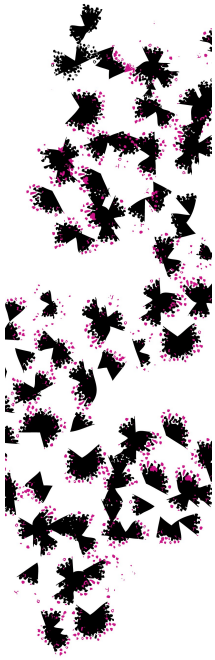


Reuse-oriented SLAM Framework using Component-based Approach

M.A. (Mohamed) Abdelhady

MSc Report

Committee:
Prof.dr.ir. S. Stramigioli
Dr.ir. D. Dresscher
Dr.ir. J.F. Broenink
Dr. M Poel

August 2017

040RAM2017
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

Simultaneous Localization and Mapping also known as *SLAM* is a well established problem in the scientific community and considered to be solved for a specific combination of environment, resources and requirements. It depicts the process of a robot creating a map of an unknown environment while simultaneously estimating its location within the self-created map. Currently, numerous SLAM algorithms have been successfully developed and applied to a multitude of applications using different platforms. The motivation for this work is the lack of a formal development methodology for SLAM and the tight coupling of the constituting sub-modules of the software artifacts, which reflects poorly on code reusability. Henceforth, a reuse-oriented framework is sought that aims at decoupling SLAM and reducing the development and deployment time. The framework is built using *component-based software development* approaches which is utilized to encapsulate the different parts in SLAM as software components and enforce the separation of concerns.

In order to validate the reusability of the developed infrastructure, *Architecture Trade-off Analysis Method* (*ATAM*) will be used to derive a number of scenarios addressing the reusability criteria based on the *reuse-readiness levels* (*RRLs*). The evaluation indicates that the proposed methodology facilitates rapid deployment of SLAM realizations with different algorithms, sensors and map representations with minimal modifications to the constituting components. Thus, contributing positively to the reusability of SLAM and offering a starting point towards a more mature SLAM from the software quality point of view.

Preface

This report constitutes as a documentation for my master thesis done at the Robotics & Mechatronics group at the University of Twente. The work is done as a part of i-Botics project.

I am thankful for all the learning opportunities I had, and for everyone who supported me along the way.

Kind regards,

Mohamed A. Abdelhady
mohamed.adel03@gmail.com

Contents

1 Introduction

1.1 Problem formulation

SLAM refers to the task of simultaneous localization and mapping, which is a well investigated problem in robotics and photogrammetry (also known as *Structure from Motion (SfM)*). It can be briefly described as a robot attempting to localize itself through an unknown environment and simultaneously create a map of this environment while navigating it. The robot is equipped with multiple information sources which provide the required perception inputs. The main challenge of the problem is the cyclic dependency between both tasks, namely, localization and mapping. i.e. the location is required to create the map and the map is in turn required to determine the location.

SLAM is applied to numerous fields such as warehouse management, port automation, seabed mapping, self-driving cars, commercial indoor robots, space exploration and many more.

Several SLAM algorithms have been successfully used in multiple applications with specific combinations of robot, environment, algorithm and performance. This includes for example indoor mapping of small areas or even buildings using mobile robot and UAVs equipped with laser scanners. Underwater mapping of seabed has also been successful to some extent. However, it is still a challenging task in rough terrains, fast dynamic behavior, large outdoor environments, etc. Cadena et al. (2016) comprehensively discusses the open challenges in SLAM and the trend towards robustness in robot perception.

As a final product, SLAM is a relatively complex software artifact. It is often developed to showcase the feasibility of a novel algorithm or to improve upon existing approaches. The result is a source code that is relatively inflexible and requires extensive efforts to be reused. Remarkably few contributions have addressed the issue of reusability and attempted to create a formal design methodology of SLAM as a software, although SLAM was firstly developed in the 1980s and witnessed many development cycles [Durrant-Whyte and Bailey (2006)].

The reuse of SLAM is becoming a pressing matter as it has been expanding towards applications in mature domains such as automotive industries. Hence, it has to comply with the industrial standards already present in these domains. These standards include non-functional specifications that relates to the maturity of the software such as reliability, reusability, security, etc.

1.2 Approach

Therefore, from the point of view of the different stakeholders it is advantageous to introduce unification and standardization to the design and development processes by creating an infrastructure which can be extended to accommodate all the variabilities of SLAM. Hence, a reuse-oriented framework is sought that adopts component-based software approaches in order to improve the quality of the software products and reduce the time of deployment, it can also provide a flexible way of handling addition or adjustments to algorithms and platforms(sensors, robots, middleware, etc.).

Through following this development strategy the different SLAM modules will be decoupled allowing easier maintenance, debugging, extension and replacement. Accordingly, these tasks can be done in a complete standalone fashion without the need to modify or write extra glue code for the already existing ones. This is in contrast to the approach of having a single code block or a monolithic architecture which performs all the different functionalities.

The main research objective is to:

" *Improve the reusability of SLAM through creating a development framework that accommodates different SLAM variations by applying relevant software engineering techniques.*"

1.3 Related work

The field of robotics can greatly make use of the tools and techniques developed in software engineering in order to analyze, test and design complex systems. It is essential to exploit the synergy between both domains.

Several projects have already been exploring the overlapping areas between both fields, a summary of which is given as follows: Brugali et al. (2012) applies software product line approach to refactor navigation libraries available in the ROS environment and the resulting architecture is shown to be more modular and middleware independent, Bruyninckx et al. (2013) presents a seminal work demonstrating best practices in developing software components from robotics by promoting the separation of concerns and model-driven engineering, Ristroph (2008) attempts at developing early steps for modular SLAM, Foxlin (2002) illustrates a general architecture for localization, auto-calibration and mapping by using a decentralized Kalman filter, Ratasich et al. (2015) demonstrates a generic sensor fusion package for ROS *sf-pkg* that is able to handle the combination of an arbitrary number of any single-dimension value sensor, Moore T. ; Stouch (2014) also develops a modulars sensor fusion package for ROS based on an extended Kalman filter *ethz_msf*, Blumenthal et al. (2011) surveys and applies refactoring techniques to commonly used robotic perception libraries yielding a reusable set of software components with harmonized interfaces, and Ball et al. (2013) presents *OpenRatSLAM* a ROS open-source version of RatSLAM in a middleware independent fashion with software architecture documentation of the different components.

1.4 Outline

Chapter 2 introduces background information on SLAM and the relevant software engineering development paradigms. Chapter 3 analyzes the main concepts and challenges. Chapters 4 & 5 outline the proposed methodology and the realization of the a number of SLAM components. The results and evaluation are given in Chapter 6 and finally Chapter 7 includes the conclusion and further recommendations.

2 Background

This chapter provides the reader with the background material regarding SLAM and the paradigms of component-based software development and software product lines.

2.1 A brief introduction into SLAM

SLAM is the process describing the navigation of a mobile robot in an unknown environment in order to create a map of the surroundings while simultaneously inferring its own location within the self-created map. This is considered to be one of the most important functionalities a mobile robot must posses in order to become truly autonomous [Siciliano and Khatib (2016)]. In order to introduce the basic concepts and notations, the classical probabilistic framework will be used. It should be noted that this specific SLAM representation is not exclusive and the general problem can be expressed using many different formulations [Strasdat et al. (2010)].

The *pose* of the robot which is to be estimated at a certain time instance is represented by the state variable $\mathbf{x}_t$, where the dimension of $\mathbf{x}_t$ is context dependent. For the simple case of a planar mobile robot $\mathbf{x}_t \in \mathbb{R}^3$ as it consists of the 2D coordinates and the heading $\mathbf{x}_t = (x, y, \theta)^T$.

The map of the environment $\mathbf{m}$ is the second variable to be estimated. It can take different forms depending on the chosen map representation which can be feature-based, volumetric or topological. It is often assumed that the map is static and therefore does not evolve with time.

The information that is provided to the robot can be classified into two classes; the *idiothetic*[1] information which is related to the self-motion cues and is regarded as control input $\mathbf{u}_t$. This information can be retrieved from velocity commands, wheel encoders, inertial measurement units, etc. The second class of information that is utilized is the *allothetic*[2] information which is related to the external cues. This could be represented as a distinct landmark in the environment that the robot can observe and deduce its relative pose. The measurements are denoted as $\mathbf{z}_t$.

The robot then seeks to calculate the conditional probability function presented in Eq. 2.1 [Thrun et al. (2005)], which estimates the robot's pose $\mathbf{x}_t$ and the map $\mathbf{m}$ based on the idiothetic and allothetic information given as $\mathbf{u}_t$ and $\mathbf{z}_t$ respectively. It is important to emphasize that in certain SLAM approaches, the term representing the map $\mathbf{m}$ is often incorporated within the state vector $\mathbf{x}_t$.

$$p(\mathbf{x}_t, \mathbf{m} | \mathbf{z}_{1:t}, \mathbf{u}_{1:t}) \tag{2.1}$$

The graphical model shown in Fig. 2.1 shows the four main variables involved in SLAM. It also highlights the desired map and pose estimates. One can also distinguish between the values which are directly observed by the robot and which are estimated. Finally, the arrows indicate a causal relationship, for example, the pose of the robot at time t is influenced by the pose at time t_1 and the control input $\mathbf{u}_t$.

2.1.1 Taxonomy

A great number of SLAM variants have been developed; they can be categorized along different dimensions that as explained extensively in Thrun et al. (2005) and Siciliano and Khatib (2016), the commonly used classifications are given as follows:

[1] Also known as proprioceptive
[2] Also known as exteroceptive

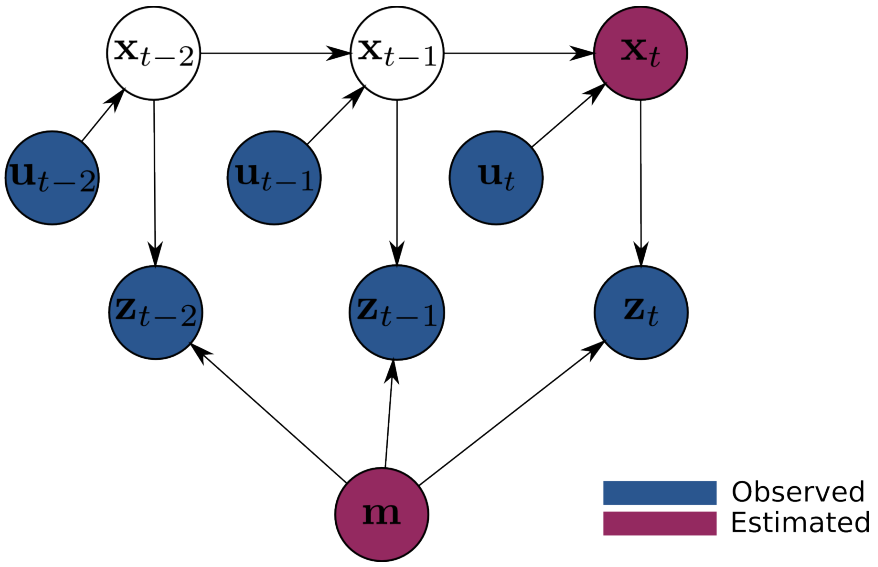


Figure 2.1: Graphical model of the online SLAM problem. The robot is provided with control input $\mathbf{u}_t$ and measurements $\mathbf{z}_t$, and it is required to estimate its own pose $\mathbf{x}_t$ and the map of the environment $\mathbf{m}$ [Thrun et al. (2005)].

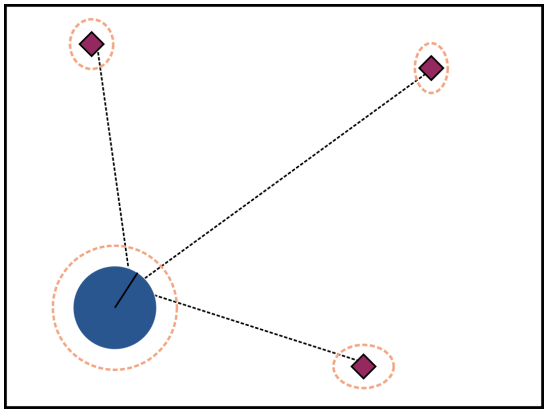
- Online versus offline, where online approach tries to estimate only the current pose of the robot and disregards the past poses, in contrast with the offline approach which tries to find the optimal path of the robot consisting of all previous poses. Both variants are of equal importance in practice.
- Active versus passive, where in active SLAM the direction of the robot motion is controlled purposefully to produce an more accurate estimate. On the other hand, the passive approach does not interfere with the control of the robot and allows for free roaming.
- Static versus dynamic, where the environment is assumed to either change over time and the dynamic effects can be detected or as static where it does not change.

Several strategies has been developed in order to solve the conditional probability described by Eq. 2.1 and will later be classified in Chapter 3. However, the classical filter approach in combination with feature-based maps is best-suited to introduce the basics of SLAM and build an intuitive understanding of the process. The main steps in the classical approach are shown in Fig.2.2 and outlined briefly as follows:

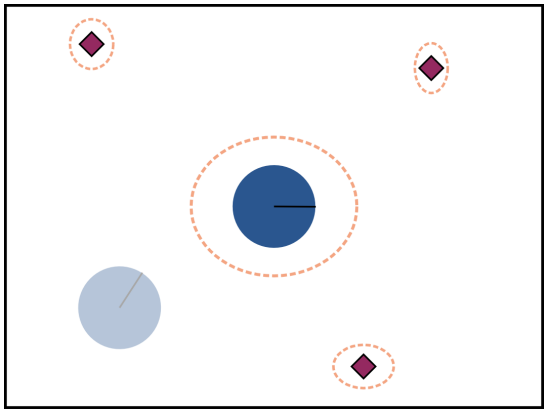
- The robot initializes the pose with the information regarding its current location and orientation. In addition to the landmarks detected during the first observation as seen in Fig. 2.2a. The uncertainties regarding the pose and the landmarks are initialized.
- A control command is given to move the robot to a new position. The motion is not executed ideally and idiothetic information $\mathbf{u}_t$ is used to *predict* the current state of the robot. Furthermore, the uncertainty in the pose increases.
- The robot observes the environment utilizing allothetic sensors which can be used to estimate the robot location and orientation with respect to external landmarks.
  - Landmarks are distinct features in the environment which can be *extracted* from an image, a laser scan or any other allothetic sensor information.
  - Landmarks are also unique and can be *associated* and recognized during motion.

- The robot *updates* its belief regarding its own location, orientation (localization) and also regarding the state of the environment and the extracted landmarks (mapping).

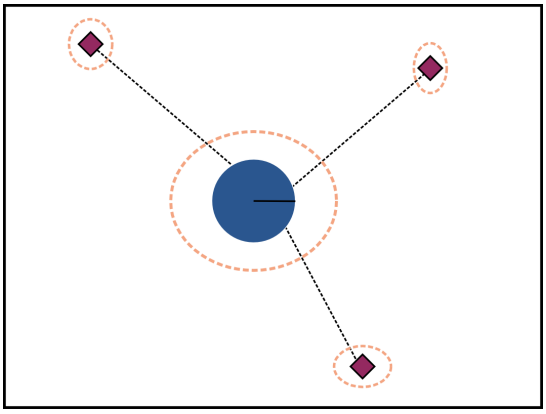
An extra step is often required to detect *loop closure*, which occurs when the robot recognizes that it has returned to one of the previous points on the path. This is also known as *long-term data association*.



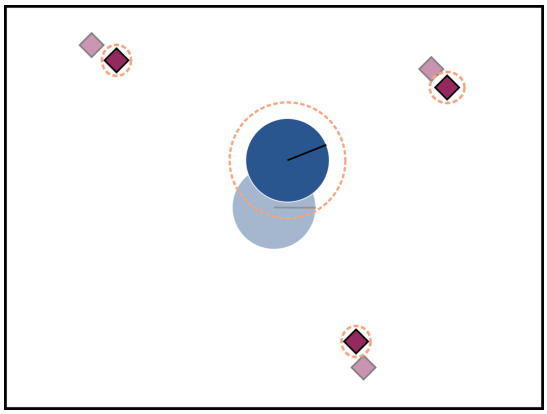
(a) The robot initializes its pose and the landmarks locations along with the corresponding uncertainties.



(b) Motion is executed leading to an increase in the robot's pose uncertainty.



(c) The robot re-observes the landmarks.



(d) The robot corrects its pose along with the belief regarding the landmarks.

Figure 2.2: Graphical representation of the SLAM process within the classical filtering approach. The robot is shown in blue, landmarks in dark magenta and the sensing process as dashed lines.

2.2 Component-based software development

Component-based approaches are considered to be the current trend in developing modular software. It is set to be one level of abstraction higher than *object-oriented programming* (OOP) by promoting the separation of concerns. As it was noted that classes and objects in classical OOP techniques are not efficiently reused [Szyperski (2002)]. Component-based approaches are expected to be the *de-facto* standards among software engineers. Figure 2.3 shows a brief history of software progression towards higher levels of encapsulation and abstraction.

According to Szyperski (2002) the definition of a component is:

"*A unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*"

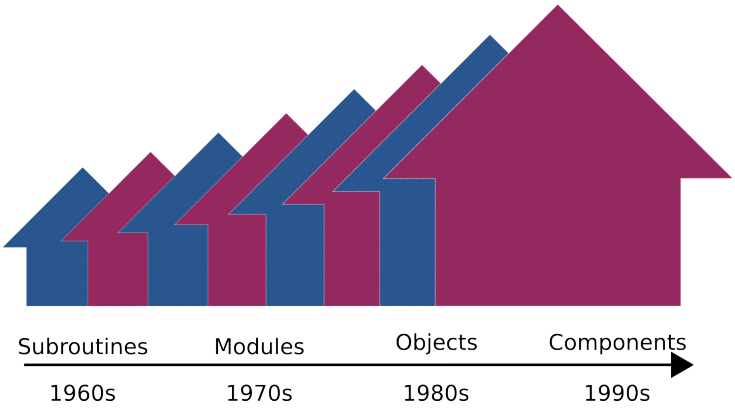


Figure 2.3: The trend of software development approaches towards further encapsulation and reusability. Source: Northrop (2008).

Following the definition, a component is regarded as an executable unit which can be initialized, configured and deployed independently. It encapsulates computation and data, and interacts with the system only through predefined interfaces.

Software components follow a specific component *meta-model* that specifies its properties and internal structure. A great number of meta-models exist such as: CORBA [Wang et al. (2001)], BRICS [Bruyninckx et al. (2013)], Orocos [Bruyninckx (2011)], OPRoS [Song et al. (2008)], etc.

Component-based design (*CBD*) has its origins in different industries other than software, as it is heavily utilized in mechanical and electronics sectors, where a group of readily available component can be assembled into products. By avoiding the reinvention of components, CBD shortens the time-to-market of the product and improves maintainability. Hence, it is favorable as a design tactic in software. The adoption of CBD is advantageous not only in the industry but also in the research community as it allows researchers to focus on the core problem and avoid re-writing code.

2.3 Software product line

Software product lines (SPL) depicts a software engineering development paradigm which is used to create a family of applications that share a common set of functionalities. The main concept behind SPL is the identification of the stable and variable components within a group of applications in a specific domain. SPL then seeks to develop the components, the architectures and the tools in order to assemble the created software artifacts into a complete application.

Such a development methodology has been employed in many engineering fields such as automotive, consumer electronics, avionics, etc. It has shown to improve the productivity and decrease the time-to-market of the products. Therefore, it is worthwhile to investigate product lines within the context of software engineering as it allows the developers to systematically and effectively reuse existing components, leading to mass customization of the software products [Gorton (2011)].

Software product lines have been successfully applied in many fields as listed in the SPL *hall of fame*[3].

[3] http://splc.net/fame.html

The primary concept of the product line development is visualized in Fig. 2.4, where the core and variable assets are identified, developed and assembled to create a family of applications.

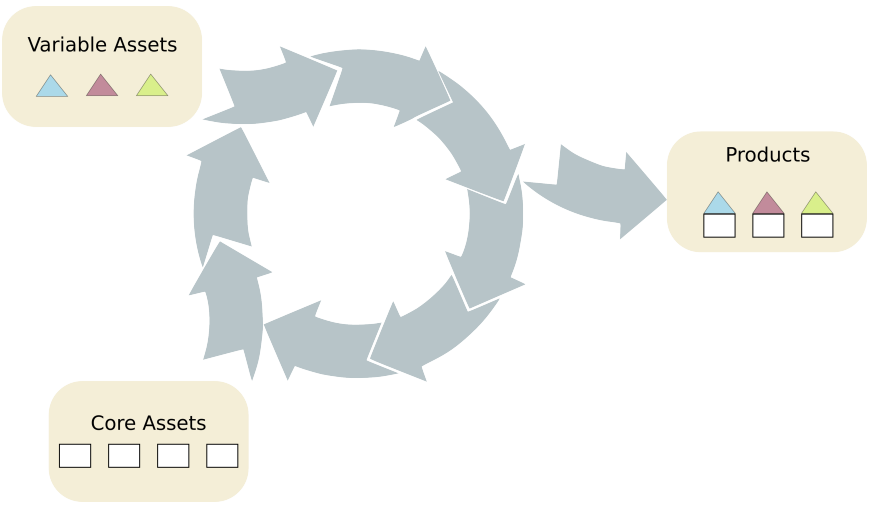


Figure 2.4: Software product line approach; the core and the variable assets are developed and assembled in a product line fashion to realize different product instances.

3 Analysis

Considering the research objective stated in Chapter 1, the analysis aims at investigating the key concepts. Starting with the criteria that qualifies a software asset to become reusable. In addition to the common techniques and development paradigms in software engineering that promote reusability. Furthermore, this chapter provides a domain analysis regarding SLAM through surveying the fundamental components, the different strategies and the challenges in the current SLAM software implementations. Finally, an overview of the approach is given.

3.1 Reusability criteria

To develop reuse-oriented software, certain criteria have to be identified that will lead to a detailed description of software reusability. To this end, The *Reuse Readiness Levels* (RRLs) were introduced by Marshall et al. (2010) at *NASA*. Where 9 topics are presented as the main pillars that indicate the level of the reusability of a software product.

The 9 topics of the RRLs are visualized in Fig. 3.1. They are divided on a scale from 1 through 9 to match the commonly used *Technological Readiness Levels* (TRLs). A score of 1 means that the software is not recommended for reuse and 9 indicates that the software is supported and can be extensively reused. The following are the topics: documentation, extensibility, intellectual property issues, modularity, packaging, portability, standards compliance, support and verification & testing [Marshall et al. (2010)].

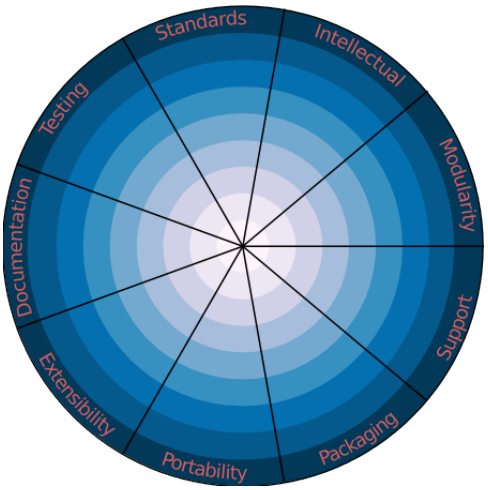


Figure 3.1: The 9 RRLs topics visualized along with the 9 levels per topic.

A highly reusable software will generally address all the mentioned RRLs. However, this work only considers the following topics due to timing constraints.

- Modularity: The degree of encapsulation and segregation of software modules.
- Standard Compliance: Following best practices and keeping the software modules coherent.
- Extensibility: The ability of the system to grow past its current capabilities.
- Verification & Testing: The ability to test the functionalities of the system in order to verify its fidelity.

There exist several software quality measures that can be applied such as *ISO25000* and the reusability metrics proposed in [Frakes and Terry (1996)]. However, the RRLs are chosen because they are exhaustive and they regard reusability of the software as the main concern.

3.2 Challenges

This section highlights the main challenges that inhibit the reusability of the currently developed SLAM software.

One of the most demanding obstacles to software reuse in SLAM is the tight coupling among the different components and the ill-defined interfaces which leads to unmaintainable *spaghetti code*. Furthermore, most of the development is mainly concerned with showcasing a particular improvement or a novel algorithm in SLAM. Therefore, the implementation disregards the long term support of the product and focuses on the computational aspect alone. The source code in these cases would require extensive resources by experts to understand, modify and extend [Brugali et al. (2012)]. The dense coupling can be clearly seen in Fig. 3.2 which shows the dependencies between one component and the rest of the system of a commonly used SLAM implementation *ORB-SLAM2*[1]. It can be seen that any changes to this component would trigger a chain of modifications to the majority of the system. In addition to the instance presented in Fig. 3.2, several examples showing architecture diagrams for a number of open-source SLAM implementations are available in Appendix A.

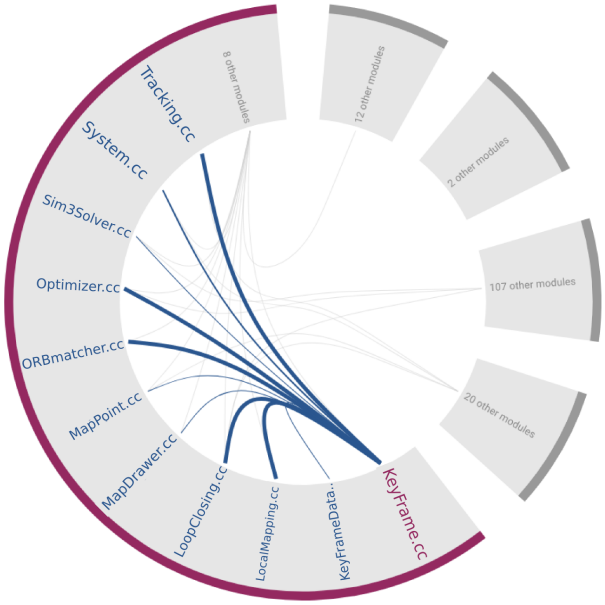


Figure 3.2: Function calls between a single component and the rest of the system in one of the commonly used SLAM open-source implementations [2].

The challenges are exacerbated by the diversity and the rapid development of different SLAM algorithms which makes it a cumbersome task to evaluate, choose and integrate the suitable algorithm to a specific application. Moreover, SLAM is also customized to fit available hardware and middleware resources that leads to a suboptimal software realizations, where platform-specific features are hard-coded within the source code and entangled with computational parts. Table 3.1 shows a selection of commonly used SLAM implementations. More extensive lists can be found on different code repositories [3], however, this collection only serves as a demonstration to the diversity of SLAM approaches.

[1] github.com/raulmur/ORB_SLAM2
[2] The visualization is generated using www.bettercodehub.com
[3] An example for such list is found at github.com/tzutalin/awesome-visual-slam

Library Name	Description
ORB SLAM	Key-frame bundle adjustment for monocular, RGB-D and stereo cameras [Mur-Artal and Tardos (2017)]
Google-cartographer	Scan-to-submap optimization [Hess et al. (2016)]
Hector SLAM	Scan stabilization with multi resolution grids [Kohlbrecher et al. (2011)]
MonoSLAM	Probabilistic approach to natural visual landmarks [Davison et al. (2007)]
RatSLAM	Pose cell state representation with experience map [Ball et al. (2013)]
gmapping	Rao-Blackwellized particle filter implementation to grid maps [Grisetti et al. (2007)]

Table 3.1: A number of open-source SLAM implementations using various approaches.

Moreover, The lack of a formal development methodology leads to an undesirable diversity in the software implementations; including data representations, interfaces and system architectures. The diversity hinders the interchangeability of the different implementations.

3.3 Reuse-oriented software development paradigms

After identifying the criteria of a reusable software and the shortcomings in the current SLAM development process, different paradigms from software engineering can be introduced in order to overcome the aforementioned challenges as they have been proven, tested and perfected over time. A number of development paradigms that promote reusability already exist such as component-based design, service-oriented design, layered-oriented approach, pipe and filter architecture, design patterns, software product lines, aspect-oriented architecture and many more.

The *Software product line* (SPL) approach has been proven to improve the software quality and decrease the development and deployment time of certain applications, provided that the applications share some commonalities. Additionally, SPL is often suited for growth scenarios where extra features, updates or modifications can be easily realized. The SPL approach has been applied successfully in many industrial sectors, furthermore, it has already been applied to the field of robotics by developing a reuse-oriented robot navigation [Brugali et al. (2012)].

As a complementary paradigm to SPL, Bruyninckx et al. (2013) provides best practices for developing robotics software components based on the *separation of concerns* "5C's", which promotes the splitting of 5 key aspects of a software artifact, namely, computation, communication, configuration, coordination and composition.

3.4 Domain analysis

This section describes the domain analysis regarding SLAM in order to explore the different approaches and identify the fundamental components.

3.4.1 SLAM front-end

SLAM can be classified into *front-end* and *back-end*, where the front-end is responsible for interfacing all the available information resources with the back-end which is considered as an abstract information processing module [Cadena et al. (2016)]. There is no clear description for the front-end since it varies according to the sensors and the algorithms employed, however, in the scope of this work, the front-end will be mainly composed of the following:

- Allothetic Information Unit: Which is responsible for the all the data processing for exteroceptive sensors that include: laser scanners (LiDARs), monocular cameras, RGB-D cameras, RADAR systems, etc. Two main operations are identified:
  - Landmark Extraction: The operation tries to identify distinguishable landmarks from the raw sensor data which could be depth or visual data. The output is the detected landmarks which can be expressed by range $\mathbf{r}$ and heading ϕ relative to the sensor's reference frame. Considering the simple case of 2D landmarks the detections are expressed as $[(r_1,\phi_1),\ldots,(r_n,\phi_n)]^T$ for n detected landmarks. In the case of visual data a large number of feature detectors can be employed such as *FAST*, *SURF*, *MSER*, *fiducial markers*, etc. Tuytelaars and Mikolajczyk (2008) offers a survey over the commonly used visual feature detectors including scale, rotation and illumination invariant detectors. Additionally, depth data can be processed in a similar fashion in order to extract the landmarks. Edge detectors and *RANSAC* are often used to extract corners and line segments respectively, moreover, G. Tipaldi (2010) proposes *curvature* and *blob* detectors for 2D laser scan.
  - Data Association: The operation attempts to find *correspondences* between the current observation and the already existing map. Considering the case of sparse landmarks, the matching occurs between k previously saved landmarks and the current n detections. The matching process is done according to different criteria such as: *euclidean distance* and *landmark descriptor*. Moreover, association can be used to match raw measurements such as laser *scan matching* and *image registration*.
- Idiothetic Information Unit: Which is responsible for converting the raw data from wheel encoders, inertial measurement units, cameras (for visual odometry) etc. into motion commands that can be processed by the back-end.

The front-end often includes additional data processing such as noise removal, temporal alignment, coordinate transformation, scaling, etc. These tasks are essential during the preprocessing of sensor data [Cadena et al. (2016)].

3.4.2 SLAM back-end

The back-end of SLAM is described as the computational module responsible for the higher level processing of semantically sound information. It infers the states of the robot and the map of the environment given idiothetic and allothetic information. The back-end of SLAM is the abstract information processing unit which has seen several taxonomies and classifications across the literature. The three main identified approaches are:

1. The filter-based approach which is considered to be the classical and a popular technique for SLAM. It maintains a probability distribution over the states of the robot as well as the map representation. A few examples are: *extended Kalman filter* (EKF), *sparse extended information filter* (SEIF), *particle filter* (PF), *unscented Kalman filter* (UKF), etc. Chapter 2 presents the basic intuition behind this strategy. The filter-based approach is derived based on the *recursive Bayes filter* formulation which is briefly described as follows:

   Considering that the required variables to be estimated are $\mathbf{x}_t$ and the map $\mathbf{m}$ given idiothetic and allothetic information as $\mathbf{u}_t$ and $\mathbf{z}_t$ respectively. *Bayes rule* can be formulated in Eq. 3.1, where the conditional probability is broken down into as the product of prior distribution, the measurement likelihood function and a normalizing factor η.

$$p(\mathbf{x}_t,\mathbf{m}|\mathbf{z}_{1:t},\mathbf{u}_{1:t}) = \eta\, p(\mathbf{z}_t|\mathbf{x}_t,\mathbf{m},\mathbf{z}_{1:t-1},\mathbf{u}_{1:t})\, p(\mathbf{x}_t,\mathbf{m}|\mathbf{z}_{1:t-1},\mathbf{u}_{1:t}) \tag{3.1}$$

By applying the *Markov* assumption and the *law of total probability* the expression can be formulated in a recursive form. Where, $\mathbf{u}_t$ and $\mathbf{z}_t$ are considered to be *conditionally independent* from one another and from past values.

$$p(\mathbf{x}_t, \mathbf{m}|\mathbf{z}_t, \mathbf{u}_t) = \eta \overbrace{p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m})}^{update} \int_{\mathbf{x}_{t-1}} \overbrace{p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)}^{predict} \underbrace{p(\mathbf{x}_{t-1}, \mathbf{m}|\mathbf{z}_{1:t-1}, \mathbf{u}_{1:t-1})}_{recursive\ term} d\mathbf{x}_{t-1} \tag{3.2}$$

Equation 3.2 is considered to be the general expression for the *recursive Bayes filter*. The implementation of the update and predict expressions is dependent on the type of filter, the robot, the map representation and the sensor. However, in all realizations the prediction encodes the motion and propagate the probability density function through a motion model, while the update takes into account the allothetic information and corrects the prediction estimate. It should be noted that the map term $\mathbf{m}$ is missing from the predict term due to the static map assumption. i.e. the environment is not affected by the robot motion [Mullane et al. (2008)].

Figure 3.3 presents the main building blocks of the filter-based approach showing the distinction between the front-end and the back-end.

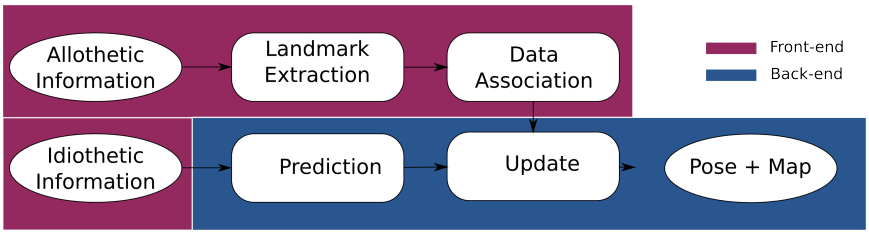


Figure 3.3: Basic building blocks of filter-based SLAM algorithms and the information flow.

2. The optimization-based approaches formulate SLAM as a non-linear optimization problem. The underlying structure can be interpreted as a graph representation, where the *nodes* in the graph encodes the robot poses and the location of the landmarks. The nodes are linked by two types of *constraints* (edges); firstly, the *relative motion constraints* that link the different poses of the robot expressing idiothetic information. Secondly, the *relative measurements constraints* that link the poses of the robot with landmarks in the map expressing allothetic information. Solving the optimization problem yields the optimal set of poses $\mathbf{x}^*$ and map $\mathbf{m}^*$ such that the squared error of the constraints is minimized [Siciliano and Khatib (2016)].

   The approach is used in many of the dominant SLAM implementation such as [Mur-Artal and Tardos (2017)] which runs the optimization problem on a number of key frames captured by vision sensors (also known as *bundle adjustment*). Furthermore, [Strasdat et al. (2010)] shows that optimization-based approaches outperforms classical filtering techniques in most cases.

3. Neural network-based approaches make use of the dynamics of *Continuous Attractor Networks* (CAN) in order to describe the motion and measurement updates. The states of the robots are represented as pose cells and the environment as an experience map. Ball et al. (2013) provides an open-source implementation for the *RatSLAM* which has successfully been utilized to map a suburban area using a single camera.

   Alternative techniques based on

Although the underlying strategy for each of the aforementioned approaches is different, the interfaces are still comparable. The back-end requires idiothetic and allothetic information in order to generate the estimated the robot pose and the map.

3.4.3 Models

Models are considered to be sub-components utilized in both the front and the back-ends. The models encode the robot kinematics and the properties of the sensors along with their uncertainties. They are classified into *motion* and *measurement* models.

- The motion model encodes the kinematics of the robot. Within the probabilistic context, motion models accepts as input idiothetic information generated from encoder data, velocity commands, IMU data, etc. And converts it into a probability density representation of the predicted robot pose, given the robot kinematics (state transition function), the previous pose and the process noise. The most commonly used models are the *velocity model* and the *odometry model* [Thrun et al. (2005)].
- The measurement model encodes the properties of the depth or vision sensors. Within the probabilistic context, measurement models accept allothetic information such as detected landmarks or laser scans and calculates the likelihood of an observation given the robot pose, the map and the noise characteristics of the sensor. The commonly used models are *likelihood field* and *range-bearing* models [Thrun et al. (2005)].

3.4.4 Additional modules

The aforementioned elements are fundamental to perform SLAM. However, a number of additional functionalities could be implemented to further process the output of SLAM such as:

- Semantic Mapping: Is used to Provide a higher-level of understanding of the environment. It can be briefly described as a categorization problem that tries to match semantic information to specific parts of the map allowing the system to add labels to these parts such as rooms, doors, corridors, etc [Cadena et al. (2016)].
- Scene reconstruction: Is used to render a dense point cloud representation of the environment. Which can be utilized in many different applications such as virtual or augmented reality. The image reconstruction functionality requires a sequence of images capturing overlapping areas in the scene along corresponding camera pose [Mur-Artal and Tardos (2017)]. This can be easily achieved by using the trajectory of the robot and the point clouds available in the map generated by SLAM.
- Volumetric map visualization: Utilizes depth data in order to obtain a semi-dense visualization of the environment. For example, a 2D floor plan can be obtained using a laser scanner. Each measurement is inserted into the map according to the current robot pose which is estimated by the back-end of SLAM.
- Path planning for *active SLAM*: Is used to generate the trajectory that the robot should follow in order to improve the localization and the map estimates.

The aforementioned functionalities do not affect the pose and the map estimates within the back-end of SLAM and operate based on the output of the back-end, therefore, they are considered as optional and additional components.

3.5 Approach overview

The analysis presents well-defined criteria for reuse-oriented software along with the development approaches and design tactics that achieve the identified requirements. It also shows that SLAM - although very diverse - has some stable components and interfaces which are always needed in order to correctly preform localization and mapping [Siciliano and Khatib (2016)]. The identified commonalities are considered to be well-suited candidates for decoupling and reuse; as they need not be designed and developed from scratch.

To this end, *component-based design* augmented with the *software product line* approach and the *separation of concerns* are proposed as a development methodology to SLAM.

In order to evaluate the developed framework, several use-case and growth scenarios will be formulated inspired from the *Architecture Trade-off Analysis Method* (ATAM) which is often used to formally assess the performance of a certain architecture against designated quality attributes (which is reusability in this case) [Kazman et al. (2000)]. The scenarios can help to provide the scope and concertize the RRLs mentioned in 3.1.

4 SLAM Framework Design

This chapter details the proposed methodology in 3.5 in order to create a reuse-oriented SLAM framework. The methodology is based on software product lines, component-based approach and the separation of concerns.

4.1 Feature model

The *feature model* is set to be the first step in developing a software product line for a certain application. Figure 4.1 shows the developed feature model for SLAM. The model is considered as a graphical representation for the domain analysis performed in 3.4 as it identifies stable and optional functionalities. Mandatory functionalities are identified with a black circle notation at the link and optional functionalities with a white circle notation. Furthermore, a certain functionality can have different implementations which is represented as a variation point. The cardinality at the variation points specify if the alternative implementations are mutually exclusive ([1..1]) or inclusive ([1..∗]) [Badger et al. (2016)].

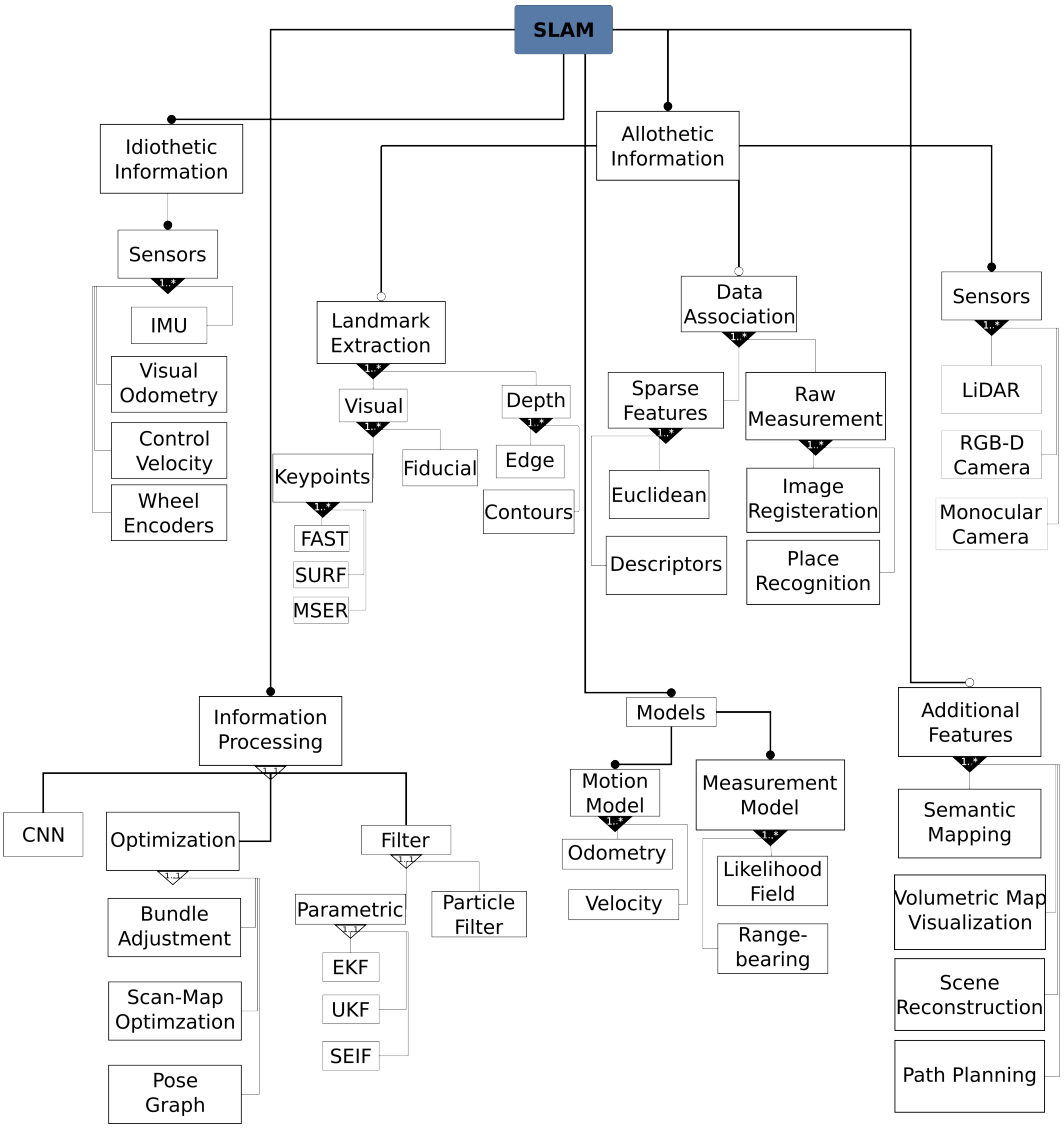


Figure 4.1: A feature model using the *HyperFlex* tool-chain to identify the higher level stable components and the variation points in SLAM.

The model incorporates the front-end which is split into allothetic and idiothetic information units, the back-end as the information processing and the models which are commonly used in the front and the back-ends. The aforementioned elements are considered to be stable SLAM functionalities [Siciliano and Khatib (2016)]. Furthermore, a number of optional features are listed in the model.

Each functionality is detailed further according to the various implementations discussed in the domain analysis 3.4.

The developed feature model is by no means exhaustive nor definitive, however, it demonstrates the possibility of applying SPL to SLAM. Additionally, it does not include the variation points at the realization phase. Such as sensor configuration, middleware, communication protocols and the robotic platform used.

4.2 Software design

The feature model splits SLAM into functional blocks which are translated into software components and considered as *primitives* building blocks (as described in Chapter 5). The combination of a number of components is labeled as *composites*. Finally, the integration of compatible components into a working SLAM realization is considered as the final *applications* [Kraetzschmar et al. (2010)]. The gradual construction strategy of components, composites and applications is chosen to ensure standalone deployment and extensive testing before reaching the final product.

4.2.1 Component level design

A *component* is a piece of software that implements a certain functionality within SLAM[1]. From the practical point of view, a component is a group of classes that are managed by a component coordinator. The coordinator also offers the main entry point to the component and manages its public interfaces.

The following are the proposed requirements for each component in order to comply with the reusability criteria (RRLs) :

- A component as an entity should be treated as a black box with public interfaces to ensure information encapsulation as shown in Fig. 4.2.

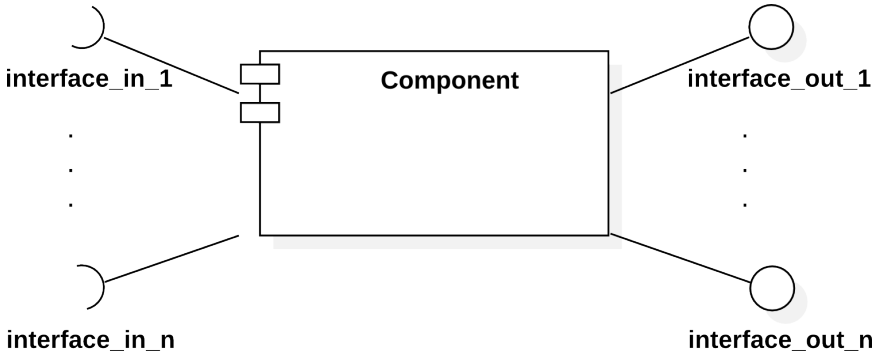


Figure 4.2: An abstract representation of a component that encapsulates all the internal computation and variables. It is only accessible through public interfaces.

- A component's main entry point is the coordinator (also known as the mediator or the manager) which handles the creation, configuration, interfaces and destruction of the component. This can be considered partly as a mediator or a factory design pattern that

[1] Important point to clarify since SLAM itself can be considered as a component in different contexts

inherently promotes loose coupling. Hence, Modification to the internal classes of the component can occur without the need to change the generic public interfaces of the coordinator such as `configure()` or `run()` functionalities. An instance of a generic component is shown in Fig. 4.3. This design choice is in-line with the best practices provided by Bruyninckx et al. (2013) in the BIRCS component model, where each component includes a coordinator.

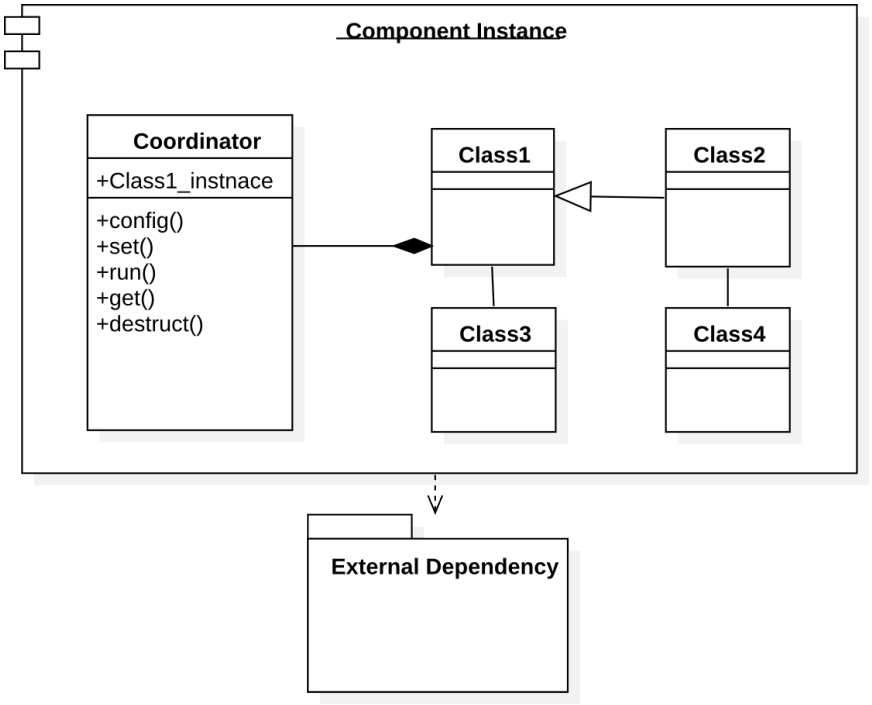


Figure 4.3: An instance of a component as a group of classes that are managed by a coordinator which is also the main entry point to the component's functionalities. Different relationships bind the classes together, such as inheritance, composition and association. Furthermore, the external dependencies are specified explicitly.

- A component should be deployable standalone without dependency on other components. This property is considered to be a cornerstone in the component-based development approaches [Szyperski (2002)] (In contrary with common open-source implementations of SLAM that offer a single executable entity).
- Within the boundaries of each component the separation of concerns should be followed as a best practice [Bruyninckx et al. (2013)]. The separation of the coordination, communication and composition is achieved by introducing the coordinator (The separation could also be done within the middleware). Additionally, the segregation between the computation and the configuration is achieved by assigning the parameter values through external files at deployment-time.
- The component should be middleware independent. Nonetheless, wrapping into a specific middleware such as ROS or Orocos can be achieved with minimal glue code. This is favorable to avoid major changes if a different middleware is chosen in future implementations.
- The internal classes within the components should be balanced to avoid having a *"god-class"* that handles all the logical computation of the component since smaller units of core are easier to maintain, test and modify [Visser (2016)].

4.2.2 System level design

The system-level design introduces the structure of the system and the approaches used to compose the developed components. Two composition strategies are proposed in order to retain component segregation:

- A higher level coordinator (*meta-coordinator*) is used to build the complete application. It interfaces with all the different components' coordinators through a *caller/provider* «C/P» communication [Brambilla et al. (2012)]. The meta-coordinator is wrapped by the middleware in order to communicate with the sensors. This centralized architecture highlights components segregation, consequently, any addition / modification / removal of a component will only be reflected in the meta-coordinator. The proposed structure is shown in Fig. 4.4.

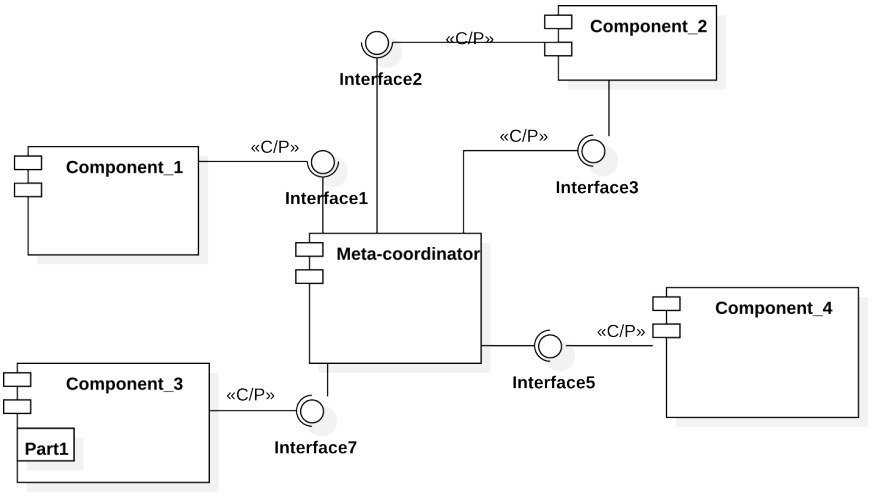


Figure 4.4: Components composition through the meta-coordinator to deploy a complete application. The centralized approach allows components to be oblivious to one another.

- Wrapping each of the individual components within the middleware such as ROS, allowing the components to communicate on *publish-subscribe* «P/S» basis as show in Fig. 4.5. Such composition exploits the distributed nature of the middleware; also any addition / modification / removal of component will only affect the corresponding wrapper.

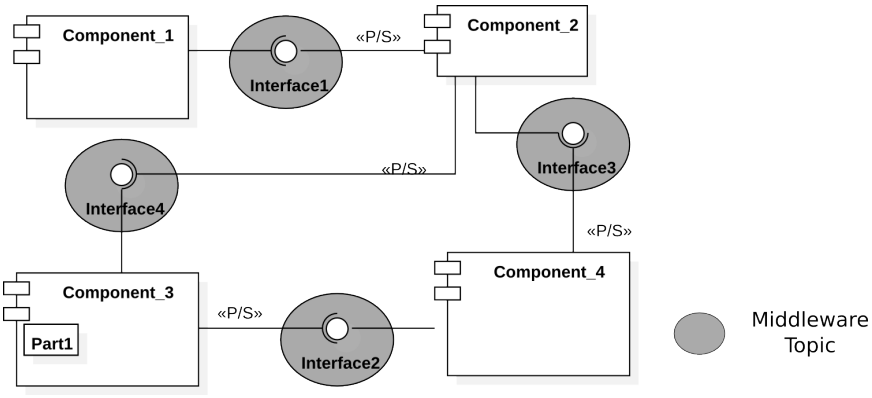


Figure 4.5: Components composition using the distributed nature of the middleware such as ROSs.

5 Components Development

This chapter describes the development of a number of components used to deploy SLAM according to the methodology defined in Chapter 4.

The derivation of a product from the software product line starts by selecting the constituting components. The selection is constrained by a set of rules defined by the developer to indicate that not all components combinations are valid. For example, non-parametric state representation can not be used with an EKF back-end.

Following the feature model shown in Fig. 4.1 several components will be developed in order to serve as building blocks to deploy multiple SLAM instances. Figure 5.1 highlights the developed components within the feature model. It should be noted that not all the components are deployed in a single instance as some of the selected blocks are mutually exclusive such as the EKF and the PF. Therefore, the figure only serves as road-map indicating the classification of the developed components.

The software framework is developed using *C++* and a number of external libraries, namely, *OpenCV*[1], *ArUco*[2] and *MRPT*[3].

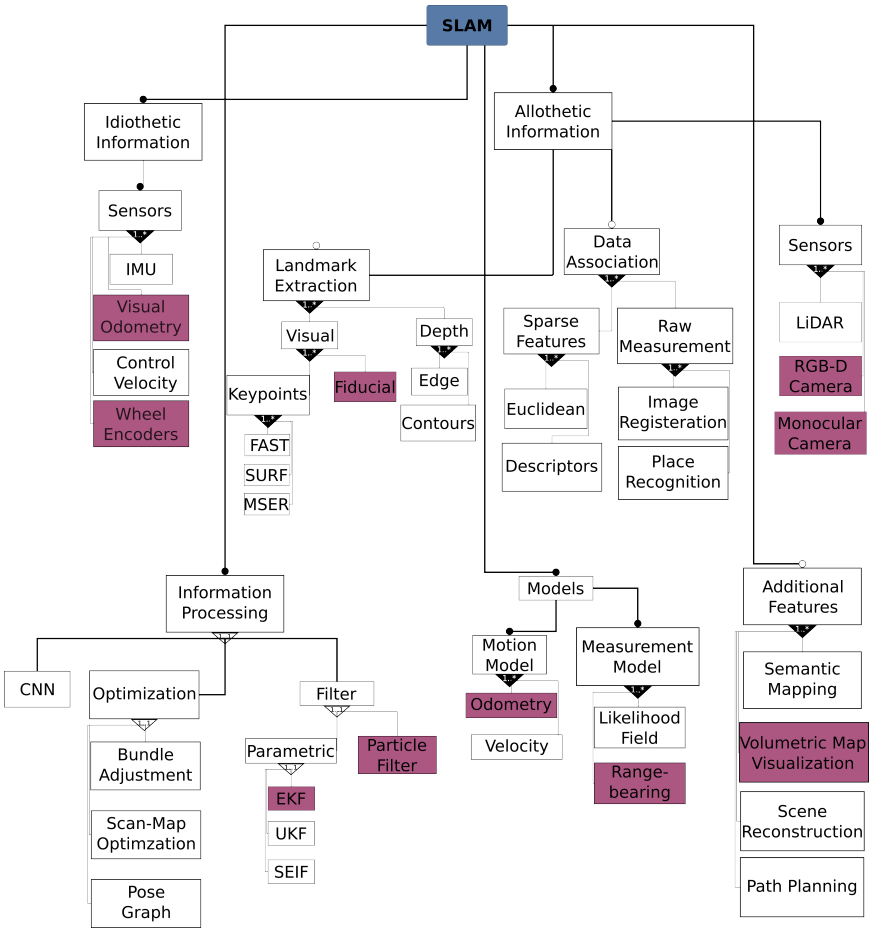


Figure 5.1: Instance view in *HyperFlex* after selecting the components required to perform SLAM based on extended Kalman filter (EKF).

[1] http://opencv.org/
[2] https://www.uco.es/investiga/grupos/ava/node/26
[3] http://www.mrpt.org/

Within the boundaries of the components, the separation of concerns is applied by distributing the computation across a group of classes. The communication, configuration and composition is done by means of a component coordinator. Configuration is handled through separate files that are parsed during deployment-time.

5.1 Extended Kalman filter for feature-based map

Extended Kalman filter (EKF) is considered to be at the core of the earliest SLAM implementation in Smith ' et al. (1987). EKF realizes the recursive Bayes equation in Eq. 3.2 by maintaining a parametric Gaussian representation over all the states including the detected landmarks in the form of a state vector $\mathbf{x}_t$ and a covariance matrix Σ_t [Thrun et al. (2005)].

The exchange of data to and from the component is done through the main interfaces shown in Fig. 5.2. It accepts idiothetic information $\mathbf{u}_t$ representing odometry measurements. which is used to propagate the states and the covariance matrix through the motion model; the result is the prediction $p(\mathbf{x}_t|\mathbf{u}_t,\mathbf{x}_{t-1})$. The landmarks are assumed to be static, therefore, the motion does not affect their locations. Furthermore, the component processes the measurement vector $\mathbf{z}_t$ containing the range r and the bearing ϕ observed landmarks at time t along with their associated id. new landmarks are appended to the state vector and previously seen landmarks are used to update the corresponding entries in the state vector and the covariance matrix.

The output is the estimated robot pose $[x,y,\theta]^T$ and the landmarks in the environment represented by their 2D coordinates (m_x,m_y).

The motion and measurement models used in the EKF implementation are discussed in 5.3.

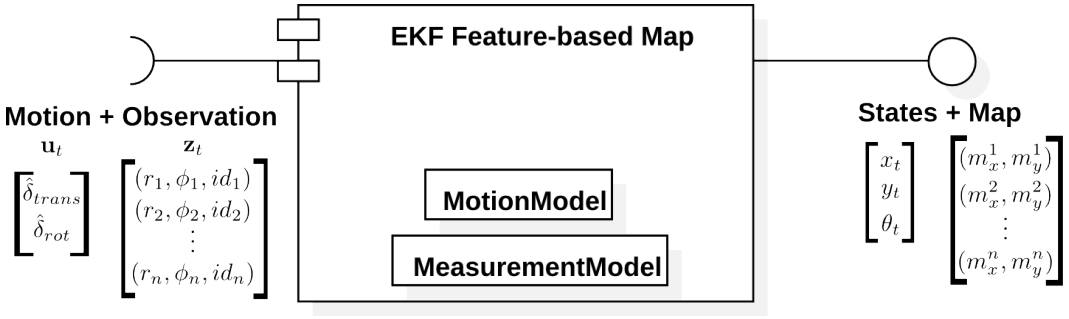


Figure 5.2: An abstract view of the developed EKF component with the main interfaces.

5.2 Fiducial marker detector

A Visual landmark extraction component is developed in order to extract discernible visual features from the environment. Fiducial markers are used as landmarks since they are easily detected and identified; which eliminates the need for data association component. The input image is inspected for square-like objects which is classified as a marker if the inner codification matches with one of the predefined patterns (*dictionary*) [Garrido-Jurado et al. (2014)]. The detection of the markers provide the location of the 4 corners w.r.t image coordinates along with the marker ID. The 3D location of the markers relative to the camera (robot's local frame) can be inferred given the camera intrinsic parameters and the actual square size of the printed markers. Figure 5.3 shows an image frame with several highlighted markers indicating that they are correctly detected and identified.

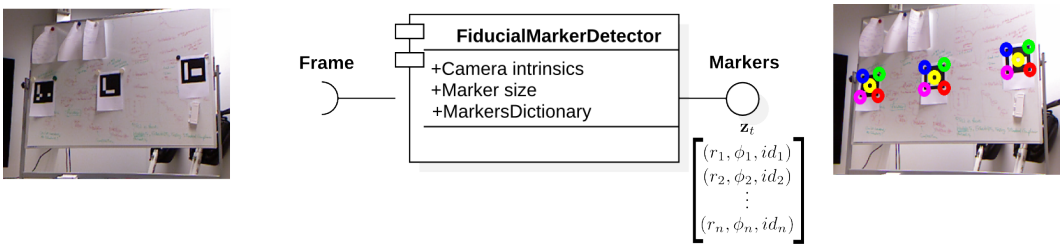


Figure 5.3: Fiducial marker detection using OpenCV ArUco module. The detected markers are highlighted.

5.3 Models

In order to encode the kinematics and the properties of the sensors along with their uncertainties, a probabilistic odometry motion model and range-bearing measurement model are developed for a 2D mobile robot mapping an environment populated with 2D landmarks.

5.3.1 Odometry motion model

The odometry motion model is used to calculate the predicted robot pose $p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{u}_t)$ in the recursive Bayes equation presented in 3.2. The input consists of incremental odometry values for translation and rotation[4] $\mathbf{u}_t = (\hat{\delta}_{trans},\hat{\delta}_{rot})^T$ that corresponds to the motion of the robot. The $(\hat{\cdot})$ notation indicates that the odometry is noisy. The noise is assumed to be additive Gaussian noise with zero-mean expressed as ϵ_{trans} and ϵ_{rot} added to each of the measurements.

The odometry increments are expressed relative to the robot's local coordinate frame, therefore, the state transition function in Eq. 5.1 resolves the motion to obtain the pose w.r.t the global reference frame.

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \hat{\delta}_{trans}\cos(\theta+\hat{\delta}_{rot}) \\ \hat{\delta}_{trans}\sin(\theta+\hat{\delta}_{rot}) \\ \hat{\delta}_{rot} \end{bmatrix} \tag{5.1}$$

Where, $(x,y,\theta)^T$ is the robot's previous pose and $(x',y',\theta')^T$ is the predicted pose.

Furthermore, in the case of an EKF back-end a linearized motion model is required in order to predict the covariance associated with the motion and to preserve the Gaussian assumption. A detailed derivation of the linearization is presented in [Thrun et al. (2005)].

The additive noise for the translation is expressed as $\epsilon_{\alpha_1\delta^2_{trans}+\alpha_2\delta^2_{rot}}$ and for the rotation as $\epsilon_{\alpha_3\delta^2_{trans}+\alpha_4\delta^2_{rot}}$. Where, ϵ is a Gaussian distribution with variance specified in the subscript. Equation 5.2 simply formulates the addition of the noise terms.

The model offers $\alpha_1,\ldots,\alpha_4$ as process noise parameters that can be tuned according to the robot platform used. The parameters can be described as follows:

- α_1 relates the translational motion to the uncertainty in translation estimate.
- α_2 relates the rotational motion to the uncertainty in translation estimate.
- α_3 relates the translational motion to the uncertainty in rotation estimate.
- α_4 relates the rotational motion to the uncertainty in rotation estimate.

[4] An extra rotation term δ_{rot_2} is occasionally added to the formulation which assumes that the robot performs an additional rotation after it arrives to the final pose [Thrun et al. (2005).]

$$\begin{bmatrix} \hat{\delta}_{trans} \\ \hat{\delta}_{rot} \end{bmatrix} = \begin{bmatrix} \delta_{trans} + \epsilon_{\alpha_1 \delta^2_{trans} + \alpha_2 \delta^2_{rot}} \\ \delta_{rot} + \epsilon_{\alpha_3 \delta^2_{trans} + \alpha_4 \delta^2_{rot}} \end{bmatrix} \tag{5.2}$$

5.3.2 Range-bearing model

The range-bearing model is implemented for 2D landmarks, where the model is used to calculate the likelihood of a certain measurement given the robot pose and the map $p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m})$, which is the update term in the recursive Bayes formulation[5] presented in Eq. 3.2. For 2D landmarks the measurement $\mathbf{z}$ consists of a range r^i and a heading ϕ^i for the i^{th} landmark and the model is formulated in Eq. 5.3 [Thrun et al. (2005)].

$$\begin{bmatrix} r^i \\ \phi^i \end{bmatrix} = \begin{bmatrix} \sqrt{(m^i_x - x)^2 + (m^i_y - y)^2} \\ atan2((m^i_y - y), (m^i_x - x)) - \theta \end{bmatrix} + \begin{bmatrix} \epsilon_{\sigma^2_r} \\ \epsilon_{\sigma^2_\phi} \end{bmatrix} \tag{5.3}$$

Where, $(x, y, \theta)^T$ is the robot pose, $(m^i_x, m^i_y)^T$ is the world 2D coordinates for the i^{th} landmark, $(r^i, \phi^i)^T$ is the estimated measurement and $(\epsilon_{\sigma^2_r}, \epsilon_{\sigma^2_\phi})^T$ is the assumed added Gaussian noise which has a zero mean and a variance of σ^2_r and σ^2_ϕ for the range and the bearing respectively. The noise parameters are sensor specific configuration values indicating the uncertainties in the measurements.

Furthermore, the inverse of the model is used to initialize the location of the landmarks, where the range and bearing are expressed w.r.t the sensor local frame and the landmarks needs to be initialized in the global frame. Therefore, Eq. 5.4 is used to initialize the landmarks location.

$$\begin{bmatrix} m_x \\ m_y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x_{sens} \\ y_{sens} \end{bmatrix} + r \begin{bmatrix} cos(\theta + \phi) \\ sin(\theta + \phi) \end{bmatrix} \tag{5.4}$$

Where, $(x, y, \theta)^T$ is the robot pose, $(x_{sens}, y_{sens})^T$ is the sensor's pose w.r.t the robot's local frame of reference and $(r, \phi)^T$ is the range and bearing of a landmark w.r.t the sensor's reference frame [Thrun et al. (2005)]. Figure 5.4 illustrates the different variables in Eq. 5.4.

5.4 Particle filter for feature-based map

The developed component for particle filtering is based on the *Rao-Blackwellized* particle filter also known as `FastSLAM` [Thrun et al. (2005)]. The particle filter represents the robot pose through a set of weighted particles. The clustering of the particles at a certain location in the state space is proportional to the probability of the robot being in that state. Each particle maintains an independent estimate of the robot's path and N extended Kalman filters (EKFs) over the N landmarks as shown in Eq. 5.5.

$$\begin{aligned} particle^1 &= \{(x, y, \theta)^T\}^1_{1:t} \quad (\mu^1_1, \Sigma^1_1)\ (\mu^1_2, \Sigma^1_2)\ \ldots\ (\mu^1_N, \Sigma^1_N) \\ particle^2 &= \{(x, y, \theta)^T\}^2_{1:t} \quad (\mu^2_1, \Sigma^2_1)\ (\mu^2_2, \Sigma^2_2)\ \ldots\ (\mu^2_N, \Sigma^2_N) \\ &\vdots \\ particle^M &= \{(x, y, \theta)^T\}^M_{1:t} \quad (\mu^M_1, \Sigma^M_1)\ (\mu^M_2, \Sigma^M_2)\ \ldots\ (\mu^M_N, \Sigma^M_N) \end{aligned} \tag{5.5}$$

[5] the map $\mathbf{m}$ is considered to be a part of the state vector $\mathbf{x}$ in the EKF formulation and the measurement update term reduces to $p(\mathbf{z}_t|\mathbf{x}_t)$

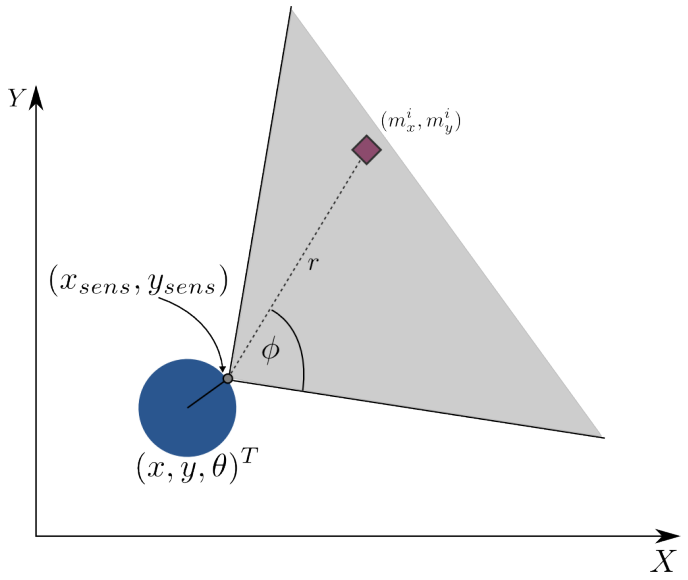


Figure 5.4: .

In similarity to the previously developed EKF, the particle filter component accepts idiothetic information $\mathbf{u}_t$ representing odometry measurements. which is used to propagate all the particles through the motion model. Furthermore, the particle filter uses the likelihood of the landmarks observations to calculate the *weight* for each particle. Finally, the particles are *re-sampled* proportional to their weights. The re-sampling achieves the update step since the unlikely particles are eliminated from the new set of particles. The output consists of the robot pose and the detected landmarks according to the most likely particle (highest weight). Figure 5.5 shows the main interfaces.

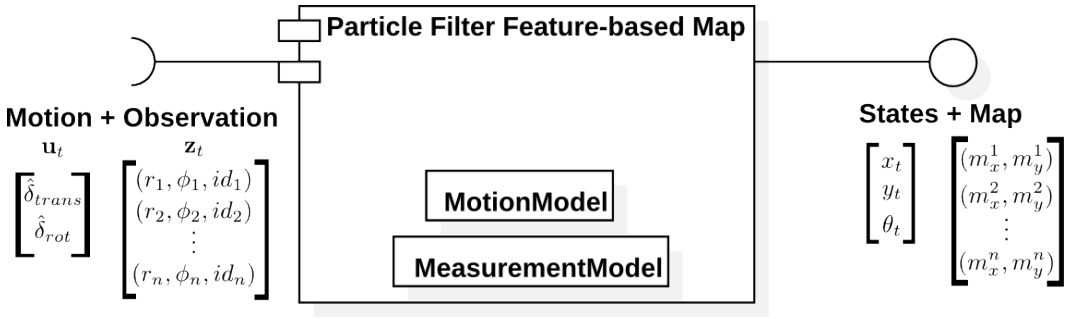


Figure 5.5: The main interfaces of the particle filter component. It accepts odometry increments and range-bearing measurements. The output is the estimated robot pose and the feature-based map according to the most likely particle.

5.5 Visual odometry

Visual odometry refers to the process of calculating the incremental translation and rotation of the robot using a sequence of images. In this approach, *SURF Keypoints* are detected in two subsequent frames along with their descriptors. The descriptors are used to find matches between the two frames, afterwards a variant of *RANSAC* is used for outliers rejection. The remaining matches are used to calculate a *similarity transformation* between the two frames. The transformation encodes the relative translation and rotation (δ_{trans},δ_{rot}). The visual odometry component is developed to be a replacement to the wheel encoders. An instance of two frames with the matched keypoints is shown in Fig. 5.6. It should be noted that different visual

odometry algorithms can be used such as *KLT* trackers [Fraundorfer and Scaramuzza (2012)] and *deep convolutional neural networks* [dee (2016)].

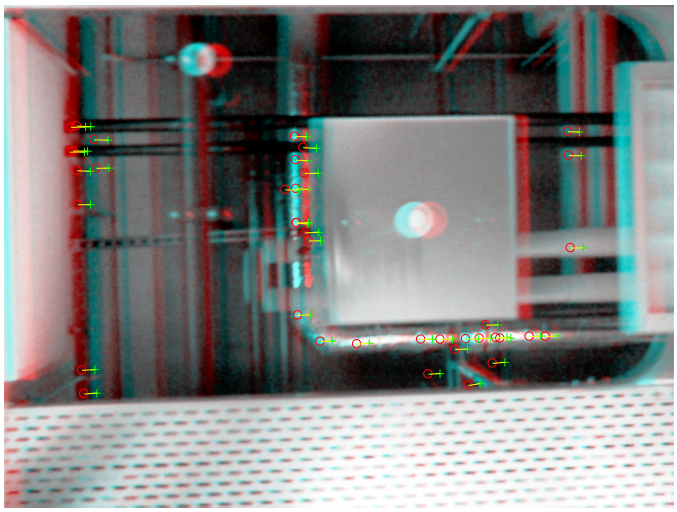


Figure 5.6: Keypoints matching between two consecutive frames captured by an upward looking camera.

5.6 Point-map visualization

A Point-map visualization component represents the environment as a 2D floor plan with each point in the map corresponding to an object detected by the laser scanner (different ranging sensors can be used). The developed component accepts 2D laser scan data as input along with the estimated robot pose. Each range data is transformed into the global frame in similarity to Eq. 5.4 and compared with each existing point in the map. Euclidean distance is compared against a threshold to determine whether the range measurement is fused with the current map or appended to it.

As an additional component, the point-map visualization has no effect over the pose estimate nor the map within the back-end of SLAM.

The component interfaces and the sample visualization are shown in Fig. 5.7.

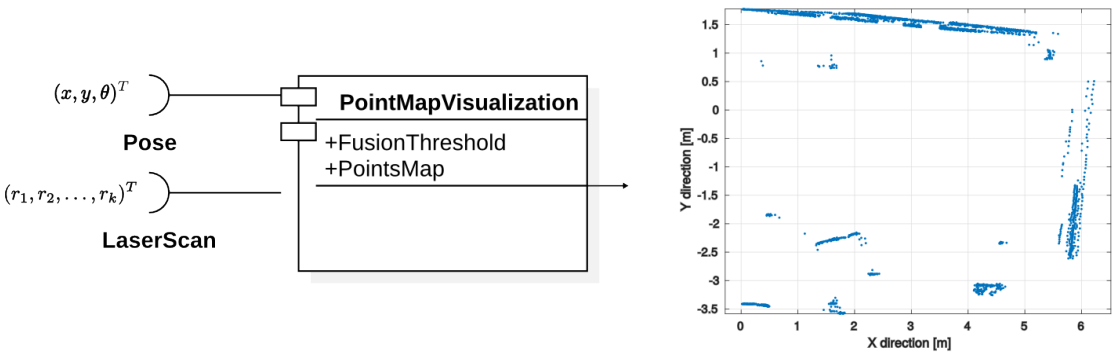


Figure 5.7: Point-map visualization component along with the main interfaces. The components generates a 2D representation of the obstacles detected in the environment based on a laser scan data.

5.7 Particle filter for grid-based map

Occupancy grid maps are often favored over feature-based maps as they do not require an explicit definition of the landmarks and could offer a more informative representation of the environment [Grisetti et al. (2007)]. Occupancy grid maps discretize the environment into a finite number of cells each carrying a probability value indicating if the cell is considered as free-space or occupied.

The previously developed particle filter can be extended to handle occupancy grid maps by adapting the calculation of the likelihood function that computes the likelihood of a certain measurement based on the pose of the particle and its corresponding grid map $p(\mathbf{z}_t|\mathbf{x}_t^i, m^i)$ for the i^{th}-particle. Similarly to the feature-based map, the likelihood is used to update the weight of each particle. Figure 5.8 shows the list of particles, where each particle consists of a robot path and an independent map.

Furthermore, the grid map for each particle is updated through calculating a posterior function for each grid cell j within the map $p(m_j^i|\mathbf{z}_t,\mathbf{x}_t^i)$ in order to obtain the full map posterior $p(m^i|\mathbf{z}_t,\mathbf{x}_t^i)$ for the i^{th} particle. The detailed derivation is presented in [Thrun et al. (2005)].

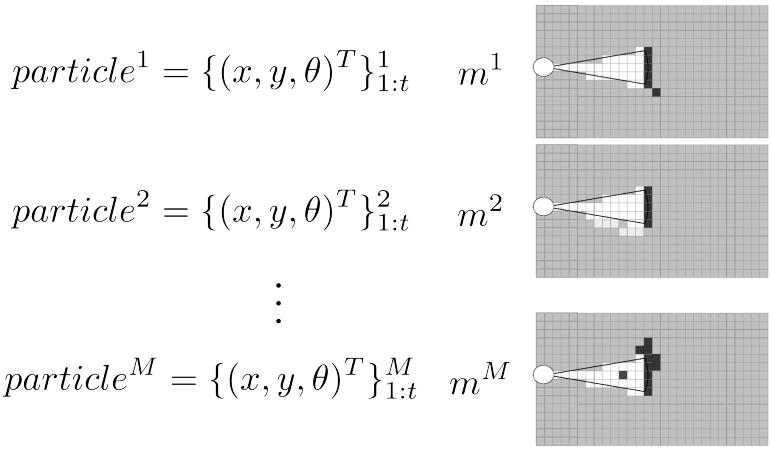


Figure 5.8: Particle list for grid-based maps. Each particle maintains a separate occupancy grid that can be used to calculate the likelihood of an observation and update the corresponding weight of the particle.

6 Evaluation

The results and the evaluation of the proposed framework is presented in this chapter along with the experimental design and the hardware information.

The evaluation consists of two aspects; firstly, the derivation of several use-cases from the product line is assessed from the reusability point of view according to the RRLs discussed in 3.1. The evaluation will adopt the concept of scenarios from *ATAM* in order to come up with quantitative measures for the chosen RRLs, namely, modularity, extensibility, standards compliance (5Cs) and testing & verification.

A number of *use-case* and *growth* scenarios are created in order to evaluate each of the chosen RRLs. A summary is presented in Table 6.1.

Target RRLs	Modularity	Extensibility	Testing	Standards
Scenarios				
Component replacement	+++			++
Component addition		+++		+
Logic modification	++	++		
Standalone deployment	++		+++	
Deployment-time configuration			+	+++

Table 6.1: A general description of the proposed scenarios and the target RRL to be evaluated accordingly.

In each of the scenarios mentioned in 6.1 a quantitaive measure will be given based on:

- The number of replaced / added / modified components
- The number of replaced / added / modified interfaces

The second aspect of the evaluation is the performance of the different SLAM realizations. A qualitative measure is proposed based on the ability to generate a consistent path and map estimation during the presence of multiple loop-closures i.e. ability to close the loop and correctly recognize previously visited location, which is considered to be the differentiating factor between SLAM and plain odometry [Cadena et al. (2016)], therefore, it is considered as a *de-facto* criterion to assess SLAM systems.

6.1 Experimental setup

A Segway RMP 50 omni-directional platform is used to run the experiments; it can be controlled through on-board intel NUC. The platform is tele-operated using a Logitech extreme 3D pro joystick. The sensors used are:

- Kinect for XBOX-360 (Only RGB camera is used).
- Hokuyo LiDAR urg-04lx-ug01.
- Upward-looking camera Logitech HD C615 webcam.
- Segway RMP 50 built-in wheel encoders.

The main SLAM computation is done on a Lenovo P50 running Ubuntu 14.04. ROS indigo is used as a middleware to interface with the sensors and to control the Segway platform.

6.2 SLAM deployment

This section describes the deployment of 5 different SLAM realizations. The different instances will be used to estimate the robot path and the map for a relatively small indoor office environment ($\approx 10m^2$) using the same dataset. The environment is randomly populated with fiducial markers and the robot is manually driven in approximately the same trajectory twice to evaluate loop-closure capabilities. The dataset does not include ground truth measurements, therefore, no quantitative evaluation of SLAM is provided.

The developed components are composed according to the system architecture proposed in Fig. 4.4, where a meta-coordinator instantiates and communicates with the different components through a *caller/provider* communication (highlighted as «C/P»). Furthermore, it interfaces with the sensors using ROS through *publisher/subscriber* communication (highlighted as «P/S»). The alternative system composition strategy proposed in Fig. 4.5 is not implemented due to time constraints.

6.2.1 Extended Kalman filter feature-based SLAM

The component diagram shown in Fig. 6.1 presents the integration of a number of selected components from the feature model in Fig. 5.1. The application uses wheel odometry as an idiothetic information source and a front camera to detect fiducial markers as the source for allothetic information. The back-end is based on an extended Kalman filter that includes odometry and range-bearing models.

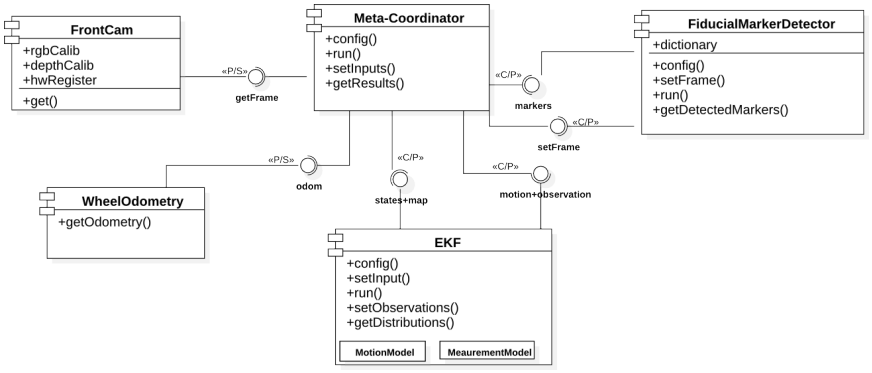


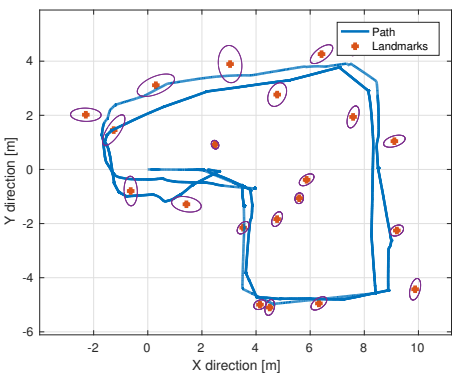
Figure 6.1: EKF feature-based SLAM component diagram.

Figure 6.2a shows the path and the map generated from SLAM instance described in 6.1 using the defined dataset.

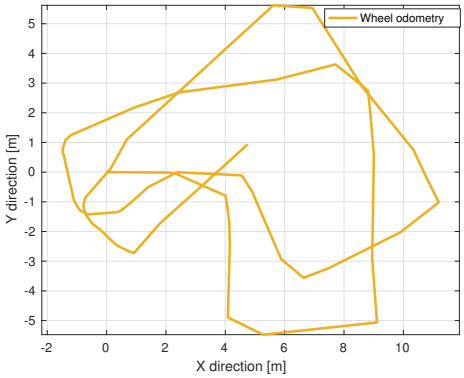
Figure 6.2b is presented in order to highlight the differences between the path estimate generated by SLAM and by dead reckoning which is based solely on wheel encoders (idiothetic information). The error accumulation in dead reckoning produces an inconsistent path as the second loop trajectory clearly does not align with the first one, while in the case of SLAM, external landmarks (allothetic information) are used to correct for the drift in odometry and generate a more consistent path.

6.2.2 Particle filter feature-based SLAM

As a part of component replacement scenarios, a different SLAM variant is deployed by exchanging the extended Kalman filter with a particle filter . This is reflected in the feature model in Fig 4.1 at the variation point for the filter functionality. Figure 6.3 shows the component diagram and highlights the modifications compared to the component diagram in Fig. 6.1. The changes are located in the meta-coordinator which replaces the EKF component with the par-



(a) EKF feature-based SLAM trajectory estimation along with the detected landmarks and the corresponding error ellipses.



(b) Dead reckoning using only wheel encoders. The error accumulation leads to an inconsistent trajectory estimation.

Figure 6.2: Robot's path estimation using EKF SLAM and dead reckoning.

ticle filter component. The interfaces are identical as the particle filter implements a feature-based map and processes the same motion and observation data.

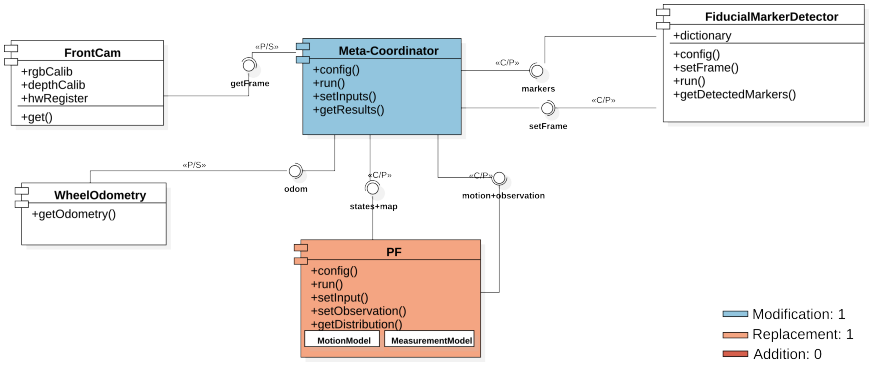


Figure 6.3: Particle filter feature-based SLAM component diagram.

Figure 6.4 shows the estimated path and map using the particle filter. The estimates are slightly more consistent compared to the EKF results in Fig. 6.2a.

6.2.3 Visual odometry-based SLAM

A visual odometry component is used to replace the wheel encoders. Figure 6.5 shows the corresponding component diagram highlighting the replacement of wheel encoders by a visual odometry component and the addition of an upward-looking camera along with two interfaces to receive the image data from the camera and pass it to the visual odomerty component.

The results in Fig. 6.6b show that both odometry techniques suffer from drift, however, the visual odometry results in a relatively more consistent path estimation compared to the wheel encoders. It is also noted that the error accumulation in both techniques is not identical as it introduces different biases leading to a slightly rotated version of the trajectory.

Figure 6.6a shows the corresponding path and map generated by the particle filter with visual odometry used as motion data. The result shows a more consistent path compared to the wheel encoders PF.

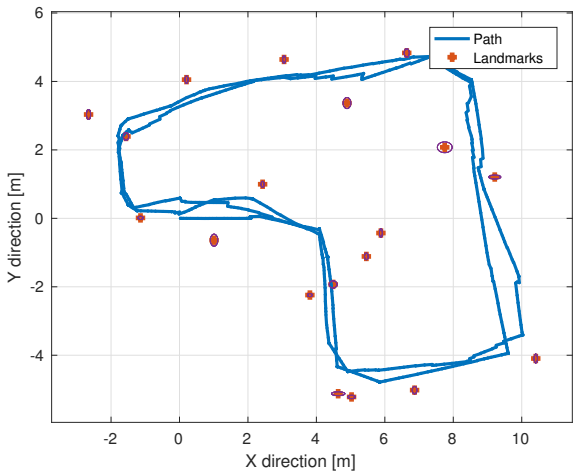


Figure 6.4: Particle filter feature-based SLAM path estimation along with the detected landmarks and their corresponding uncertainty ellipses.

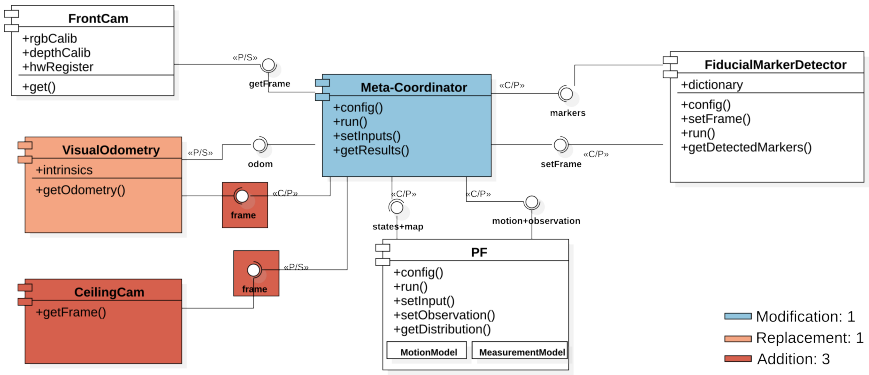
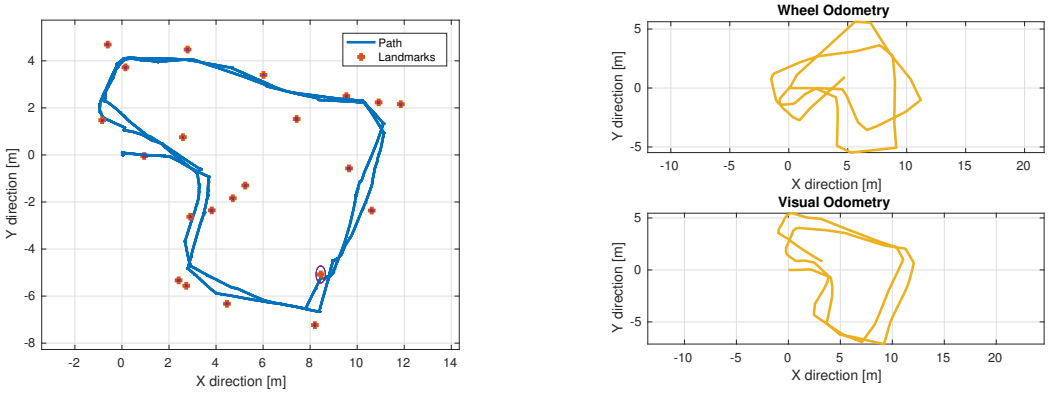


Figure 6.5: Wheel encoders replaced by visual odometry in the particle filter SLAM instance.



(a) Particle filter path estimation with feature-based maps using visual odometry.

(b) Visual odometry versus wheel odometry path estimations.

Figure 6.6: Robot's path estimation using EKF SLAM and dead reckoning.

6.2.4 Point-map visualization

A points map visualization component is added to the EKF SLAM realization in Fig. 6.1 as a part of component addition scenarios. Figure 6.7 shows the corresponding component diagram that describes the addition of a laser scanner and the additional visualization component along with their interfaces. The estimated pose by the EKF is used in the to the visualization component.

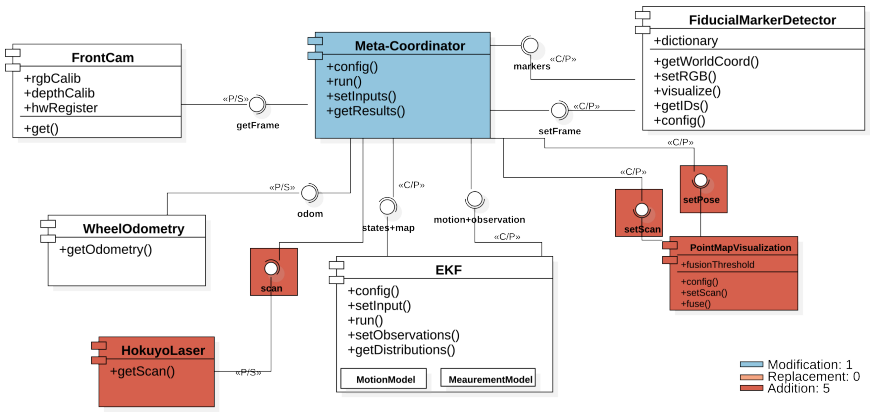


Figure 6.7: Points map visualization.

The generated points map is presented in Fig. 6.8 along with the EKF path estimate. The result shows an inconsistent map as the trajectory overlaps occupied areas at some parts of the map. However, the added map provides as relatively more intuitive description of the environment compared to the feature-based map.

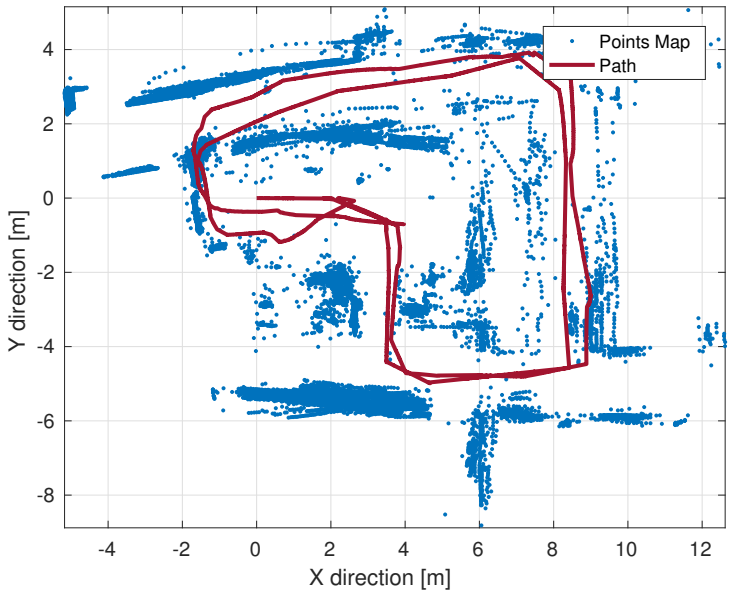


Figure 6.8: Points map visualization.

6.2.5 Particle filter grid-based map

As a part of logic modification scenarios, the particle filter will be extended to incorporate grid-based representation of the environment. The component diagram is presented in Fig. 6.9. It shows the added interface with the laser scanner, the modified meta-coordinator and the particle filter components along with their interface.

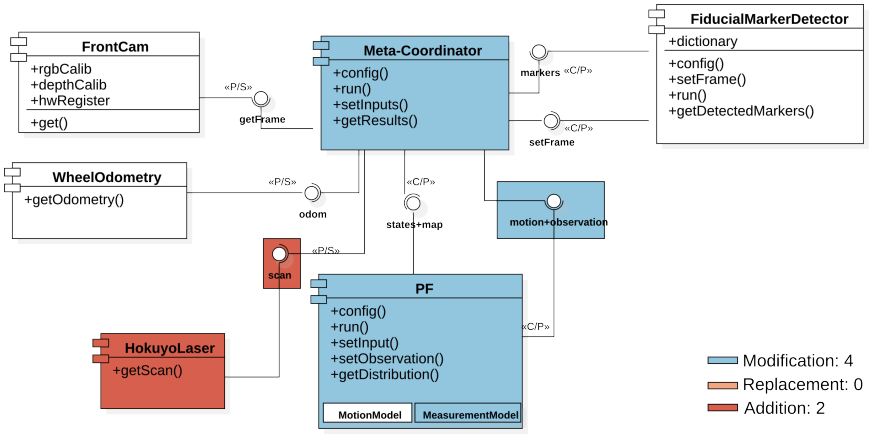


Figure 6.9: Modifying particle filter component to accommodate the addition of occupancy grid maps.

Figure 6.10 shows the generated occupancy grid map as a part of the particle filter along with the estimated robot trajectory and the detected landmarks. The result shows consistent path estimation as it aligns with the free spaces. The generated map is relatively consistent as well.

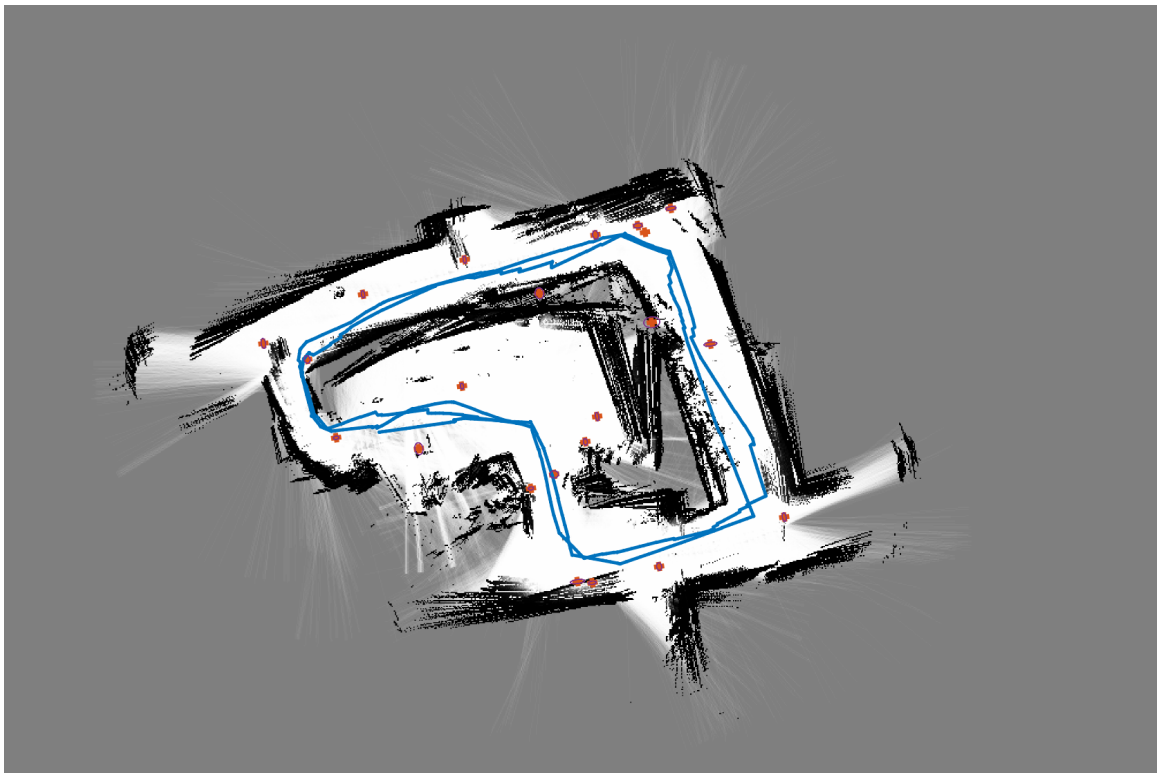


Figure 6.10: Occupancy grid map augmented with the estimated trajectory and the detected landmarks generated by the particle filter.

6.3 Discussion

This section discusses the results according to two aspects as specified in the introduction of the chapter, namely, the evaluation of SLAM and the reusability scenarios.

6.3.1 Trajectory & map estimation

Figures 6.2a, 6.4, 6.6a and 6.10 show the results of running different SLAM realizations using the same dataset. The outcome consists of the robot's path and the generated map of environment, where it can be seen that the different realizations were able to detect all the landmarks placed in the environment, estimate the path of the robot consistently, recover from odometry drift and perform loop-closures.

The volumetric map visualization component added to the EKF instance as shown in Fig. 6.7 did not yield a consistent map as the result indicates in Fig. 6.8. Due to the elementary fusion method employed that matches the map points based solely on euclidean distance. The map can be improved by using a more sophisticated technique such as scan-to-map optimization as Hess et al. (2016) proposed.

6.3.2 Use-case & growth scenarios

The component replacement scenario had two occurrences; the EKF was replaced with a particle filter and the visual odometry was used instead of wheel encoders. In the first occurrence The replacement effects are confined to the meta-coordinator which had to simply be adapted to instantiate the PF component instead of the EKF as shown in Fig. 6.1. In the second scenario the meta-coordinator was modified to instantiate the visual odometry component and add two new interfaces to pass the image data as presented in Fig. 6.5. The fact that the replacement did not have system-wide effects indicates that *modularity* as a first target RRL has been addressed.

A component addition scenario was realized by adding a point-map visualization to the EKF SLAM. The meta-coordinator was simple modified to instantiate a points map component and pass the laser scan and pose information through the added interfaces as shown in Fig. 6.7. An additional growth scenario is implemented by extending the particle filter to accommodate a grid map representation. Figure 6.9 shows the modifications in the particle filter component and the meta coordinator which is required to pass the laser scan data from the sensor to the filter.

The component addition and the logic modification indicates that the system is capable of growing beyond the current implementation without the need to rebuild the majority of the components to be adapted to the expansions, which implies that the system is *extensible*.

Generally, the derivation of the developed components into a working SLAM instance in the previous scenarios required minimal glue code and integration time. The interfacing consists of approximately 6 function calls per component. The common operations offered by all components are `construct()`, `configure()`, `set(...)`, `execute()`, `get(...)` and `destruct()`. Additionally, the performance of each component is configured through a separate configuration file that is parsed at deployment-time with no need to modify or re-compile the source code. Therefore, the components are considered coherent and adhering to the specified design patterns and the 5Cs, which improves the *standards compliance* RRL.

Finally, The hierarchical deployment of components as primitives (standalone C++ executables) followed by composites and finally as a fully working SLAM application helped expose faulty logic or interfaces early during the development phase, leading to a minimal integration phase in terms of time requirements and complexity. Improving the *testing & verification* RRL.

A summary of the correspondences between the results and the chosen RRLs is presented in Table 6.3.2

Scenarios	Experiments	Results	Target RRLs
Component replacement	• Replacing EKF with PF • Replacing wheel encoders with visual odometry	• Minimal changes; only meta-coordinator is adapted • Added interfaces to pass image data for visual odometry	Modularity & standards compliance
Component addition	A point-map visualization component is added to the EKF feature-based SLAM	Meta-coordinator is modified to add the component and the required interfaces for the laser scan and the pose.	Extensibility
Logic modification	The particle filter is modified to accommodate occupancy grid map	PF computation is adapted and laser scan interface is added.	Extensibility & modularity
Standalone deployment	Execute and test each primitive component	The individual components are separate C++ projects, thus, can be deployed independently from one another and from the middleware	Modularity & testing
Deployment-time configuration	Tune each individual component parameters	Configuration files are parsed at deployment time without modifying or re-building the source code	Standards compliance.

Table 6.2: A summary of the scenarios, experiments and the results showing the improvements to each of the traget RRLs: modularity, extensibility, standards compliance and testing & validation.

7 Conclusion & Future Recommendations

Considering the research objective mentioned in chapter 1:

" *Improve the reusability of SLAM through creating a development framework that accommodates different SLAM variations by applying relevant software engineering techniques.*"

A development methodology is proposed based on component-based software design, software product lines and the separation of concerns. Accordingly, a feature model is created highlighting the stable components, the variation points and the different strategies often found in SLAM. Furthermore, a number of elements are selected from the feature model and translated into software components in accordance with the proposed guidelines for component-level and system-level development.

RRLs are used as quality attributes to develop *use-case* and *growth scenarios* in order to test each chosen criterion of the *RRLs*. The evaluation shows that the *modularity*, *extensibility*, *standards compliance* and *testing & verification* have been positively addressed as the results of each scenario indicates.

Additionally, the derived SLAM realizations were able to generate consistent maps and path estimations for a robot navigating in a relatively small indoor environment with multiple loop-closures; confirming the applicability of the components.

The analysis along with the results achieve the stated research objective and show the added value and the potential of adopting component-based development paradigms in creating a reuse-oriented SLAM.

7.1 Future recommendations

Considering ongoing work in SLAM and software engineering paradigms, several themes for future work can be proposed:

- Extending the framework by refactoring the readily available open-source SLAM implementations in order to yield a library of coherent and reusable components that could be used to deploy and benchmark different SLAM strategies.
- Automatic code generation from the developed feature model to further shorten the deployment phase.
- Improving the RRLs which are not addressed in the framework, namely, *documentation*, *portability*, *packaging*, *support* and *intellectual property issues* in order to further enhance the framework.
- Run-time optimization for the developed components in order to allow for large scale mapping which requires extensive computational resources.
- Adopting the framework as a *"de-facto"* standard in the SLAM community, where it can be further extended to a more mature level in order to become a *commercial off-the-shelf* software product.

Appendices

A Appendix 1

This appendix showcases a number of auto-generated UML class-diagrams for open-source SLAM implementations. The architectures presented in figures A.1 and A.2 illustrates two different development styles commonly found, namely, an approach that utilizes object-oriented programming, however, no clear components' boundaries could be identified and the modules are tightly coupled and unbalanced. The second approach is a monolithic approach where a single code block performs the majority or all of the different SLAM functionalities.

The two instances serve as an indication to the current state of SLAM as a software product. Such designs violate the reusability criteria discussed previously as they require extensive efforts to be modified or extended.

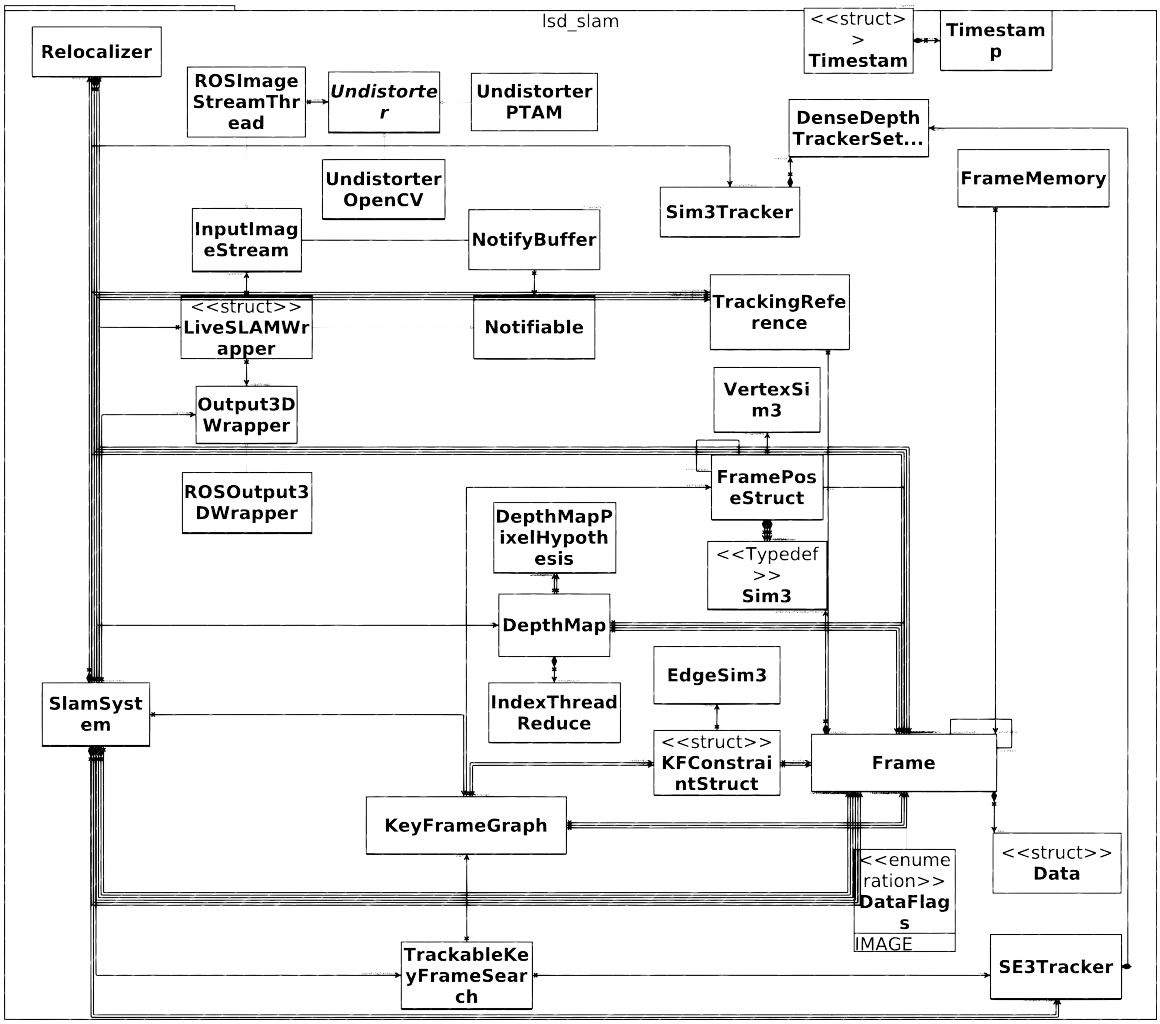


Figure A.1: An auto-generated UML class diagram from an open-source implementation of dense visual SLAM with tight coupling among components. All the attributes and functions are hidden to allow for visualization.

In contrast with the proposed framework that offers balanced components with a single point of entry as given in Fig. A.3.

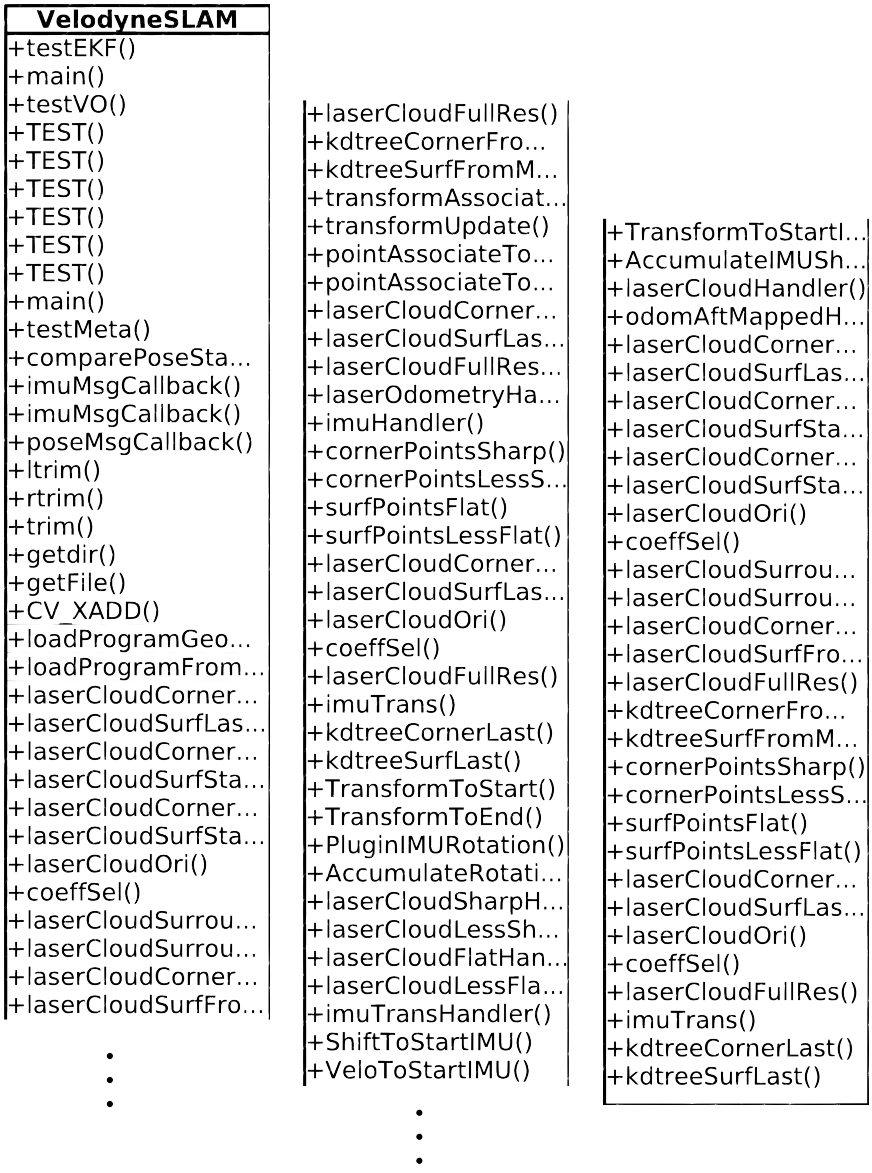


Figure A.2: An auto-generated UML class diagram from an open-source SLAM implementation using 3D laser scanners. The software is developed as a single class to implement all the different functionalities.

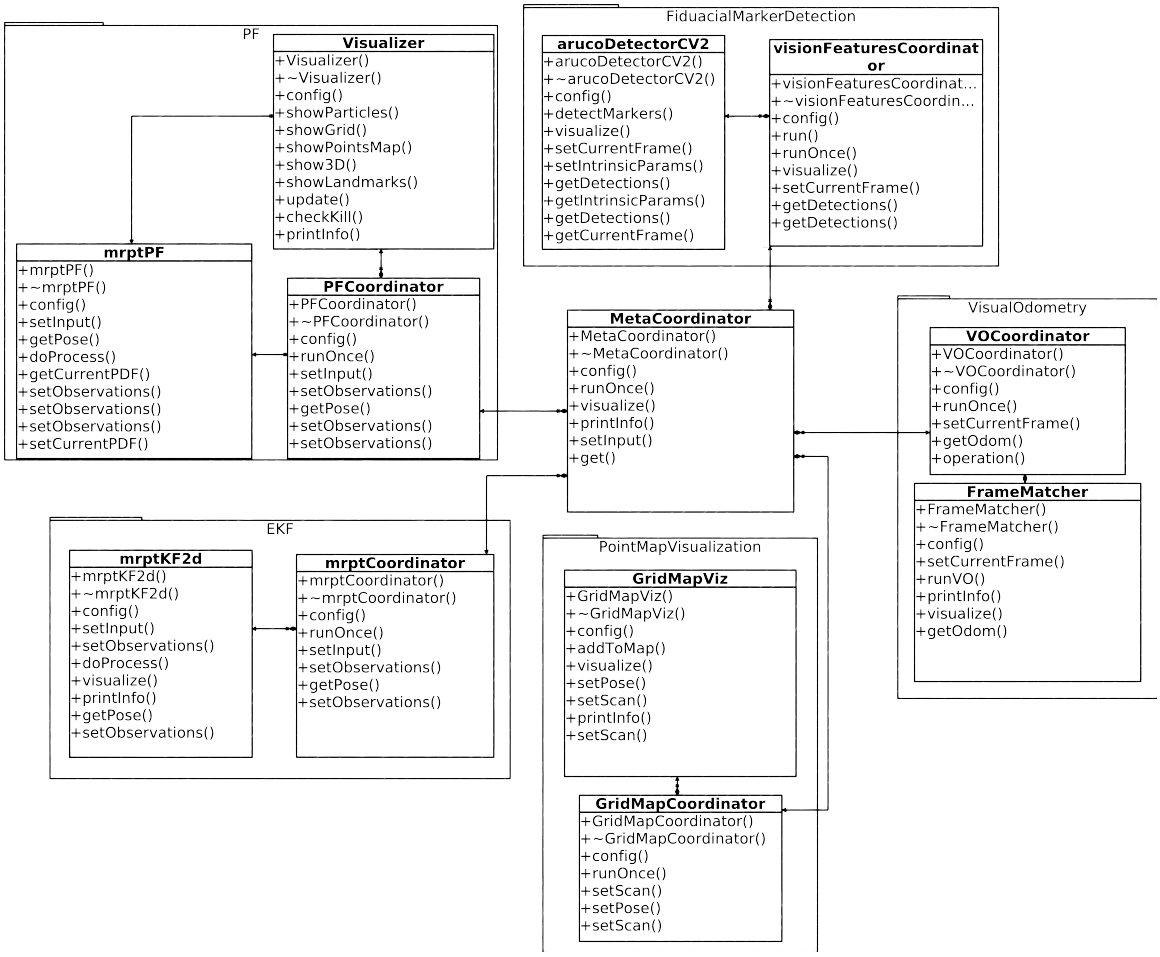


Figure A.3: Auto-generated UML class diagram showing SLAM realization using the proposed component-based paradigm. The

Bibliography

(2016), Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation, **vol. 1**, no.1, pp. 18–25, ISSN 23773766, doi:10.1109/LRA.2015.2505717.

Badger, J., D. Gooding, K. Ensley, K. Hambuchen and A. Thackston (2016), *HyperFlex: A Model Driven Toolchain for Designing and Configuring Software Control Systems for Autonomous Robots*, volume 625 of *Studies in Computational Intelligence*, Springer International Publishing, Cham, ISBN 978-3-319-26052-5, doi:10.1007/978-3-319-26054-9. http://www.scopus.com/inward/record.url?eid=2-s2.0-84958580682{&}partnerID=tZOtx3y1http://link.springer.com/10.1007/978-3-319-26054-9

Ball, D., S. Heath, J. Wiles, G. Wyeth, P. Corke and M. Milford (2013), OpenRatSLAM: An open source brain-based SLAM system, **vol. 34**, no.3, pp. 149–176, ISSN 09295593, doi:10.1007/s10514-012-9317-9.

Blumenthal, S., E. Prassler, J. Fischer and W. Nowak (2011), Towards identification of best practice algorithms in 3D perception and modeling, *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3554–3561, ISSN 10504729, doi:10.1109/ICRA.2011.5980106.

Brambilla, M., J. Cabot and M. Wimmer (2012), Model-Driven Software Engineering in Robotic, *IEEE Robotics & Automation Magazine*, p. 185, ISSN 1070-9932, doi:10.2200/S00441ED1V01Y201208SWE001. https://books.google.fr/books?id=2tVu-wC4XkAC

Brugali, D., L. Gherardi, A. Biziak, A. Luzzana and A. Zakharov (2012), 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, ISSN 03029743, doi:10.1007/978-3-642-34327-8_33.

Bruyninckx, H. (2011), Open RObot COntrol Software. http://www.orocos.org/

Bruyninckx, H., M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi and D. Brugali (2013), 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, doi:10.1145/2480362.2480693.

Cadena, C., L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid and J. Leonard (2016), Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age, **vol. 32**, no.6, p. 1309âĂŞ1332.

Davison, A. J., I. D. Reid, N. D. Molton and O. Stasse (2007), MonoSLAM: Real-Time Single Camera SLAM, **vol. 29**, no.6, pp. 1052–1067, ISSN 0162-8828, doi:10.1109/TPAMI.2007.1049. http://dx.doi.org/10.1109/TPAMI.2007.1049

Durrant-Whyte, H. and T. Bailey (2006), Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms, *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, **vol. 2**, p. 2006.

Foxlin, E. M. (2002), , no.IROS.

Frakes, W. and C. Terry (1996), Software Reuse: Metrics and Models, **vol. 28**, no.2, pp. 415–435, ISSN 0360-0300, doi:10.1145/234528.234531. http://doi.acm.org/10.1145/234528.234531

Fraundorfer, F. and D. Scaramuzza (2012), Visual odometry: Part II: Matching, robustness, optimization, and applications, **vol. 19**, no.2, pp. 78–90.

G. Tipaldi, K. A. (2010), FLIRT - Interest Regions for 2D Range Data, *International Conference on Robotics and Automation*, pp. 1–7, ISSN <null>, doi:10.1109/ROBOT.2010.5509864. papers2://publication/uuid/B7F0E570-4ACD-490E-97DA-F10D639F6380

Garrido-Jurado, S., R. Muñoz-Salinas, F. Madrid-Cuevas and M. Marín-Jiménez (2014), Automatic generation and detection of highly reliable fiducial markers under occlusion, **vol. 47**, no.6, pp. 2280–2292, ISSN 00313203, doi:10.1016/j.patcog.2014.01.005. http://linkinghub.elsevier.com/retrieve/pii/S0031320314000235

Gorton, I. (2011), *Essential Software Architecture*, Springer Publishing Company, Incorporated, 2nd edition, ISBN 3642191754, 9783642191756.

Grisetti, G., C. Stachniss and W. Burgard (2007), Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters, **vol. 23**, no.1, pp. 34–46, ISSN 1552-3098, doi:10.1109/TRO.2006.889486.

Hess, W., D. Kohler, H. Rapp and D. Andor (2016), Real-Time Loop Closure in 2D LIDAR SLAM, in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271–1278.

Kazman, R., M. Klein and P. Clements (2000), ATAM : Method for Architecture Evaluation, **vol. 4**, no.August, p. 83, ISSN CMU/SEI-2000-TR-004, doi:(CMU/SEI-2000-TR-004,ADA382629). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.6014{&}rep=rep1{&}type=pdf

Kohlbrecher, S., O. V. Stryk, J. Meyer, U. Klingauf, O. Von Stryk, J. Meyer, U. Klingauf, O. V. Stryk, J. Meyer and U. Klingauf (2011), A Flexible and Scalable SLAM System with Full 3D Motion Estimation, *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011*, pp. 155–160, ISSN 2374-3247, doi:10.1109/SSRR.2011.6106777.

Kraetzschmar, G., A. Shakhimardanov, J. Paulus, N. Hochgeschwender and M. Reckhaus (2010), Specifications of Architectures, Modules, Modularity, and Interfaces for the BROCRE Software Platform and Robot Control Architecture Workbench, *Best Practice in Robotics Brics Deliverable*.

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

Moore T. ; Stouch, D. (2014), A Generalized Extended Kalman Filter Implementation for the Robot Operating System, *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*.

Mullane, J., B.-N. Vo, M. D. Adams and W. S. Wijesoma (2008), A random set formulation for Bayesian SLAM, in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, IEEE, pp. 1043–1049.

Mur-Artal, R. and J. D. Tardos (2017), ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras, *IEEE Transactions on Robotics*, ISSN 15523098, doi:10.1109/TRO.2017.2705103.

Northrop, L. (2008), Software Product Lines Essentials.

Ratasich, D., B. Fromel, O. Hoftberger and R. Grosu (2015), Generic sensor fusion package for ROS, *IEEE International Conference on Intelligent Robots and Systems*, **vol. 2015-Decem**, pp. 286–291, ISSN 21530866, doi:10.1109/IROS.2015.7353387.

Ristroph, M. (2008), A Component-Oriented Approach to Simultaneous Localization and Mapping, pp. 1–14.

Siciliano, B. and O. Khatib (2016), *Springer Handbook of Robotics*, Springer Publishing Company, Incorporated, 2nd edition, ISBN 3319325507, 9783319325507.

Smith ', R., M. Self^ and P. Cheesemans (1987), Estimating Uncertain Spatial Relationships in Robotics*.

https://www.frc.ri.cmu.edu/{~}hpm/project.archive/reference.file/Smith,Self{&}Cheeseman.pdf

Song, B., S. Jung, C. Jang and S. Kim (2008), An introduction to robot component model for opros (open platform for robotic services), in *Proceedings of the International Conference Simulation, Modeling Programming for Autonomous Robots Workshop*, pp. 592–603.

Strasdat, H., J. M. Montiel and A. J. Davison (2010), Real-time monocular SLAM: Why filter?, *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2657–2664, ISSN 10504729, doi:10.1109/ROBOT.2010.5509636.

Szyperski, C. (2002), *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, ISBN 0201745720.

Thrun, S., W. Burgard and D. Fox (2005), *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*, The MIT Press, ISBN 0262201623.

Tuytelaars, T. and K. Mikolajczyk (2008), Local Invariant Feature Detectors: A Survey, **vol. 3**, no.3, pp. 177–280, ISSN 1572-2740, doi:10.1561/0600000017.
http://dx.doi.org/10.1561/0600000017

Visser, J. (2016), *Building Maintainable Software*, ISBN 9781491944349.

Wang, N., D. C. Schmidt and C. O'Ryan (2001), Component-based Software Engineering, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, chapter Overview of the CORBA Component Model, pp. 557–571, ISBN 0-201-70485-4.
http://dl.acm.org/citation.cfm?id=379381.379581