Reuse-oriented SLAM Framework using Software Product Lines

Mohamed A. Abdelhady*, Douwe Dresscher*, and Jan F. Broenink*
*Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente, 7500 AE Enschede, The Netherlands
Emails: m.adel.abdelhady@gmail.com, {d.dresscher, r.cobosmendez, j.f.broenink}@utwente.nl

***Abstract*—Simultaneous Localization and Mapping (SLAM) is a widely investigated problem in robotics. It depicts the process of a robot creating a map of an unknown environment while concurrently estimating its location within the self-created map. In recent years, many solutions have been proposed to the SLAM problem based on numerous approaches such as probabilistic filters or graph optimization. This work recognizes that, with the growing complexity and the active development in the field of SLAM, reuse is becoming an essential quality as researchers often have to solve architectural issues that are secondary to the core of the problem, which leads to sub-optimal realizations in the final SLAM product from the software point of view. Therefore, a *component-based* framework is introduced that regards reusability as a primary requirement of SLAM software, which highlights the core separable modules and implements them as encapsulated interchangeable components forming a *software product line*. The reusability of the framework is evaluated according to the *reuse-readiness levels* criteria and the results show improved modularity and reduction in the development and deployment time and effort.**

I. Introduction

SLAM is a well-defined problem in robotics and photogrammetry, which describes a mobile robot or even a single camera system that navigates into an unknown environment, and builds a map of the surrounding while simultaneously estimating its location and orientation within the created map. This process is essential for autonomous and semi-autonomous mobile systems in order to be able to perceive the environment and act accordingly. Hence, many approaches have been proposed and successfully implemented to solve SLAM under a certain set of limited conditions, such as small indoor static environments [1]. However, it is still an active research area as there are multiple unaddressed challenges such as dynamic environments, large scale mapping, and scene understanding.

As a final product, SLAM can be regarded as a complex software artifact, which is often developed in a research context to showcase the feasibility of a novel algorithm or to improve upon an existing one with no attention to long term reuse and support. The outcome is a monolithic piece of software that is tailored towards a specific application with high dependency on a certain platform or a class of sensors. Consequently, variations of SLAM, being a different algorithm, different sensors, or a different platform, need to be developed with little or no reuse of previous effort. This is in contrast to more established domains that adopt systematic software reuse in order to improve quality, efficiency, and provide customizability with little overhead.

To address the aforementioned points and to be able to comply with such non-functional industry standards, we propose a reuse-oriented framework that decouples the main functional elements of SLAM, and introduces unification to its design and development processes. The proposed framework aims at enhancing the quality of the software, in terms of reusability, by introducing a structured development process and creating a SLAM infrastructure. As well as providing a methodology to assess reusability of SLAM instances. It should be noted that the framework does not consider additional requirements other than SLAM functionality. Therefore, components with real-time constraints or other performance conditions will have limited reusability and will need to be separately addressed.

This paper is organized as follows: in Section II an overview of related work is provided. In Section III the main concepts of the applied reuse-oriented framework and the reusability criteria are introduced. In Section IV the domain analysis of SLAM, which is essential for software product lines, is detailed. In Section V, several building blocks of SLAM are developed and assembled into different instances. Finally, the experiments, discussion, and conclusion, are presented in Sections VI, VII and VIII.

II. Related Work

A common theme in relevant work is the utilization of reuse-oriented software-development principles in robotics. This growing area of research is driven by the need for standardized development methodologies that enable the reuse of large-grained software components, which is demonstrated by the popularity of component-based middlewares such as *ROS* [2] and *OROCOS* [3] [4]. These frameworks are used to deal with the increasing complexity in robotic tasks.

Brugali et al. [5] propose a reuse-oriented motion planning library for robotics applications that utilizes the component-based framework provided by the Robotic Operating System (ROS). Their work shows the added value of utilizing software product lines (SPL) in designing the framework and having a more balanced implementation of the different components with reduced coupling. The proposed development process is based on refactoring open source implementations. Similar work is presented in [6], demonstrating refactoring techniques for perception libraries for robotics. The refactoring procedure includes a domain analysis in order to encompass the

978-1-7281-5237-0/20/$31.00 ©2020 IEEE
DOI 10.1109/IRC.2020.00037

commonly used perception algorithms and data structures. The results yield a perception framework with standalone atomic software components and harmonized interfaces that can easily be used to interchange algorithms and benchmark them. Thus, allowing the developers to decide in an early stage which is the most suitable algorithm for the application and showcasing the significance of variability modeling. In addition, [7] combines the two previously mentioned contributions into a perception software product line for robotics applications. Another SLAM related standardization is also introduced by [8], which proposes a standard map representation that can enable different systems to be able to utilize shared map data.

III. Methodology

In order to create a reuse-oriented SLAM framework, a number of well-established reuse methodologies are utilized from software engineering.

A. Software product lines

The software product line (SPL) paradigm [9] aims at identifying a family of applications in which common parts are distinguished from variable parts. Where, the former is available in every instance of the product family, while the latter can be regarded as add-ons that change with every instance. Hence, allowing for a software stack to be assembled from these common and variable parts in a product-line manner as long as the parts have unified interfaces. In order to apply SPL within the SLAM domain, three main procedures are followed in the workflow, namely, domain analysis, product line development, and product derivation.

B. Reuse-readiness levels

Evaluation metrics are proposed as an integral part of the framework in order to provide a quantitative feedback to judge the reusability aspect. The adopted criteria for reusability are primarily developed by the software working group at *NASA* [10]. They devised Reuse-Readiness Levels (RRLs) that describe 9 essential factors, which influence the reusability aspect of any software product. Within the scope of the work described in this paper, only 3 RRLs are considered during the design, development, and evaluation, namely, Modularity, Extensibility, and Standard compliance.

The chosen reusability criteria are considered to be of higher priority as their impact would be more significant compared to the rest of the RRLs.

IV. Domain Analysis

Domain analysis of SLAM is performed in order to capture the commonly used functionalities and algorithms. The result of the domain analysis is encoded in the form of a feature model, which is a graphical representation that shows the different decision points in the design of a SLAM software. The *HyperFlex* toolchain [11] is used in order to create the feature model for the SLAM framework.

A number of different approaches to SLAM can be found, such as the commonly used implementations presented in [12], [13], [14], [15], [16], which can be seen as products belonging to the same SLAM product line family. Through closer inspection, a number of major concepts can be recognized with respect to the software implementation. The details of each major concept are presented hereafter and the generated feature model used in this work is presented in Figure 1.

A. Idiothetic Information Handler

Idiothetic information handler is the class of software components concerned with processing interoceptive sensors to estimate incremental pose measurements, which is also known as dead-reckoning [17]. It is used in combination with internal state sensors such as wheel encoders, gyroscopes, compasses, etc. However, this class of components can also be used with cameras or other exteroceptive sensors in order to perform visual odometry by estimating the transformation between two consecutive measurements. Although the raw data of the sensors is processed differently according to the type of the sensor, the provided interface can be standardized as a vector of incremental pose measurements.

B. Allothetic Information Handler

Allothetic information handler is the class of software components responsible for processing the measurements of exteroceptive sensors such as cameras, laser scanners, stereo cameras, sonar sensors, etc. The information provided by the allothetic sensors are complementary to the dead-reckoning approach as they are used to update the map and the pose given an observation of the environment. The interface of the allothetic handler can provide aggregated information or the actual raw sensor measurements to the inference back-end.

C. Inference Back-end

Inference back-end is the software component that incorporates the idiothetic and allothetic information and maintains an estimate of the pose and the map. As the core of SLAM, there are different approaches to achieve the estimation, such as probabilistic filters (EKF, UKF, PF, etc.) [17] that perform recursive prediction and correction steps, optimization and graph based techniques [18], and artificial neural networks [15]. The back-end estimates the posterior probability distribution $p(x_t, m|z_t, u_t, x_{t-1})$, where x_t and x_{t-1} are the current and previous pose estimates respectively and m is the created map [17].

The inference of the pose and the map can be considered as a standalone problem, therefore, it can be tackled in a sensor-agnostic fashion. Furthermore, different map representations (feature maps, volumetric maps, topological maps, etc.) are directly linked to the choice of the back-end, which can be encoded as constraints in the feature model.

D. Platform-Specific Parts

Platform parts are models that represent platform-specific information as they describe motion models, sensor models, observation models, and noise models, which differ according

to the system configuration, sensor characteristics, etc. These models are often implicitly included into each of the previously mentioned concepts. However, the platform parts can be made to be interchangeable and shared among different approaches. For example, the odometry motion model can be used in combination with and EKF or a particle filter inference back-end [17].

V. Product Line Development & Derivation

In this section, the process of developing the components forming the SLAM product line is described, as well as the derivation of multiple instances by assembling these components together.

A. Product Line Development

Following the domain analysis, a number of components are implemented as software components as highlighted in the feature model in Figure 1. Components belonging to the same type (allothetic handler, idiothetic handler, inference back-end, platform-specific parts) are developed such that they provide and depend on the same interfaces.

B. Product Derivation

The derivation of the SLAM products is achieved by assembling different variations of the 4 essential components as shown in Table I. The different combinations are chosen based on the most commonly used SLAM realizations, namely, feature-based EKF and particle filter.

#	Idiothetic	Allothetic	Inference	Platform Parts
1	Wheel encoders	Kinect - Fiducial detection	EKF	Odometry - Intrinsic camera
2	Visual odometry	Kinect - Camera - Fiducial detection	EKF	Velocity - Intrinsic camera
3	Visual odometry	Lidar - Feature extraction & matching	Particle filter	Odometry - Likelihood field
4	Wheel encoders	Kinect - Fiducial detection	Particle filter	Odometry - Intrinsic camera
5	Visual odometry	Kinect - Camera - Fiducial detection	Particle filter	Odometry - Intrinsic camera

TABLE I: A list of the assembled SLAM instances showing the selection for each of the essential components.

VI. Evaluation

The derived instances are deployed onto an actual mobile platform in an indoor environment to consolidate the entire development life cycle, where, each component is executed as a standalone *ROS* node and the different instances are configured through *launch* files. The output map and trajectories generated by each of the instances described in Table I are consistent with the actual trajectory traversed by the robot, without the need for an explicit loop closure component.

The changes needed to create a different SLAM instance based on another instance are shown in Figure 2. For example, in one of the transitions the inference back-end is switched from an EKF to a particle filter, and the result is an operational SLAM with no modifications required to the rest of the components. Similarly, only a single component is impacted by the replacement of the wheel encoders with visual odometry, or the addition of a laser scanner. A summary of the required changes for all transitions is summarized in Table II.

#	1	2	3	4	5
1	-	{1,0,1}	{3,1,3}	{0,0,1}	{1,0,2}
2		-	{1,0,2}	{0,0,2}	{0,0,1}
3			-	{0,0,2}	{0,0,2}
4				-	{0,0,1}

TABLE II: The required changes in order to transition from one instance to another. The numbers encode {Addition, Modification, Replacement} of the different components.

VII. Discussion

Observing the results of the evaluation, it is shown that the different realizations of SLAM exhibits locality of change. This is reflected in Figure 2 and Table II, where, the transition from one instance to another only affects the components that are changed. Additionally, the derived products in Table I can be easily extended to include graph-based instances that work with the same allothetic and idiothetic interfaces as they can be interpreted as measurement constraints needed for optimization-based SLAM. Thus, the framework allows for modifications with a minimal impact on the software as it does not need to be re-designed and developed from scratch for every new instance. Such locality of change in the software indicates enhanced modularity and extensibility, which are identified as target RRLs.

Furthermore, the number of different realizations created shows that combining different components into a working SLAM instance is simplified by the standardization of the interfaces and the separation of concerns.

VIII. Conclusions

This work presents a software product line, which is designed in order to improve the reusability of SLAM. Accordingly, the domain variations that characterize SLAM are analyzed and a feature model is created that highlights the different implementations and algorithms used in the domain of SLAM. The developed feature model provides a structured way to make design choices and validate them early in the development life cycle.

Furthermore, the clear segregation among different functionalities enables various components to be developed and tested in separation, in contrast with monolithic architectures. In addition, standardization allowed for a minimal integration efforts as well as improved interoperability of arbitrary information sources be it, idiothetic sensors, or allothetic sensors, as well as the algorithms used as the back-end of SLAM.

Finally, the presented framework can be further extended with additional features for each of the main components to generate higher resolution maps, more accurate trajectories, and generalize to larger environments. And refactoring techniques can also be used to adapt widely used implementations into the framework for better standardization.

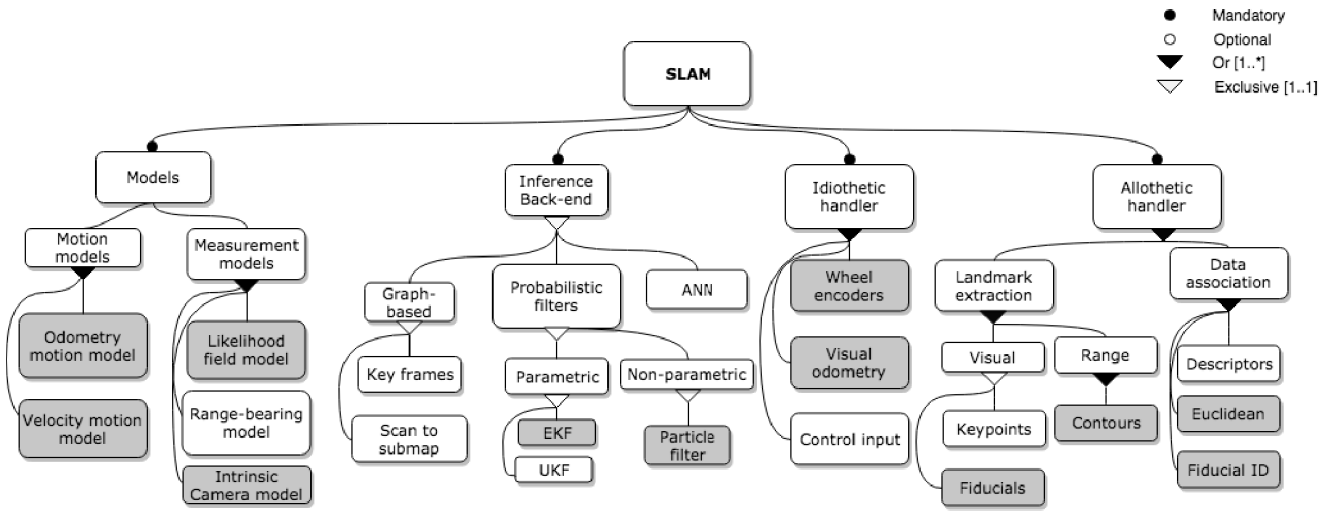

Fig. 1: A feature model for a SLAM software product line. The model has 4 essential components, where variations for each of the functionalities are listed with their corresponding cardinality. A number of highlighted components indicate the ones implemented within the scope of this work.

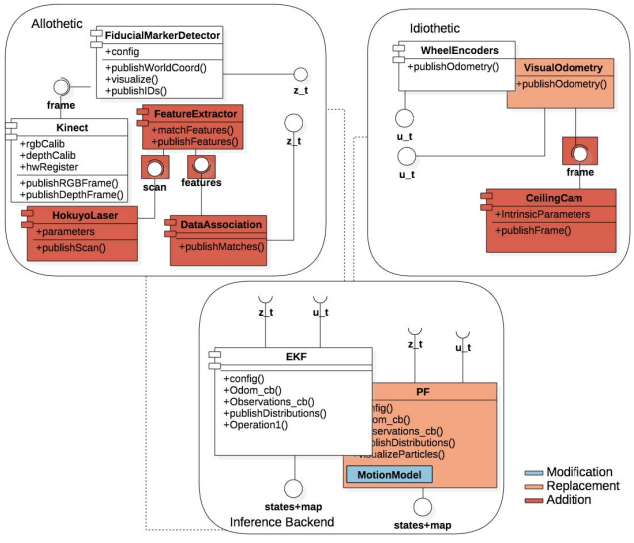

Fig. 2: Superimposed component diagrams of the deployed SLAM instances showing instance 1 as the base configuration, as well as the subsequent modification and addition of different components.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard, "Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, p. 1309–1332, 2016.

[2] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[3] H. Bruyninckx, "Open robot control software: the OROCOS project," in *Robotics and Automation (ICRA), 2001. IEEE International Conference on*, vol. 3. IEEE, 2001, pp. 2523 – 2528.

[4] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The BRICS component model: a model-based development paradigm for complex robotics software systems," *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 1758–1764, 2013.

[5] D. Brugali, L. Gherardi, A. Biziak, A. Luzzana, and A. Zakharov, "A reuse-oriented development process for component-based robotic systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7628 LNAI, pp. 361–374, 2012.

[6] W. Nowak, S. Blumenthal, and E. Prassler, "Framework for identification and re-factoring of best practice algorithms in robotics," 2013.

[7] D. Brugali and N. Hochgeschwender, "Software product line engineering for robotic perception systems," *International Journal of Semantic Computing*, vol. 12, no. 01, pp. 89–107, 2018.

[8] F. Amigoni, W. Yu, and D. Holz, "Ieee 1873-2015: A standard for map data representation," 2018.

[9] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Heidelberg: Springer-Verlag, 2005.

[10] J. Marshall, S. Berrick, and a. Bertolli, "Reuse Readiness Levels (RRLs)," *Science Data Systems*, 2010.

[11] J. Badger, D. Gooding, K. Ensley, K. Hambuchen, and A. Thackston, *HyperFlex: A Model Driven Toolchain for Designing and Configuring Software Control Systems for Autonomous Robots*, ser. Studies in Computational Intelligence, A. Koubaa, Ed. Cham: Springer International Publishing, 2016, vol. 625, no. January.

[12] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[13] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, "Fast and accurate slam with rao-blackwellized particle filters," *Robot. Auton. Syst.*, vol. 55, no. 1, pp. 30–38, Jan. 2007.

[14] M. J. M. M. Mur-Artal, Raúl and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[15] D. Ball, S. Heath, J. Wiles, G. Wyeth, P. Corke, and M. Milford, "OpenRatSLAM: An open source brain-based SLAM system," *Autonomous Robots*, vol. 34, no. 3, pp. 149–176, 2013.

[16] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[17] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[18] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.