from the program, T is the set of directed edges denoting possible control transitions from one block to another, and $b_0 \in B$ is the initial block from which the control begins when the program is executed. An execution of p in this model is defined as a path through this CDFG.

An approximate version of a program p is one in which one or more lines of code or variable types are approximated, i.e., downsizing the variables to reduce code size and/or cutting down iterations of a loop to improve timing while ensuring the final program (degraded) output remains within a certain error threshold. Let us denote the maximum degradation of the output measured as e due to this modification. We introduce a function f that assigns to each node in the CDFG, its contribution of error to the output. Thus the (approximate) model of a program, G, can be expressed as $C = < B, T, b_0, f >$, where $f : B \rightarrow \mathfrak{R}$ and error is measured as a value $e \in \mathfrak{R}$.

Our objective now is to model the composed behavior of a set of n such approximated programs executing in a set of n independent components. Let these n programs be denoted as $p_1, p_2, \ldots p_n$ such that p_i drives the i^{th} component and let us denote $C^{(i)} =< B^{(i)}, T^{(i)}, b_i^{(0)}, f^{(i)} >$ to be the corresponding CDFG for p_i. The behavior of the composed system can be characterized as the composition of these CDFGs, $\widehat{C} = C^{(1)} \circ C^{(2)} \circ \cdots \circ C^{(n)}$. The composition is defined as follows. $\widehat{C} =< \widehat{B}, \widehat{T}, \widehat{b_0}, \widehat{f} >$, where $\widehat{B} = B^{(1)} \times B^{(2)} \times \ldots \times B^{(n)}$. In the composite model, a transition from a composite state $\widehat{b_i}$ to another $\widehat{b_j}$ exists, iff there is at least one transition from one elemental state in $\widehat{b_i}$ to another elemental state in $\widehat{b_j}$. We now define the error model, $\widehat{f}$, in this composite model. At any state in $\widehat{B}$, the error measurement is a composition of the error functions of its elemental states, i.e., $\widehat{f}(\widehat{b}) = f^{(1)}(b_{k_1}^{(1)}) \oplus f^{(2)}(b_{k_2}^{(2)}) \oplus \cdots \oplus f^{(n)}(b_{k_n}^{(n)})$ where $\widehat{b} = \left(b_{k_1}^{(1)}, b_{k_2}^{(2)}, \ldots, b_{k_n}^{(n)}\right) \in \widehat{B}$.

Example: Consider a pair of devices working collaboratively as a distributed system and both run two instances of the same approximated version of a program. Let the CDFG be $C = < B, T, b_0, f >$ as depicted in Figure 1(a), where $B = \{1, 2, 3, 4, 5\}$, $T = \{(1,2), (1,3), (2,4), (3,4), (4,5)\}$, $b_0 = 1$. The error function f is defined as $f(4) = 5$ and $f(i : i \in B \backslash 4) = 0$. The statements/module corresponding to the basic block 4 of the program has been approximated and annotated with the expected error value of 5 from the program. The implication of this model of an approximated program is that the expected error in the output of the program is 5%.

For this composite system, we construct a composition of two instances of the same CDFG as $C \circ C$ (shown in Figure 1(b)). Let $C \circ C =< \widehat{B}, \widehat{T}, \widehat{b_0}, \widehat{f} >$, where $\widehat{b_0} = (1, 1)$. We consider the error composition function as additive, defined as $\widehat{f}(i, j) = e_1 + e_2$, where $f(i) = e_1$ and $f(j) = e_2$. Error models with other non-trivial semantics have been extensively studied in the literature. In this paper, we adopt a simple intuitive additive error model to show its effects in approximate programming. As an example of our model, the error attribute for the composed state $(4, 4)$ is 10, while the error attribute of the elemental state 4 in C is defined as 5. The implication of this composed model is as follows. In the composed CDFG, the state with the maximum expected error denotes the maximum expected error of the composed system provided there exists a path to this state from the initial state of the composed CDFG. In this example composition, however, all paths are bounded by a 10% total error.

It is not difficult to see that in a distributed system, where n instances of the same application are running and collaborating, the expected error in an additive model can often be magnified n fold, unless the algorithm inherently avoids/compensates error paths of its collaborating counterparts. In the following section, we present our experiments on the additive error model on the composition of the approximate versions of a multi-robot SLAM algorithm. Through our experiments, we wish to establish that collaborative approximation, if carried out independently, may lead to violations of the expected accuracy in the overall application execution.

4 EXPERIMENTAL SETUP AND RESULTS

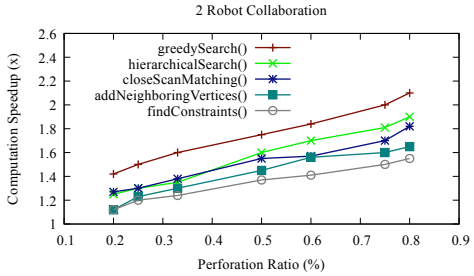
We use the MR-SLAM algorithm [7] available in GitHub[1] as a ROS-package [12] that runs on the stage simulator[2]. Multiple robots can be simulated as different processes. Each simulated robot can be supplied with scan data that simulates its laser scan in a given space. The scan data is also available along with the ROS-package. Our simulation setup uses ROS Indigo running on an Intel® Core™ i5-6500T single-core CPU running at 2.50GHz clock, with 6MB cache and 4GB memory.

Approximation targets. Experimentation with approximate computing requires – identifying the target locations for approximation within code and the extent of approximation. These two related aspects can be defined as an optimization problem in terms of the accuracy of the system. The problem is known to be intractable [14]. In these experiments, we target the part of the code that is most expensive and related to collaboration among robots. The code is profiled for execution time with Valgrind [1]. From the profiled data, we identify the following set of functions that have high computational resource consumption and therefore are good candidates for approximation: `closeScanMatching()`, `greedySearch()`, `hierarchicalSearch()`, `addNeighboringVertices()`, and `findConstraints()`. The `closeScanMatching()` function is used to find common areas among the previously scanned areas and the current scan by the same robot and provides alignment information to merge two scanned areas. The function is re-used when the robot receives scanned data from another in the vicinity and merges the partial maps constructed by the pair. Function `greedySearch()` is used to match scans based on some features of the scanned data and make the scan matching process faster. Function `hierarchicalSearch()` is used in the map merging process and performs on the graph-based model of the map. Function `addNeighboringVertices()` is used for inserting a node in the graph model of the area based on the scanned data. Function `findConstraints()` determines whether a scan has any potential to be close to another scan data. This is used as an entry-level filter for the scan matching function.

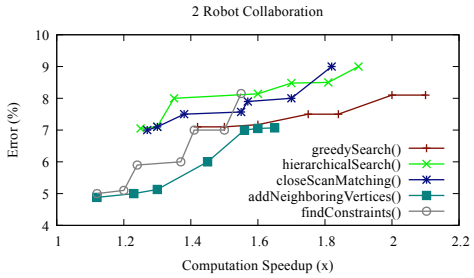
We employ the loop perforation technique [19] to infuse inexactness in code. In our experiments, the top-most loop of the aforementioned functions is perforated in several perforation ratios (number of times the loop runs in the approximated code versus the original one) and they are 0.2, 0.25, 0.33, 0.5, 0.6, 0.75, and 0.8.

During experiments, two or more robots are simulated with the objective that they explore the space simultaneously and collaboratively to build a composite map of the region. We report the amount of savings in terms of computation along with the error in the map image generated by the individual robots. The error is calculated as the image difference against the one generated by the exact version. The image difference is calculated by the method of Jaccard's similarity measure [6], which is a statistical measure of (dis)similarity between sample sets. In the context of image (dis)similarity, the measure is the fraction of the image that is (dis)similar with respect to the basis image.

Approximation results. First, we experiment with two robot collaborative setup. Both the exact and the approximate versions of MR-SLAM are used for guiding the robot in the space one after another. The approximation of the application is achieved by perforating the top-most loop in different degrees as detailed above. In this experiment, we only perforate one function at a time. Figure 2(a) shows the speedup obtained for perforations for each function. As expected, as the perforation amount increases, the code runs faster. Figure 2(b) shows the compromise in quality in terms of the error in the map generated by the robots. The result follows the expected trend that both speed-up and error increase as the perforation ratio increases. Figure 2(a) shows that the function `greedySearch()` is the most effective for approximation and shows a consistent behaviour with varying perforation. The error-performance profile of the function in Figure 2(b) also shows consistent behavior. The approximation of this function produces a higher speedup with lower error as compared to any other function examined in this experiment.

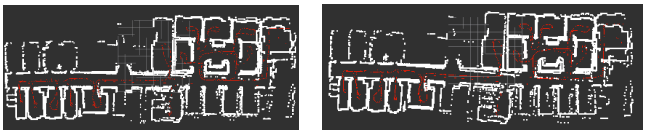


(a) Perforation vs speedup profile



(b) Error profile

Figure 2: Approximation profile for perforation in a single function in a 2-robot system.

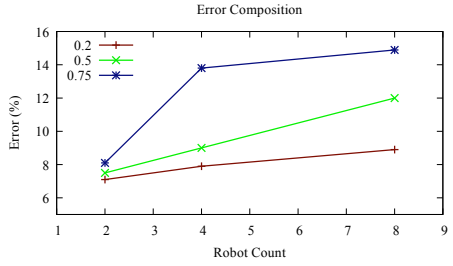


(a) exact MR-SLAM (b) approximate MR-SLAM

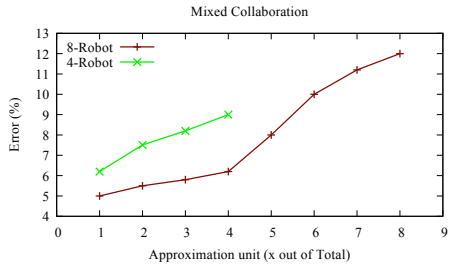
Figure 3: Approximate maps generated in the 2-robot system.

Figure 3(b) depicts the approximate version of the map in contrast to the map generated by the exact version of the application (Figure 3(a)), where the function `greedySearch()` has been perforated with perforation ratio of 0.5. The approximate version results in a 2x speed-up in this case. By the similarity measure, the error is below 10% for all the generated maps. In the corridor area, the generated maps are slightly curved due to imperfection in estimating the location and pose by the individual robots and imperfection in the alignment of the maps during map-merging. Naturally, the effect is magnified in the map generated by the approximate version.

[1] https://github.com/mtlazaro/cg_mrslam
[2] http://playerstage.sourceforge.net/



(a) Under approximation of function `greedySearch()`



(b) Approx-exact mixed robots

Figure 4: Degradation patterns of output.

However, these imperfections do not impart any overwhelming distorting effect. The basic structure of the obstacles, as captured, is the same in both the exact and approximate versions. Moreover, the SLAM algorithm is inherently robust against minor errors from sensor data and its perception about the organization of the space.

Further, we experiment with the collaborative setup of more number of robots (4 and 8). Since a higher number of agents in collaboration implies higher communication overhead and we apply approximation in computation alone, the overall speed-up factor decreases with respect to the total run-time. Result shows, the `greedySearch()` function is still effective as an approximation target. However, the overall error in the map increases in comparison with the two-robot system in both the cases, which is expected from the program composition model described in Section 3. Figure 4(a) shows the pattern of degradation of output as increasing number of robots participate in the collaboration. Each curve represents one perforation ratio (PR) induced in the function `greedySearch()`. This figure shows the shape of the error composition function ($\widehat{f}$). The composition function in this case, although is not strictly linear, but is very close to one. At-least at perforation ratio of 0.5 the error composition seems to behave linearly. Figure 4(b) shows error variation when a fraction of the robots are approximated (function `greedySearch()` by PR = 0.5), while the other robots run an exact version of the application. As the number of robots, running an inexact version of the SLAM code, increases in the fleet, the amount of error infused in the collaborative map increases, which by our model is expected to increase. This figure also presents an idea of error composition function and can be approximated to be a linear function.

5 CONCLUSION AND FUTURE WORK

We study the behavior of approximate computing on the MR-SLAM algorithm in the context of the multi-robot system. We present a model for approximation in the collaborative system. Our experiments uphold the model and show that the approximation error increases as the number of collaborators increases in the system. Also, there are some specific execution paths in which the error magnifies. To avoid such paths, in order to reduce error in the overall outcome, the agents in the system need to coordinate and hence communicate more. Since communication cost for mobile robots is significantly less than the motion cost, this would not impact negatively on the overall efficiency of the system. Our future work includes modeling such interaction and optimizing the trade-off between communication and error in the outcome of the collaborative system.

REFERENCES

[1] 2018. *Valgrind*. http://www.valgrind.org/

[2] Hrishav Bakul Barua and Kartick Chandra Mondal. 2019. Approximate Computing: A Survey of Recent TrendsâĂŤBringing Greenness to Computing and Communication. *Journal of The Institution of Engineers (India): Series B* (2019).

[3] S. Cheemalavagu, P. Korkmaz, and K. V. Palem. 2004. Ultra low-energy computing via probabilistic algorithms and devices: CMOS device primitives and the energy-probability relationship. In *Proc. of The Intl. Conf. on Solid State Devices and Materials*. 402–403.

[4] Isaac Deutsch, Ming Liu, and Roland Siegwart. 2016. A framework for multi-robot pose graph SLAM. In *IEEE Intl. Conf. on Real-time Computing and Robotics (RCAR)*. IEEE, 567–572.

[5] Christian Forster, Simon Lynen, Laurent Kneip, and Davide Scaramuzza. 2013. Collaborative monocular slam with multiple micro aerial vehicles. In *2013 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. 3963–3970.

[6] Paul Jaccard. 1901. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull Soc Vaudoise Sci Nat* 37 (1901), 547–579.

[7] M. T. Lazaro, L. M. Paz, P. Pinies, J. A. Castellanos, and G. Grisetti. 2013. Multi-robot SLAM using condensed measurements. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 1069–1076.

[8] S. Mittal. 2016. A survey of techniques for approximate computing. *Comput. Surveys* 48, 4 (2016), 62.

[9] J. Oh, J. Choi, G. C. Januario, and K. Gopalakrishnan. 2016. Energy-Efficient Simultaneous Localization and Mapping via Compounded Approximate Computing. In *2016 IEEE Intl. Workshop on Signal Processing Systems (SiPS)*. 51–56.

[10] J. Park, H. Esmaeilzadeh, X. Zhang, M. Naik, and W. Harris. 2015. Flexjava: Language support for safe and modular approximate programming. In *Proc. 10th Joint Meeting on Foundations of Software Engg.* ACM, 745–757.

[11] R Venkatesha Prasad, Chayan Sarkar, Vijay S Rao, A Rahim Biswas, and Ignas Niemegeers. 2012. Opportunistic service provisioning in the future internet using cognitive service approximation. In *28th WWRF Meeting, Athens, Greece*.

[12] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and Andrew Y. Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

[13] A. Reiter, H. Gattringer, and A. Müller. 2017. Real-Time Computation of Inexact Minimum-Energy Trajectories Using Parametric Sensitivities. In *Intl. Conf. on Robotics in Alpe-Adria Danube Region*. Springer, 174–182.

[14] A. Sampson. 2015. *Hardware and Software for Approximate Computing*. Ph.D. Dissertation.

[15] Adrian Sampson, André Baixo, Benjamin Ransford, Thierry Moreau, Joshua Yip, Luis Ceze, and Mark Oskin. 2015. Accept: A programmer-guided compiler framework for practical approximate computing. *University of Washington Technical Report UW-CSE-15-01* 1, 2 (2015).

[16] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. 2011. EnerJ: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 164–174.

[17] Chayan Sarkar, Vijay S Rao, R Venkatesha Prasad, Abdur Rahim, and Ignas Niemegeers. 2012. A distributed model for approximate service provisioning in internet of things. In *Proceedings of the 2012 international workshop on Self-aware internet of things*. ACM, 31–36.

[18] Patrik Schmuck and Margarita Chli. 2017. Multi-uav collaborative monocular slam. In *2017 IEEE Intl. Conf. on Robotics and Automation (ICRA)*. IEEE.

[19] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. 2011. Managing performance vs. accuracy trade-offs with loop perforation. In *Proc. of 19th ACM SIGSOFT Symp. and 13th European conf. on Foundations of Software Engg.*

[20] S. Thrun. 2002. Probabilistic robotics. *Commun. ACM* 45, 3 (2002), 52–57.

[21] Danping Zou and Ping Tan. 2013. Coslam: Collaborative visual slam in dynamic environments. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 35, 2 (2013), 354–366.