Sensitivity Analysis For Building Evolving & Adaptive Robotic Software

Prasad Kawthekar **Christian Kästner**
Carnegie Mellon University

Abstract

There has been a considerable growth in research and development of service robots in recent years. For deployment in diverse environment conditions for a wide range of service tasks, novel features and algorithms are developed and existing ones undergo change. However, developing and evolving the robot software requires making and revising many design decisions that can affect the quality of performance of the robots and that are non-trivial to reason about intuitively because of interactions among them. We propose to use sensitivity analysis to build models of the quality of performance to the different design decisions to ease design and evolution. Moreover, we envision these models to be used for run-time adaptation in response to changing goals or environment conditions. Constructing these models is challenging due to the exponential size of the decision space. We build on previous work on performance influence models of highly-configurable software systems using a machine-learning-based approach to construct influence models for robotic software.

1 Introduction

Designing robotic software requires making many design decisions. These software-level design decisions (or options) can impact the quality of performance of the robots. Decision making in practice is often guided by a mix of intuition, experience, and some experimentation. However, such informal techniques may not be reliable and can lead to suboptimal quality of performance. This can be a serious issue for service robots, as they can often work in close proximity to humans and thus need to operate under stringent requirements on performance such as safety and reliability. For instance, poor decisions may cause service robots to run out of battery power while delivering tours to visitors.

Therefore, a (formal) technique to guide good software-level design choices that can ensure high quality of performance in robots is desirable. Such a technique can also speed up *evolution* of robotic software by helping revisions and decision making involved in the integration of new features and algorithms with the existing platforms, thereby allowing researchers to devote more time to the creative process of designing new algorithms than on fine-tuning robotic algorithms.

Software-level design choices are not the only factors that affect the quality of performance of a robot. Changes in the robot's environment or in its high-level intents may also significantly affect the robot's quality of performance. However, negative effects can possibly be mitigated by reconfiguring the robot's software at run-time (by reconsidering its design choices) in response to the external changes. Such adaptation techniques are important to service robots, as they are increasingly deployed in dynamic environments to perform tasks with diverse performance requirements.

All these use cases, from initial decision making to evolution and runtime adaptation require making 'good' design decisions at the software level. However, searching for good design choices (or a good software configuration) is not straightforward. The brute-force technique of iterating over all possible configurations to select the best performing one is infeasible in practice, because the search space of possible configurations is exponential in the number of design decisions. Multiple design decisions can interact with each other to determine the quality of performance, such that their combined effect is different from the effect of changing them one at a time.

This size of the search space motivates a machine-learning strategy for *sensitivity analysis* [Saltelli *et al.*, 2000] of robotic software, sampling a few configurations from the search space to predict the quality attributes for the remaining configurations. A function learned in the process describes the sensitivity or influence of different design decisions on quality attributes, which can be used to guide optimization, evolution, and runtime adaptation, as well as, for understanding and debugging the software implementation of robotic algorithms.

We build on previous work on using machine learning for building influence models of highly-configurable software systems [Siegmund *et al.*, 2015]. We demonstrate feasibility by applying this approach to the robotics domain with a preliminary analysis of decisions in the simulator of the CoBot robotic platform [Veloso *et al.*, 2015]. Specifically, we focus our preliminary analysis on a subset of design choices in the particle-filter based autonomous localization component of the CoBot software [Biswas *et al.*, 2011; Biswas and Veloso, 2013]. We demonstrate the possibility of building evolving and adaptive robotic software by discussing tradeoffs in the quality requirements for the localization mod-

ule and the possibility of countering changes in the external environment by reconfiguring the module; the necessity for machine learning due to the exponential configuration space of the module; and a simple linear-regression based influence model for the localization module.

2 Motivation

A robot's performance requirements for completing a task are defined by its high-level goals and intents. The robot's fidelity to its high-level goals can be measured using appropriately defined quality attributes. For instance, the high-level goals of an autonomous robot while performing a navigation task would be to complete the task as fast as possible while using minimal processing power (corresponding to longer battery life). The robot's quality of performing the navigation task can thus be measured in terms of the total time to complete the task, its total power consumption, and the accuracy of its localization during the task. Optimizing these quality measures would thus enable the robot to more effectively achieve its high-level goals.

The software of a robot implements algorithms that enable the robot to achieve its goals, such as state estimation and planning. The quality of the algorithms and their implementation is thus responsible for determining the quality of performance of the robot. Designing this software involves many choices, which can manifest as constants or variables in the source code and can range in granularity, from selecting between different types of algorithms for a task to selecting values for some algorithm's parameters. Consider the particle filter based localization algorithm of CoBots [Biswas *et al.*, 2011; Biswas and Veloso, 2013]: Implementing this algorithm requires making a choice for the number of particle estimates and the number of gradient descent refinement steps applied to each of these particles on sensor updates. Different values for these options can impact the quality of performance of a robot. For example, in Figure 1a and Figure 1b, we show that different decisions for these two parameters can have a significant effect on the average localization error (the euclidean distance between the true location and the estimated location) and average CPU utilization of the robot in a sample navigation task.

An influence model that describes the effects (or sensitivity) of different decisions on the quality of performance can be useful in various stages of software development:

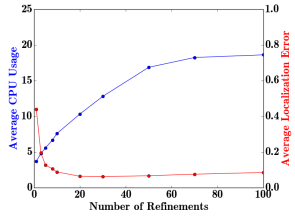
- **Evolving Functions and Goals.** Developing software is an iterative process. Novel features are developed over time and added to the software. Evolution is common in rapidly evolving fields such as service robots. For example, new sensors might be added, or newly developed algorithms might be integrated with the existing software. Implementing new features involves revising prior decisions at the software level as well as making new ones. Influence model based optimization can help to accurately *bootstrap* new features by using empirical evidence to make appropriate design decisions.

  The high-level goals and intents of the robot may also be updated as the robot is deployed in new environments. Changes in the high-level goals can lead to conflicting optimization problems for different performance requirements of the robot. It is also possible that adding new features or algorithms introduces tradeoffs in the performance requirements of the robot, which can cause the previously selected design choices to not be suitable for the new performance requirements. For example, Figure 1a and Figure 1b illustrate how the configuration space defined by the number of particles and number of refinements in the localization module exposes a tradeoff between localization error and CPU utilization: Increasing the number of particles (resp. refinements) reduces the localization error (up to a threshold), but it comes at the expense of higher CPU utilization. To handle tradeoffs between different performance requirements, different influence models can be learned for each corresponding quality attributes and then used together to guide the *(re)configuration* of the robotic software.

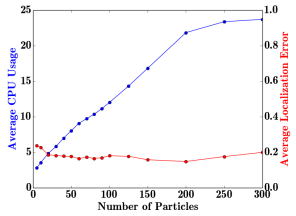
- **Runtime Adapting to Change.** Influence models can also be used to build robotic software that can purposefully *adapt* to changes at runtime. Both environment conditions and performance requirements can vary at runtime depending on the task assigned to the robot. For adaptation, the robot can use a learned influence model to tweak its software configuration options and move to a more optimal point in the configuration space. For instance, our experiments in Figure 2 indicate that an increase in the number of refinements can lead to lower localization error in the face of higher odometry noise and miscalibration, at the expense of higher CPU utilization. At a more coarse granularity, the robot could adapt by switching between different types of robotic localization algorithms altogether, each designed for slightly different optimization problems and potentially better suited for some environments than others. Rather than implementing a single localization algorithm in the robot's software, multiple algorithms can be implemented and the choice between them be left open depending on the environment conditions. Such choice can be incorporated in influence models as a single variable. Quality measures are likely to be more sensitive to coarser configuration options than fine-grained ones, thereby implementing coarser configurations can open up avenues for greater adaptability and larger gains in optimization. However, these gains could come at the expense of higher development costs for implementing coarser configuration choices.

  It is also worth noting that software design choices can control some aspects of hardware design as well, since the software implements the interface of a robot's algorithms to its sensors and actuators. For instance, a robot's hardware may contain multiple (possibly redundant) sensors that are used for localization (such as Kinect, Lidar etc.). Under different environment conditions, such as changing sensor noise, the robot can increase or decrease the number of its active sensors to ensure high localization accuracy, while simultaneously trying to minimize the power consumption by the sensors. This kind of adaptation can be made possible by including the (boolean) choice of activating a sensor in the robot's influence model.

- **Understanding and Debugging Software.** The influence model for different performance requirements can

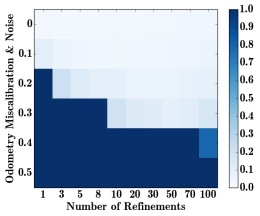


(a) Varying number of refinements and constant number of particles (20).

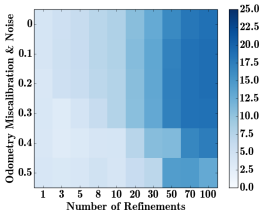


(b) Varying number of particles and constant number of refinements (3).

Figure 1: *Performance tradeoffs in the localization algorithm.* The two y-axis measure the average localization error (in meters) and average CPU time consumed by the robot's software (in % for a quad-core machine) for a sample navigation task of the CoBot in a simulator. A noise model is applied to the odometry for the purpose of these experiments (uniform distribution with 20% standard deviation around mean corresponding to 20% miscalibration). Results averaged over 5 iterations.



(a) Average localization error for combinations of odometry noise models and number of refinements.



(b) Average CPU utilization for combinations of odometry noise models and number of refinements.

Figure 2: *Effect of number of refinements on localization error due to odometry noise.* The heatmaps measure the average CPU utilization and average localization error for a sample navigation task in the CoBot simulator, with darker shades corresponding to higher values. y-axis represents different odometry noise models with equal values for noise and miscalibration for visualization purposes. Results averaged over 5 iterations.

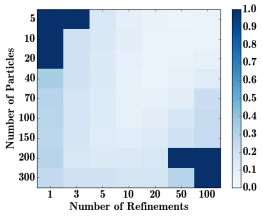
also be used by the robot software developer to test the implementation of algorithms in the software by comparing the expected (analytical) model, if it is known, to the observed (empirical) model. For instance, an analytical model to adapt the number of particles in response to localization uncertainty has been developed [Fox, 2002]. The implementation of this adaptation strategy in software can be tested in practice by comparing it with the observed adaptation using the empirically learned influence model. In addition, influence models are useful to discover and understand interactions among choices.

3 Challenges

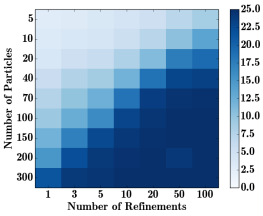
Sensitivity analysis of software configuration options on quality of performance of robots can be used to build influence models that are useful for offline optimization, online adaptation, and debugging software. However, building these influence models is challenging due to the following two reasons:

- **Costly Measurements.** Measuring the performance of the robot for some task with some software configuration and environment conditions requires running the robot physically or in a simulator. Even parallelizing measurements, the long running times can limit the number of measurements that are feasible in practice.
- **Large Configuration Space.** In practice, there are likely to be several hundreds of configuration options in robotic software systems. Many of these configurations options, including the number of particles and refinements, can have large numeric ranges. In addition, the options can interact, resulting in an exponentially growing configuration space. Two options interact if the effect of changing both options together is different from the expected effect of changing both options individually. This means that effects of changes in combinations of configuration options need to be measured and modeled just as effects of changes in individual configuration options.

  For instance, the number of particles and number of refinements interact in the localization algorithm. As visible in Figure 3b and Figure 3a, there are combinations of decisions regarding the number of particles and number of refinements that outperform the individual decisions observed in Figure 1a and Figure 1b. Understanding this interaction can lead to better overall decisions, but it requires exploring an exponential search space.



(a) Average localization error for combinations of number of refinements and number of particles.



(b) Average CPU utilization for combinations of number of refinements and number of particles.

Figure 3: *Interactions between parameters of localization algorithm.* The heatmaps measure the average CPU utilization and average localization error for a sample navigation task in the CoBot simulator, with darker shades corresponding to higher values. Same noise model as in Figure 1. Results averaged over 5 iterations.

4 Approach & Preliminary Results

To overcome the challenge of time-consuming and large-scale measurements, we suggest to use sensitivity analysis, based on supervised machine learning, to learn influence models.

We learn influence models using iterative linear regression, following a method originally developed for highly-configurable software systems [Siegmund *et al.*, 2015]. It uses feature selection, starting with the base set of configuration options, iteratively adding interactions of configurations in successive steps in order to learn interactions and non-linear effects on the quality metrics. For details regarding the learning procedure, we refer the interested reader to the original publication [Siegmund *et al.*, 2015]. We use linear regression for our preliminary experiments, because linear models are easy to understand and can intuitively explain the effects of individual configuration options and interactions on quality metrics, which aids debugging. A linear regression model is also useful as it can easily incorporate domain knowledge (such as from known analytical models) during learning.

To illustrate feasibility, we learned influence models after measuring 3000 different configurations of the CoBot's localization algorithm, varying 9 numeric parameters, including odometry noise and miscalibration, laser noise and miscalibration, odometry and laser update frequency, number of particles and refinements, and maximum robot velocity. We learned models for 5 quality attributes, including CPU utilization, wall

clock time to completion, laser and odometry update processing time, and localization error. The following is an excerpt from a learned influence model that describes part of the variation seen regarding CPU utilization:

$$\begin{aligned} cpu\_utilization = 19.20 &+ 9.051 * odometry\_noise \\ &+ 12.034 * odometry\_miscalibration \\ &+ 0.026 * num\_particles \\ &- 7.32 * 10^{-5} * num\_particles^2 \\ &+ 0.063 * num\_refinements \\ &- 0.061 * odometry\_noise * num\_ref. \end{aligned} \quad (1)$$

The above model can be read as follows: By plugging in different values for the variables on the right hand side of the equation, we can compute the expected CPU utilization for a given configuration. As expected, we see that CPU utilization increases with an increase in odometry noise and miscalibration and with the number of particles and refinements. The negative factors indicate that this increase is not strictly linear. The above model also indicates an interaction between odometry noise and number of refinements (last term).

Discussion & Future Work. We have not conduct a rigorous analysis of the obtained models yet. We plan to follow up this preliminary analysis with additional validation, discussions with domain experts, more accurate and larger scale measurements, and more refined machine learning techniques.

Our current approach uses linear regression in order to generate a model that can intuitively explain the expected effect of different configuration options on the robot's quality of performance. Machine learning can be combined with search-based strategies for learning potentially more accurate models with fewer measurements. Active machine learning can also be a viable option in this direction. All the preliminary results presented in this paper are based on measurements conducted offline on a simulator. Eventually, we hope to integrate online measurements on the physical robot.

Once we obtain an accurate influence model based on these large-scale measurements, we plan to integrate it with optimization strategies to obtain high performing configurations for the software. This optimization strategy can also be moved online to enable reasoning about runtime adaptation in response to environment changes.

5 Related Work

Our work directly builds on the approach of using machine learning for sensitivity analysis of software configurations, specifically using the tool SPLConqueror [Siegmund *et al.*, 2015]. While their approach uses linear regression to learn the influence models, other machine learning models such as Classification and Regression Trees have also been used in this context [Guo *et al.*, 2013]. We decided to use the former approach for our preliminary analysis since linear models are easy to understand and the learning can incorporate domain knowledge about expected relationships. However, we note that the approach we describe for building evolving and adaptive robotic software is agnostic the exact choice of machine learning model used. Learning performance models of software configuration options on different non-functional properties of the software has been applied on various real-world applications [Thereska *et al.*, 2010; Kwon *et al.*, 2013; Hoffmann *et al.*, 2011; Siegmund *et al.*, 2015].

For the more specific goal of *optimization*, both optimizing only for performance and multi-objective optimization, several approaches have been explored, typically using search-based techniques to find good parameter combinations. Approaches include combining seeding with evolutionary learning [Sayyad *et al.*, 2013] and combining that with constraint satisfaction solving [Henard *et al.*, 2015], hill-climbing [Gratch and Dejong, 1992], and genetic algorithms [Terashima-Marn and Ross, 1999]. A general discussion on the merits of optimizing software using configuration options is also available [Hoos, 2012]. While optimization techniques are suitable to (and potentially more effective at) find good settings, they are less suitable for facilitating understanding and adaptation, as they do not actually attempt to identify the influence of individual decisions or their interactions.

Similar to optimization, several researchers have explored approaches to automatically calibrate vision sensors [Li and Chen, 2003] and odometry [Roy and Thrun, 1999]. Our work is more generic, but potentially also less precise than such more specialized solutions. Considerable amount of effort has been devoted to building resource-efficient localization algorithms, by reducing the number of particles [Biswas *et al.*, 2011; Fox, 2002]. Also analytical performance models are available for some parameters [Fox, 2002]. In contrast, our approach is agnostic to the implementation and applicable to a large range of options and measurable quality attributes.

Self-adaptive systems have received significant attention in the software engineering community. These approaches usually involve adaptation at the software architecture level [Garlan *et al.*, 2004; Salehie and Tahvildari, 2009]. Architecture based techniques have also been applied to the domain of robotics [Ingls-Romero *et al.*, 2011]. Approaches to learning based self-adaptation using configuration options has also received attention in literature [Elkhodary *et al.*, 2010]. The influence models that we learn can be used to guide the adaptation strategy using the different approaches above.

6 Conclusion

We proposed to build influence models that explain how design decisions and their interactions affect the quality of performance of robotic software based on sensitivity analysis. We motivated how such influence models can guide optimization of the software, help debugging and testing during the development, and support adapting the robots to external changes at runtime. We presented preliminary results of applying a machine learning based approach in the context of the CoBot robotic platform and discuss future directions.

7 Acknowledgements

We thank Manuela Veloso for comments on earlier drafts of the paper, Norbert Siegmund and Joydeep Biswas for their helpful discussions on the subject matter, Alexander Grebhahn for his help with measurements, and the anonymous reviewers. This research was supported by the DARPA BRASS project.

References

[Biswas and Veloso, 2013] Joydeep Biswas and Manuela Veloso. Localization and Navigation of the CoBots over Long-term Deployments. *International Journal of Robotics Research*, 32(14):1679–1694, December 2013.

[Biswas *et al.*, 2011] Joydeep Biswas, Brian Coltin, and Manuela Veloso. Corrective gradient refinement for mobile robot localization. In *Proceedings of IROS'11, the IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, CA, September 2011.

[Elkhodary *et al.*, 2010] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. Fusion: A framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 7–16, New York, NY, USA, 2010. ACM.

[Fox, 2002] Dieter Fox. Kld-sampling: Adaptive particle filters and mobile robot localization. In *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[Garlan *et al.*, 2004] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecturebased self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[Gratch and Dejong, 1992] Jonathan Gratch and Gerald Dejong. Composer: A probabilistic solution to the utility problem in speed-up learning. pages 235–240, 1992.

[Guo *et al.*, 2013] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej W asowski. Variability-aware performance prediction:a statistical learning approach. *IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, 2013.

[Henard *et al.*, 2015] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 517–528, Piscataway, NJ, USA, 2015. IEEE Press.

[Hoffmann *et al.*, 2011] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 199–212, New York, NY, USA, 2011. ACM.

[Hoos, 2012] Holger H. Hoos. Programming by optimization. *Commun. ACM*, 55(2):70–80, February 2012.

[Ingls-Romero *et al.*, 2011] Juan F. Ingls-Romero, Cristina Vicente-Chicote, Brice Morin, and Olivier Barais. Towards the automatic generation of self-adaptive robotics software: an experience report. *20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2011.

[Kwon *et al.*, 2013] Yongin Kwon, Sangmin Lee, Hayoon Yi, Donghyun Kwon, Seungjun Yang, Byung-Gon Chun, Ling Huang, Petros Maniatis, Mayur Naik, and Yunheung Paek. Mantis: Automatic performance prediction for smartphone applications. In *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 297–308, San Jose, CA, 2013. USENIX.

[Li and Chen, 2003] Y. F. Li and S. Y. Chen. Automatic recalibration of an active structured light vision system. *IEEE Transactions on Robotics and Automation*, 19(2):259–268, Apr 2003.

[Roy and Thrun, 1999] N. Roy and S. Thrun. Online self-calibration for mobile robots. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 3, pages 2292–2297 vol.3, 1999.

[Salehie and Tahvildari, 2009] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42, May 2009.

[Saltelli *et al.*, 2000] Andrea Saltelli, Karen Chan, and E. Marian Scott. *Online self-calibration for mobile robots*, volume 1. Wiley, New York, 2000.

[Sayyad *et al.*, 2013] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 465–474, Nov 2013.

[Siegmund *et al.*, 2015] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kstner. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 284–294. ACM, August 2015.

[Terashima-Marn and Ross, 1999] Hugo Terashima-Marn and Peter Ross. Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99*, pages 635–642. Morgan Kaufmann, 1999.

[Thereska *et al.*, 2010] Eno Thereska, Bjoern Doebel, Alice X. Zheng, and Peter Nobel. Practical performance models for complex, popular applications. *SIGMETRICS Perform. Eval. Rev.*, 38(1):1–12, June 2010.

[Veloso *et al.*, 2015] Manuela Veloso, Joydeep Biswas, Brian Cotlin, and Stephanie Rosenthal. Cobots: robust symbiotic autonomous mobile service robots. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 4423–4429. AAAI, 2015.