UNIVERSITY OF BREMEN
INSTITUTE FOR
ARTIFICIAL INTELLIGENCE

Automated Design Synthesis of Perception Systems for Industrial Applications

Vincent Dietrich

Vollständiger Abdruck der vom Fachbereich 3 (Mathematik und Informatik) der Universität Bremen zur Erlangung des akademischen Grades eines

Doktors der Ingenieurswissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr.-Ing. Udo Frese *Universität Bremen*
1. Prüfer:	Prof. Dr. h.c. Michael Beetz, PhD *Universität Bremen*
2. Prüfer:	Prof. Dr.-Ing. Jens Lambrecht *Technische Universität Berlin*
Beisitz:	M.Sc. Franklin Kenghagho Kenfack *Universität Bremen*

Die Dissertation wurde am 16.03.2021 bei der Universität Bremen eingereicht und durch den Prüfungsausschuss am 26.05.2021 angenommen.

Abstract

The capability to perceive the working environment is crucial for autonomous and automated systems. For instance, an assembly robot needs to know accurately the position of the assembly parts in order to grasp them successfully. However, the design synthesis of perception systems is associated with high engineering efforts due to a large number of interdependent design choices and high reliability and accuracy requirements. This is especially problematic for flexible assembly system, where an effortless adaptation to new products is necessary. The design process includes different steps. First, one or multiple sensors must be chosen from a large set of alternatives and placed in the working environment. Then, data processing steps must be chosen, combined, and parameterized in order to form a working perception pipeline. The space of possible system configurations is large and the solution space comparatively small. Noise and uncertainty are immanent in the data and have a strong effect on the system performance. Additionally, small variations in task, environment, and parameterization can decrease the performance or render the entire system dysfunctional.

In this thesis, models and methods are introduced and investigated in order to automate the perception system design process for flexible assembly systems. First, a common representation grounded in set-theory is introduced, which allows to represent procedural and declarative models for perception systems hierarchically, i.e. on different levels of abstraction. The hierarchical structure allows to represent different levels of computationally less demanding approximations of the models, which facilitates an efficient exploration of the space of system configurations. Based on the common representation a design synthesis method is introduced, which allows to jointly optimize the structure and parameterization of perception pipelines. In order to address geometric uncertainties, we propose an uncertainty model based on factor graphs, which is embedded in a planning-based synthesis approach and allows to efficiently estimate pose uncertainties of sequences of perceptual actions. Finally, a synthesis method based on hierarchical planning is introduced. This hierarchical approach increases the synthesis efficiency and enables the runtime adaption and failure handling of perception system. The contributions are validated on the example of industrial assembly applications in simulated and real-world environments.

Zusammenfassung

Umgebungswahrnehmung ist eine Kernfähigkeit für autonome und automatisierte Systeme, wie beispielsweise Roboter und Fertigungsanlagen. Ohne eine zuverlässige und ausreichend genaue Lokalisierung von beweglichen Objekten im Arbeitsraum ist ein vollautomatischer Betrieb nicht möglich. Die Auslegung von Wahrnehmungssystemen ist jedoch mit hohem Aufwand verbunden. Passende Sensoren müssen platziert und eine Vielzahl von Datenverarbeitungsschritten in geeigneter Weise parametrisiert und kombiniert werden. Dies ist beim Einsatz von flexiblen Automatisierungssystemen problematisch, da hier eine schnelle Anpassung an neue Produkte benötigt wird. Der resultierende Konfigurationsraum ist riesig und der Lösungsraum klein. Rauschen und Unsicherheit, insbesondere auch geometrischer Natur, sind omnipräsent in Roh- und weiterverarbeiteten Daten, was Fehlinterpretationen nach sich zieht. Schon kleine Änderungen an Aufgabe, Umgebung und Parametrisierung des Wahrnehmungssystems können die Erfolgsquote erheblich reduzieren oder zum Scheitern der Wahrnehmung führen.

Im Rahmen dieser Arbeit werden daher neuartige Modelle und Methoden für die Automatisierung des Auslegungsprozesses eingeführt und untersucht. Zunächst einmal werden gemeinsame Repräsentationen und Schnittstellen auf Basis der Mengenlehre eingeführt, die es erlauben prozedurale und deklarative Modelle für Wahrnehmungssysteme auf verschiedenen Abstraktionsebenen hierarchisch zu beschreiben. Dies ermöglicht eine effiziente Exploration des Konfigurationsraumes mithilfe abstrakter und weniger rechenintensiver Surrogat-Modellen. Aufbauend auf der gemeinsamen Repräsentation wird ein Ansatz zur Synthese von Datenverarbeitungsketten eingeführt, der es ermöglicht die Struktur der Datenverarbeitungskette und die individuellen Parameter der Operatoren gemeinsam zu optimieren. Zur Berücksichtigung von geometrischen Unsicherheiten wird ein Ansatz eingeführt, der es ermöglicht Unsicherheiten explizit mithilfe von Faktor-Graphen zu repräsentieren, abzuschätzen und in der Synthese zu berücksichtigen. Abschließend wird eine Synthesemethode auf Basis von hierarchischer Planung beigetragen, die durch Ausnutzung des hierarchischen Modells die zeitliche Effizienz der Synthese steigert.

Dadurch wird eine Anpassung und Reaktion auf Fehler zur Laufzeit ermöglicht. Die wissenschaftlichen Beiträge und Ansätze werden in Simulation und auf echten Robotersystemen am Beispiel von Montageaufgaben validiert.

Acknowledgments

This thesis has been written at Siemens Corporate Technology in Munich in collaboration with the Institute for Artificial Intelligence from the University Bremen. It would not have been possible without the support of many people, whom I want to thank for their contributions and their help.

First, I would like to express my gratitude to my supervisor Prof. Dr. h.c. Michael Beetz for numerous fruitful discussions and his guidance, which made this thesis possible. Michael shaped my scientific approach and always provided me with insights to improve my research.

Furthermore, I would like to thank the entire team at the Institute for Artificial Intelligence for supporting me in organizational and technical matters and for making my stays in Bremen very enjoyable.

Moreover, I am deeply grateful for the opportunity to write this thesis in the research group of Dr. Georg von Wichert. It is a great place to work, do research, and implement cutting edge robotics solutions. For the splendid time, a lot of proof reading, and always interesting scientific and non-scientific exchange, I would like to thank the entire research group and supporting team. A tribute goes to Georg, who always supported me personally and in my dissertation project and provided me with all necessary resources. A special thank, furthermore, goes to my friends and fellow doctoral candidates Bernd Kast, Philipp Schmitt, and Florian Wirnshofer with whom I spent days and nights implementing and debugging robotics software, discussing, and, of course, playing competitive matches of table soccer. I also want to display my deep gratitude to Dr. Sebastian Albrecht and Michael Fiegert, who supervised and guided my research and spent countless hours for discussions and corrections. A particular appreciation goes to Dr. Wendelin Feiten for inspiring discussions and always opening up out-of-the-box views.

Finally, I am deeply grateful for my parents Gabriele and Wolf and my sisters Larissa and Karolin, who raised and supported me, and made me who I am. I also want to thank my girlfriend Michelle from all my heart for her love and support, without which this thesis would not have been possible.

July 20, 2021,

Vincent Dietrich

Contents

Symbols

d	data point in data set
$\mathbb{D}$	dataset
I_{in}	operator input set
i	instance
I	instance set
O	operator
$\mathrm{I}_{\mathrm{out}}$	operator output set
P	perception pipeline
$\mathbb{P}$	perception pipeline set
f_{IO}	abstract operator input to output function
f_{T}	abstract operator runtime function
m	metric
$\mathbb{M}$	metric space
e	factor graph edge
$\mathcal{E}$	factor graph edge set
x	factor graph variable
$\mathcal{V}$	factor graph variable set
C	concept
B_{Γ}	concept base

Γ	concept class
ϕ	factor
F_{G}	factor graph
$\mathcal{U}$	factor set
$\mathcal{T}$	task
γ	configuration
Γ	configuration space
Σ	standard deviation matrix
$\mathcal{F}$	frame
K	intrinsic camera matrix
λ	operator parameterization
Λ	operator parameterization space
ψ	pipeline structure
Ψ	pipeline structure space
θ	sensor positioning
Θ	sensor positioning space
p	pose
$\oplus$	pose composition operator
$\ominus$	pose inversion operator
p_{rot}	rotation
ϕ	sensor choice
Φ	space of sensor choices
p_{pos}	translation
$\mathcal{A}$	action set
bel	belief

$\mathcal{F}$ fact set

g goal

$\mathcal{G}$ goal set

o object

$\mathrm{Post}(,)$ post-condition

$\mathrm{Pre}()$ pre-condition

σ standard deviation

s state

S state set

a action

f fact

List of Figures

List of Tables

Chapter 1

Introduction

Automation in manufacturing enables the efficient production of industrial and consumer goods. From basic products such as screws up to highly complex products such as airplanes and cars, numerous steps are required in the manufacturing process. In a simplified example of a gearbox, a housing must be cast, machining has to provide accurate surfaces, housing and gears must be transported to an assembly station, where the gearbox can finally be assembled. Automation ensures consistent quality levels, high production output and low production costs. It therefore enables to efficiently satisfy the demands of modern production systems.

The term automation is rooted in the Greek word "automatos", which signifies "self-acting", (Gupta and Arora, 2009). An automation system is therefore characterized by the ability to perform an action independently, without external control. An automation system is comprised of actuators, sensors and control units. Actuators, such as robot arms and grippers are used to physically interact with the world, e.g., to grasp parts and to assemble products. Sensors provide information about the state of the world, for instance the locations of available objects or attributes thereof. The control units interpret incoming sensor data and provide commands for the actuators.

Three different types of automation in manufacturing can be differentiated: fixed, programmable and flexible automation (Gupta and Arora, 2009), see also Fig. 1.1. Fixed automation is realized using custom designed, special purpose hardware and typically addresses very high production volumes and specialized processes. An example is the production of car bodies, where high capital investment is required for the special machinery, which is then able to produce the same product at very high throughput for multiple years. Programmable automation is characterized by the reuse of general-purpose equipment, such as a robot, which is reprogrammed manually for each product. It is characterized by hardware that can be adapted by rewriting the program. A milling machine with computerized numerical control (CNC), for example, can be reprogrammed for each part to be processed. Due to the programming effort, it is suitable especially for medium production batch sizes, such as a dozen or few hundred pieces and upwards. With additional software tooling for the automated generation of a CNC program flexible automation is feasible. Flexible automation addresses production environments where the product or the environment changes often and only low number of parts are produced. Here, the cost to adapt the automation system to a new product must be minimal, even if this means that the production process itself is slower. In the ideal case, the adaptation to a new product is software-defined and fully automatic. However, an adaptation of the hardware is often necessary and can be realized for instance via automated tool changing systems.

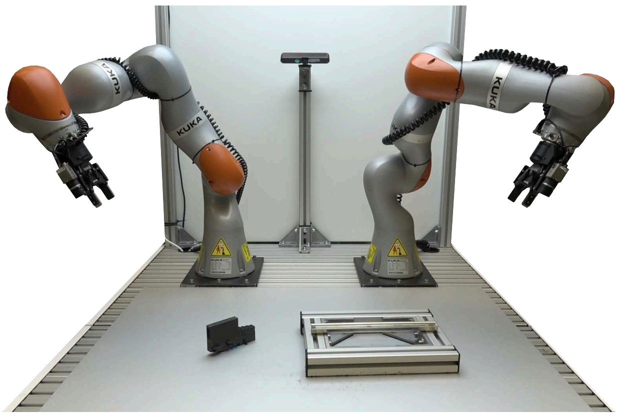


Figure 1.1: *A flexible automation system for assembly. It does not use fixtures at all and is characterized by an advanced control system which allows to adapt the program automatically to new tasks. Here, the task is to mount an electric cabinet module on a hat rail. No fixtures are given and the parts must be localized. Therefore, perception is key for this kind of flexibility.*

The need for automation and especially flexible automation is increasing due to several factors. First, most western countries have an aging population. According to (Fuchs et al., 2017), the workforce in Germany will decrease by 6 million persons until 2030 if only demographic effects are taken into account. This will result in a major labor shortage if automation cannot fill the gap. Another important aspect is the global competition, which forces local manufacturers to increase their productivity. Often, automation is the most effective solution to increase productivity. Furthermore, product life-cycles are decreasing with higher innovation speeds. At the same time, product customization and variability increase. This results in an increasing need for flexible automation.

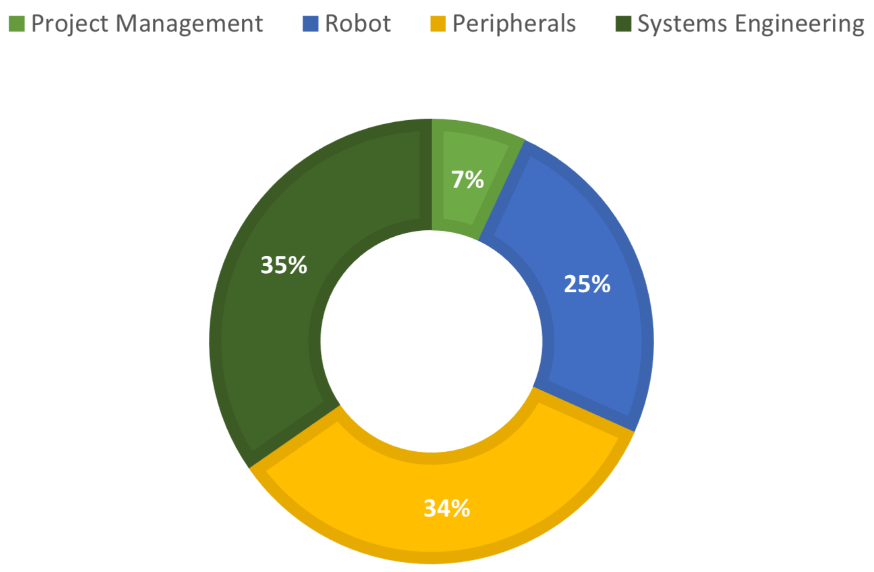


Figure 1.2: *Relative costs for the design and deployment of a robotic spot welding automation solution (Boston Consulting Group, 2015).*

However, the cost for automation is still high, especially due to the engineering effort involved. In Fig. 1.2 a distribution of the costs of a robotic automation system for spot welding in the automotive industry is shown. The systems engineering, which includes programming, is a major cost point, which inhibits cost effective adaptation to varying products. Therefore, software-defined design approaches are required in order to reduce and automate the engineering for automation systems. Here, *software-defined* means, that new designs are created and validated in a software environment without the use of hardware and real tests.

Ideally, such approaches can be used and reused in the distinct phases of *offline engineering* and *online adaptation*. Offline engineering addresses the initial design of an automation system, where there is still freedom in the design decision. Online adaptation addresses the reconfiguration of an automation system, when the product or production process changes, or when unexpected situations or failures occur.

Perception plays a crucial role for cost effective, flexible automation. Flexible automation systems have to cope with unstructured environments, where the exact locations of the parts can be unknown and have to be inferred using a perception system, see Fig. 1.1. In a fixed automation system on the contrary fixtures are used, which are special product specific hardware parts that cannot be re-used across different products. The flexible automation system in Fig. 1.1 consists of two industrial robot arms with grippers as actuators and multiple cameras as sensors. The task is to assemble an electric cabinet, which is a good example for a task that requires flexible automation. The typical production volume for electrical cabinets is low and the degree of customization high. The required parts are placed loosely on the table. Without a way to estimate the position of the parts, successful manufacturing is impossible. This is what the perception system provides. The perception system alone does require significant engineering effort and as key enabler for flexible automation, this thesis will address the automated design synthesis of perception systems.

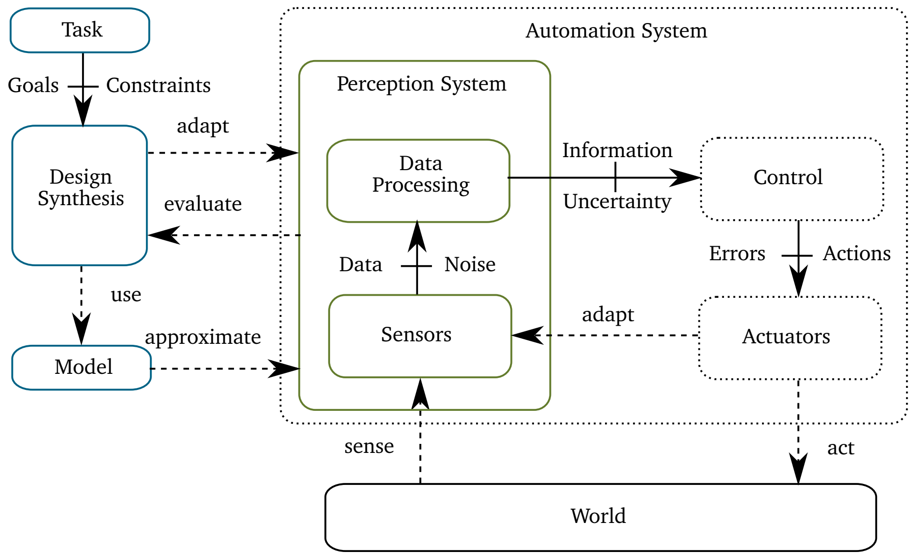


Figure 1.3: *High-level overview of design synthesis of perception systems in industrial applications. The perception system is part of the control loop of an automation system, with inherent noise and uncertainty. The design synthesis considers goals and constraints from a task definition. It can adapt and evaluate the perception system in order to satisfy the task.*

A high-level overview of design synthesis of perception systems is given in Fig. 1.3. An automation system interacts with the world via a control loop which involves sensing and acting. Sensing is done via *sensors*, which we define as:

Definition 1 - Sensor

A sensor is a physical device, which is capable of converting physical properties or environmental attributes into machine readable data (Mitchell, 2012).

There are many different types of sensors, such as radar sensors, lidar sensors, and cameras. Let us consider the example of a camera. A common 2D camera converts light that is reflected from the surfaces of a scene into an image, an array of digital values which encode the light intensity at different positions on the screen of a camera. So light is converted into intensity data.

The raw intensity data of a camera is not useful for most applications, unless the application requires an estimate of the light intensity. Therefore, data processing is required in order to extract information. For instance, object recognition algorithms can be applied in order to recognize the type of an object seen in the image. The description of a scene via an object category is a more condensed representation as the intensity data and allows more complex decision taking. We define data processing as:

Definition 2 - Data Processing

Data processing is the “collection and manipulation of items of data to produce meaningful and condensed information”, Oliver et al. (1992, p. 2).

Information in general is the structured, organized, and processed data, which allows to answer relevant questions or take decisions. Information can be associated with uncertainty. In the context of automated industrial applications, such as assembly, logistics, and bin-picking, relevant information includes the location of objects, product identifiers, and the presence of defects.

A perception system is now the sum of components which are required in order to derive the necessary information of a task. More formally, the term perception system can be defined as:

Definition 3 - Perception System

A perception system provides task relevant information about its environment. It is comprised of sensors, data processing, and computation units.

An exemplary perception system is depicted in Fig. 1.4. The data from the sensor is processed by a sequence of parameterized steps, which yield the required information, for instance the position of an object.

In Fig. 1.3 the relation between task, design synthesis and perception system is highlighted as well. The task defines goals and constraints, which are to be fulfilled within a design synthesis process. Therefore, the perception system needs to be adapted and evaluated. This can be done in virtual environments if a simulation of sufficient quality is available. The high-level goal of this work is to reduce the engineering effort for automation systems, specifically the perception sub-system. In the next section the engineering and synthesis process are introduced in more detail.

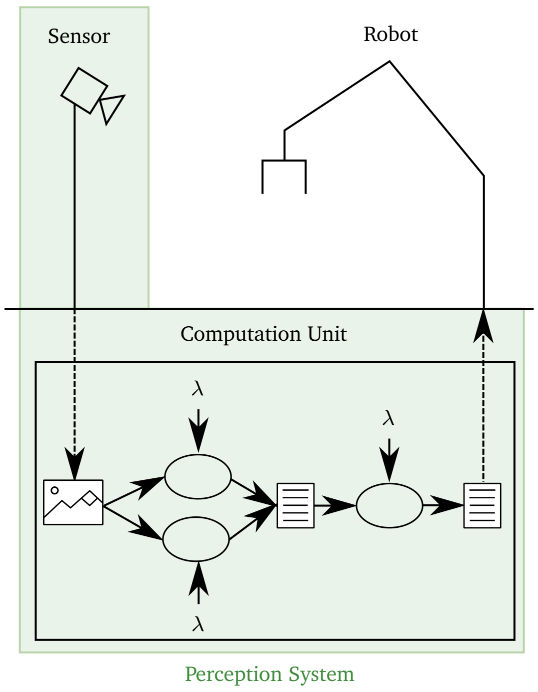


***Figure 1.4:** Exemplary perception system with a single sensor and a computation unit, where a data processing pipeline comprised of parameterized operators is executed in order to generate task relevant information. The symbol λ denotes the parameters for the operator.*

1.1 Perception System Design Synthesis

The design synthesis of perception systems is embedded in the overall systems engineering process. According to (Lightsey, 2001) the process can be structured via three major steps, which are in brief:

- **Requirement analysis**: Definition of system requirements by considering customer, engineering team, and management, as well as laws and regulation.
- **Functional analysis / allocation**: Definition of functional interfaces as well as a functional architecture by a decomposition and distribution of the system requirements.
- **Design synthesis**: Transition from functional architecture to physical architecture by selecting and parameterizing suitable components.

The steps are interconnected and an iteration between them can be required. We can define system design synthesis as:

Definition 4 - System Design Synthesis

System design synthesis is a part of the overall systems engineering process. “The objective of design synthesis is to combine and restructure hardware and software components in such a way as to achieve a design solution capable of satisfying the stated requirements.”, Lightsey (2001, p. 57).

The result of the system design synthesis is a system configuration within the available configuration space. It can be defined as:

Definition 5 - System Configuration

A system configuration denotes the selection, parameterization, and combination of hardware and software components for a specific task. It represents a point in the multi-dimensional configuration space.

We now discuss the systems engineering process for perception systems based on an example. The engineering of perception systems is closely linked with the target application and is typically performed by a perception engineer. Let us consider an assembly application as displayed in Fig. 1.5. Here, a peg must be inserted into a hole, which is a common assembly task. For instance within the manufacturing of gear boxes, a shaft must be inserted into gears, bearings or the housing.

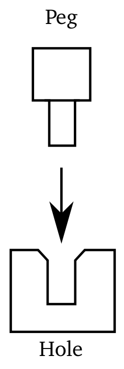


Figure 1.5: *Peg in hole task*

We target a flexible automation system, where no fixtures can be designed in an economically feasible manner. Here, the parts are delivered loosely without exact information about their positions. How does a perception engineer approach the task of designing a perception system for such an application?

First, the application requirements must be clearly defined. For the assembly procedure it is important, that both objects are well localized and aligned, such that the assembly procedure can be performed successfully. As depicted in Fig. 1.6 on the left side, when peg and hole are well aligned, successful assembly is possible. If not, a collision may occur, which prevents the assembly and might even destroy or deteriorate the product. In general, the task defines a dependency of the success probability of the assembly process on the localization accuracy provided, c.f. Fig. 1.6, right. This dependency can be used to determine a threshold for the accuracy of the localization as requirement. Note here, that force-feedback controlled robot movements can be used in order to decrease the accuracy requirements. However, search motions may leave scratches on the surface and can therefore not be applied in arbitrary applications. Furthermore, tactile sensing is also part of the perception domain.

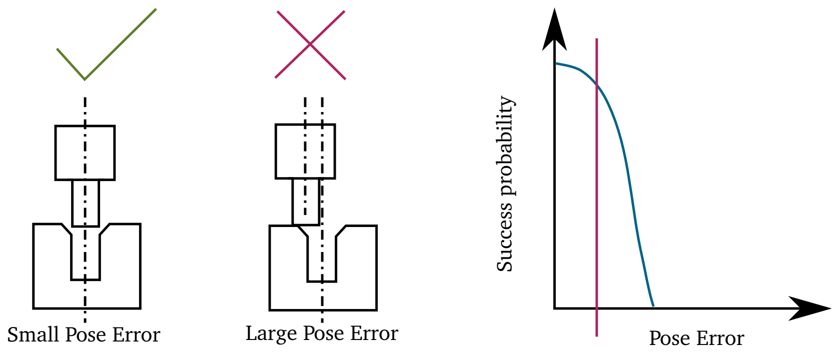


Figure 1.6: *Process requirements for peg-in-hole task. Large pose errors can lead to failures within the assembly.*

In practice, we have an interdependency between the product design, the design of the overall automation system, and the design of the perception system, which go hand in hand. The product designer can enlarge the chamfers of the peg in order to reduce the requirements of the localization accuracy. The automation system designer can provide fixtures for improved localization. This is especially true for the offline engineering phase, when a new product and automation system is created from scratch. Here, it would be desirable to represent all the different design decision in a single software-defined framework, that may consider all at once. But a prerequisite thereof is that the individual design processes can be automated as well in a software-defined manner. Consequently, for the design of the perception system we can assume, that the requirements are quantified, and the problem is to design a system that fulfills them. Additionally, different types of requirements exist. There are functional requirements like the minimum accuracy and there are non-functional requirements such as the overall cost of a system and preferred hardware or software standards.

With the set of requirements at hand, the perception engineer now must select appropriate hardware and software modules, i.e., sensors, data processing and computing units in order to fulfill the task requirements. The computing unit is often given by the automation system, but also specialized hardware might be required such as graphics processing units and accelerators for neural networks. But the selection of the different components depends on each other and the task. Algorithms that require color images cannot be applied on depth data. Sensors which produce high quality data can be too expensive for the task at hand. The sensor might not be suitable for the environment or the targeted objects. Specific computing units may not be suitable or too slow to run the selected algorithms. Additionally, the individual components must be parameterized. The camera must be positioned and a large number of individual parameters of the data processing pipeline must be set.

A perception engineer uses different sources of information for this task. First, during his training he would learn mathematical models and abstractions, which allow to estimate system characteristics. For instance, he would learn to quantify the error characteristic of stereo cameras and how it depends on the distance between object and camera. Furthermore, he would determine to read and interpret datasheets of sensors and how to understand performance evaluations of different perception algorithms. In general, we can define a model as:

Definition 6 - Model

A model is an abstract description of the real world giving an approximate representation of more complex functions of physical systems (Papalambros and Wilde, 2000).

So, a perception engineer uses models in order to make predictions about the performance of specific perception system configurations. But not only mathematical models are used, also experience plays an important role, which can be seen as a data-informed model. The engineer for instance knows by experience, that specific algorithms do not work well for small or symmetric objects. Both types of models can help to take promising design decisions early on.

Another important source of insight are experiments. At some point it is required to try out and test the system configuration in order to reduce the error induced by model approximations. Here it is required that the tests cover the range of situations, that the perception system might be confronted with. So even system tests can only approximate the system's performance across its application spectrum, as it is often not possible to test every variation.

In Fig. 1.7 the overall task of the perception engineer is visualized. First, he receives a description of the task together with a specification of the requirements, for instance, the targeted pose estimation accuracy. Then he chooses appropriate hardware and software components from the available set. Finally, he assembles the components together and parameterizes each of them. This includes the positioning of the sensor, the combination of different algorithms as well as the choice of individual parameter values.

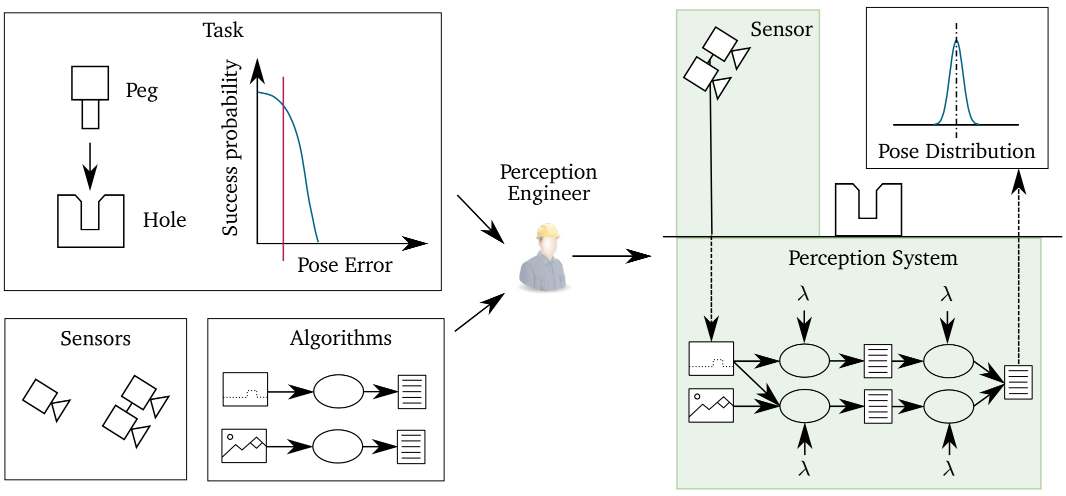


Figure 1.7: *Overview of the system design by a perception engineer. A perception engineer chooses sensors and algorithms in order to design a perception system which fulfills the task requirements.*

Different system configurations have impact on the performance of the system. Consider the three different setups in Fig. 1.8. In the solution instance A, a stereo camera is used that provides point clouds and RGB images. Therefore, a pipeline can be designed that works on both modalities and provides small pose errors. In solution B only a RGB camera is used and placed at the same position. This setup results in a lower pose accuracy. A repositioning of the RGB camera closer to the object, as shown in solution C can improve the accuracy to a level of solution A.

The configuration space consists of the sensor choice, the system setup, the pipeline structure, and the pipeline parameterization, as depicted in Fig. 1.9. The choice of the sensor primarily defines the type of the available sensor data and the noise characteristic that it exhibits. Depth and RGB are two examples for commonly used data modalities in the context of industrial automation. Others include radar or infrared images, which for instance are used for autonomous driving. Different sensors can have distinct noise characteristic (Pomerleau et al., 2012). A good example are stereo cameras that provide depth data. Here, the depth error is inverse proportional to the distance between the cameras, the baseline (Gallup et al.,

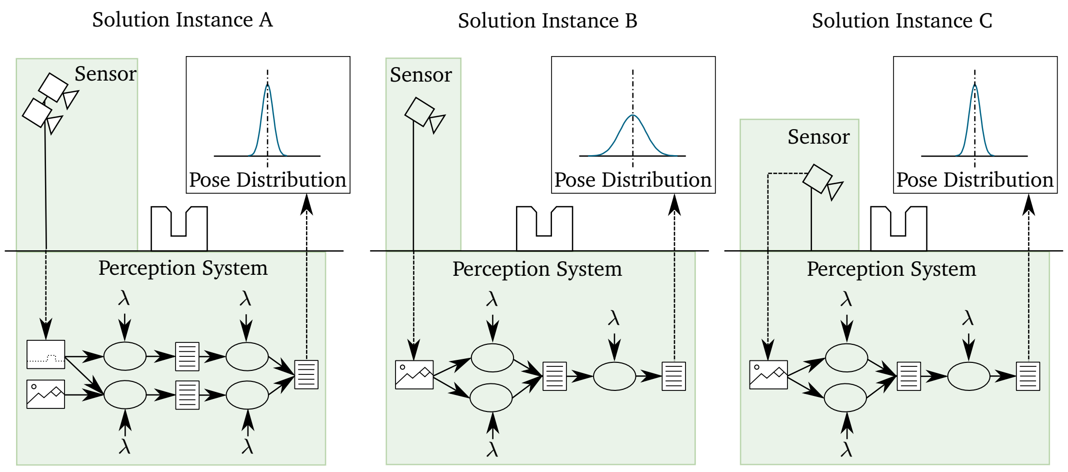


Figure 1.8: *Performance comparison of different perception systems. Depending on the sensor type, placement, and data processing different pose error distributions are achieved.*

2008). Thus, different stereo cameras with different baselines have a different noise characteristic.

Another configuration space dimension is the hardware placement, which comprises the positioning of the camera and possibly object fixtures and additional lighting. Especially the positioning of the camera has a strong influence on the noise, which increases with higher distance between camera and scene for common depth sensors (Nguyen et al., 2012). Therefore, it is important to choose the positioning well according to the task.

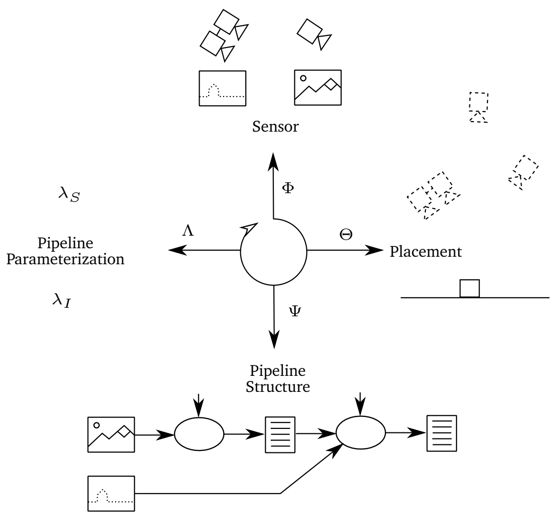


Figure 1.9: *Configuration space dimension: sensor choice, sensor placement, pipeline structure, and pipeline parameterization.*

Furthermore, the third dimension of the configuration space is the perception pipeline structure. Different perception algorithms for filtering, object detection, pose estimation, pose refinement, etc. can be combined in order to form a pipeline. Often it is necessary to combine multiple algorithms, as an individual one does not perform sufficiently. Finally, the algorithms must be parameterized, which is the configuration space dimension of the pipeline parameterization.

1.1.1 Problem Statement

The perception system design problem can be formalized as a design optimization problem (Papalambros and Wilde, 2000). The task is denoted as $\mathcal{T}$. First, we define a perception system configuration γ from the configuration space $\Gamma(\mathcal{T})$, as a tuple:

$$\gamma = \langle \phi, \theta, \psi, \lambda \rangle, \tag{1.1}$$

where

- ϕ denotes a sensor from the set of available sensors $\Phi(\mathcal{T})$,
- θ denotes a sensor placement from the set of possible sensor placements $\Theta(\mathcal{T})$,
- ψ denotes the structure of a data processing pipeline from the set of possible data processing pipelines $\Psi(\mathcal{T})$, and
- λ denotes a parameterization of the data processing pipeline from set of possible parameterizations $\Lambda(\mathcal{T})$.

The configuration space and its dimensions depend on the application. Now, the optimization problem can be formalized as (Papalambros and Wilde, 2000):

$$\begin{aligned}
&\min_{\gamma} && f(\gamma, \mathcal{T}) && \text{(1.2a)}\\
&\text{subject to} && h_i(\gamma) = 0, i \in H(\mathcal{T}) && \text{(1.2b)}\\
& && g_j(\gamma) \geq 0, j \in G(\mathcal{T}) && \text{(1.2c)}\\
&\text{and} && \gamma \in \Gamma(\mathcal{T}), && \text{(1.2d)}
\end{aligned}$$

where

- γ is the system configuration vector,
- $f(\gamma, \mathcal{T})$ is the objective function,

- $h_i(\gamma)$ are a set of m_1 equality constraints,
- $g_j(\gamma)$ are a set of m_2 inequality constraints, and
- Γ is the configuration space for the presented problem.

For the application in industrial pose estimation tasks, different target variables can be addressed as for instance the system cost, the reliability and the runtime. This, however, is application specific. An exemplary objective function which addresses the aforementioned variables is given in the following:

$$f(\gamma, \mathcal{T}) = c(\gamma) + w_1 * r(\gamma, \mathcal{T}) + w_2 * t(\gamma, \mathcal{T}) \tag{1.3}$$

where

- c represents the cost of the system,
- r represents the reliability of the system,
- t represents the perception runtime of the system, and
- w_1 and w_2 are weights which allow to prioritize between the different parts of the objective function.

The objective of this thesis is to provide modeling and synthesis methods in order to facilitate and automate the design synthesis process. The overall approach should address the main challenges, which arise in the problem domain and which are introduced in the following.

The domain which serves as concrete example and evaluation target within this thesis is pose estimation for programmable and flexible automation. Pose estimation denotes the process of determining the translation and orientation of a known object or object primitive with respect to a reference coordinate frame. It may entail the use of different information sources, such as sensory data, e.g., color images, point clouds, and force data, and available scene knowledge, such as supporting planes. Pose estimation can be performed passively, by evaluation of already available sensor data, or may involve active perception steps, such as a camera viewpoint adaptation.

Programmable and flexible automation pose special requirements especially regarding the offline engineering and online adaptation phase. Generally, the offline engineering use case addresses the initial design of a system and therefore the full configuration space is available. In the online adaptation case, the hardware setup is fixed and only software components and their parameterizations can be adapted.

Additionally, the online adaptation may require quick response, while the offline engineering may take more time. Both use cases should be supported for automated design synthesis. But the focus of the offline synthesis lies in adaptations without hardware changes, which can typically be realized in programmable and flexible automation systems.

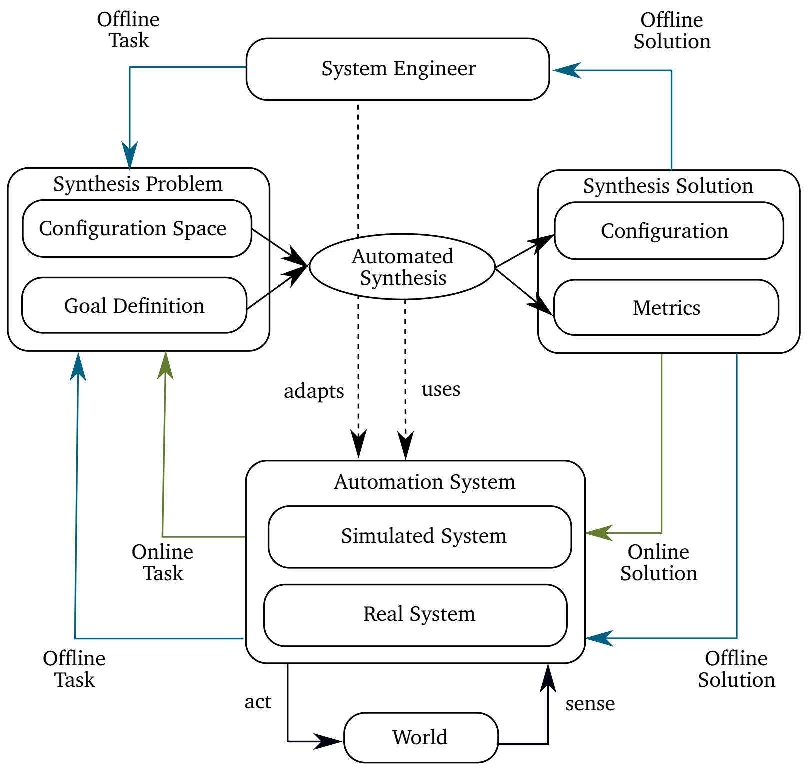


Figure 1.10: *Automated synthesis for offline engineering and online adaptation. Offline engineering and offline adaptation tasks are converted into a synthesis problem, which is solved by an automated synthesis approach. The latter can leverage a simulation of the automation system as well as directly interact with a real automation system.*

In Fig. 1.10 an overview over the actors, components, and synthesis loops is given. The central steps are the setup of the synthesis problem, the automated synthesis and the use and evaluation of the synthesis results. At the offline engineering phase, the system engineer needs to pose and solve synthesis problems. For automated synthesis, the problem is formalized via the configuration space and a goal definition. The solution of the synthesis procedure is a set of configurations and associated metrics, such as runtime and accuracy. The automation system can be instantiated in simulation, especially in early design phases, or in real world. Once the real system is deployed design synthesis is required to adapt the system to new situations and tasks. This adaptation phase covers online tasks, which require fast feedback and offline tasks, which are less time sensitive. The automated synthesis can access the simulated and or real automation system if required.

1.1.2 Challenges

The challenge of the task stems from multiple origins. First, the typical configuration space is very large. The four dimensions of sensor, positioning, pipeline structure, and parameterization form a mixed discrete-continuous space with infinitely many different configurations, due to the continuous variables. Even if continuous variables are discretized, there are easily millions of possible configurations.

Furthermore, sensor data and algorithm performance are subject to noise and uncertainty. Noise in the input of operators naturally leads to degraded and varying results for the operators which are applied on the input. Additionally, every operator does have its own characteristic. Consider Fig. 1.11, where the distribution of the pose estimation error for different operators is plotted. Here the operator *poseEstimationFPFP*, based on fast point feature histograms (Rusu et al., 2009), produces a broad spectrum of results given the parameter set. In order to achieve higher accuracies, if required, additional operators can be employed that allow for higher accuracy. For instance, a pose refinement based on iterative closest points (*refineICP*) or a pose refinement based on physics simulation (*refinePhysics*). Even though *refineICP* produces results with a positional distance below $5\,\mathrm{mm}$, accuracies below $1\,\mathrm{mm}$ are hard to achieve. Here a combination with the operator *refinePhysics* can help. Performing a reliable online estimate of uncertainties is still subject to research and is addressed for instance within the Probabilistic Object Detection Challenge (Hall et al., 2020) for spatial and semantic uncertainty of bounding boxes.

In an automation task we can distinguish five different sources of uncertainty:

- **Environments:** The environment of the automation system can change. For instance, at different times of the year, different lighting conditions can be given.
- **Sensors:** Sensor data is inherently subject to noise and errors. The effects range from drift, bias, nonlinearity, white noise to discretization errors.
- **Actuators:** When the automation system is interacting with the world, e.g., via robotic actuators it may induce uncertainty itself.

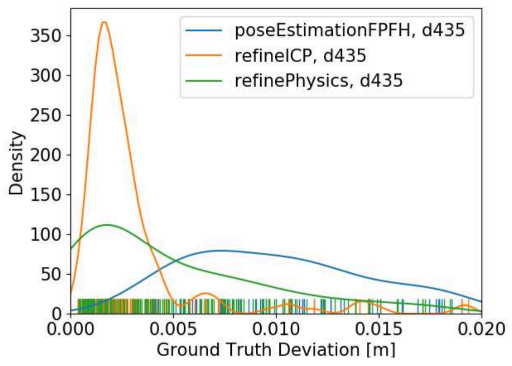


Figure 1.11: *Distribution of the positional error of different perception operators for a single object and multiple scenes (Dietrich et al., 2020). Different operators clearly have different capabilities and properties. The operator refineICP provides low error results but requires a good initial estimate in order to work.*

- **Models:** As noted before, models are abstractions which are inherently inaccurate. Every model contains simplifications, often deliberately in order to keep the computational effort low.
- **Computation:** Reaction times and computational resources are constrained, which can require to algorithmically trade accuracy versus speed. Furthermore, algorithms and their implementations can be non-deterministic, which may lead to variations in the results. And, as noted before, due to the uncertainty in the input, there is inherent uncertainty in the output as well.

The list is inspired by Thrun et al. (2005), where robots are denoted as source of uncertainty instead of actuators.

Due to noise and uncertainty, it is required to assess a specific configuration for a potentially large number of scenes, i.e., object poses and environmental conditions, and a potentially large number of noisy inputs. This leads to high computational effort to evaluate a single system configuration. The state-of-the-art BOP benchmark (Hodan et al., 2018) for 6D pose estimation operators uses 7450 images from seven datasets in order to capture the variability of scenes and poses. It furthermore shows that the performance of an operator depends on the scene and object.

Industrial applications are challenging due to additional requirements as summarized in a study for pose estimation by Hagelskjær et al. (2018). The runtime of the perception needs to be low in order to enable high throughput. High robustness is required especially if erroneous detections can lead to damage. Furthermore, it is desired to have low costs associated with the components and the engineering process. Finally, the overall development time needs to be as low as possible.

The problem of synthesizing perception systems in an offline and online fashion lies at the intersection of multiple specialized research fields. It requires techniques from, among others, computer vision, data fusion, model-driven systems engineering, and design optimization. An engineer can manually plug and combine different components, write the necessary glue code, parameterize the components, and test the system against given use cases. For an automated design synthesis, however, these steps must be performed automatically. Therefore, it is required that different components and techniques have interoperable interfaces and can be tested with available or simulated data. Furthermore, the generation and use of abstract models is required in order to exploit prior knowledge and reduce the effort for the configuration space exploration. Here a tight integration between modeling and execution can reduce the engineering effort which is induced by tool and system boundaries.

Overall, the challenges in the design of perception systems can be summarized as:

- **Variability**: On one hand there exists a large variety of perception tasks and applications. On the other hand, there is a diverse set of sensing domains, perception operators and data types. This variability needs to be representable and searchable.
- **Uncertainty**: Perception operators are generally subject to uncertainty and errors in their predictions. Therefore, single data points are not enough to assess the system performance. Rather large datasets must be used, which are computationally demanding to evaluate. Furthermore, a quantification and estimation of the uncertainty is desired, which would allow to react appropriately.
- **Sensitivity**: Despite current advances, perception pipelines are still sensitive to the application setting. Varying lighting conditions, sensor noise, ranges, and texture can all contribute to decreasing performance in different application settings. This is true for model-based (classical) operators and data-driven operators such as neural networks, which often highly depend on the provided training data. The dependency of pipeline performance on object properties can be clearly seen in the BOP benchmark (Hodan et al., 2018).
- **Complexity**: The number of possible sensors, sensor placements, operators, and pipeline structures as well as the parameter space are large, which leads to a combinatorial explosion. Not all variants and parameter settings can be evaluated in a brute force fashion. Therefore, approaches are required to efficiently structure and reduce the configuration space.

1.2 Contribution

The core contributions of this thesis are automated methods for the design synthesis of perception systems for the phases of offline engineering and online adaptation, which are grounded in a common, hierarchical modeling formalism.

Contribution C1:
Declarative and procedural hierarchical model for perception systems based on set theory which facilitates configuration space exploration for design synthesis.

The hierarchical modeling formalism has multiple advantages. First, it allows to model and generate compatible interfaces for components and algorithms from different origins and thus reduce the need for glue code. The generated models are executable, which allows to apply meta-techniques for configuration space exploration, such as optimization and planning. The meta-techniques for configuration space exploration are represented as first class citizens as so-called meta-operators. Therefore, a reuse within configuration space exploration methods is facilitated. Furthermore, model abstraction hierarchies are represented, which allows to model the perception system domain at different levels of abstraction. This allows to guide the configuration space exploration via more abstract and computationally less expensive models, as exploited within hierarchical planning. A superior synthesis runtime with respect to a non-hierarchical baseline model is enabled and validated.

Contribution C2:
Design synthesis approach based on pipeline templates and sequential model-based optimization which allows to jointly optimize the structure and parameterization of perception pipelines for offline perception system design synthesis.

For offline design synthesis a pipeline structuring and operator matching approach is introduced which builds upon the hierarchical modeling formalism from **Contrib. C1**. A pipeline is modeled as a template, which consists of multiple structural elements which can accommodate a set of connected perception operators. Operator matching is used to match operators from an available set to the individual structural elements. The overall structuring approach allows to reduce the configuration space and to employ black box optimization techniques for the pipeline structure as well as the operator parameters. It is modeled as meta-operator, which can be easily transferred to different domains. The optimization is performed on annotated datasets, which allows to consider scene variability and sensor uncertainty. The applicability is validated in an industrial assembly setting, where accurate pose estimates are required for successful assembly. The design synthesis approach allows to find pipeline configurations and associated operator parameterizations, which improve the overall success rate.

Contribution C3:
Uncertainty model based on factor graphs, which allows to consider graphs of heterogeneous pose observations and can be successfully integrated in online design synthesis.

For offline synthesis, uncertainty can be considered implicitly via the scene distribution of a dataset. Explicit representations of uncertainty, however, are beneficial when performing online synthesis. They can allow to assess the current belief quality as well as the outcome of additional perceptual actions. Therefore, we introduce an approach to handle geometric uncertainties across different coordinate frames and observations in an automation system, which can be used for online synthesis. It is based on a factor graph representation to fuse different observations and prior beliefs while considering their respective uncertainty. The approach is integrated into a hierarchical model, based on **Contrib. C1**. For the evaluation we formulate an online synthesis problem in an assembly setup with multiple sensors and a robot arm which allows to adjust the sensor pose.

Contribution C4:
Hierarchical design synthesis approach based on hierarchical planning, which includes a quality metric model and enables online synthesis with superior runtime properties with respect to a non-hierarchical baseline.

Furthermore, online perception system synthesis requires quick responses especially for industrial automation use cases. Therefore, we introduce a design synthesis method based on hierarchical planning, where the abstraction hierarchy in the model is leveraged in order to synthesize perception pipelines fast enough. This synthesis method builds upon the contribution **Contrib. C1**. We introduce quality metric models across the abstraction hierarchy, which allows to assess sensory and belief data at runtime. The approach is validated in an industrial assembly scenario for multiple objects and sensors and compared with a non-hierarchical baseline planner.

1.3 Experimental Validation Setup

In the following we introduce the hardware setup for the experimental validation, which is embedded in a research demonstrator for flexible autonomous assembly. In Fig. 1.12 the hardware setup is displayed, consisting of two KUKA IIWA robot

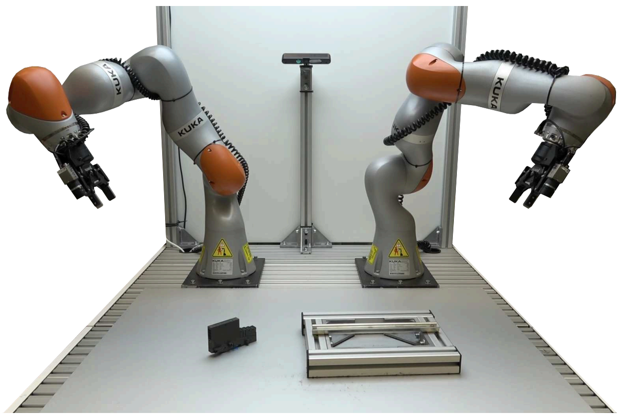


Figure 1.12: *Flexible assembly cell with two manipulators, a static RGBD camera and two end-effector mounted RGB cameras. The system is designed to autonomously perform assembly processes without pre-programming and fixtures. Perception is a core capability for this process. Therefore, this flexible assembly cell is used for the experimental validation.*

arms with 2-finger grippers. The sensors of the perception system are industrial RGB cameras at the end-effectors and an ASUS Xtion consumer RGBD camera mounted in between the robot arms. The system is designed as a research demonstrator for flexible and autonomous assembly. The vision is to have an automation system that needs no programming at all in order to adapt to a new task. It takes a description of the task, the objects involved, and a description of its own capabilities as input and can infer all necessary motions and perceptual actions in order to fulfill the task.

Exemplary tasks are visualized in Fig. 1.13. On the left is an assembly task of an electrical cabinet, where different individual parts must be mounted on a hat rail. In the middle is the assembly task of a box, whose lid is fixed using four screws. On the right is a peg in hole task from the Siemens Robot Learning Challenge (Siemens, 2017).

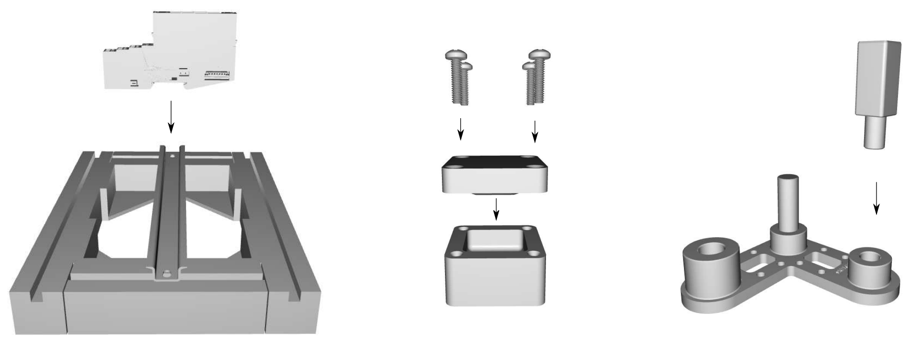


Figure 1.13: *Different assembly tasks which can be performed by the flexible assembly system.*

1.4 Outline

Including the introduction, this thesis is comprised of 7 chapters, which are listed and briefly introduced in the following.

- **Chapter 2:** Introduction of mathematical models and fundamental concepts.
- **Chapter 3:** Introduction of the common approach to represent procedural and declarative models and to incorporate hierarchy in order allow for hierarchical task decomposition and planning. Furthermore, it is presented how the model generation can be partially automated via the use of harvesting methods. Addresses **Contrib. C1**.
- **Chapter 4:** Presentation of a pipeline structuring approach, that builds on the common model and enables the offline optimization of perception pipelines. Depends on Chapter 3 and addresses **Contrib. C2**.
- **Chapter 5:** Building on the semantic model, introduction of an online configuration approach rooted in planning, where geometric uncertainty across different coordinate systems can be considered. Depends on Chapter 3 and addresses **Contrib. C3**.
- **Chapter 6:** Extension of the hierarchical model regarding metric modeling and application of the hierarchical planning approach of Chapter 5 to an online perception system synthesis task. Depends on Chapter 3 and addresses **Contrib. C4**.
- **Chapter 7:** Discussion of the presented work and conclusion

1.5 Publications

In the following, the publications which contribute to this thesis are listed and sorted by their importance for this work. A detailed reference is done within the respective chapters.

- V. Dietrich, B. Kast, S. Albrecht, and M. Beetz. Data-Driven Synthesis of Perception Pipelines via Hierarchical Planning. In *Int. Conf. on Robotics in Alpe-Adria Danube Region*, Springer, 2020
- V. Dietrich, B. Kast, M. Fiegert, S. Albrecht, and M. Beetz. Automatic Configuration of the Structure and Parameterization of Perception Pipelines. In *Int. Conf. on Advanced Robotics*, IEEE, 2019
- V. Dietrich, B. Kast, P. S. Schmitt, S. Albrecht, M. Fiegert, W. Feiten, and M. Beetz. Configuration of perception systems via planning over factor graphs. In *Int. Conf. on Robotics and Automation*, IEEE, 2018
- B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang. A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems. In *Int. Conf. on Informatics in Control, Automation and Robotics*, 2019
- B. Kast, V. Dietrich, S. Albrecht, and J. Zhang. Domain Optimization for Hierarchical Planning Based on Set-Theory. In *Int. Conf. on Informatics in Control, Automation and Robotics*, 2020 (Best Student Paper Award Winner)
- B. Kast, S. Albrecht, V. Dietrich, F. Wirnshofer, W. Feiten, and G. von Wichert. Der digitale Zwilling in der autonomen Robotik. In *atp magazin*, 61(5), 2019
- M. Galanis, V. Dietrich, B. Kast, and M. Fiegert. RTFM: Towards Understanding Source Code using Natural Language Processing. In *Int. Conf. on Informatics in Control, Automation and Robotics*, 2020
- N. Hafez, V. Dietrich, and M. Zwick. Probabilistic orientation resolution for near symmetrical objects using depth images. In *Int. Conf. on Robotics in Alpe-Adria Danube Region*, Springer, 2019
- N. Hafez, V. Dietrich, and S. Roehrl. Analysis of Different Methods to Close the Reality Gap for Instance Segmentation in a Flexible Assembly Cell. In *Int. Conf. on Robotics in Alpe-Adria Danube Region*, Springer, 2020

- V. Dietrich, D. Chen, K. M. Wurm, G. von Wichert, and P. Ennen. Probabilistic multi-sensor fusion based on signed distance functions. In *Int. Conf. on Robotics and Automation*, IEEE, 2016
- D. Chen, V. Dietrich, Z. Liu, and G. von Wichert. A probabilistic framework for uncertainty-aware high-accuracy precision grasping of unknown objects. In *J. of Intelligent & Robotic Systems*, 90(1-2), 2018
- D. Chen, V. Dietrich, and G. von Wichert. Precision grasping based on probabilistic models of unknown objects. In *Int. Conf. on Robotics and Automation*, IEEE, 2016

Chapter 2

Fundamentals of Perception and Design Synthesis

In this chapter we introduce fundamental concepts, that are required to understand the task at hand. Furthermore, we sketch the architecture of our approach and the distinct components. The purpose is to provide background knowledge and show how components interact and depend on each other.

Due to the large variety of perceptual problems available in industry, we focus on industrial pose estimation as the running example and evaluation domain. We define it as:

Definition 7 - Pose Estimation

Pose estimation denotes the process of determining the translation and orientation of a known object or object primitive with respect to a reference coordinate frame. It may entail the use of different information sources, such as sensory data, e.g., color images, point clouds and force data, and available scene knowledge, such as supporting planes. Pose estimation can be performed passively, by evaluating already available sensor data, or may involve active perception steps such as camera viewpoint adaptations.

The translation and rotation of the object determine its location in the three-dimensional space and allow to interact with the object. This allows to grasp an object or perform an assembly task. An important aspect is the reference frame of the translation and rotation, which means, with respect to which coordinate system both are defined. Depending on the application a global frame may be defined, or local frame, such as the robot end-effector may be used. We will go into more detail on reference frames in the following sections. An exemplary setup for a pose estimation task is shown in Fig. 2.1, where an object from a peg-in-hole task is to be localized accurately.

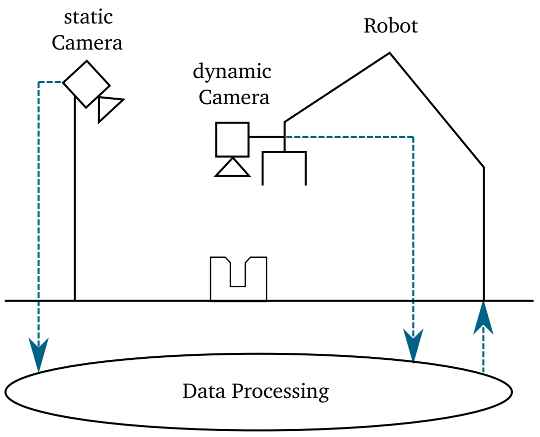


Figure 2.1: *Perception system consisting of two different cameras, where one is static and the other one is mounted on a robot arm. The task is to estimate the pose of the presented object with sufficient accuracy.*

In this chapter, we start with the introduction of fundamental mathematical concepts which are needed within the thesis. Then, individual components of perception systems and models to approximate their behavior are discussed. Subsequently, the overall engineering process is introduced in more detail and the role of design synthesis is specified. Finally, we give an overview of the automated synthesis procedure and necessary models. The sections 2.2.2.1, 2.2.2.2, 2.2.2.4, and 2.2.2.5 are based on the prior publication (Dietrich et al., 2019).

2.1 Fundamental Concepts

Fundamental concepts for the modeling of perception systems for pose estimation tasks are the mathematical models for coordinate frames, poses, pose uncertainty, and measurement error. In the following, the models are introduced and discussed with respect to their role for the design of automation systems. The introduction of the mathematical apparatus for handling poses is restricted to the needs within the thesis. The formalization is partially based on (Corke, 2017) and (Barfoot, 2017), to which we refer to for an in-depth introduction.

2.1.1 Model

A general definition of the term model has been already given in Def. 6 in Chapter 1. Models are omnipresent and used throughout all scientific disciplines and engineering activities. Different types of models can be distinguished. Mathematical models for instance can be used to describe physical processes, systems, and relationships via mathematical formulae which model the relationship between variables and parameters. Data models standardize how pieces of data are represented and how they relate. In the context of programming, data models can refer to actual data structures. On the software engineering side, there are models and modeling languages such as the Unified Modeling Language (UML) (Rumbaugh et al., 2004), which allows to model different aspects of software systems in a graphical way. This facilitates the understanding and communication between engineers.

For the automated design synthesis, we define two major classes of models: procedural and declarative models.

Definition 8 - Procedural Model

A procedural model represents procedural knowledge of how to perform a task or process. It is executable and thus allows to predict the outcome of or actually perform an action. Procedural models do not necessarily have to provide information about their inner working and therefore can be black box models.

Definition 9 - Declarative Model

A declarative model states facts and relations about a system. Depending on the formalization, interpreters can be executed on the model to extract additional information. Such interpreters are part of the procedural knowledge and can be represented using procedural models.

Examples for declarative models are flow diagrams and data models. Mathematical models defined as formulae are declarative as well and can be transformed into an executable procedural model. If a machine-readable formula description language is used, this transformation can be automated.

Procedural models, as defined, play an important role for automation systems and autonomous systems. Every possible action of a system can be represented using a procedural model, such as the change of a viewpoint, the acquisition of a camera image or the conversion of data types. As the procedural model is executable it allows for instance to predict the outcome of actions as part of a system simulation.

A real-world execution of an action can similarly be performed using a procedural model.

2.1.2 Coordinate Frame

A coordinate frame is a set of axes which are orthogonal to each other and intersect at the origin of the coordinate frame. It is also denoted as Cartesian coordinate frame and can have an arbitrary number of dimensions, even though we will only need the two-dimensional and three-dimensional case. The position of a point in space with respect to a coordinate frame $\mathcal{F}_{\mathrm{A}}$ can be described as a vector $\vec{v}_{\mathrm{A}}$, where A is an identifier. Different coordinate frames can have different positions and rotations in space and therefore the position vector $\vec{v}_{\mathrm{B}}$ with respect to a coordinate frame $\mathcal{F}_{\mathrm{B}}$ can differ from $\vec{v}_{\mathrm{A}}$, as depicted in Fig. 2.2.

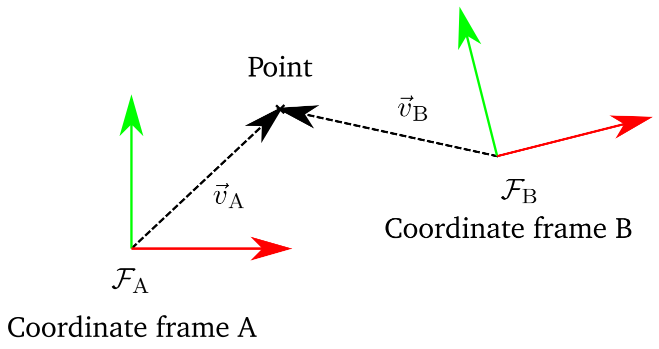


Figure 2.2: *The location of a point can be described by a vector within a coordinate frame. Here, two different two-dimensional coordinate frames $\mathcal{F}_{\mathrm{A}}$ and $\mathcal{F}_{\mathrm{B}}$ are given, which results in two different vectors $\vec{v}_{\mathrm{A}}$ and $\vec{v}_{\mathrm{B}}$ to describe the location of the point.*

Coordinate frames can be static or dynamic within the targeted scope. A dynamic coordinate frame changes its relative location with respect to static frames over time.

2.1.3 Pose

The combination of the position and orientation of a coordinate systems is named *pose*. It requires a reference coordinate frame and is therefore also denoted as relative pose or transform between coordinate frames. If no reference coordinate frame is explicitly given a hypothetical world frame is assumed. In order to denote the relative pose of the coordinate frame $\mathcal{F}_{\mathrm{B}}$ with the reference frame $\mathcal{F}_{\mathrm{A}}$ we use the notation ${}^{\mathrm{A}}p_{\mathrm{B}}$, see Fig. 2.3.

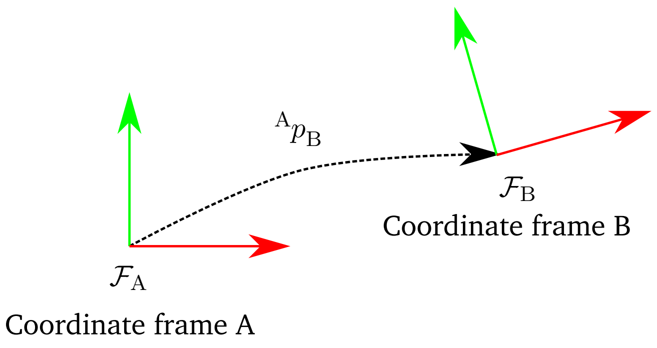

${}^{\mathrm{A}}p_{\mathrm{B}}$
$\mathcal{F}_{\mathrm{B}}$
Coordinate frame B
$\mathcal{F}_{\mathrm{A}}$
Coordinate frame A

Figure 2.3: *Visualization of the relative pose ${}^{\mathrm{A}}p_{\mathrm{B}}$ between the coordinate systems $\mathcal{F}_{\mathrm{A}}$ and $\mathcal{F}_{\mathrm{B}}$.*

Poses can be composed, which yields a new pose. As the operation differs with respect to addition in the realm of numbers, the symbol $\oplus$ is used. In Fig. 2.4 the pose composition is visualized. A new coordinate frame $\mathcal{F}_{\mathrm{C}}$ is introduced and the pose of $\mathcal{F}_{\mathrm{C}}$ with respect to $\mathcal{F}_{\mathrm{A}}$ can be denoted as

$$ {}^{\mathrm{A}}p_{\mathrm{C}} = {}^{\mathrm{A}}p_{\mathrm{B}} \oplus {}^{\mathrm{B}}p_{\mathrm{C}}. \tag{2.1}$$

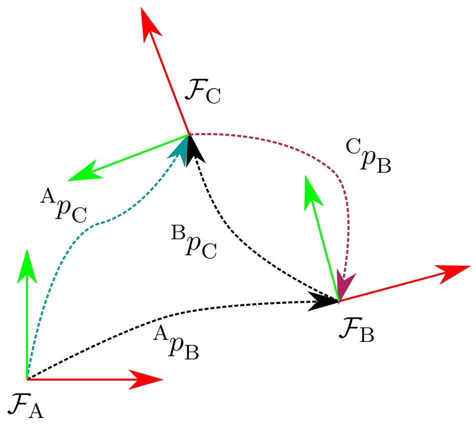

$\mathcal{F}_{\mathrm{C}}$
${}^{\mathrm{C}}p_{\mathrm{B}}$
${}^{\mathrm{A}}p_{\mathrm{C}}$
${}^{\mathrm{B}}p_{\mathrm{C}}$
$\mathcal{F}_{\mathrm{B}}$
${}^{\mathrm{A}}p_{\mathrm{B}}$
$\mathcal{F}_{\mathrm{A}}$

Figure 2.4

Furthermore, poses can be inverted, which is denoted by the operator $\ominus$. The inverse of ${}^{\mathrm{B}}p_{\mathrm{C}}$ is

$$ \ominus {}^{\mathrm{B}}p_{\mathrm{C}} = {}^{\mathrm{C}}p_{\mathrm{B}}. \tag{2.2}$$

The operator symbol $\ominus$ is also used to denote a composition with the inverse:

$$\begin{aligned} {}^{\mathrm{A}}p_{\mathrm{B}} &= {}^{\mathrm{A}}p_{\mathrm{C}} \oplus (\ominus {}^{\mathrm{B}}p_{\mathrm{C}}) \\ &= {}^{\mathrm{A}}p_{\mathrm{C}} \ominus {}^{\mathrm{B}}p_{\mathrm{C}} \\ &= {}^{\mathrm{A}}p_{\mathrm{C}} \oplus {}^{\mathrm{C}}p_{\mathrm{B}}. \end{aligned} \tag{2.3}$$

Mathematically, poses constitute a group, where identity, an inverse and an associative binary operator, the composition, are defined. For the two- and three-dimensional case the groups are denoted as the special Euclidean groups $\mathrm{SE}(2)$ and $\mathrm{SE}(3)$. Additional algebraic rules hold for poses:

$$\begin{aligned} p \oplus 0 &= p, \\ p \ominus 0 &= p, \\ p \ominus p &= 0, \\ \ominus p \oplus p &= 0, \\ {}^{\mathrm{A}}p_{\mathrm{B}} \oplus {}^{\mathrm{B}}p_{\mathrm{A}} &\neq {}^{\mathrm{B}}p_{\mathrm{A}} \oplus {}^{\mathrm{A}}p_{\mathrm{B}}, \end{aligned} \tag{2.4}$$

where 0 is a zero pose without relative translation or rotation. The composition is not commutative as stated in the last rule.

A pose can be represented as a tuple of translation p_{pos} and rotation p_{rot}:

$$p = \langle p_{\mathrm{pos}}, p_{\mathrm{rot}} \rangle. \tag{2.5}$$

Note here, that a pose can as well be represented via a transformation matrix acting on homogeneous coordinates. The different representations have different advantages and disadvantages, depending on the application. The translation p_{pos} in three dimensions is a vector of length three:

$$p_{\mathrm{pos}} = (x, y, z), \text{ with } x, y \text{ and } z \in \mathbb{R} \text{ and } p_{\mathrm{pos}} \in \mathbb{R}^3. \tag{2.6}$$

For the rotation in three dimensions we introduce the representation as Euler angles and quaternion. Euler angles represent individual rotations via the angles α_1, α_2 and α_3 around the coordinate axes which are applied successively. There exist twelve different conventions, which can lead to serious interfacing problems, if the convention is not ensured.

Quaternions are an alternative rotation representation. They are a generalization of complex numbers and represent a four-dimensional vector space. They consist of a real component and three imaginary components denoted as $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$. Thus, every quaternion q can be represented using four real numbers x_0, x_1, x_2 and x_3:

$$q = x_0 + x_1\mathbf{i} + x_2\mathbf{j} + x_3\mathbf{k} \tag{2.7}$$

Among different applications, quaternions can be used to represent rotations. Compared to rotation matrices the multiplication of quaternions is more efficient. Additionally, quaternions are numerically advantageous compared to matrix multiplications. Due to rounding errors, regular normalization is required, which is also computationally more efficient when working with quaternions.

The matrix, Euler and quaternion representations have different advantages and disadvantages. The Euler representation is in many cases more intuitive, especially when rotations must be defined manually. Although, from a computational perspective, problems with singularities may arise. Additionally, the plethora of different conventions yield the Euler representation particularly error prone when interfacing heterogeneous systems. Quaternions on the other side are well suited for computation and represent rotations in a unique manner. However, they are more difficult to read, as the average engineer is more familiar with angular representations. Matrices have the advantage that rotation and translation can jointly be represented. Furthermore, matrices can be used to provide different types of operations, such as scaling and projection. In summary, the choice of a rotation representation depends on the use case and a mix of representations can be used via conversion.

2.1.4 Pose Uncertainty

Similar to the rotation, pose uncertainty can also be modeled in different ways. Arbitrary distributions can be represented using particles. Here, the pose distribution is characterized by a set of pose particles. The probability is implicitly given by the particle density. However, the model is not compact and requires large numbers of particles. Memory and computation demand can therefore be high, especially when a large number of chained uncertain poses have to be considered in real-time. But the particle representations is suitable for parallelization. The set of particles can be easily distributed on a set of computing resources such as threads and propagated independently. The set of independently computed particles represents the resulting pose distribution.

A more compact representation are Gaussian distributions. Here, the distribution is approximated by uni-modal or multi-modal multivariate Gaussians. This is perfectly valid for the Euclidean Space of positions. Rotations, however, can not as easily be represented using Gaussians. Depending on the representation, there are jumps and singularities, as for instance with Euler angles. Other representations, such as matrices and quaternions require to be normalized. Therefore, their parameters are not independent by design. As a result, specialized approaches have been developed such as mixtures of projected Gaussians (Feiten et al., 2009). Even specialized, discretized representations can be used for instance the case of symmetric objects as proposed in (Hafez et al., 2019). Again, the representations have different advantages and disadvantages and the suitability depends on the application.

Also, pose uncertainty may not be sufficient, for instance in the case of unknown surface geometry. Here, for instance truncated signed distance functions can be leveraged to represent surface uncertainty, as proposed and validated in (Dietrich et al., 2016; Chen et al., 2016, 2018).

2.1.5 Frames and Poses in Automation Systems

Various coordinate frames are of importance in the context of pose estimation for industrial applications such as assembly. Consider again the peg-in-hole application as introduced within the first chapter, as visualized in Fig. 2.5. First, any object such

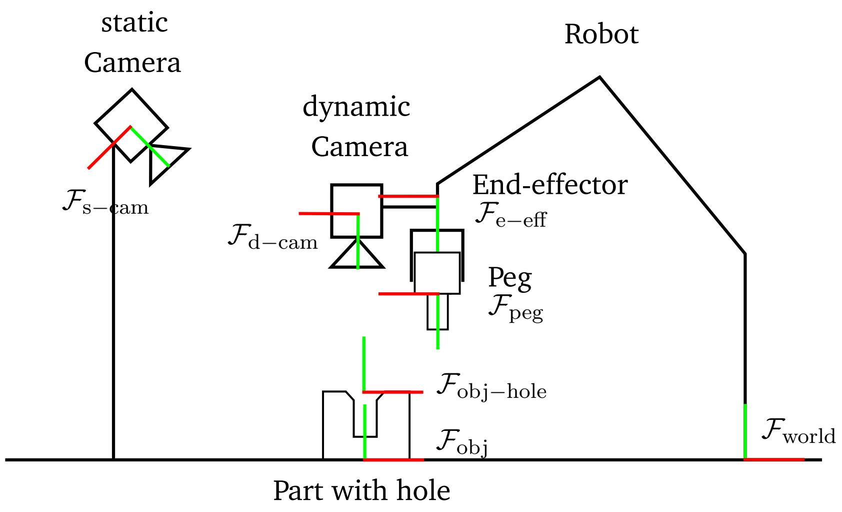

static Camera
Robot
dynamic Camera
End-effector
$\mathcal{F}_{\text{e-eff}}$
$\mathcal{F}_{\text{s-cam}}$
$\mathcal{F}_{\text{d-cam}}$
Peg
$\mathcal{F}_{\text{peg}}$
$\mathcal{F}_{\text{obj-hole}}$
$\mathcal{F}_{\text{obj}}$
$\mathcal{F}_{\text{world}}$
Part with hole

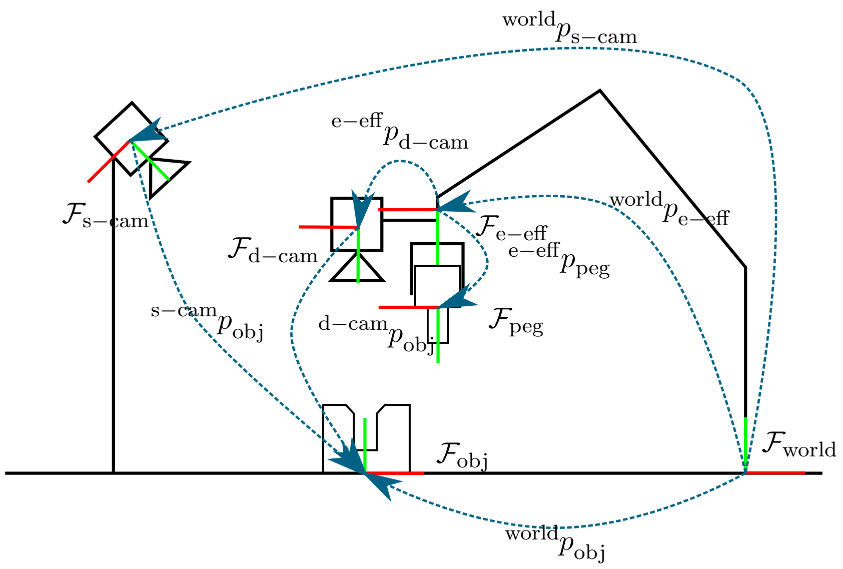
Figure 2.5: *Exemplary frames which are relevant for pose estimation in an automation system that covers a robot and a static as well as a dynamic sensor.*

as the part with the hole can have multiple frames. In fact, arbitrary frames can be defined. In the example the part with the hole has a base frame $\mathcal{F}_{\text{obj}}$, which is the coordinate frame of the CAD model. Additionally, a frame $\mathcal{F}_{\text{obj-hole}}$ is defined which specifies the location of the hole. Throughout this work, we assume rigid objects.

This means that the objects are not deformable, or the deformation can be neglected within the targeted scope. Therefore, frames which are defined for the same rigid object have constant relative poses. Compound objects with a static connection, e.g., being the results of an assembly process, can be modeled as a single rigid object.

The hardware of the automation system requires additional frames. The common reference frame is the world frame $\mathcal{F}_{\text{world}}$ which here is defined in the base of the robot. With respect to the overall automation system it is static, but in general the automation system could be placed on a moving base, so in the factory context it might be dynamic. The choice of the base of the robot arm is arbitrary it could be also the corner of the working table. The cameras have each a coordinate frame, that corresponds to the optical frames. We will go into more detail on the optical frame in the following sections. Typically, other frames such as the hardware mounting point are defined. For the robot the frame of the end-effector $\mathcal{F}_{\text{e−eff}}$ is shown, which is the mounting point for a gripper.

We now discuss relevant relative poses and how they are determined, based on Fig. 2.6. The pose between the robot end-effector and its base ${}^{\text{world}}p_{\text{e−eff}}$ is determined based



${}^{\text{world}}p_{\text{s−cam}}$
${}^{\text{e−eff}}p_{\text{d−cam}}$
${}^{\text{world}}p_{\text{e−eff}}$
$\mathcal{F}_{\text{s−cam}}$
$\mathcal{F}_{\text{d−cam}}$
$\mathcal{F}_{\text{e−eff}}$
${}^{\text{e−eff}}p_{\text{peg}}$
${}^{\text{s−cam}}p_{\text{obj}}$
${}^{\text{d−cam}}p_{\text{obj}}$
$\mathcal{F}_{\text{peg}}$
$\mathcal{F}_{\text{obj}}$
$\mathcal{F}_{\text{world}}$
${}^{\text{world}}p_{\text{obj}}$

Figure 2.6: *Relevant relative poses for pose estimation in an automation system that covers a robot and a static as well as a dynamic sensor.*

on encoder values in the joint angles and a forward kinematics calculation. The relative pose between base and end-effector is subject to errors due to measurement errors in the encoders and uncertainty in the relative poses between joints. For industrial robot arms, these errors can vary from a tenth of a millimeter up to millimeters and more. Various techniques can be used to improve this accuracy such as error compensation (Karan and Vukobratović, 1994) and stiffened mechanical design (Courteille et al., 2009).

The camera poses ${}^{\mathrm{e-eff}}p_{\mathrm{d-cam}}$ and ${}^{\mathrm{world}}p_{\mathrm{s-cam}}$ are required in order to spatially align pose estimation results with respect to the application. They are typically determined using a calibration procedure, which is performed during system setup. Although highly accurate calibrations targets are used, calibration is subject to errors as well, which results in deviations in the calibrated poses with respect to reality.

Finally, the pose estimation results ${}^{\mathrm{s-cam}}p_{\mathrm{obj}}$ and ${}^{\mathrm{d-cam}}p_{\mathrm{obj}}$ are the results of the data processing of sensor data from the static and the dynamic camera respectively. Here again, the uncertainty in the data processing results in errors, which will be discussed in more depth in the following sections.

Overall, the relative poses and the coordinate frames form a graph. Tree-like representations, such as a transform tree (Foote, 2013), can help to resolve relative transformations. But the tree is only an approximation of the underlying graph structure. Furthermore, uncertainty is prevalent in all relevant poses and it is important to consider uncertainty for positioning, calibration, and pose estimation accuracy in order to determine the object's pose sufficiently well.

2.1.6 Measurement Error

Every measurement is subject to measurement errors, which we define as follow:

Definition 10 - Measurement Error

The measurement error is the deviation of a perceived value with respect to the true value.

The *true* value is not always known. For discrete values such as the number of objects to be counted, an objectively true value can be determined. The true value of continuous values such as the illumination cannot be determined without a measurement that is subject to errors itself. Therefore, more accurate reference measurements may be used as replacement. If we define z_t as the true value of the targeted variable and z_m as the measured value, the measurement error e is:

$$e = z_m - z_t. \tag{2.8}$$

The measurement error can be classified into *systematic error* and *random error* (Balazs, 2008). Systematic error has the same constant effect on repeated measurements.

An estimate of the systematic error is denoted as *measurement bias*. Random errors on the other hand are unpredictable with respect to a single measurement and occur in a stochastic manner. Both error types may vary in their intensity depending on external factors such as temperature and time. With e_s as the systematic error and e_r as the random error, the measurement error can be described in its composite form:

$$e = e_s + e_r. \tag{2.9}$$

2.2 Perception System Components

The primary components of perception systems on the hardware side are sensors and computing units. On the software side the components are individual data processing steps and interfaces to external components. The primary effort is associated with the data processing, which itself depends on the chosen sensor hardware.

2.2.1 Sensor

Perception starts with the acquisition of data about the environment using sensors. Sensors exist in all kinds of types and sizes, from photodiodes to integrated smart sensors which consist of multiple components and elaborate data-processing, such as smart cameras with integrated image processing. In the following, we introduce common sensors for pose estimation in industrial environments, which are cameras and depth sensors.

Camera

A camera is a device, which allows to acquire images. The word *camera* originates from the Latin term *camera obscura*, which means dark chamber. With a small hole towards the outside of dark chamber, an image of the outside can be projected to a wall of the chamber. This setup is a pinhole camera. For industrial automation purposes *digital cameras* are used. They consist of optics, a digital image sensor, and a processing unit. A schematic overview of a digital camera is given in Fig. 2.7. The optics ensure, that incoming light from a point in space maps to a point on the image sensor. Furthermore, as opposed to a pinhole camera the incoming light is bundled to increase the intensity. The image sensor is a two-dimensional array of photodiodes, where the incoming light intensity is converted to an electronic signal. Using a filter pattern, such as a Bayer filter, the intensity for different wave lengths can be measured and a colorized image reconstructed using demosaicing. The processing unit can perform such post-processing and handle the communication of raw image data or higher-level information. Camera systems which provide higher level information such as object positions are typically denoted as smart cameras.

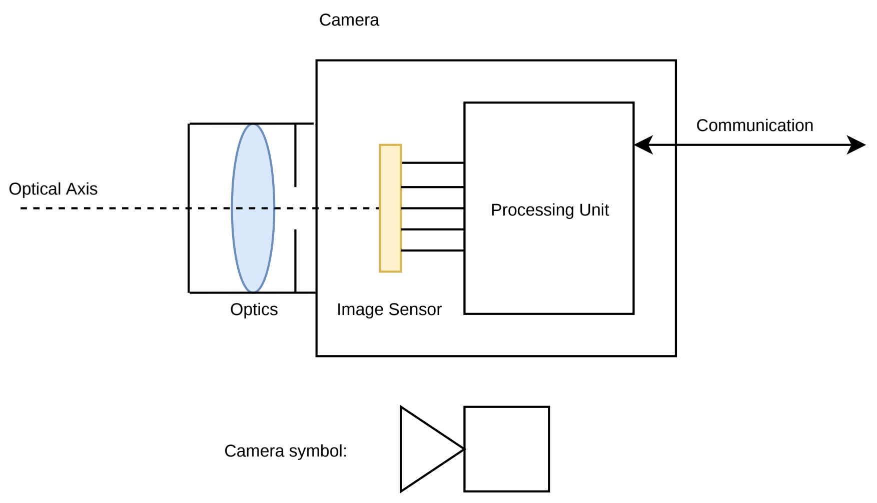
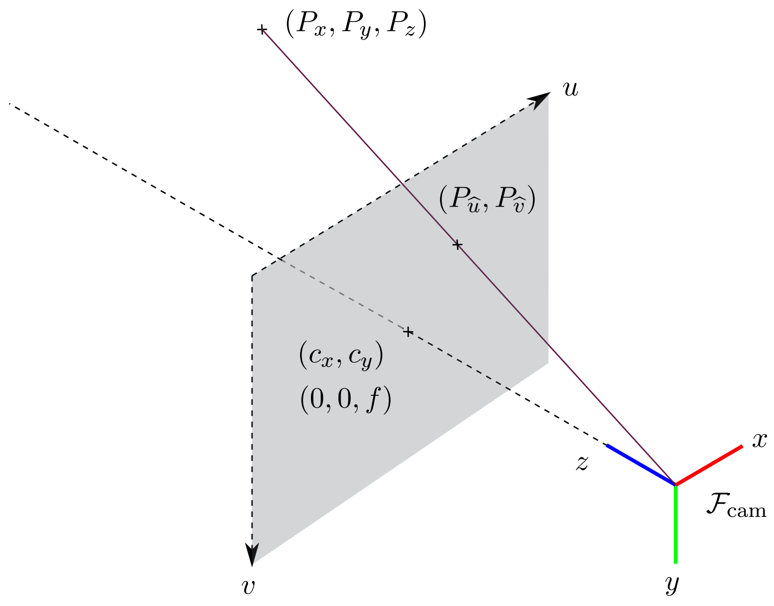


Figure 2.7: *Schematic overview of the components of a digital camera. Incoming light is focused on the image sensor using optical lenses. A processing unit reads out the individual pixels of the image sensor and transfers the resulting image via a communication bus. Smart cameras may also provide on-board processing and interpretation of the image.*

Cameras are widely used in robotic applications, such as bin-picking and quality inspection. The intensity or color image that the camera produces are the input for following data processing steps such as object or defect detection.

The relationship between the 2D coordinates on the image sensor and the view direction of the incoming light can be represented using mathematical models. In the following we present the pinhole camera model. As every model, it contains modeling errors, such as geometric distortions or object blurring due to the optical lenses. The projection from a point in the three-dimensional space to the two-dimensional image screen is visualized in Fig. 2.8. The central coordinate frame of the camera is the optical frame, which is denoted as $\mathcal{F}_{\text{cam}}$. The image plane is parallel to the xy-plane of the optical frame and in a distance of $z_{\text{screen}} = f$, where f is the focal length of the lens. In case of an actual pinhole camera without lens, the distance z_{screen} is the



(P_x, P_y, P_z)
u
$(P_{\widehat{u}}, P_{\widehat{v}})$
(c_x, c_y)
$(0, 0, f)$
x
z
$\mathcal{F}_{\text{cam}}$
v
y

Figure 2.8: *In the pinhole projection model it is assumed that the camera optics behave as a pinhole. Therefore, each visible 3D point in space can be associated with a pixel on the image sensor by projection along a light ray.*

distance between screen and hole. The z-axis of the optical frame is the so-called optical axis.

First, we represent the two-dimensional image plane by the centered coordinate axes x and y, whose origin is on the optical axis. The 2D location $(P_{\widehat{u}}, P_{\widehat{v}})$ on the image plane of a 3D point $P = (P_x, P_y, P_z)$ in the environment is geometrically the intersection of the straight line formed between the point P and the origin of the optical frame and the image plane. Based on the intercept theorem this relation can be formalized mathematically as

$$\frac{P_{\widehat{u}}}{P_x} = \frac{P_{\widehat{v}}}{P_y} = \frac{f}{P_z}. \tag{2.10}$$

Thus, the 2D projection of point P can be calculated as

$$\begin{aligned} P_{\widehat{u}} &= f\frac{P_x}{P_z} \text{ and} \\ P_{\widehat{v}} &= f\frac{P_y}{P_z}. \end{aligned} \tag{2.11}$$

By convention, image coordinates are used, which are centered in the top left corner of an image. This reflects the underlying matrix data structure and allows to map matrix indices to points in the environment. The coordinate axes for the top left origin are named u and v. Therefore, two new variables c_x and c_y are introduced, which denote the shift between the coordinate frames. Additionally, it can be considered that the scaling in x and y direction is not necessarily equal. Therefore, separate focal lengths f_x and f_y are defined. Typically, the deviations are small and depend on manufacturing inaccuracies or pixels which are non-square. This yields the adapted projections

$$\begin{aligned} P_u &= P_{\widehat{u}} + c_x = f_x \frac{P_x}{P_z} + c_x \text{ and} \\ P_v &= P_{\widehat{v}} + c_y = f_y \frac{P_y}{P_z} + c_y. \end{aligned} \tag{2.12}$$

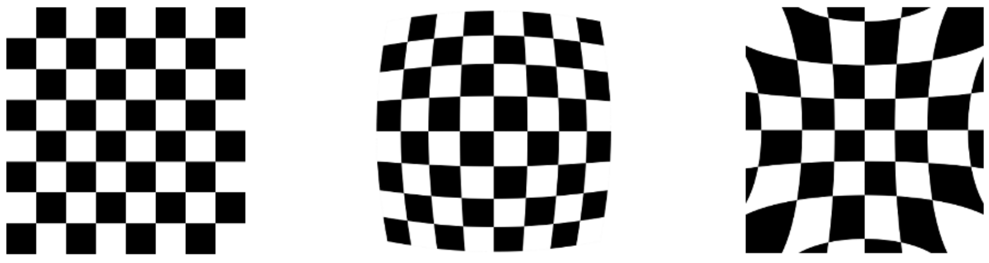
The projection can also be represented using matrix notation. Therefore, we introduce the *intrinsic matrix* K with

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.13}$$

Using the intrinsic matrix, the 3D to 2D projection can be denoted as

$$\begin{pmatrix} P_u \\ P_v \\ 1 \end{pmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix} \tag{2.14}$$

Again, this is only a basic projection model. There are additional effects such as lens distortions that can be considered, see Fig. 2.9. Models that cover lens distortion have additional parameters that need to be determined. Moreover, there are different types of camera optics, such as telecentric lenses and fisheye lenses, both of which require different mathematical models. In general, there is again a range of models with varying complexity and computational demand. The achievable error depends not only on the quality of the model, but also on the quality of estimated model parameters, which are determined using model calibration.



(a) *No distortion* ***(b)*** *Barrel distortion* ***(c)*** *Pincushion distortion*

Figure 2.9: *There are different types of lens distortions, which can be corrected algorithmically. The effect of both barrel distortion and pincushion distortion increases with distance to the image center.*

Depth Camera

Depth images and point clouds are an important input domain for robotic applications. By covering the three dimensions, more geometric information can be contained than in a 2D RGB image. There are different types of measurement techniques that may provide depth information, such as time of flight measurement and triangulation (Hartley and Zisserman, 2003). Time-of-flight cameras rely on light pulses, by light emitting diodes or a laser. The emitted light is reflected by the environment back to the image sensor, where the time of flight between the emission of a pulse to the arrival is measured for each pixel. As the speed of light in air is approximately constant, it can be used to calculate the traveled distance based on the time of flight for each pixel its corresponding view direction.

Triangulation on the contrary is a geometric approach. An exemplary setup with two cameras is depicted in Fig. 2.10. Two cameras which are placed in parallel with a distance d, also called *baseline*. If the same point of an object is seen in both images and its respective pixel locations can be identified, the distance can be calculated by applying trigonometry. The critical aspect is the matching of identical points in both images, which is denoted stereo matching and addressed by several different algorithms, see (Lazaros et al., 2008). Stereo matching errors in the range of a single pixel can already introduce depth errors, which grow with increasing distance to the object. This effect can be alleviated by using larger baselines, although this can increase the minimum distance, where depth can be perceived due to non-overlapping fields of view.

Generally, both measurement principles have advantages and disadvantages. Time-of-flight sensors do require less computation and can provide images at a higher rate. But the light pulses may interfere with ambient light and can be reflected multiple

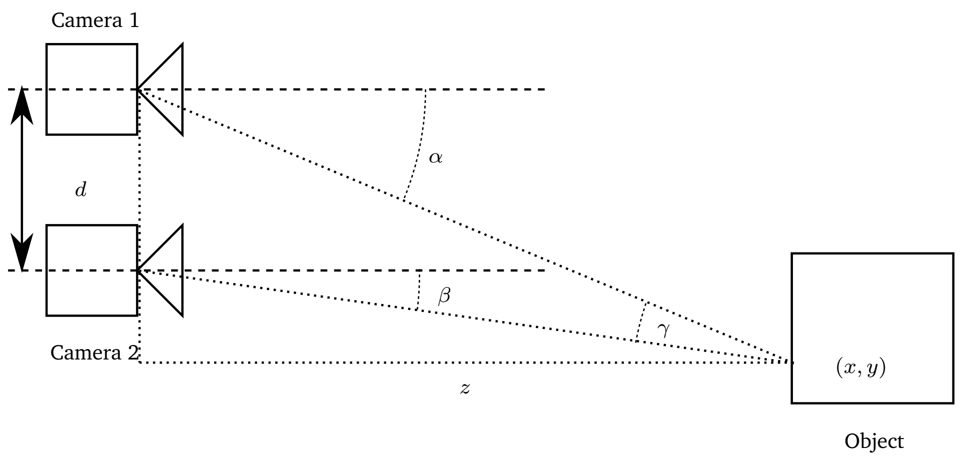


***Figure 2.10:** Exemplary visualization of stereo triangulation. If the same point on an object can be identified from different views, it's distance to the camera z can be calculated using trigonometry.*

times. Depth estimation based on triangulation is computationally demanding, due to the stereo matching. Also, the stereo matching depends on the texture in the scene and may fail for non-textured and shiny surfaces. The use of projected patterns in visible or infrared light can alleviate this. Such projected patters are also denoted as structured light and can be either static or adapted dynamically.

The properties of a depth camera clearly depend on design choices such as the measurement principle and the baseline. Additionally, there is a dependency on the scene: Triangulation and stereo matching work well for textured objects, time-of-flight sensors may not work well in bright sunlight, etc. In Fig. 2.11 the different noise characteristic of different depth sensors is displayed. The usability of a sensor clearly depends on the application and the required task accuracies.

Clearly, data processing is involved in the determination of depth, either by calculating the traveled distance by the light or by performing stereo matching and triangulation. This processing can be implemented using hardware or software.

Composite Sensors

Multiple sensors can be combined to form composite sensors. Common sensors for robotic applications are *RGBD-sensors* that provide color (RGB) and depth images (D). Popular models are depicted in Fig. 2.12. Additional to an internal processing unit, they are composed of multiple cameras and components such as infrared projectors. Sensors can be composed in a hierarchical manner, an Intel D435 is composed of

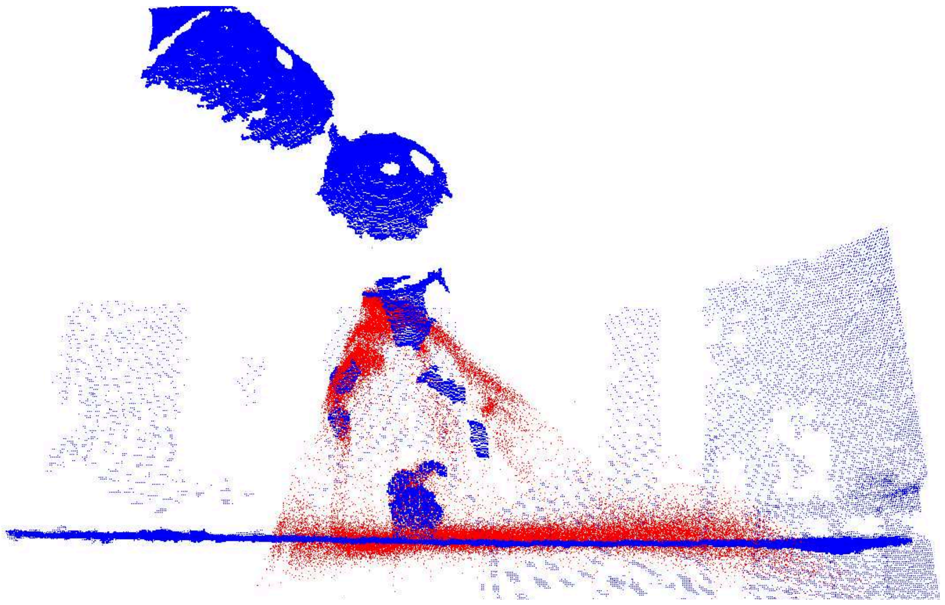


Figure 2.11: *Comparison of point clouds of a time-of-flight camera Senz3D (red) and a Kinect camera (blue), which uses structured light and triangulation (Dietrich et al., 2016). Even though the time-of-flight camera is placed closer to the object, a 3D printed Stanford Bunny, it displays more noise. The noise depends on the actual setup of the individual camera system.*

multiple cameras, which are composed of image sensors which are composed of photodiodes. The same is true for the data processing which is performed on every level of the sensor. Depending on the task, a different detail level will be chosen. For instance, an image sensor developer needs to go into detail on photodiodes, whereas a camera developer needs to consider different variants of image sensors. The developer of an automation systems for industrial applications will typically choose between different types of composite sensors, such as RGBD-cameras and will not look into detail of the actual photodiodes of the sensor.

Camera Noise and Uncertainty

Cameras introduce different sources of errors. First, the quality of the intrinsic and extrinsic calibration affects the resulting accuracy. The extrinsic calibration depends on the calibration target, the set of acquired data points and the positioning accuracy of the actuator which moves the calibration target (camera to world) or the camera (camera to robot). It can be modeled using an uncertain 3D pose, as described in Sec. 2.1.4. Errors in the intrinsic calibration lead to erroneous projections.

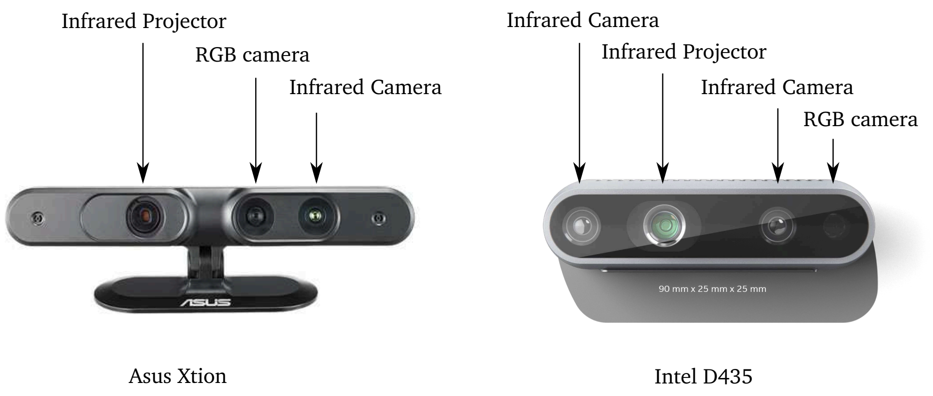


Figure 2.12: *Two exemplary composite sensors, the ASUS Xtion, and the Intel D435, based on (Li Yang Ku, 2012) and (Intel, 2020). Both consist of multiple image sensors and additionally contain infrared pattern projectors.*

An additional source of errors is the image sensor itself. Noise is induced among others due to dark currents, discretization, pixel non-uniformity, quantization, and general electronic circuit noise (Corke, 2017). Depending on the characteristics, the noise can be modeled for instance as normal distribution (Gaussian noise) or as Poisson process (shot noise, salt-and-pepper noise).

For depth sensors, different models for the noise exist as well. One approach is to represent lateral and axial noise with respect to the optical axis as Gaussian, as visualized in Fig. 2.13 (Nguyen et al., 2012). Here, axial noise characterizes the depth variation along the view axis and lateral noise is defined to characterize invalid depth values in the boundary regions of objects. For a depth camera which employs

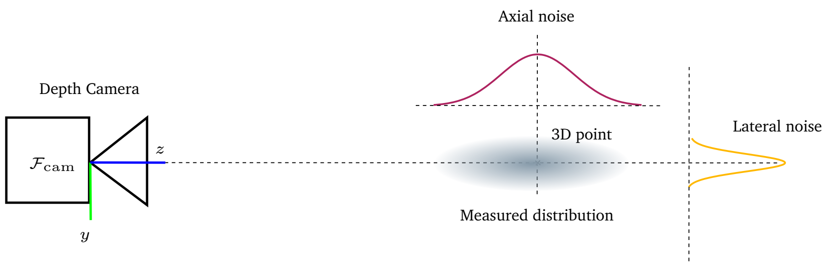


Figure 2.13: *Depth error model with Gaussian axial and lateral noise. Axial noise is defined along the view axis and lateral noise perpendicular to the view axis. Figure inspired by (Nguyen et al., 2012)*

triangulation as measurement principle, as introduced in Sec. 2.2.1.2, the standard deviation of the axial error increases quadratically. Lateral noise only increases linear as shown empirically for a Kinect camera in Fig. 2.14. For a realistic simulation of

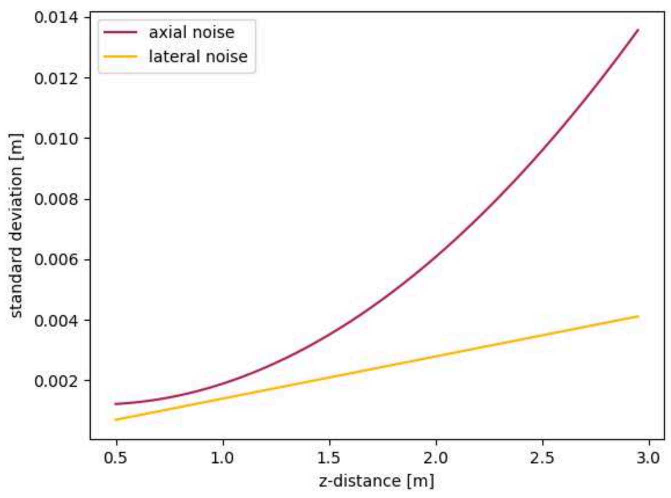


Figure 2.14: *Plot of the distance dependency of axial and lateral noise for a Kinect 3D camera as determined by (Nguyen et al., 2012). Lateral noise σ_L is proportional to the distance, while the axial noise σ_z depends on the square of the distance z.*

synthetic depth data, a large number of real-world effects have to be considered. In addition to axial and lateral noise, the interaction with specular and non-specular surfaces has to be regarded, lens distortions and quantization have to be taken into account, and the effect of motion and the shutter as well as shadows need to be considered, as shown in (Planche et al., 2017). The more accurate the model reflects real physical effects, the better it can be used to compensate them. Or, in the case of the generation of synthetic data, improved model accuracy can improve the performance of perception operator which are trained on the synthetic data.

Camera Calibration

Sensor models must be parameterized to fit to the targeted device. Therefore, we introduce and define *calibration*:

Definition 11 - Calibration

Calibration is the determination of the measurement error characteristics of a device under test and optionally the correction the aforementioned error.

The correction of the measurement error is only possible partially, as there always remains a measurement error, even if arbitrarily small. Also, the correction is optional in the base definition, but throughout the thesis the correction step is generally included as integral part of calibration.

Model calibration is the act of determining model parameters. For cameras this includes the *intrinsics* (intrinsic matrix) as well as the so-called *extrinsics*, which is the pose of the optical frame, with respect to a reference coordinate frame. The names indicate, that the intrinsics denote parameters within the camera, e.g., the optics and image sensor, and the extrinsics are parameters that describe the camera with respect to its environment.

2.2.2 Data Processing

After the general introduction of the term data processing in the introduction, see Def. 2, in this section we go into more detail on data-processing and its constituents. In a typical pose estimation context, the pose of a single or multiple target objects in the environment is required, but images from depth and RGB cameras are given as sensor data. Given sensor data as input data processing provides the means to produce the condensed information of, for example, an object pose.

First, it is to note that different types of data are involved, such as RGB images, depth images, point clouds, clusters, segments, and poses. Each of these can be represented in different ways, which depend on the mathematical model and the implementation environment. The different data types need to be represented coherently even if different implementation environments and parties are involved. The data types are denoted as *concepts* and actual pieces of data as instances.

Operator

Data processing is typically composed of individual data processing steps, which we denote as *operators*. A perception operator is depicted in Fig. 2.15, which extracts a pose from an RGB image. An operator is an executable implementation of an algorithm. It generally has one or multiple inputs and one or multiple outputs and can be parameterized using a set of parameters. Parameters of an operator are values, which influence the behavior of an operator and are often kept constant across different inputs.

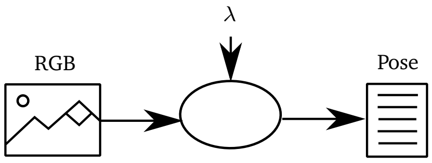


Figure 2.15: *Exemplary operator, which extracts an object pose from an RGB image.*

In the following, the operator model is formalized. An *operator* $\mathrm{O} \in \mathbb{O}$ denotes a function that maps a set of input instances to a set of output instances and optionally takes a parameter vector λ:

$$\mathrm{O} \ : \begin{cases} \mathrm{C}_{\mathrm{in}}^{1} \times \ldots \times \mathrm{C}_{\mathrm{in}}^{n_{\mathrm{in}}} \times \Lambda \rightarrow \mathrm{C}_{\mathrm{out}}^{1} \times \ldots \times \mathrm{C}_{\mathrm{out}}^{n_{\mathrm{out}}} \\ (\mathrm{i}_{\mathrm{in}}^{1}, \ \ldots, \ \mathrm{i}_{\mathrm{in}}^{n_{\mathrm{in}}}, \ \lambda) \mapsto (\mathrm{i}_{\mathrm{out}}^{1}, \ \ldots, \ \mathrm{i}_{\mathrm{out}}^{n_{\mathrm{out}}}) \end{cases} , \qquad n_{\mathrm{in}}, n_{\mathrm{out}} \in \mathbb{N}, \quad (2.15)$$

with $\mathrm{C}_{\mathrm{in}}^{j}, j = 1, \ldots, n_{\mathrm{in}},$, and $\mathrm{C}_{\mathrm{out}}^{k}, k = 1, \ldots, n_{\mathrm{out}}$, being input and output concepts of the operator, $\mathrm{i}_{\mathrm{in}}^{j}$ and $\mathrm{i}_{\mathrm{out}}^{k}$ the respective instances of aforementioned concepts, Λ the set of parameterizations, and $\mathbb{O}$ the set of operators. A set of instances is denoted as I, such as the set of input instances I_{in} and output instances $\mathrm{I}_{\mathrm{out}}$.

We can distinguish different purposes of operators in the context of perception:

- **Data acquisition**: Data needs to be gathered from available sources, such as sensors or data storages.
- **Data pre-processing**: The available data is in many cases not directly suitable for the task at hand. An algorithm may for instance require a point cloud, while only a depth image is available. The purpose of pre-processing is the preparation for the following processing steps and may include operations such as data conversion (e.g. point cloud to depth or vice versa) and noise reduction (e.g. image smoothing via sigma filter (Lee, 1983) and bilateral filters (Tomasi and Manduchi, 1998)).
- **Data extraction**: Data extraction addresses the computation of condensed representations, such as feature vectors, object bounding boxes (Redmon et al., 2016), segmentation masks (Abdulla, 2017), and also poses (Tekin et al., 2018) from given data. However, the quality of the input data does not always allow successful, fast and reliable data extraction. Therefore, additional techniques such as data pre-processing, reduction, and fusion can be required.
- **Data reduction**: Data reduction serves to reduce the size of data, which can be beneficial for the computation times. Although too much reduction leads to loss of information and reduces the quality of the outcome of the

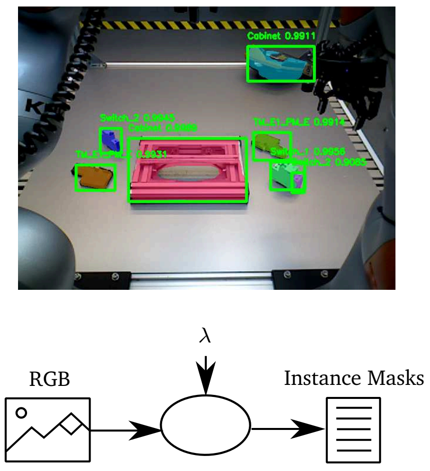
overall pipeline. Exemplary operations are cropping (Vaquero et al., 2010), down-sampling (Orts-Escolano et al., 2013), and compression (Vijayvargiya et al., 2013). Also, the use of object or instance segmentation results, e.g., for background removal falls into the category of data reduction.

- **Data fusion**: Multiple sources of data and information can be combined to improve the quality of an estimation task. This process is called data fusion and is used in many applications (Linn et al., 1991). It can be applied on all levels, from low-level sensor data to high-level interpretations. For instance, multiple raw point clouds from different viewpoints can be combined to a single and more accurate surface representation (Dietrich et al., 2016). On a higher level, object estimates from different sources can be combined (Aeberhard and Kaempchen, 2011).
- **Data generation**: The design and parameterization of perception pipelines can require large amounts of data, which is not always available for real sensors. Data generation techniques include photo-realistic rendering (Hinterstoisser et al., 2019), domain randomization (Tremblay et al., 2018), and domain adaptation (Csurka, 2017). Generated data is primarily used for offline engineering (Hafez et al., 2020), but rendered images of object hypotheses can also be used during online adaptation.

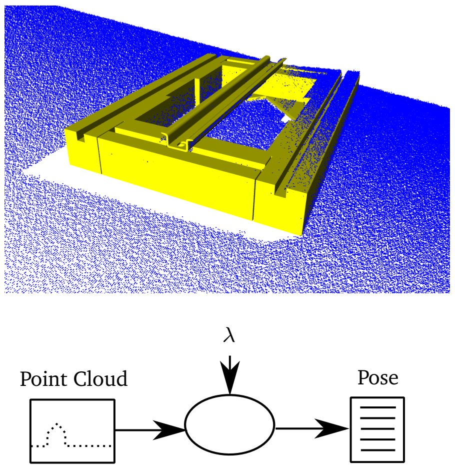
Furthermore, there are meta operators, which act upon other operators such as planning and optimization. Optimization addresses the determination of parameters of a model or system in order to improve overall performance and satisfy constraints. It covers for instance the training of neural networks (Le et al., 2011). But also inference algorithms, e.g. for factor graphs (Pfeifer et al., 2016), can be based on optimization. Planning is the process of determining a plan of actions in order to achieve a defined goal. It plays a major role for instance for the task and motion planning of autonomous systems, e.g., for autonomous object manipulation (Schmitt et al., 2017).

Pipeline

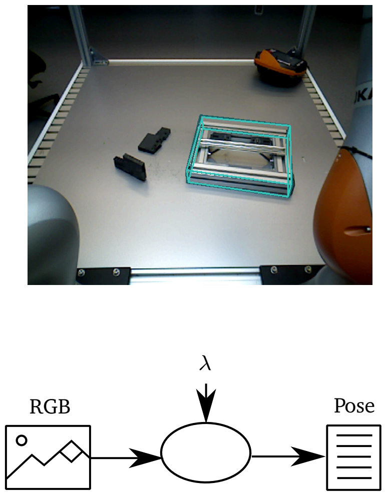
In order to solve a perception task, a single operator is often not sufficient. Therefore, multiple operators can be combined in a so-called pipeline. For pose estimation applications different operators are visualized in Fig. 2.16. The MaskRCNN operator Fig. 2.16 (a) (Abdulla, 2017) does not predict a pose, but rather masks of classified objects. But the masks can be used to reduce the search space, i.e., crop the point cloud for an operator such Fig. 2.16 (b). This benefits the pose estimation operator Fig. 2.16 (b) as the size of the point cloud is otherwise be too cluttered and large leading to false matches.



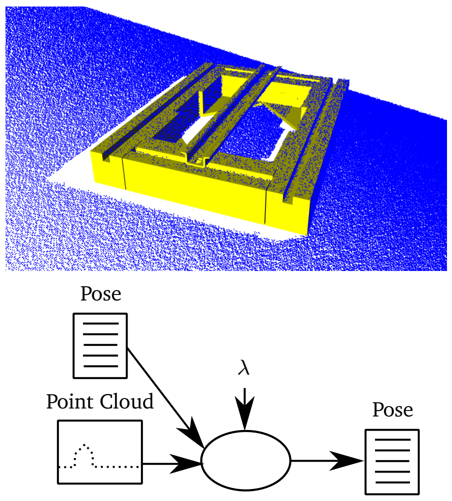
(a) MaskRCNN



(b) FastPointFeatureHistograms



(c) SingleShotPose



(d) Iterative Closest Points

Figure 2.16: *Illustration of different perception operators and their inputs and outputs. For the RGB based operators, 2D and 3D bounding boxes are used for the visualization. For the point cloud operators the estimated pose is visualized in yellow using the CAD model.*

Now consider the SingleShotPose operator Fig. 2.16 (c) (Tekin et al., 2018), which predicts the 3D pose of an object. The pose estimates are based on a 2D image only and therefore strong errors can occur due to calibration errors and projection. These can be reduced by combining the operator with an additional pose refinement step using iterative closest points Fig. 2.16 (d), although this requires a point cloud of sufficient quality. This means, the noise in the point cloud cannot be too high. Here, pre-processing can help in order to reduce the noise in the point cloud.

In general, different operators have different strengths and weaknesses and can be useless when not combined. The combination of operators is denoted as *pipeline*. A *pipeline* $\mathrm{P} \in \mathbb{P}$ is composed of several operators whose inputs and outputs are connected in a graph structure. The set $\mathbb{P}$ denotes the set of all possible pipeline structures, which depends on the available instances and operators and may be restricted by additional constraints.

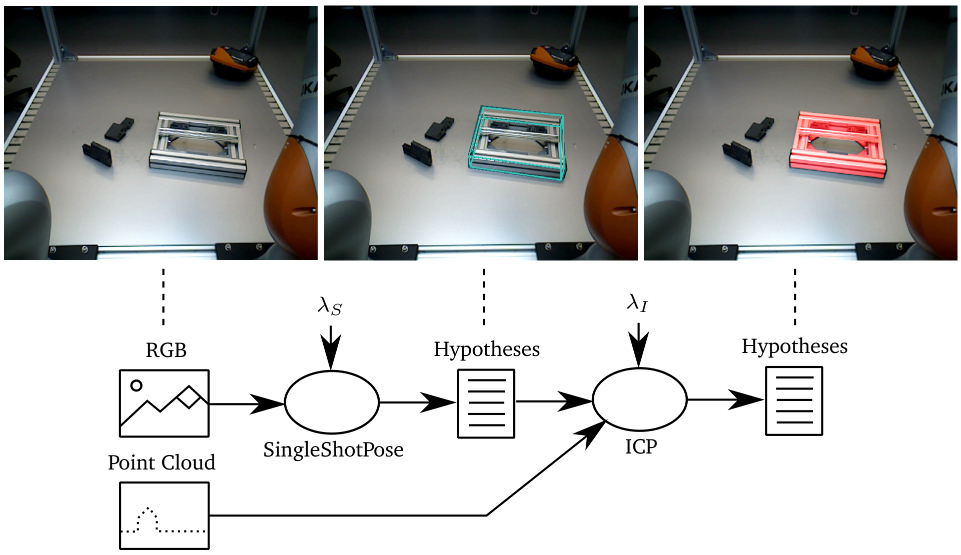


Figure 2.17: *Real perception pipeline consisting of hypothesis generation based on a neural network and a RGB image, which is refined using an iterative closest point matching on a point cloud.*

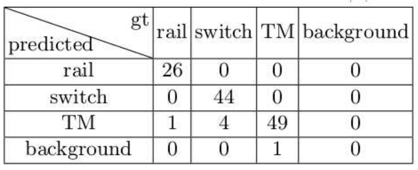
In Fig. 2.17 the combination of operator (c) and (d) of Fig. 2.16 is shown. By combining both operators, the pose uncertainty can be reduced.

In summary, data processing is prevalent at all levels of a perception system. Within the camera pixel individual illumination values are accumulated to images. Multiple images are used in order to compute a depth image and a point cloud. Images and point clouds are processed via pipelines which extract the desired information. The individual operators are itself a combination of elementary operators, which at the lowest level can be multiplications and additions.

Data Processing Uncertainty

As data-processing can cover a large field of different algorithms, models for the characterization of induced uncertainty and noise are as diverse. In the context of industrial pose estimation pipelines two kinds of metrics are especially relevant: metrics for object classification and metrics for pose estimation.

For object classification confusion matrices show the accuracy of the classifier for different object classes. An example is shown in Fig. 2.18. The rows correspond to predicted object classes and the columns to ground truth object classes, and each cell denotes the number of observations. Cells on the diagonal represent correct predictions, while non-diagonal cells contain numbers of erroneous predictions. In the example, the object called rail was interpreted once as the object TM and that switches were seen as TM four times. Condensed as a single number, metrics such as precision, recall, average precision (AP) and mean average precision (mAP) are defined. However, metrics have uncertainty on their own, as they depend on the choice and amount of ground truth data, the data split and the annotation quality of ground truth data.



gt / predicted	rail	switch	TM	background
rail	26	0	0	0
switch	0	44	0	0
TM	1	4	49	0
background	0	0	1	0

Figure 2.18: *Exemplary confusion matrix from an instance segmentation task in the assembly cell used for validation (Hafez et al., 2020).*

For object detection task as bounding boxes or segments, additional metrics such as intersection over unit are defined. It denotes the ratio between the area of the intersection of the detection and the ground truth and the area of the unit of both.

In order to characterize the error of an object pose different methods have been introduced in literature, such as visual surface discrepancy (VSD) (Hodan et al., 2018) or the average distance of the model points (ADD) (Hinterstoisser et al., 2012). In summary, different data-processing operators can require different metrics for the characterization of uncertainty. They are calculated on datasets in order to compare and rank operators and assess their performance. Note here, that it can be hard to distinguish the effects of errors introduced by an operator or by the noise and errors which are present in the input data. Also, the metrics are not always ideal, as can be seen for the case of symmetrical objects, where there is still ongoing work to improve the metrics (Hodaň et al., 2016). Metrics can have inherent uncertainty and thus have to be chosen and calculated carefully.

Dataset

The targeted application domain of a perception pipeline or perception system can be characterized via a given annotated dataset $\mathbb{D}$. The dataset needs to reflect the application properties, e.g., the expected distributions of scenes, sensor noise, and environment conditions. A dataset can be modeled as a list of scene instance sets d_j, with $j = 1, \ldots, m$ (Dietrich et al., 2019):

$$\mathbb{D} = \{\mathrm{d}_1, \mathrm{d}_2, ..., \mathrm{d}_m\}, m \in \mathbb{N}, \tag{2.16}$$

where each scene instance set contains the data for a specific scene. The scene instance sets are composed of input instances I_{in}, ground truth instances I_{gt}, and expected pipeline output instances $\mathrm{I}_{\mathrm{exp}}$:

$$\mathrm{d} = (\mathrm{I}_{\mathrm{in}}, \mathrm{I}_{\mathrm{gt}}, \mathrm{I}_{\mathrm{exp}}), \text{ with } \mathrm{I}_{\mathrm{in}}, \mathrm{I}_{\mathrm{gt}}, \mathrm{I}_{\mathrm{exp}} \subseteq \mathbb{M}. \tag{2.17}$$

Input instances include different sensor data inputs, such as images and point clouds, and prior knowledge. Ground truth instances describe the scene state in different (intermediate) representations such as object poses, masks, and bounding boxes. Expected output instances represent the information that is required by the application, such as a list of objects and their poses. The expected output instances are typically a subset of the ground truth instances.

Error Metrics

In order to formulate a synthesis problem, a metric is required to quantify the pipeline performance. Therefore, we define the metric e_{D} for the parameterized pipeline P_λ and the dataset $\mathbb{D}$ as the average of the individual error for each scene instance set $\mathrm{e}_{\mathrm{d}}(\mathrm{P}_\lambda, \mathrm{d}_j)$:

$$\begin{gathered}\mathrm{e}_{\mathrm{D}}(\mathrm{P}_\lambda, \mathbb{D}) = \frac{1}{m} \sum_{j=1}^{m} \mathrm{e}_{\mathrm{d}}(\mathrm{P}_\lambda, \mathrm{d}_j), \\ \text{with } \mathrm{d}_j \in \mathbb{D}, m = |\mathbb{D}|.\end{gathered} \tag{2.18}$$

The scene instance set error $\mathrm{e_d}(\mathrm{P}_\lambda, \mathrm{d}_j)$ requires the execution of the pipeline and computation of the instance error metric $\mathrm{e_i}$ between the pipeline output $\mathrm{I_{out}}$ and the expected instances $\mathrm{I_{exp}}$:

$$\begin{aligned}\mathrm{e_d}(\mathrm{P}_\lambda, \mathrm{d}_j) &= \mathrm{e_d}(\mathrm{P}_\lambda, (\mathrm{I_{in}}, \mathrm{I_{gt}}, \mathrm{I_{exp}})) \\ &= \mathrm{e_i}(\mathrm{P}_\lambda(\mathrm{I_{in}}), \mathrm{I_{exp}}) \\ &= \mathrm{e_i}(\mathrm{I_{out}}, \mathrm{I_{exp}}).\end{aligned} \tag{2.19}$$

The instance error metric $\mathrm{e_i}$ depends on the application and the type of expected pipeline output instances $\mathrm{I_{exp}}$ and needs to be chosen accordingly.

2.2.3 Engineering Design Synthesis

Now that the individual components of a perception system, sensor, operator, pipeline, dataset and error metrics are introduced we can address the design synthesis, c.f. Def. 4, of the perception system. Within the design synthesis the hardware and software components and their connections are chosen and parameterized. An overview of the perception system design synthesis process is given in Fig. 2.19, which builds upon prior work of Hagelskjær et al. (2018). It consists of multiple steps, targeting the different dimensions of the configuration space. Furthermore, an online and offline view with distinct optimization loops are defined. In the following, we discuss the different steps on the example of the flexible assembly system as introduced in Sec. 1.3. Note here, that the sequential ordering of the selection steps does not strictly apply in practice. The choices can be done in arbitrary order. For instance, the choice of a data processing step can be done first and influence the choice of the sensor. However, once the sensor is ordered and available a change involves higher cost. In that sense, the left to right ordering in Fig. 2.19 reflects the cost and effort to revoke a decision, which decreases towards the right.

Hardware Selection

The first step is the hardware selection. Here, appropriate sensors are selected and positioned. In the flexible assembly use case different aspects must be taken into account. First, the measurement principle and type of sensor is to be chosen. RGB cameras are more typically more compact than depth or RGBD sensors

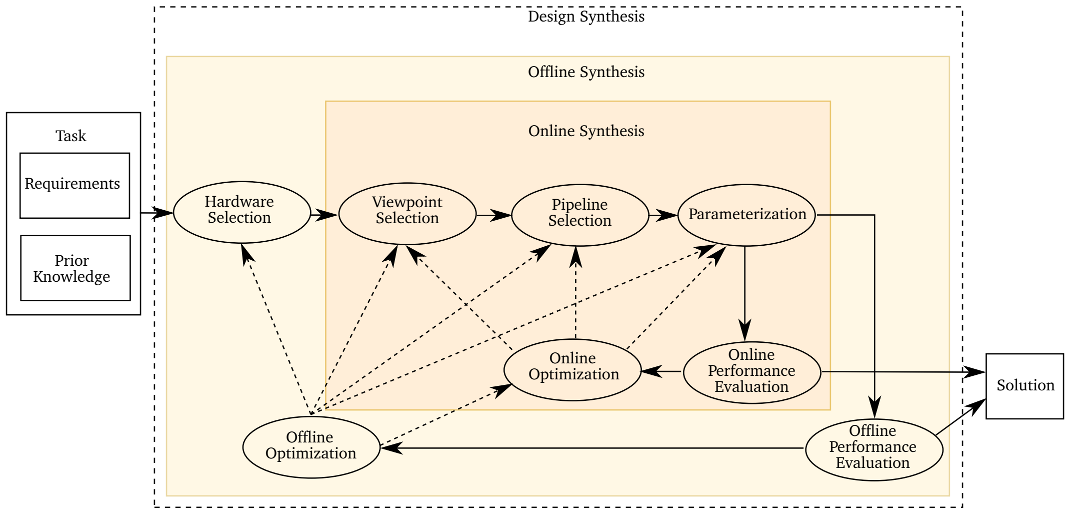


Figure 2.19: *Design Process for Perception Systems which covers an offline and an online optimization loop, based on Hagelskjær et al. (2018).*

and in most cases cheaper. Although there is a difference between industrial and consumer cameras. Consumer cameras are cheaper, but often not suitable for the harsh environments and durability requirements in an industrial application. The sensor choice does influence the set of possible data-processing steps and there is an interdependency between both. If no suitable data-processing pipeline is available for depth data, no depth sensor should be chosen. It is also possible to decide for a set of multiple sensors and select a subset of the sensors for sub-tasks.

Furthermore, hardware interfaces must be taken into account. The sensors need to be fixated such that vibrations induced by moving actuators do not move the sensor too much. For a fully automated configuration, structural simulations or calculations are required in order to assess the stiffness of a given structure. Communication interfaces may also pose constraints. For instance, the full bandwidth within the USB 3.0 standard is only specified up to a cable length of $0.5\,\mathrm{m}$. And placing a sensor on a movable robot arm requires to have reliable and endurable cabling connection.

In the exemplary flexible assembly cell, a setup of three cameras is chosen. A consumer RGBD camera is placed statically in between the robot arms and covers the working area of the robot arms. Additionally, an industrial RGB camera is placed at each end-effector, which allows to closely look at objects of interest. The industrial RGB cameras are connected via Ethernet, for which the use of spiraled cables and long distances are feasible.

Viewpoint Selection

The sensor or sensors must be positioned. Here, different aspects have to be considered. As noted in Sec. 2.2.1.4, axial depth noise for instance increases quadratically with the distance. Also, the closer the camera, the higher the ratio of pixels which show the object with respect to the overall number of pixels. Therefore, a camera should generally be placed as close as possible, while still covering the required working area. Another aspect is to place sensors statically or dynamically. A dynamic sensor is placed such that it can be repositioned at runtime, e.g., on a robot arm. A static sensor has a fixed position. Therefore, it is not subject to motion blur due to its own movement. It can be used, when the manipulators are moving. Dynamically placed sensors may require to stop the task and take sensor data from a static position. This can increase the cycle time of the task. But in some applications sufficient accuracy can only be achieved, when the sensor is placed very close to the target object. Here, a dynamic sensor placement is advantageous, as it can be adapted to the given task.

Data Processing Selection

The next step is the selection of the data processing. It depends on the hardware sensor setup, as the hardware defines the type of sensor data as well as the quality of the data in terms of noise and errors. Accessible data sources for the selection are (scientific) publications, books, manuals, tutorials, and benchmarks. Especially benchmarks are an important resource as they provide a performance comparison. For object pose estimation, the BOP benchmark is notable and clearly shows that the performance of different methods depends on the type of object and task (Hodan et al., 2018). The available information again represents abstract performance models via error metrics such as the visual surface discrepancy. The performance models may help in selecting appropriate algorithms, but an evaluation with real data from the actual task setup is required to increase the level of confidence.

In practice, performance may not be the main criterion. The task is to achieve the goal while reducing the overall engineering and system costs. Therefore, operator availability plays a crucial role. Perception operators are organized within libraries such as OpenCV, Point Cloud Library (PCL) or Halcon. But operators can also be provided as individual repositories. A reimplementation of algorithms and approaches can take significant engineering effort. Therefore, it is necessary to be able to easily integrate and quickly evaluate operators of arbitrary origin. Although in some cases a reimplementation is necessary in order to keep the software architecture of the perception system clean.

Parameterization

Once, the hardware and software are set up, the parameterization of the overall system can be addressed. As noted before, the sensors itself may contain data-processing and parameters which can be adapted such as focus, shutter and gain. The following data-processing pipeline needs a parameterization as well. Operators can have from a few, over dozens up to millions of parameters. For a low number of parameters, the manual selection of parameter values can be an option. For larger amounts, optimization based on labeled input data is required. The latter is the case for processing steps based on machine learning. Here, a hierarchy of optimization problems is given with the training of, for instance, a neural network at its core. Around the core there are other so-called meta parameters, such as the learning rate, the choice of the dataset, etc.

Additionally, some operators require the pre-computation of perception models which allow to reduce the runtime or encode object knowledge. Examples are Linemod (Hinterstoisser et al., 2011), SSD (Liu et al., 2016), and 3D shape-based matching as implemented in the Halcon library (Ulrich et al., 2011).

Offline Optimization Loop

The offline optimization loop addresses the design synthesis for a static configuration. This is the typical case in classical automation systems, where tasks do not change. Here, the system can be optimized once and remain unchanged. In a flexible automation setting additional online adaptation is required, which is outlined in the next section.

Two major steps are involved, the offline performance characterization and the choice of a new configuration. The offline performance characterization involves the computation of performance metrics based on a set of annotated data points. The annotated data points are, for instance, specific scenes or object placements which occur within a task which are annotated with expected results of the perception system. If all requirements from the task are fulfilled the current configuration is accepted as a solution. If not, a new configuration is selected and evaluated as part of the offline optimization. Here, the costs of evaluating a new configuration must be considered. The change of hardware components involves high costs and should be avoided if possible. Good performance predictions, e.g. via simulation or by experience can help to reduce the risk of having to change the hardware.

Online Optimization Loop

Once the perception system is deployed, as in the example of the flexible assembly cell, it is confronted dynamically with new tasks as well as variations and deviations in their execution. The online adaptation allows to adapt the perception system dynamically during runtime. The main difference here is that no ground truth data is available for the optimization. Therefore, metrics are required which can assess the quality of the current belief. Theses metrics are subject to error.

Furthermore, the configuration space is reduced during online optimization. For instance, no sensors can be changed and only dynamically positioned sensors can be repositioned. Also, the search in the complete parameter space can take too long and may not be feasible as in the offline optimization loop. Fast reaction is required.

2.3 Automated Synthesis

In the prior sections different aspects of the design synthesis of perception system for industrial applications are highlighted. In this section we focus on the automation of the synthesis process, which is the goal of this thesis. First, requirements are summarized and then a high-level overview over the automated procedure and involved models is given.

A summary of important aspects for the modeling and execution of automated synthesis is given in the following:

- **Abstraction via Models:** Models are required to abstract reality and to be able to evaluate configurations without costly experiments in the real world. This is true for the online and the offline use case.
- **Procedural and declarative models:** There is a demand for declarative and procedural models. The description of the properties of a sensor can done

within a declarative model. The performance model of an operator is procedural and may require its execution for the evaluation on given input data.

- **Inherent Hierarchy:** The synthesis of perception systems is done on different levels of abstraction and granularity. Sensors can range from single photodiodes, to complex integrated smart cameras with multiple image sensors and complex data-processing. In the same way, data-processing operators are composed of other data-processing operators and in that sense are pipelines of more elementary operators. Therefore, a hierarchy of descriptive granularity is given. Furthermore, models inherently represent a hierarchy of abstractions. At the most concrete level of the abstraction hierarchy is the real world and different model representations approximate the real world with different degrees of abstraction and modeling errors.

- **Uncertainty:** Models are approximations and the degree to which they are wrong can often be uncertain. In general, uncertainty is prevalent at all levels of the design synthesis, sensor data is noisy, data-processing induces errors, pose uncertainty models are only approximations, and performance estimates and metrics are subject to errors as well.

- **Heterogeneous fields:** The synthesis of perception systems covers different perception disciplines such as sensor fusion, pose estimation, and object classification. Each discipline has own models and specialized approaches which need to be represented and possibly combined.

- **Large configuration space:** The configuration space is large and cannot be evaluated in its entirety via experiments. Models allow for lowered computation demand but may induce approximation errors. Also, it does not need to be searched in its entirety all the time depending on the engineering phase.

- **Heterogeneous search methods:** On the different levels of the overall problem hierarchy different approaches for the synthesis can be required.

- **Online and offline synthesis phases:** The design synthesis has the distinct phases of online and offline synthesis, where different requirements are given. In the offline case, ground truth annotations are available, whereas in the online case no such information is given. A common infrastructure for both phases is desired.

- **Heterogeneous software environments:** Due to the variability in specialized problem domains, the software environments vary. The integration of different software environments poses a major engineering hurdle and therefore should be facilitated.

Therefore, we introduce a modeling and synthesis approach, which addresses large parts of the aforementioned aspects.

2.3.1 Synthesis Procedure

In Fig. 2.20 a general overview over the major steps in the design synthesis procedure is given. As introduced before, there is a need to cover two different phases, the offline engineering with an engineer in the loop and the adaptation phase with the automation system in the loop. We differentiate between three different steps within the synthesis, the problem instantiation, the configurations space exploration, and the evaluation. The instantiation serves to formulate the incoming task as a synthesis problem. This entails the definition of the configuration space as well as the constraints and goals. Depending on task and synthesis method, different data structures are required.

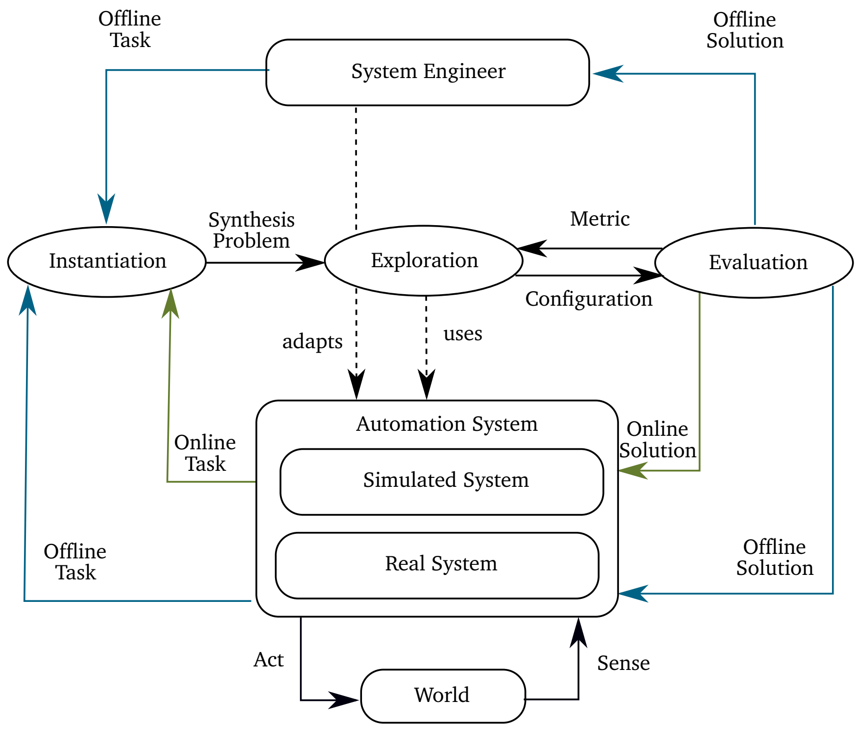


Figure 2.20: *Synthesis procedure represented as the three major steps: instantiation, exploration, and evaluation.*

Exploration and evaluation are tightly coupled. The exploration steers the overall search procedure and provides new configurations which are to be evaluated. This requires knowing the performance metrics of prior evaluated configurations. Within the evaluation step the performance of a configuration is evaluated. Furthermore, as part of the evaluation, a suitable solution is selected and returned.

Within this thesis, multiple synthesis approaches are introduced and applied to the different phases. They share a common modeling infrastructure but provide different solutions for the exploration and evaluation.

2.3.2 Model Structure

In order to facilitate reuse and abstraction, a hierarchical modeling approach is used for the procedural and declarative models of the perception domain as well as the synthesis system. This is illustrated in Fig. 2.21.

The modeling of perception systems requires multiple levels of a model hierarchy. The foundational model is the meta model, where declarative models such as operator, pipeline, and hierarchy are defined. Procedural models in the meta model contains operators which can act on instances of the declarative models. For instance, a planning or optimization algorithms composes, combines and parameterizes operators in order to achieve a goal.

Different other sets of models are defined which build on another. Elementary sets of models are mathematical models where elementary mathematical concepts and operators can be described. More complex and domain specific sets of models are defined for the robotics and the perception domain. In addition to the hierarchy of model sets, each individual model, e.g., of the sensor can be defined on different levels of abstraction itself. This means that hierarchy is given in different ways.

For each set of models declarative and procedural models are kept together. The rationale behind this decision is that with declarative models alone only a limited set of usable inferences can be performed. The combination of procedural and declarative models allows to translate models into executable actions.

Furthermore, we noted that different sub domains may employ distinct models which are not directly interoperable. Here, the approach is to allow heterogeneous models and add executable procedural models which allow to convert different representations. Furthermore, the meta-model allows techniques for model exploration, transformation, and generation.

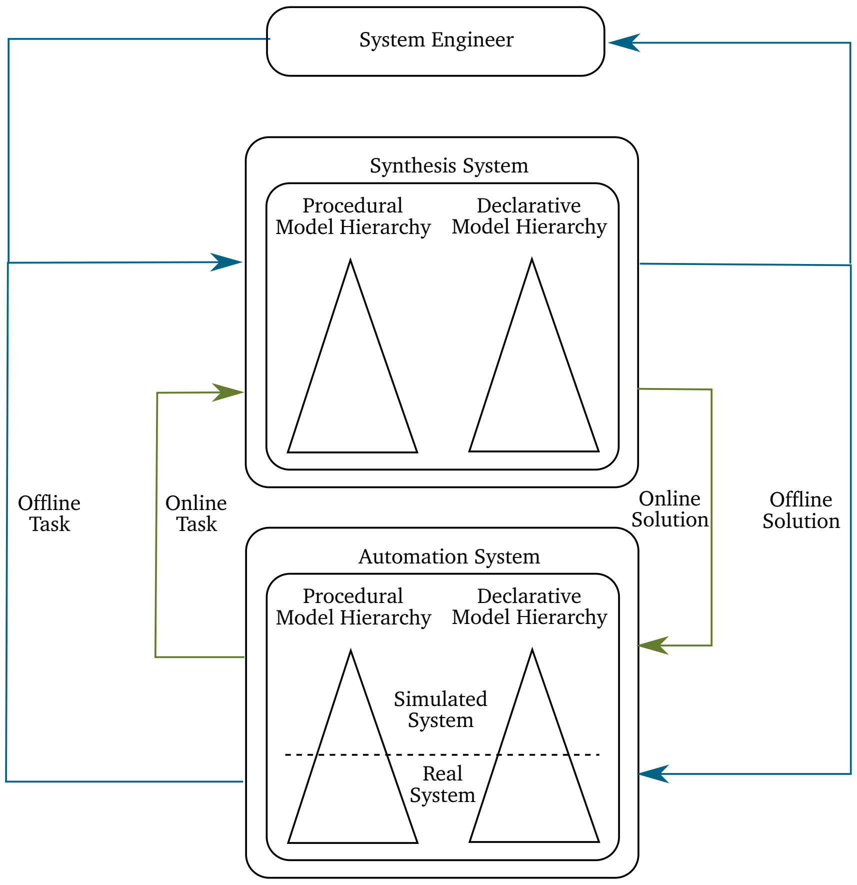


Figure 2.21: *Procedural and declarative hierarchical models for the automation system and the synthesis system.*

In the next chapter, the modeling approach and infrastructure are introduced in more detail. In Chapter 4 an approach for efficient offline optimization is introduced and evaluated in a real-world assembly application on the research demonstrator. In Chapter 5 online reconfiguration will be addressed and how uncertainty across the coordinate frames can be considered. In Chapter 6 the exploitation of hierarchical models for online reconfiguration will be presented and evaluated in an assembly context.

Chapter 3

Hierarchical Modeling Approach and Infrastructure

In the prior chapters, the importance of *models* for the perception system synthesis has been highlighted. In this chapter the modeling approach that we employ, as well as employed models for the perception system domain are discussed. An overview of the modeling infrastructure and associated processes is given in Fig. 3.1. At the center is the hierarchical model. It is comprised of a hierarchy of procedural models and a hierarchy of declarative models. Both define abstract representations of procedural and declarative knowledge.

The declarative and procedural models can be differentiated into meta models and domain models. Meta models include concepts such as operator and pipeline as well as executable procedural models such as planners and optimizers which can act upon operators and pipelines. In order to render models executable a code generation procedure is employed where the implementation agnostic models are converted into executable procedural models in one of different available implementation languages.

Models can be created in different ways. A manual model engineering approach involves a combination of engineering knowledge and experience for the implementation of procedural and declarative models. But also an automated generation of models is possible, which we denote as *model harvesting*. Here, the source code and documentation of a library are used in order to generate declarative and procedural models with minimal manual intervention.

The hierarchical model is then employed for the synthesis of perception systems as shown in the lower part of Fig. 3.1. It has two major steps, the problem instantiation

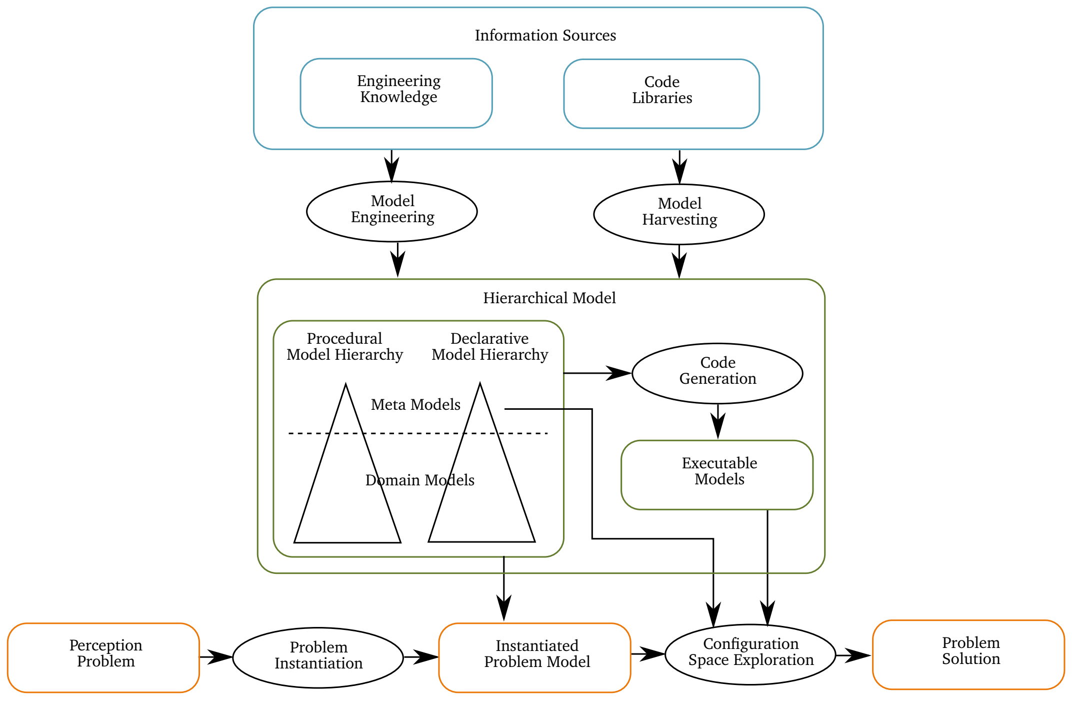


Figure 3.1: *Overview of the modeling system. Manual model engineering and automated model harvesting are used in order to generate domain models. The hierarchical model is comprised of procedural and declarative models on different levels of abstraction. A code generation step provides executable models. The hierarchical model allows to instantiate the perception synthesis problem and provides meta models for configuration space exploration.*

and the configuration space exploration. The former formalizes the perception problem and represents it using concepts and operators of the hierarchical model. The latter step, the configuration space exploration denotes the search and evaluation of different solutions variants. It reuses meta models for exploration such as planning and optimization algorithms and furthermore requires executable procedural models of the domain. The result is a solution for the given perception problem.

In this chapter the modeling approach as well as the different procedures of the modeling system are introduced and discussed for the perception domain.

3.1 Set-Theoretic Formal Model

The formalization of declarative and procedural models in this section is based on and was first presented in the prior publications (Kast et al., 2019a,b,c, 2020). In order to allow handling and representation of different levels of abstraction, both model categories provide consistent hierarchical structures. No distinction is done between symbolic and sub-symbolic entities, which therefore facilitates the transition between both worlds. As mathematical apparatus, set-theory is used to formalize the models. The declarative models are represented via so-called *concepts* and the procedural models via so-called *operators*, which are both introduced in the following.

3.1.1 Concepts

A concept describes entities in our domain. Let's consider the example of the concept *sensor*. Different sensors, such as RGB cameras and temperature probes are sensors, but represent distinct sets within the sensor concept. Thus, the concept *sensor* is more abstract than the concepts of *camera* and *temperature probe*. We capture the desired properties with the following set-theoretic definitions:

- A concept base B_Γ is the set of instances, not necessarily finite.
- A concept C is a subset of B_Γ, i.e., $C \subseteq B_\Gamma$. Therefore, it also represents a set of instances.
- A concept class Γ is the set of concepts C_i that have a common concept base B_Γ, i.e., $\forall\ C_i \in \Gamma, b \in C_i : b \in B_\Gamma$.

- A partial order $\mathcal{M}$, describing *more detailed than*, can be defined on Γ: $(C_i, C_j) \in \mathcal{M}$ iff $C_i \subset C_j$.

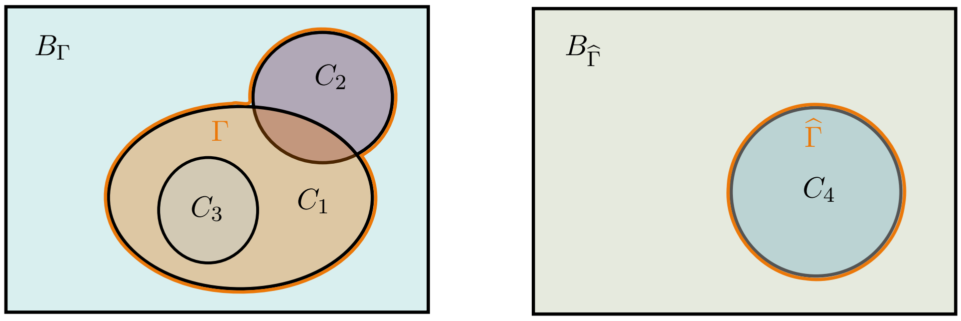


Figure 3.2: *Exemplary visualization of the set-theoretic relations between base, class, and concept. The three concepts C_1, C_2, C_3 are subsets of a common concept base B_Γ and therefore form the concept class Γ. Concept C_4 in contrary has the concept base $B_{\widehat{\Gamma}}$ and concept class $\widehat{\Gamma}$. Concept C_3 is more detailed than C_1, which is not true for any other pair of concepts in the example.*

This definition allows to capture the sensor example from the previous paragraph, where the camera and temperature probe concepts are more detailed than the sensor concept. Concepts can also be composed of different sub-concepts. For instance, an industrial camera is typically composed of a body and a lens. Additionally, application relevant properties such as the camera intrinsics need to be modeled. In order to capture composition and properties, we introduce so-called *composite concepts*:

- Given an ordered set $\mathcal{R}$ of identifiers, e.g. strings, its elements $r \in \mathcal{R}$ are called roles. Thus, for each subset $\mathcal{R}_i \subseteq \mathcal{R}$ with $n_{\mathcal{R}_i} := |\mathcal{R}_i|$ there exists a bijective mapping $\mathcal{J}$ to $\mathbb{N}_{n_{\mathcal{R}_i}} := \{1, \dots, n_{\mathcal{R}_i}\}$, i.e., $\mathcal{J}(r) \in \mathbb{N}_{n_{\mathcal{R}_i}}\ \forall r \in \mathcal{R}_i$ and $\mathcal{J}^{-1}(n) \in \mathcal{R}_i\ \forall n \in \mathbb{N}_{n_{\mathcal{R}_i}}$.
- A composite, recursively defined concept $C = \Pi_{r \in \mathcal{R}_C} C_r = C_{r_1} \times C_{r_2} \times \dots \times C_{r_j}$ with $\mathcal{R}_C$ being the specific set of roles for this composite concept and $j := |\mathcal{R}_C|$.

Therefore, a composite concept can be represented as a (directed) graph. An exemplary and oversimplified concept graph for a camera is depicted in Fig. 3.3. Nodes correspond to concepts and edges correspond to roles. Edges point from a concept to a sub-concept. The extension of the scope of roles to ordered, finite lists and unordered finite sets is possible and is denoted by curly and square brackets: $r[]$ and $r\{\}$. The ordering and uniqueness of the roles allows for efficient instance and concept comparison, which is important for meta-operators such as planning. The partial order $\mathcal{M}$ can be generalized for composite structures using recursion. Given

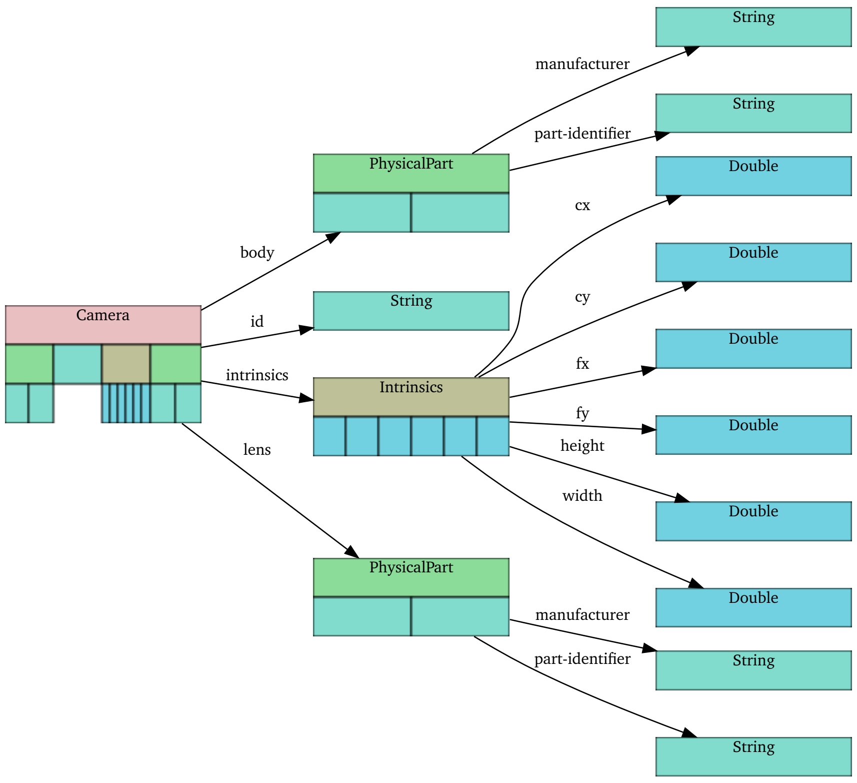


Figure 3.3: *Exemplary composite concept of a camera that consist of body and lens, which are both specified as a PhysicalPart concepts with manufacturer and part-identifier given. Additionally the property with the identifier intrinsics is modeled as a concept of type Intrinsics. Each graph vertex corresponds to a concept and each graph edge to a role. Colors and box structures are used to facilitate the identification of type and structural relations.*

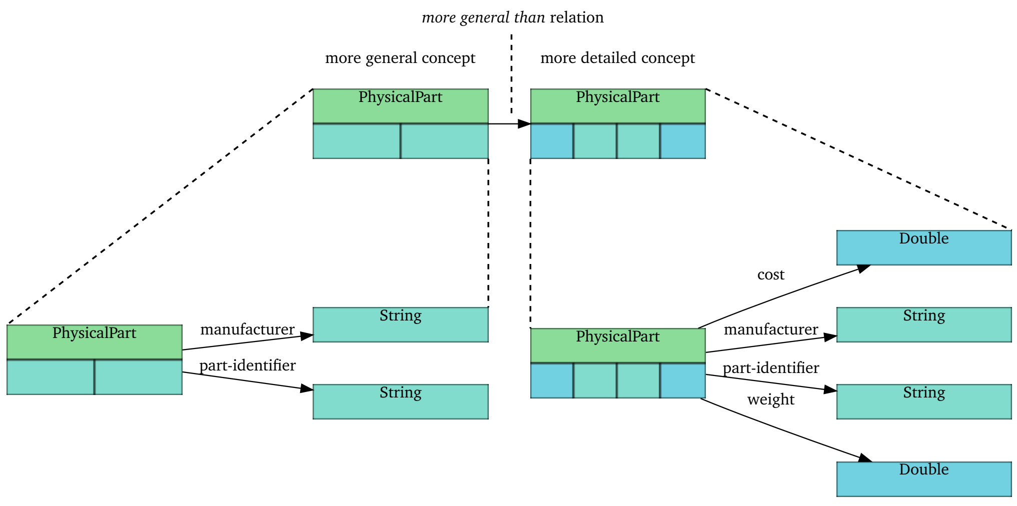


Figure 3.4: *Two variants of the concept PhysicalPart are displayed as a graph with two vertices and an edge that points from the more general to the more detailed concept. The more detailed concept has the additional roles cost and weight and the roles from the more general concept are preserved.*

$C, \overline{C} \in \Gamma$, we define $\mathcal{M}$ as

$$(C, \overline{C}) \in \mathcal{M} \text{ iff } (C_r, \overline{C}_r) \in \mathcal{M}\ \forall r \in \mathcal{R}_{\overline{C}}.$$

Therefore, the sub-concept C requires all roles from the concept $\overline{C}$, which can be formally represented as:

$$C \cong \Pi_{r \in \mathcal{R}_C \cap \mathcal{R}_{\overline{C}}} C_r \times \Pi_{r \in \mathcal{R}_C \setminus \mathcal{R}_{\overline{C}}} C_r \subseteq \overline{C} \times \Pi_{r \in \mathcal{R}_C \setminus \mathcal{R}_{\overline{C}}} B_{\Gamma(r)}.$$

A minimal example of a concept hierarchy is depicted in Fig. 3.4. Here, two additional roles for *cost* and *weight* have been added to the original concept of *PhysicalPart*. A more complex example is illustrated in Fig. 3.5 with different variants of the *Camera* concept. For the visualization a directed acyclic graph is used whose vertices represent concepts and whose edges point from more general to more detailed concepts.

An actual element of the set of a concept is an *instance*. In order to define an instance, the values of all leaf concepts of the concept tree need to be specified. In the example of the more general definition of the *PhysicalPart* in Fig. 3.4, this means that an instance specifies the *String* concepts *manufacturer* and *part-identifier*, for example with values "Industrial Camera Manufacturer Corporation" and "cam-12345". This means that an instance of a more detailed concept can be converted to or extends an

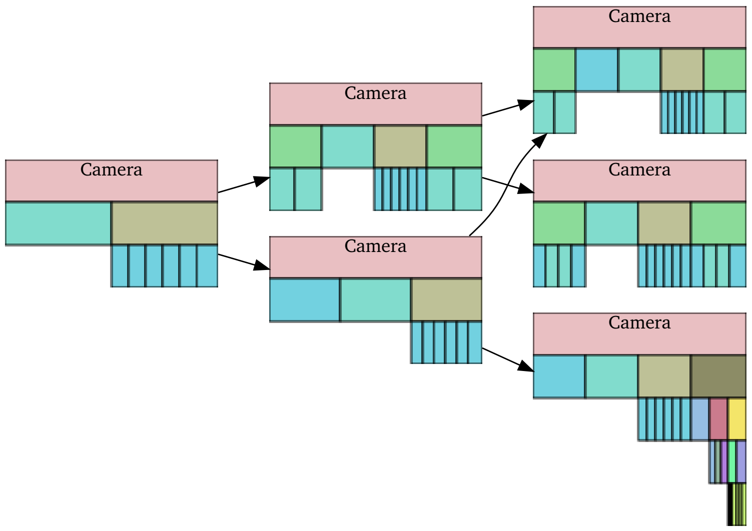


Figure 3.5: *A concept hierarchy of multiple variants of the Camera concept. The definition of the partial order allows that a a concept specializes multiple concepts, as for instance the top right Camera concept.*

instance of a more general concept. This is similar to inheritance in object oriented programming languages.

An important aspect, e.g. for planning purposes, is the ability to compare instances. Therefore, a *compare relation* F_C is defined. Two instance b_i, b_j of concept C are similar if $(b_i, b_j) \in F_C$. For instances of composite concepts similarity applies when all leaf instances are similar. The compare relation does not necessarily require two instances to be identical, but can be defined differently. The intended meaning is that the second instance does not add information on the level of abstraction of the domain.

3.1.2 Operators

The declarative models as introduced only provide limited use, unless there exist executable procedural models acting on them. These procedural models are given in the form of so-called operators, which we define as follows:

- An operator $\pi \in P$ maps given input concepts I_{r_i} to output concepts O_{r_j} with given input roles $r_i \in \mathcal{R}_{\pi,I}$ and output roles $r_j \in \mathcal{R}_{\pi,O}$, i.e., $\pi : \Pi_{r_i \in \mathcal{R}_{\pi,I}} I_{r_i} \rightarrow \Pi_{r_j \in \mathcal{R}_{\pi,O}} O_{r_j}$.

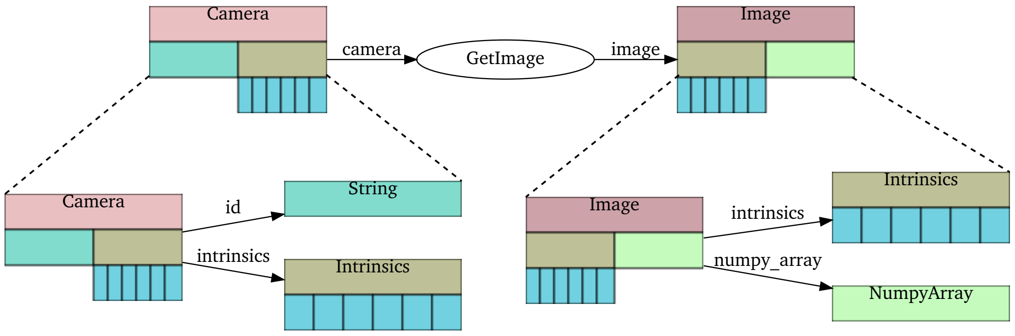


Figure 3.6: *Example of an operator. It is named GetImage and takes an instance of the Camera concept as input and produces an Image instance. The input is not consumed, as a camera instance is still available after the operator execution.*

- The operator mapping can be defined explicitly in the form of a symbolically-representable mapping, e.g. a mathematical formula, or implicitly as the result of a computation or real world execution.
- Operators can modify instances, which is realized by invalidating the input instance. Invalidated inputs are called *consumed*. The set of roles corresponding to inputs that are consumed by the operator is denoted by $\mathcal{R}_\pi^c \subseteq \mathcal{R}_{\pi,I}$.
- Operators do not have an internal state and are thus fully functional.

An example for an operator is shown in Fig. 3.6. The operator takes an instance of the *Camera* concept as input and produces an *Image* instance.

Operators are elements of a functional space and therefore are modeled as instances of an *Operator* concept on a higher level of abstraction. This higher abstraction layer will be denoted as *meta domain*. On this meta domain, operators can be defined that act on other operator instances as inputs, e.g. planning algorithms. The actual operator concept from the meta domain as used throughout this work is depicted in Fig. 3.7. The *Operator* concept contains input and output roles, as well as reference to executable code. The actual executable code is referenced within *operator_callable*.

A hierarchy of operators can be established automatically based on the concept hierarchy of inputs and outputs. If the following conditions hold true, operator π_1 is more detailed than π_2:

- all input and output roles of operator π_2 are elements of the role set of operator π_1: $\mathcal{R}_{\pi_2,I} \subset \mathcal{R}_{\pi_1,I}$ and $\mathcal{R}_{\pi_2,O} \subset \mathcal{R}_{\pi_1,O}$,

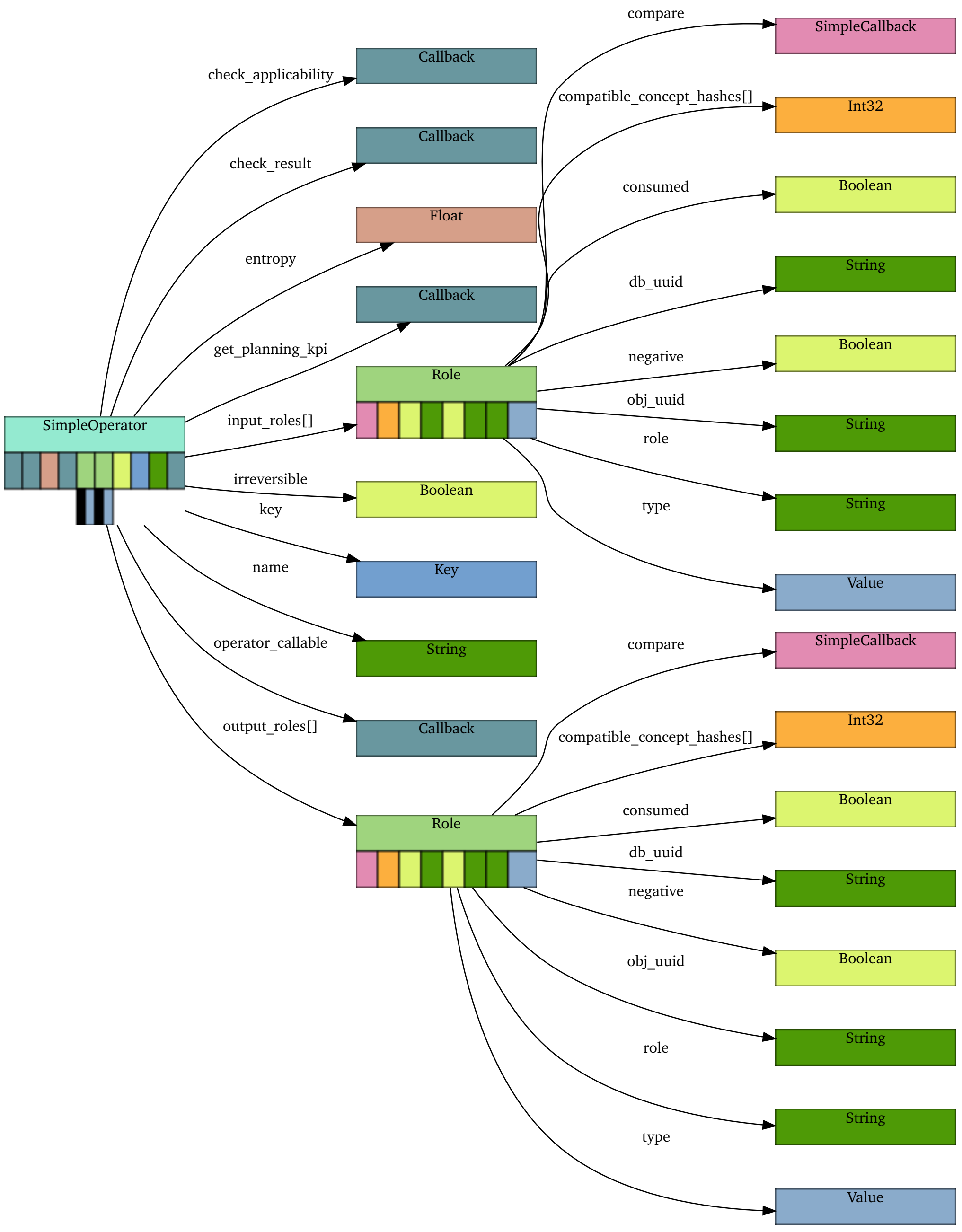


Figure 3.7: *The Operator concept in the meta domain as used throughout this work. Among others, it contains references to different concepts of executable functions, e.g. operator_callable. The consumed property of the input roles as introduced as part of the operator definition is part of the Role concept.*

- all (common) input concepts $I_{r,1}$ and outputs $O_{r,1}$ of operator π_1 are more detailed than those of operator π_2: $(I_{r,1}, I_{r,2}) \in \mathcal{M}\ \forall r \in \mathcal{R}_{\pi_2,I}$ and $(O_{r,1}, O_{r,2}) \in \mathcal{M}\ \forall r \in \mathcal{R}_{\pi_2,O}$.

3.2 Perception System Models

In this section we cover how the perception system related components as introduced in Chapter 2 can be represented within the hierarchical modeling formalism. Based on the examples from the perception system domain different design decisions and alternatives are discussed. We start with pose representations and object models, which are elementary and reused multiple times. We proceed with camera models, sensor data types, state, and belief representations. Furthermore, models for parameters and perception algorithms are introduced, which are an important part of the configuration space. Finally, models for datasets and simulation are introduced. The presented models are used throughout the following chapters. Additionally some implementation aspects are covered, as the compatibility of the modeling approach with existing operators of different kind is important to its usability and applicability.

3.2.1 Pose Model

Different representations for poses, which represent different abstraction models are defined. In Fig. 3.8 an excerpt of the pose hierarchy as modeled is visualized, which covers two dimensional as well as three dimensional pose models.

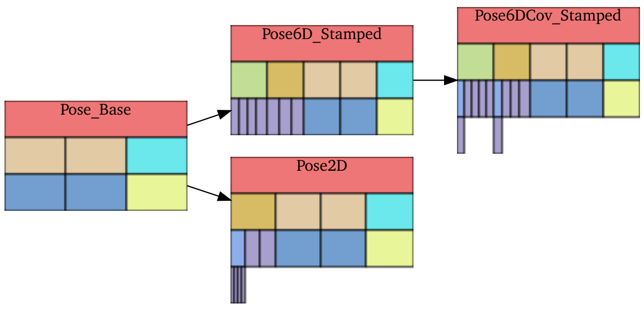


Figure 3.8: *Excerpt of the pose model hierarchy, which represents 2D and 6D poses with and without associated uncertainty.*

A detailed model of the most detailed concept *Pose6DCov_Stamped* is shown in Fig. 3.9. Here it is composed of a position and a quaternion, which is one of the possible representations, which are introduced in Sec. 2.1.3. Additionally, each of them is associated with a 3x3 covariance matrix, which models the inherent uncertainty. Furthermore, identifiers are given for the start frame and the target frame, which are represented by the pose. Additionally a time stamp is given in this model instance. The structure is similar to the pose message definitions in the Robot Operating System (ROS) (Quigley et al., 2009) and in Sec. 3.3.2 the automated import of concepts from a middleware such as ROS will be highlighted. The main difference with respect to the message definitions of a middleware with focus on communication and deployment are the inherent hierarchical abstractions which allow the planning and execution of operators on different levels of granularity.

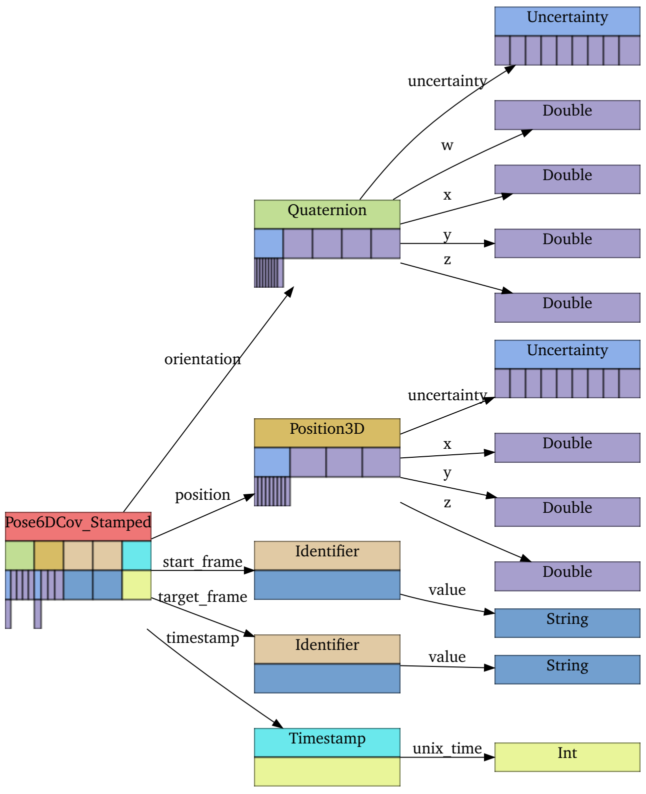


Figure 3.9: *Model of the most detailed pose from the presented hierarchy. The pose models position and orientation in the three dimensional space which are associated with uncertainty.*

Common procedural knowledge involved in handling poses includes the composition and the inversion, as introduced in Sec. 2.1.3. In Fig. 3.10 the pose composition is displayed. Each operator can operate on the given input concepts as well as more detailed instances of the input concepts.

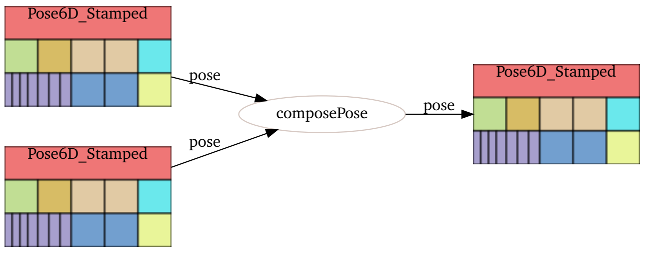


Figure 3.10: *Procedural model for the composition of poses.*

The operator model does contain not only the definition of inputs and outputs, but also the definition of internal code in a programming language of choice. Both are used to generate executable code. For the interfacing of operators in different implementation languages a code generation layer is implemented, which provides auto-generated conversion code based on (Beazley et al., 1996). This modeling approach allows to modify and create models, as required on the meta level.

The presented declarative and procedural models only provide an excerpt of the domain. More operators exist, e.g., for conversion and visualization.

3.2.2 Object Model

The model for objects reuses the pose definition. A realization is visualized in Fig. 3.11 and shows the basic components, an identifier, a mesh, and a pose. Certainly, more properties can be required and easily added to the model. But the modeling approach premise is to only model declarative knowledge, which is actually used, e.g., within procedural models. Unless it is used, it could be denoted as "useless". Therefore, the model is kept lean to the concepts which are actually used in the application context.

The object model hierarchy is visualized in Fig. 3.12. The differentiation between the different models are the attributes and the modeling details of the attributes. At an abstract level, the pose of an object is not required, for instance to state which types of objects an assembly process requires. A camera is also modeled as an object and is introduced in the following.

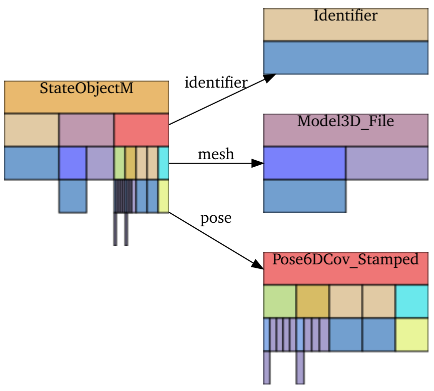


Figure 3.11: *Basic object model, which is comprised of an identifier, a mesh, and a pose.*

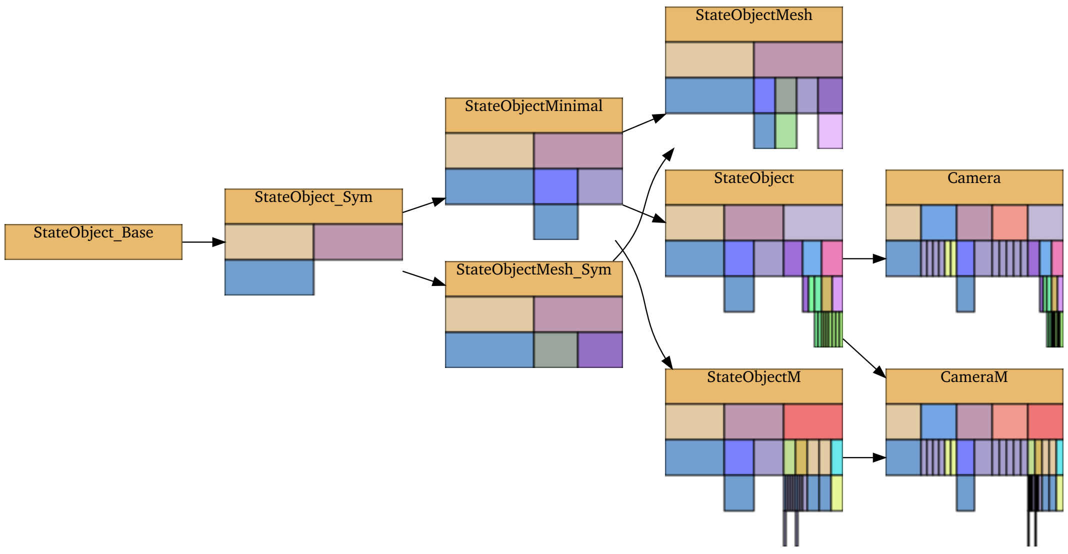
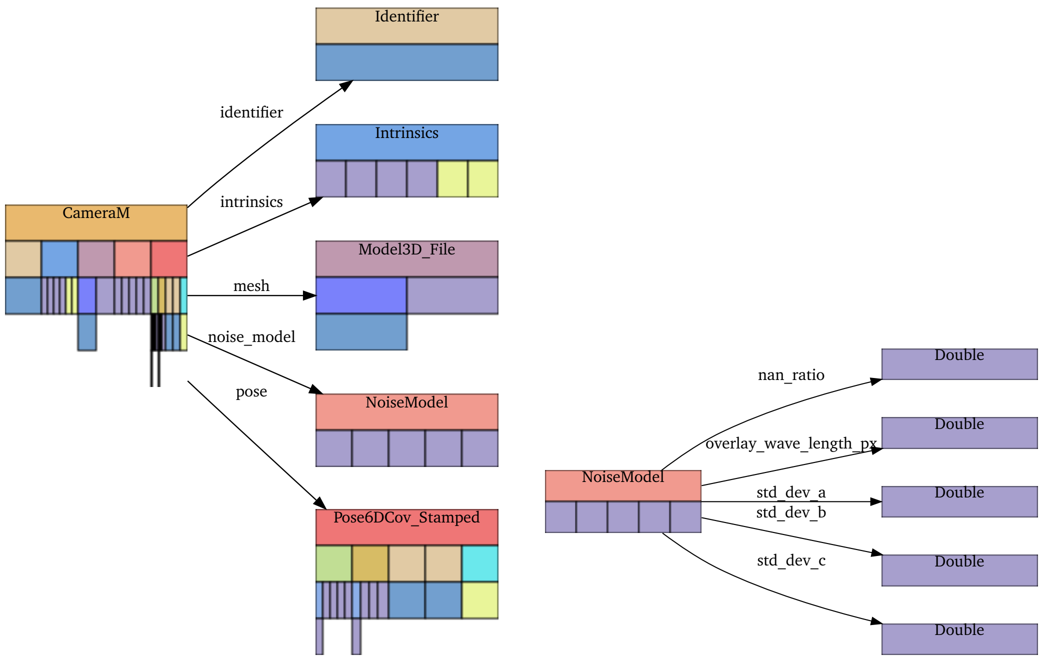


Figure 3.12: *Hierarchy of object models on different levels of abstraction. A camera for instance is a specialized type of object and has additional attributes.*

3.2.3 Camera Model

The camera model is based on the mathematical model as introduced in Sec. 2.2.1.1 and shown in Fig. 3.13a. In addition to the object it contains attributes for intrinsics and a noise model. Different abstractions and representations of intrinsic or noise models are available and can be used to model different camera variants.



(a) *Camera Model* ***(b)*** *Camera noise model*

Figure 3.13: *Models for a camera and the noise of the camera. Here the parameters std_dev_a, std_dev_b, and std_dev_c describe a quadratic approximation of the dependency of the standard deviation on the distance as introduced in Sec. 2.2.1.4.*

The depth camera noise model is shown in Fig. 3.13b with different parameters for different kinds of noise types. The values *std_dev_a*, *std_dev_b*, and *std_dev_c* for instance represent a quadratic model of the axial standard deviation depending on the distance.

Furthermore, procedural knowledge is defined to act on the camera model. For instance, in Fig. 3.14 an operator model is visualized for the acquisition of a depth image from a camera.

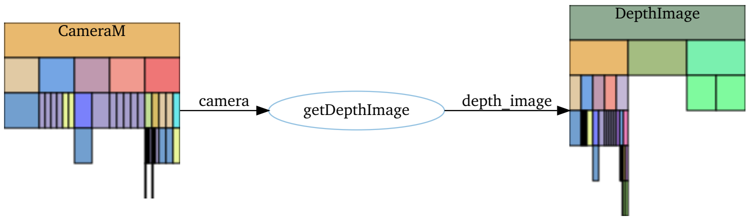
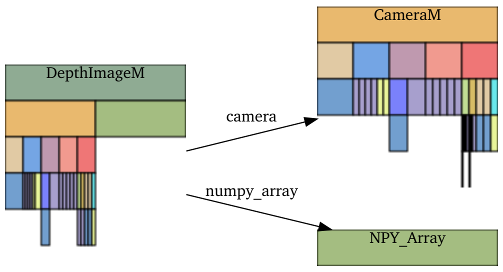


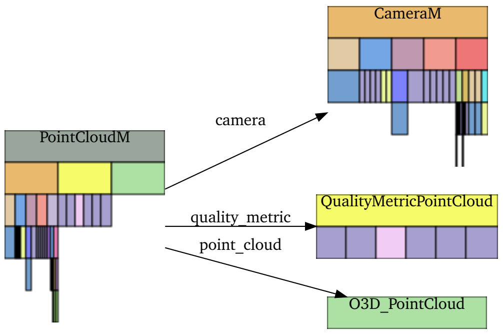
Figure 3.14: *Procedural model for the acquisition of a depth image from a camera. This operator applies to a real camera and internally uses the communication interface of the camera. There also exists a simulated variant, where a depth image is rendered based on camera model and scene, as shown in Fig. ??.*

3.2.4 Sensor Data Models

Depending on the sensor and data-processing, different data types and models can be defined. In the context of industrial pose estimation the most important data types are point clouds, RGB images and depth images. As example, we show the models for a depth image and a point cloud in the following Fig. 3.15a and Fig. 3.15b respectively. For theses models different design decisions have to be noted. The



(a) *Depth image model*



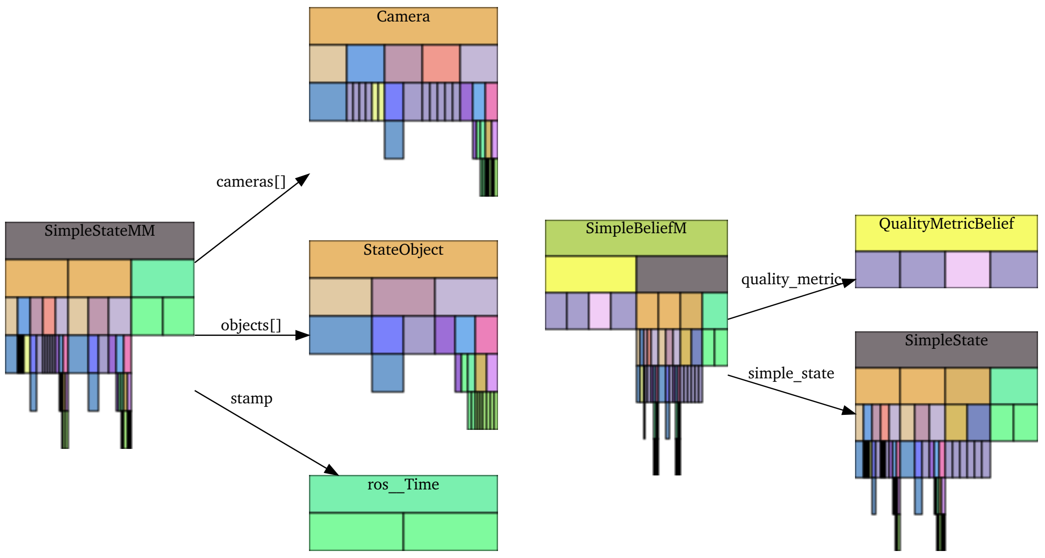
(b) *Point cloud model*

Figure 3.15: *Models for the common sensor data types depth image and point cloud. Both contain a model of the originating camera and the actual sensor data.*

model of the originating camera is added to the concept. Alternatives are to model solely the identifier of the camera and access the camera properties via a database or to leave out the camera at all. Both can be implemented with the given modeling formalism. Furthermore, in order to represent low-level data, it is possible to add new library specific base types such as a Numpy array (Walt et al., 2011) or an open3d point cloud (Zhou et al., 2018). This allows to use data types which allow high performance for certain operations. It is of course possible to model a point cloud as a list of points with coordinates. But with a custom model every operator has written from scratch or significant overhead for type conversion to and from the target library is introduced. For that reason, the modeling environment supports the reuse of existing data types of external libraries, such as Numpy. This easy integration of custom and specialized types in a modeling environment allows for higher performance and reuse of existing operators. Also note here, that in case no pre-computed point cloud is provided by the sensor additional operators are required to perform for instance stereo-matching in order to produce a point cloud. This is fully compatible with the presented models and modeling approach.

3.2.5 State and Belief Model

In the robotics domain world state and belief have to be represented. The actual state is the ground truth state, which is only known exactly for simulations or with errors for annotated scenes. The belief denotes the state representation which is available at runtime within the control system of an autonomous systems.



(a) *State model* **(b)** *Belief model*

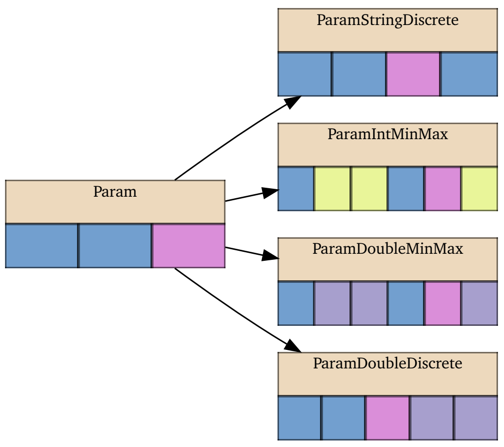
Figure 3.16: *Models for state and belief. For the belief representation, the state model is reused. Furthermore a metric is added the assess and estimate the quality of the belief.*

In Fig. 3.16a a state model is visualized which is suited for perception tasks and covers cameras and objects in a scene. Here the implicit assumption is that poses are given with respect to a common world frame. If this cannot be assured in the task context, additionally a transform tree or graph can be added to the state model.

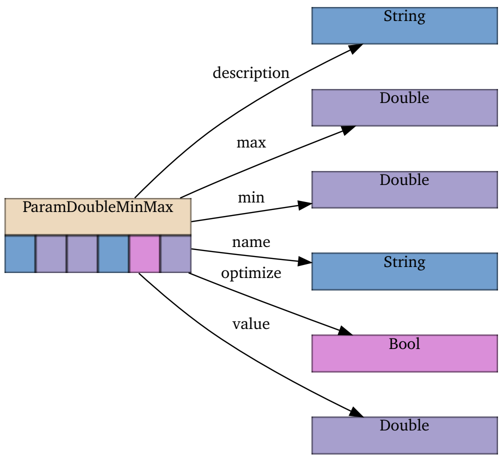
The corresponding belief model is shown in Fig. 3.16b. Here, the state model is reused, which does not include uncertainty representations. If needed, particle representation of states or uncertainty representations for each object can be modeled and added.

3.2.6 Parameter Model

Parameters play an essential role for the configuration of perception operators. In Fig. 3.17a an excerpt of the parameter model hierarchy is shown, of which the parameter *ParamDoubleMinMax* is visualized in more detail in Fig. 3.17b. Common properties are the name, the description and the optimize flag. The latter allows to keep a parameter fixed during optimization.



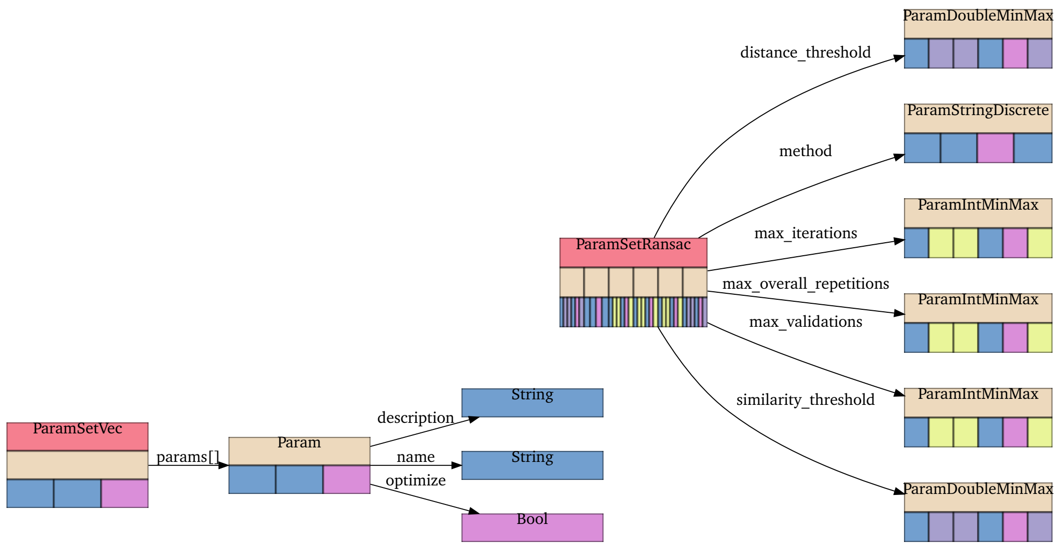
(a) *Model of parameter hierarchy*



(b) *Continuous parameter model*

Figure 3.17: *Parameter model hierarchy and example. The individual parameters contain range descriptions, such as minimum and maximum values, such that the configuration space can be restricted.*

Parameters can be bundled to parameter sets in different ways. A generic way is shown in Fig. 3.18a, where a parameter set is specified as a list of individual parameters. This allows iteration over the parameters in the set and is well suited, when generic operations are to be performed. Additionally, parameter sets can be modeled as named attributes as shown in Fig. 3.18b for the parameter model of a RANSAC operator (Fischler and Bolles, 1981). This approach is well suited for model the specific parameter sets for operators. Both modeling approaches coexist and can be converted into each other depending on the need.



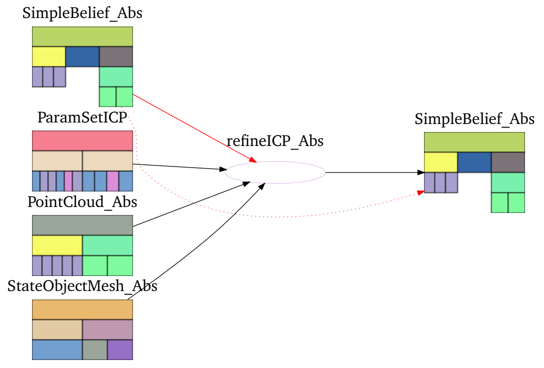
(a) *Parameter set model* ***(b)*** *RANSAC parameter model*

Figure 3.18: *Generic and specialized parameter set models. Specialized parameter sets can be converted to the generic representation which allows more generic definition of operators acting on the parameter set.*

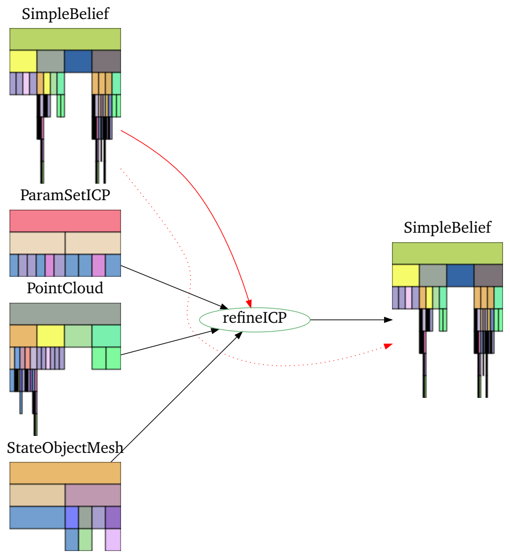
The weights of neural networks are similarly modeled. Due to the size and amount of weights for typical computer vision the storage as a file on disk is common practice. Therefore, the weight location is modeled as a path parameter.

3.2.7 Perception Algorithms

Perception algorithms are modeled as operators based on the introduced concepts on different abstraction levels. In Fig. 3.19a and Fig. 3.19b two different abstractions of the *refineICP* operator are displayed. It is clearly visible how, the input and output concepts of the more abstract operator *refineICP_abs* are less detailed than the concepts of the operator *refineICP*. The more detailed operator works on an instantiated point cloud and can therefore be used on the lowest level for a real world execution. The abstract operator only acts on an abstract model of a point cloud.



(a) Abstract ICP operator model



(b) Refine ICP operator model

Figure 3.19: *Exemplary models for the ICP operator at different levels of abstraction. The abstract operator does not receive the actual points as input, but rather works on an abstract representation, where the instance quality is approximated via metrics.*

The operator *refineICP* is furthermore modeled on a relatively high level if compared to raw perception operators as given in perception libraries. It takes a belief as input and output, which are custom concept and not available in a typical perception library. Internally it is a pipeline of operators, which are modeled at an even more elementary level as shown in Fig. 3.20. Here, elementary data types, such as point clouds from the open3d library are used. So a composition hierarchy as well as an abstraction hierarchy can be represented.

An abstraction can be modeled for any kind of operator as shown in the excerpt of the operator hierarchy as shown in Fig. 3.21. Here single abstraction layer is added above the operators which act on lowest level data. But in general, the modeling

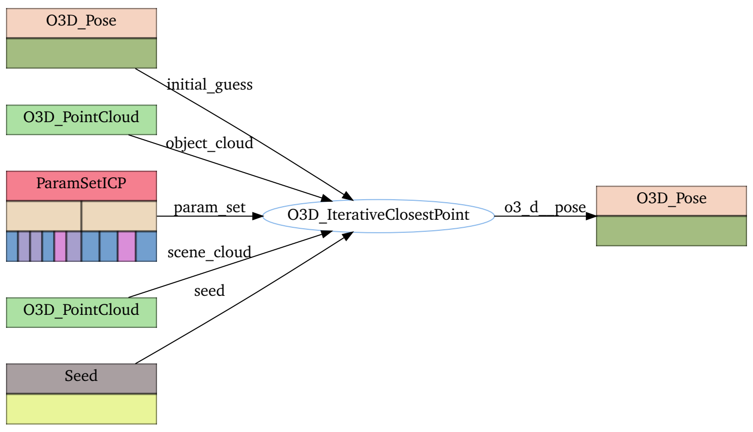


Figure 3.20: *Open3d ICP operator model, which is used internally within the refineICP operator*

approach allows an arbitrary number of abstraction layers. The choice and modeling of abstraction layers is so far a manual approach and in the responsibility of the perception system engineer. But the possibility to generate models using an API is given, which allows the design of meta operators which learn and automatically generate abstract model layers.

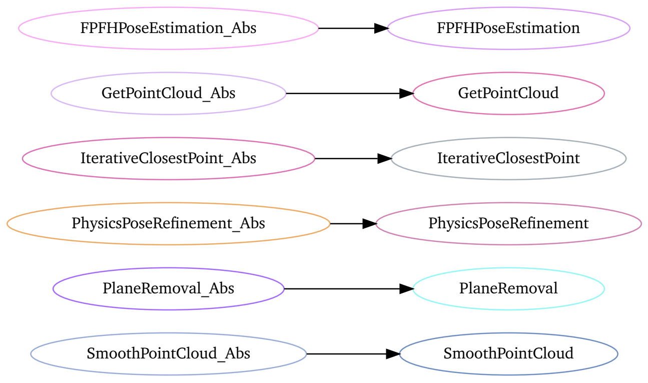


Figure 3.21: *Operator hierarchy between abstract operator approximations and the actual operator implementations.*

3.2.8 Datasets

Finally, the model of datasets is introduced, as it plays a central role in the design synthesis of perception systems. We structure datasets with so-called *SceneInstanceSets*, in accordance with Sec. 2.2.2.4 and based on (Dietrich et al., 2019). Each instance set contains data which describes a static object and sensor setup. As instance types we distinguish input instances, ground truth instances and expected instances. Input instances include different sensor data inputs, such as images and point clouds and prior belief knowledge. Ground truth instances describe the scene state in different (intermediate) representations such as object poses, masks, and bounding boxes. Expected output instances represent the information that is required by the application, such as a list of objects and their poses. The expected output instances are typically a subset of the ground truth instances.

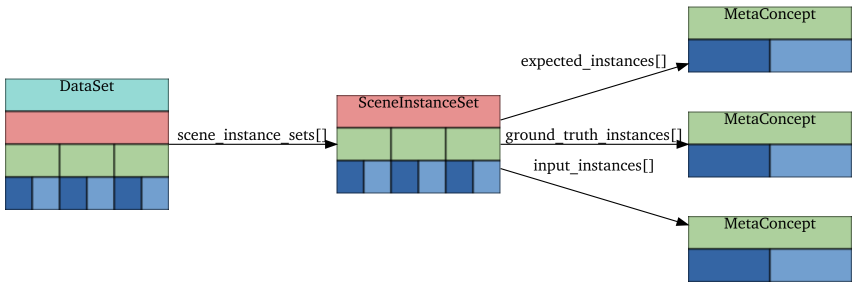


Figure 3.22: *Dataset model, which consists of a list of SceneInstanceSets.*

The dataset model is generic and therefore allows to cover a wide range of perception tasks. For an exemplary image classification task, the input instances contain an image and the ground truth and expected instances contain a class label or object instance. Pose estimation tasks can be modeled accordingly. The differentiation between ground truth instances and expected instances allows to encode the task in a dataset. In a pose estimation task an instances of objects with poses are given within the expected instances. And the ground truth instances additionally contain the object masks in the image. This allows to train an instance segmentation network as prior to a subsequent pose estimation operator instead of enforcing a end to end approach.

Furthermore, datasets need to be generated, which requires additional operators such as simulators or interfaces to data annotation services. These are equally modeled within the presented formalism.

3.3 Concept and Operator Acquisition

A core question is how operators and concepts are added to the system. The current speed of technological development leads to an strong growth of the number of available perception operators, as can be seen in the growth numbers of publications and open source repositories. A core requirement is to keep the engineering effort low, while working with heterogeneous software environments.

One approach is to manually model and integrate operators and concepts from different origin. While this approach provides flexibility, it does not cost effectively scale when significant amounts of manual labor are involved. Therefore, we introduce two different examples where existing software environments, notably software libraries and middlewares, are automatically processed and used for the automatic generation of executable declarative and procedural models.

3.3.1 Library Parsing

This section is based on the joint work of Galanis et al. (2020). As noted before, different perception libraries such as HALCON, PCL and OpenCV are used especially for pose estimation in industrial tasks (Hagelskjær et al., 2018). In order to generate executable models of such libraries, implementation details have to be taken into account. Furthermore, a large knowledge source for generating usable models lies in the documentation of the individual operators. In the following, we present a model extraction system (MES) which addresses the automatic generation of executable models via the interpretation of documentation and automated code generation. An overview is given in Fig. 3.23, based on (Galanis et al., 2020).

The model extraction system is designed to be modular and expandable. This is achieved by using the abstract semantic graph (ASG) of the targeted library as knowledge base throughout the analysis. The ASG is extracted from the target library using static source code analysis and represents all the code components that are part of the source as nodes and the semantic relationships between the nodes as edges. The system is designed in such a way that all modules have access to the current knowledge in the form of the ASG. Any information that is extracted by the modules, like the extracted semantic value for each parameter or whether a parameter is an input or output, is then added directly to the correct nodes in the ASG themselves. Figure 3.23 illustrates this design and the implemented modules. In the following, the three different phases are outlined in more detail.

Preprocessing In the preprocessing phase, the abstract semantic graph of the targeted library is built and filtered. The ASG generation relies on the header files only and thus allows to parse libraries which are distributed in binary format. The filtering

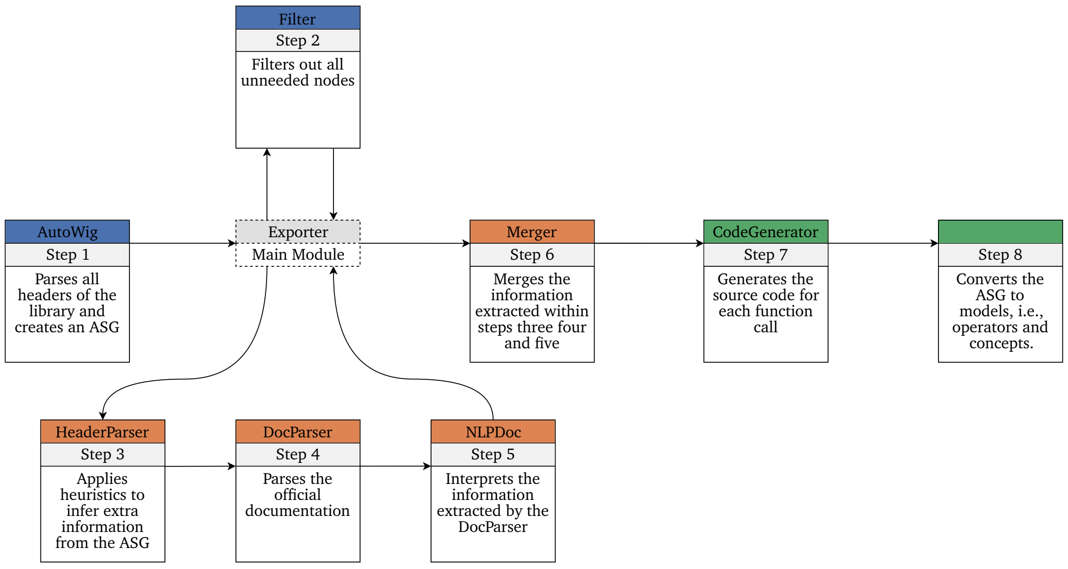

Filter
Step 2
Filters out all unneeded nodes
AutoWig
Step 1
Parses all headers of the library and creates an ASG
Exporter
Main Module
Merger
Step 6
Merges the information extracted within steps three four and five
CodeGenerator
Step 7
Generates the source code for each function call
Step 8
Converts the ASG to models, i.e., operators and concepts.
HeaderParser
Step 3
Applies heuristics to infer extra information from the ASG
DocParser
Step 4
Parses the official documentation
NLPDoc
Step 5
Interprets the information extracted by the DocParser

Figure 3.23: *Overview of the different modules used in the model parsing. Every block corresponds to a module of the harvester. The structure of this system can be divided into three phases. Preprocessing (blue), annotation (orange) and export (green). Each phase is divided into several modules.*

step reduces the size of the ASG and removes entries belonging to dependencies of the library, as those are not user-facing code that requires generated models.

Annotation In the annotation phase, background knowledge is extracted and annotated to the ASG. This is an important aspect of the model harvesting, as the code itself does not guarantee, that it contains enough information for the model generation. For instance, the ASG does not specify whether a parameter is an input or output of a function, especially if passed as reference. Furthermore, a language related fundamental type such as *string* can have higher level semantic meaning such as being a file path. This information is desired on the modeling level. Therefore, the annotation phase allows to annotate the ASG using different methods. First of all, heuristics can be applied, which can be language and library specific. For instance, a *const* qualifier for a parameter in C++ can denote an input. Subsequently, the documentation is parsed in order to be interpreted using natural language processing. The parsing depends on the documentation formalism and representation and can be customized. Natural language processing then provides the means to extract information, which is provided in a human-readable format within the function description. In particular, we analyze the parameter comments, as they contain the information about a parameter's mode (input or output) as well as its semantic content. To extract this knowledge, a pre-trained neural network that is based on the transformer architecture, is used. This network is then fine-tuned on two tasks:

1. Input/Output classification and
2. Semantic type classification.

The trained network is then used to classify the parameters and the resulting labels are added to the corresponding node in the ASG.

The final step of the annotation phase merges possibly contradictory annotations. Consider the information whether a function parameter is an input or output. During the analysis in the MES, several modules like the *HeaderParser*, or the *NLPDoc* module might classify the parameter into input and output. In case these information sources contradict each other, a way to fuse both information sources is needed. To accomplish this, the *Merger* module uses a weighted majority vote to determine the final result for each of the two classification tasks given the results of several modules. This implementation was chosen to be flexible enough to allow for additional modules that might be added in the future.

Export Once the needed information is extracted and merged, the MES generates the necessary glue code. This code is responsible for calling the underlying function using the inputs provided by the input concepts of the operator as well as passing the outputs of the function to the output concepts. By using automatically generated bindings, the underlying C++ library can be called directly within Python, which simplifies the code generation process. Finally, the concept and operator models are generated, which can then be reused within the automated perception system synthesis.

The general feasibility of the approach has been demonstrated in Galanis et al. (2020), where models not only are harvested, but also successfully used for the automated planning of a perception task. However, the natural language processing is still far away from reaching human level understanding of code and its documentation. Nonetheless, approaches like the presented have large potential to reduce the general engineering effort like writing boilerplate code in the future.

3.3.2 Middleware Parsing

A middleware is a “software that acts as a bridge between an operating system or database and applications, especially on a network”, Lexico (2021). It mediates between different pieces of software or applications and allows communication and interaction. In the robotics domain, the Robot Operating System (ROS), is a commonly used middleware, which provides standardized interfaces for communication and service provisioning (Quigley et al., 2009). Furthermore an entire ecosystem with a large number of tools is available based on ROS. As it addresses robotics tasks such as motion planning and perception it is a valuable source of concepts and operators. Therefore, ROS serves as a middleware example for which the automated generation of concepts and operators is demonstrated in the following.

Within ROS a compositional hierarchy of messages is defined for the exchange of information between nodes using an implementation independent interface definition language (Quigley et al., 2009). Language specific code generation provides language specific instantiations and serialization. The message definition is used for the different communications patterns via topics, services and actions.

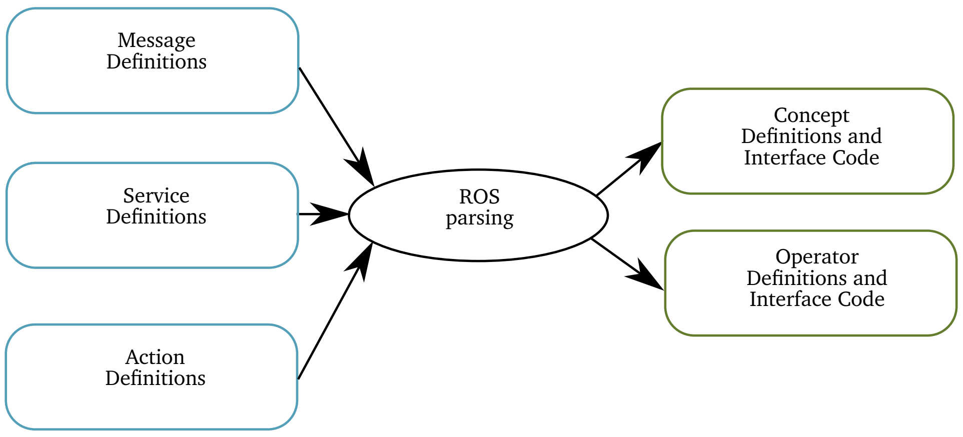

Message Definitions
Service Definitions
Action Definitions
ROS parsing
Concept Definitions and Interface Code
Operator Definitions and Interface Code

Figure 3.24: *High level view on the ROS parsing. Message, service and action definitions are interpreted in order to generate concepts and operators.*

Our approach for automated model generation is visualized in Fig. 3.24. It accesses the message and service definitions via the ROS API for a specific package and can generate concepts for messages and operators for services and actions. Therefore, it builds upon the modeling API as introduced before. A basic requirement for this procedure is to have common primitive types, such as string and float, which are represented in the std_msgs packages within the ROS message type system. A manual mapping between the ROS primitive types and primitive types within the modeling systems is performed within the automated model generation.

An exemplary auto-generated declarative model for a stamped transform is given in Fig. 3.25. The hierarchical composition of different concepts of different types is clearly visible. The primitive type std_msgs__String is mapped internally to the base type String.

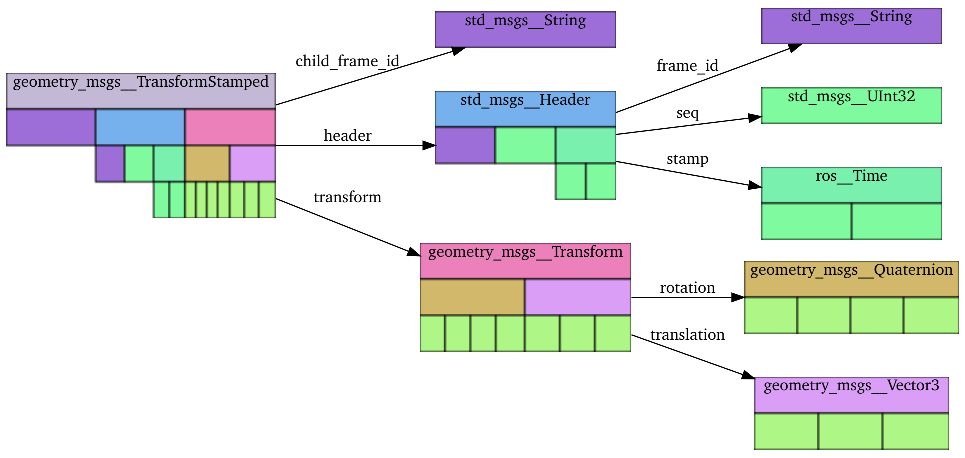


Figure 3.25: *Auto-generated model of ROS geometry_msgs\TransformStamped.*

In summary, it is equally possible to auto-generate large parts of the model infrastructure for middlewares to avoid re-engineering and allow heterogeneous systems. It is furthermore generally feasible to also auto-generate conversion operators between model from different origin with the same purpose, e.g., of modeling a pose. Although this meta-modeling capability is not yet implemented. The main distinction of the general modeling formalism with respect to the message definitions of a middleware with focus on communication and deployment are the inherent hierarchical abstractions. They allow the planning and execution of operators on different levels of granularity and thus enable efficient design synthesis. Furthermore, it is to note, that the interface generation for other perception related middlewares and communications standards such as GEN<i>CAM, (EMVA, 2021), is conceptually feasible, but associated with an initial engineering effort. It provides benefit if different users define individual messages.

3.4 Models for Configuration Space Exploration

In this section a high-level modeling view on different configuration space exploration approaches is provided. The approaches are applied in the following chapters to different task setups.

First, we define the term configuration space exploration in the following.

Definition 12 - Configuration Space Exploration

Configuration space exploration is part of the overall optimization process. It denotes the *efficient* search of suitable points in the configuration space with respect to their performance for a given task.

For perception system synthesis the computational demand for the evaluation of a single configuration is very high, due to the dimensionality of data, uncertainty and state variability. The latter means, that, e.g., the objects in a scene can be arranged in often arbitrary ways. Therefore, large amounts of test data are required to ensure performance for the possible object arrangements. Overall, computational efficiency within the exploration is paramount.

There exist different approaches to reduce the computational effort. If the overall performance model can be differentiated with respect to the configuration, gradients are available to steer the search process. This is done for the training of neural networks or differentiable algorithm networks (Karkus et al., 2019). In general, the evaluation of a perception system configuration is not differentiable. Although sub-problems in the overall problem hierarchy can be differentiable. This also motivates the possible use of different exploration methods at different layers of the problem hierarchy.

For non-differentiable problems black-box optimization methods can be applied, for instance using a surrogate model. A surrogate model is an approximation model of the performance of a process, which is either difficult or (computationally) expensive to measure or calculate. In the perception system synthesis context, it allows to predict the performance of a configuration with less computational effort. But it has to be taken into account, as with any model, that only an approximation is given, whose error depends on the type of model, the calibration and the modeled system. In the one dimensional case, curve fitting is equivalent to building a surrogate model.

Multiple steps are involved when working with surrogate models. First, the model has to be chosen according to the task. Exemplary models are random forests, support vector machines and artificial neural networks. Furthermore, the model parameters have to be calibrated based on given data point from more accurate models or real systems. Here the approximation error clearly depends on the amount of data available as well as the coverage of the configuration space. Therefore, often iterative schemes are used, where the costly evaluation of the detailed model is sequentially alternated with model calibration (Hutter et al., 2011).

In the following we introduce the models for different exploration approaches for the online and the offline phase.

3.4.1 Black-Box Optimization

In order to integrate black-box optimization generically into the modeling framework, interfaces have to be defined. A typical interface for black-box optimization operators consists of the function to be minimized, which receives a parameter vector as input and returns the cost as a scalar value. Overall, three major steps are required for the integration:

1. The configuration space has to be represented as a parameter vector with specified bounds for the different parameters as required by the optimizer.
2. During optimization, an instantiated parameter vector needs to be converted back to an executable system or pipeline instantiation, which can be evaluated for a specific set of inputs.
3. The cost of the configuration has to be calculated via a given cost function, which may depend on the application at hand.

The last two steps are integrated within the target function to be minimized.

In Fig. 3.26 the meta operator for optimization as modeled is visualized. It takes as input an instance of the *ConfigurationSpace* model as well as a *ParameterSet* which allows to set optimization parameter, such as the maximum number of iterations. Additionally, operators have to be specified for the conversion between the configurations space models and the parameter specification of the optimizer. The target function needs to be given as well. A template is given in Alg. 3.1. The parameter specification from the optimizer is converted back to a configuration, which is executed for each item of the data set. Then, the configuration is executed for each data point of the dataset. Finally, the cost metric is computed for the overall output and returned.

Algorithm 3.1 TargetFunction

```
function TARGETFUNCTION(ParamVector)
    Config ← convertToConfig(ParamVector)
    OutputList ← []
    for DataPoint in DataSet do
        Output ← executeConfig(Config, DataPoint)
        OutputList.append(Output)
    end for
    Cost ← calculateCosts(OutputList)
    return Cost
end function
```

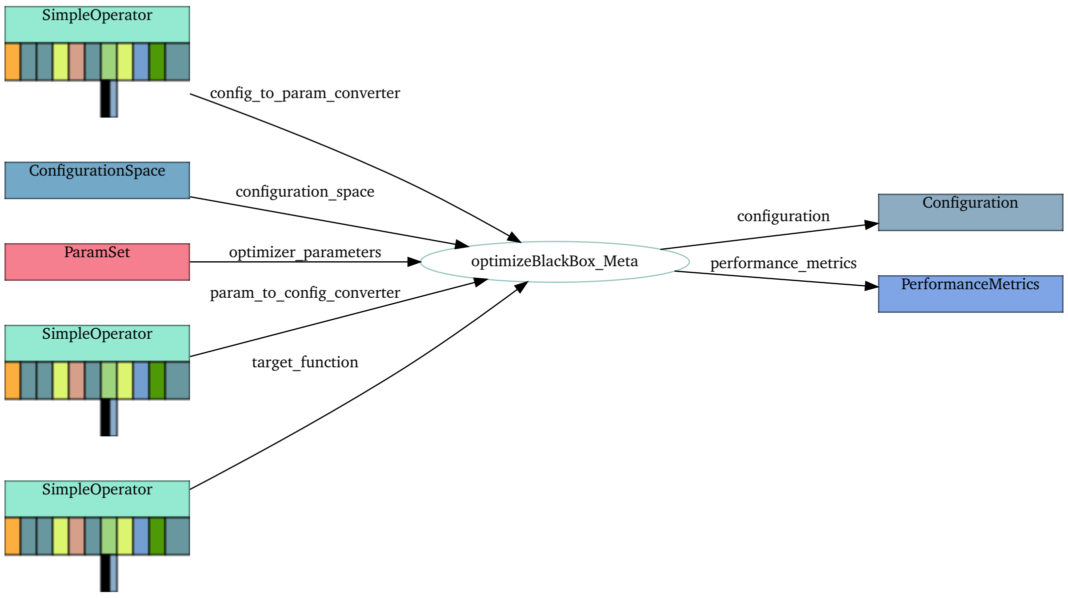


Figure 3.26: *Black-box optimization model, which takes as input the configuration space and a target function, which is executed during optimization. The output is a system configuration as well as associated performance metrics.*

Within the optimization procedure as shown in Alg. 3.2, the target function is passed to the parameterized optimizer. Conversions between the configuration space model and the parameter modeling approach as required by the optimizer are needed as well.

Algorithm 3.2 OptimizeMeta

```
function OPTIMIZEMETA(ConfigSpace, TargetFunction, OptParameters)
    ParamSpace ← convertToParam(ConfigSpace)
    ParamVector, Cost ← optimize(ParamSpace, TargetFunction, OptParameters)
    Config ← convertToConfig(ParamVector)
    return Config, Cost
end function
```

On this level of modeling, the optimizer is a black box itself. Depending on the optimization approach a surrogate model is used, as for instance in sequential model based algorithm configuration (Hutter et al., 2011). The surrogate model can be represented with the presented modeling formalism as well. One approach is shown in the following section about hierarchical planning.

3.4.2 Planning

An alternative way to explore the configuration space is planning, which can be employed for offline or online phases of the synthesis process. Planning algorithms are commonly used for sequencing robot actions (Kast et al., 2019c) and for the generation of feasible motions and trajectories for robotic actuators (LaValle, 2006). But planning can be applied as well for a very broad set of tasks, such as solving a rubic's cube and optimizing the logistics of a warehouse.

A planning problem can be defined via a model of state, actions and a goal. Planning algorithms are used in order to determine suitable sequences of actions in order to satisfy the goal. Actions act on a state and lead to a new state. Continuous and discrete action and state spaces can be targeted, depending on the task and the abstraction level of the model. So for instance, a task can be described on a solely symbolic level with discrete state descriptions, such as stating that an object is localized. Here, no information is given about the quality of the localization and the actual pose and its reference frame, which are needed for real world interaction. But if the planning model is well designed for a task, it can still be used to reduce the overall search space. The suitability depends on the problem. If in a warehouse discrete storage locations for pallets are given and the storage and retrieval process are reliable and deterministic, they can be abstracted as discrete actions which only require the identifier of the storage location. So the abstract model requires real world equivalent actions, which satisfy the sub-symbolic, continuous nature of reality.

Common strategies for planning are forward chaining and backward chaining. In the former the search is started at the initial state until a goal is reached. In the latter case the search is started at the goal and actions are explored backwards from the goal until the start state is reached. Furthermore, the search graph can be build in a breath-first or a depth-first manner. For a detailed introduction and formalization we refer to (LaValle, 2006).

Planning can be considered as a meta-operator in the presented model hierarchy. Exemplary, the procedural model for a breath-first forward chaining planner is visualized in Fig. 3.27. It takes a planning box as input, which describes the planning problem via facts, goals and available operators and is detailed in Fig. 3.28. Furthermore, the maximum number of plans can be specified. The output is a list of plans and the planner state, or, if applicable an exception. The model is kept generic, and allows to handle different planning domains, as given by the fact, goal and operator instances.

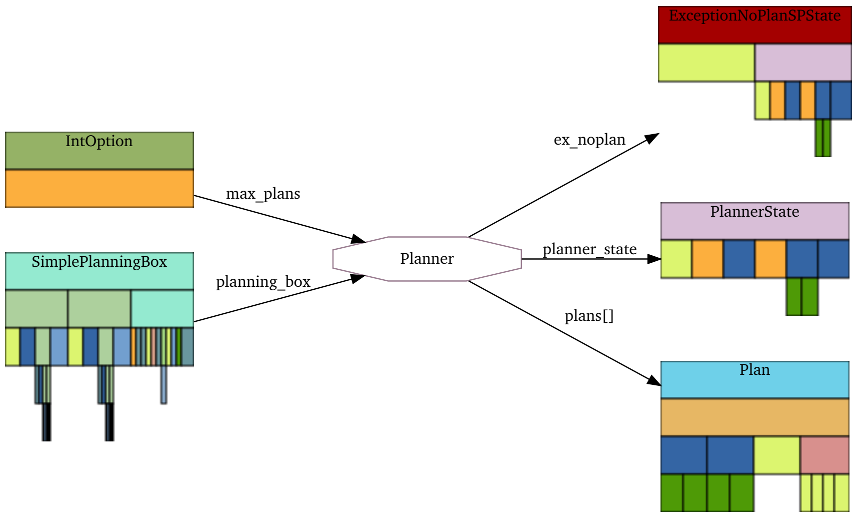


Figure 3.27: *Planner model, which acts on a planning box, where the planning problem is described. The output is a planner state and a set of plans or, in case of failure, an exception.*

Modeling meta-operators such as planners and optimizers provides the ability to construct a hierarchy of problems which can be solved with different methods at different layers. This is done with the hierarchical planning procedure as presented in the following.

3.4.3 Hierarchical Planning

As discussed before, a planning domain is generally an abstract model of a real world process and therefore is subject to modeling and abstraction errors. There are multiple ways to address this issue. Either, highly detailed and accurate models can be used within the planning domain, which increases the computational effort associated with the planning. The reason is that the modeled actions in the planning domain are executed during planning and detailed models require higher computational effort.

Another approach is to use backtracking, where a plan is executed in the real world, until a mismatch between the modeled state and the expected state occurs. In this situation, the planner is triggered again and can provide an alternative solution. Although, this is not possible for arbitrary modeling errors in the planning domain and it can lead to non-recoverable situations, such as unreachable objects in a scene.

The computational effort involved in planning can be reduced by exploiting the model abstraction hierarchy in the procedural and declarative knowledge as introduced

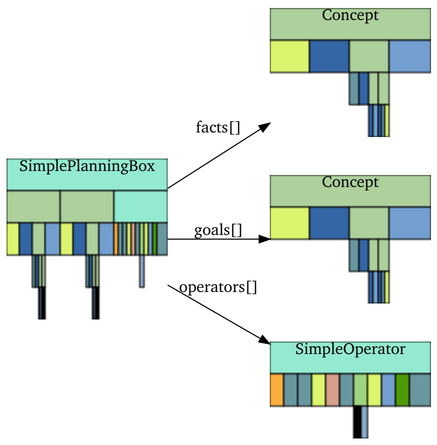


Figure 3.28: *Simple planning box, which describes a planning problem via facts, goals and given operators.*

in Sec. 3.1. In the following we will introduce the hierarchical planning approach based on the joint work (Kast et al., 2020). The basic idea of is to divide the overall planning problem into small subproblems. This process is performed repeatedly and allows to face the curse of dimensionality by formulating subproblems which do not address the entire problem dimension at once. This is done by planning in an abstract domain first. In this abstract domain the goal can be reached with a relatively small number of steps, as the abstracted operators cover huge changes of the state. For instance, a state change can be modeled on a purely symbolic level on this domain. Additionally, this reduces the branching factor, as only a small number of operators exist and can be applied in the abstract domain.

Once a solution on the coarse level is found, each applied operator in this plan by itself defines a new sub-planning problem with its inputs as starting values and outputs as goals. The creation of subproblems, also called refinement, is applied recursively to each of the newly generated planning problems until there is no further refinement for the operators. However, there can be errors and unsolvable subtasks on any level, as the abstracted domains and their planned solutions are only approximations of the real behavior. In our approach the downward refinement property (Bacchus and Yang, 1991) is not strictly enforced, which means that the successful execution of an abstract operator does not guarantee the successful execution of its more concrete operator counterparts. This property is hard to ensure in real world robotic systems and enforces strong constraints on the abstract models. Our solution to avoid dependency on the downward refinement property is backtracking. Here, if a refinement fails, plans on the abstract level are dismissed and new plan sequences to the goal on the abstract level are calculated. In our system, the real execution is the final refinement. Error handling requires no special care, it is a case of backtracking. Therefore, the planning approach represents a model predictive control scheme, where abstract domain representations are used for the prediction. Both execution and error handling are first class citizens within our hierarchical planning approach.

Under ideal conditions, when the coarse level is a good approximation to the behavior of the real-world, the planning approach can scale linearly with the length of the task, as shown in (Kast et al., 2020). This, however, holds only true if the downward refinement property is always guaranteed. For a bad approximation the number of visited nodes still grows exponentially as the full problem is np-hard.

Additionally, the solutions are not necessarily optimal. The solution quality depends on the model of the abstract domains to propose good intermediate goals for factorization. This, however, can be a burden to the overall system engineering, especially when different stake holders model the levels of the domain according to their respective user roles.

The hierarchy within the planning domain model can also be seen as a hierarchy of surrogate models, where each level is a surrogate model of the underlying level. Equally, its generation and parameterization can require a calibration procedure, where the model is calibrated to the reality it is to represent.

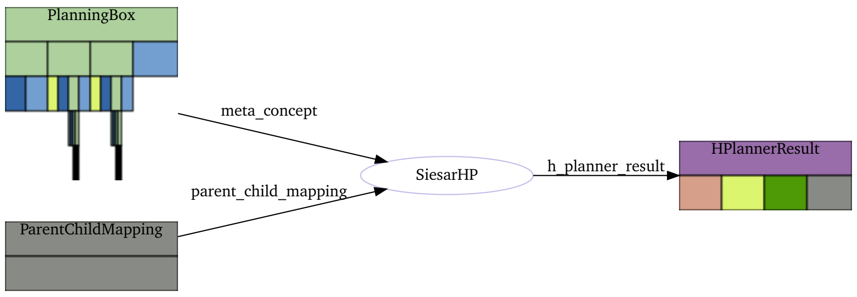


Figure 3.29: *Top level view on the hierarchical planner.*

The hierarchical planner equally has a model representation as operator in the meta-domain, as shown in Fig. 3.29. Internally it uses the executable model of the single level planner as introduced in Sec. 3.4.2 in order to solve sub-planning problems.

3.5 Related Work

The modeling approach addresses different lines of research, from general knowledge representation to specific modeling for automation and robotics. Knowledge representation is a subfield of artificial intelligence with the goal of designing representations of knowledge which allow to solve complex problems, typically using automated reasoning. Knowledge is therefore represented using specialized languages which are declarative models of the domain. A knowledge representation language is tightly coupled with an inference engine, which allows to derive conclusions and new facts from the declarative model, also called reasoning. The inference engine is therefore a procedural model which acts on the declarative model. According to Davis et al. (1993) five different roles can be identified for knowledge representation:

1. A knowledge representation is a surrogate
2. A knowledge representation is a set of ontological commitments
3. A knowledge representation is a fragmentary theory of intelligent reasoning
4. A knowledge representation is a medium for efficient computation
5. A knowledge representation is a medium of human expression

First-Order-Logic (FOL) is a common formalism being used for knowledge representation and reasoning. It can be used to model facts, objects and their relations and infer new instances thereof using logical inference. Although the expressive power is high, its use on the shop floor is limited so far. It is mostly adopted in research (Radig et al., 1992) and for expert systems (Metaxiotis et al., 2002). One reason for this is the fact that the average software developer is used to different programming paradigms, such as procedural and object oriented programming. Furthermore, the engineer brings in a lot of background knowledge when designing an automation system, which allows to reduce the programming tasks to problems, which can be represented well using the aforementioned paradigms. A crucial part of that background knowledge is a symbolic and sub-symbolic action-effect simulation, which an engineer can perform intuitively.

An important concept for knowledge representation and reasoning are ontologies, which on a high level are "an explicit specification of a conceptualization", Gruber et al. (1993), which is formal and can be shared (Studer et al., 1998). An ontology is typically represented in First-Order-Logic or in the Web Ontology Language (OWL) and is comprised of individuals, classes, functions, relations and axioms. In a robotics context it allows to state knowledge such as *a camera is a sensor* and *an automation system has a manipulator and a sensor*. But also more complex knowledge can be represented such as constraints between individuals and general rules.

Ontologies have been successfully applied to the robotics and automation application domain (Stenmark and Malec, 2013; Tenorth and Beetz, 2013; Diab et al., 2019). Especially Stenmark and Malec (2013) target industrial applications, while most other approach are focused on service robotics, where a larger variety of scenes, tasks, and objects to interact with is given. For a review of different approaches we refer to (Olivares-Alarcos et al., 2019). Ontologies provide especially declarative knowledge, which can be used and queried by the automation system at runtime. But recent developments target the integration of simulation to form a hybrid knowledge processing architecture (Beetz et al., 2018; Haidu et al., 2018). Here, parts of the robot control program are leveraged in order to gather experiential knowledge in simulation.

Standardization efforts are undertaken in order to specify common ontologies for the robotics and automation domain (Fiorini et al., 2017). However, an ontology alone is not sufficient to create the control system for a robotic system. Parameterizable and executable software components are required in addition, which are linked and connected with the ontology.

In the fields of industrial automation and systems engineering other specialized models and representations are available. For instance the automation modeling language AutomationML (Drath et al., 2008) is a standardized set of representations for topology, geometry, kinematics, and controller logic based on the Extensible Markup Language (XML) (Bray et al., 2000). It allows to declaratively model automation systems as a hierarchy of objects and their attributes and relations. Procedural knowledge can be represented declaratively using the PLCopen standard (van der Wal, 2009) and executed using specialized execution engines. However, programming techniques targeted for programmable logic controllers (PLC) do not fit well for programming the perception of advanced autonomous robots. Currently, the perception domain is not yet represented within AutomationML, but first approaches regarding sensor modeling exist (Gonçalves et al., 2019).

For systems engineering the Systems Modeling Language (SysML) (Hause et al., 2006) was designed based on the Unified Modeling Language (UML) (Rumbaugh et al., 2004). It can be used to model system requirements and exchange system information without disambiguation between different stakeholders. It is mainly a tool for system engineers and is not targeted to be used directly in the control system in the online phase. In the field of system modeling and simulation the Functional

Mock-up Interface (FMI) (Blochwitz et al., 2012) is defined. It allows to exchange executable simulation models between different tools and developers. However, it mainly targets the simulation of dynamic system.

Other approaches like OPC-UA (Leitner and Mahnke, 2006) thrive to standardize data formats and communication on the factory floor. While it is focused on the low-level data exchange, it provides data models and semantics, which can be leveraged for data analysis and automated configuration. There is a specialized OPC-UA standard for the machine vision domain (VDMA, 2021), which especially covers the recipe and configuration management. The modeling of so-called vision skills such as object detection are still future work (VDMA, 2021).

The aforementioned approaches from knowledge representation research and specialized modeling tools provide excellent tools for declarative modeling of different domains. But the modeling and representation of procedural knowledge, i.e., executable procedural models, are not in the general focus. The procedural knowledge is encoded in inference, or execution engines which act on the abstract models. It is not straight forward how to integrate existing code and libraries, which are the core procedural knowledge about perception operators. Therefore, integration effort is reduced, but automated design synthesis is only possible in a limited way. Only executable procedural models allow for a detailed evaluation and automated design synthesis down to the low level engineering decisions required for perception systems.

The presented hierarchical modeling approach does only provide a subset of the expressiveness, as for instance of OWL. This is due to the tree-based structures rather than graphs. However, the executable procedural models, which are tightly integrated with the declarative models allow to address robotic tasks without major tool boundaries. In general there is a trade-off between expressiveness and ease of use. It is desirable future work to ground the presented hierarchical modeling approach in a formal ontology and leverage synergies. However, the presented modeling approach allows to model hierarchies of abstraction levels which can be leveraged across different tasks ranging from manipulation planning to perception system synthesis. Furthermore meta-operators such as planning and optimization are first class citizens, which targets the requirements of offline and online engineering for the design and operation of autonomous systems.

One example of the integration of knowledge representation and inference for perception in automation and robotics is the RoboSherlock system (Beetz et al., 2015a), (Bálint-Benczédi et al., 2019). It builds on the Unstructured Information Management Architecture (UIMA) (Ferrucci and Lally, 2004). The perception system is modeled using an ontology in OWL within the KnowRob knowledge base (Tenorth and Beetz, 2013) and First-Order-Logic is used in order provide perception pipelines given input queries. Uncertainty in the object classification can be taken into account using markov-logic-networks (Nyga et al., 2014), which are trained on logged data. The approach is not targeted for systems engineering, but focuses on robot autonomy in unstructured environments. Furthermore, declarative and executable procedural perception models are separated into different systems and the handling of ontologies poses a challenge for industrial shop floor applications.

Different modeling languages exist in the context of planning, which relate to the presented hierarchical approach. The following planning literature review is based on the joint work (Kast et al., 2019c). There are many languages such as PDDL (McDermott et al., 1998) and its variants that focus on the procedural knowledge of a domain on the symbolic level. This language standardization combined with a long series of planning competitions have resulted in a large set of fast planners for PDDL domains such as (Helmert, 2006). Extensions to PDDL add support for certain sub-symbolic properties such as time. An example is PDDL+ (Fox and Long, 2002) that introduces events and processes to model exogenous change and support domains with mixed discrete and continuous dynamics. Corresponding solvers, such as (Cashmore et al., 2016) or (Piotrowski et al., 2016), make use of this representation and handle domains with nonlinear continuous change. They approximate the dynamics of the system and handle the resulting discretized model with uniform time steps and step functions.

Another approach is partial-order planning (POP) (Young et al., 1994) which involves partially specified action decompositions. In this approach, the planner is only allowed to fill in missing pieces of a fixed plan template. This hugely reduces the search space. However, it is difficult to ensure the separation of system and task description with this approach, which reduces the flexibility.

A large community addresses hierarchical task networks (HTN) that refine each abstract skill by a network of sub-methods, e.g. (Castillo et al., 2006), (Goldman, 2006), (Nau et al., 2003). In their basic form they were state-oriented and could not deal with time constraints or concurrent actions. Nevertheless, the formalism is more expressive than that of first principle planners (Erol et al., 1994), HTN methods can improve planning times (Nau et al., 2003) and even support plan reparation (Gateau et al., 2013). In (Bercher et al., 2016) an overview of HTN methods is provided that discusses the expressiveness of hierarchical planning formalisms as well as implications of preconditions and effects of abstract methods. A mandatory and limiting condition for HTNs is the downward refinement property that enforces refinements to all abstract solutions (Bacchus and Yang, 1994). Yet, the generation of suitable abstract models for a domain is a challenging or even impossible task and often results in a coupling between task and system description.

To overcome the limitations induced by the downward refinement property, several variants of HTN planning and hybrid planning (Kambhampati et al., 1998), (Schattenberg, 2009) have been proposed. The hierarchical partial-order planner, introduced in (Bechon et al., 2014), uses additional knowledge that describes sets of abstract actions with optional methods to increase flexibility during refinement. Another approach to improve versatility is the combination of HTN and POP in a domain-specific planner with a strict separation between several hierarchical levels (Castillo et al., 2003). In both approaches the effects of the downward refinement property are mitigated at the cost of models which are dependent on the system and the task at the same time.

In summary, the proposed modeling formalism stands out with respect to the state-of-the-art in several points. First, it allows to natively model abstraction hierarchies and declarative as well as procedural models. This allows efficient configurations space exploration via abstract layers for mixed discrete continuous problems such as the perception domain. Furthermore, the approach successfully combines techniques for code generation and model harvesting in order to reduce engineering effort. Finally, meta modeling allows to represent operators for configuration space exploration, which can be easily recombined in sequential and even hierarchical manner.

3.6 Summary and Discussion

In this chapter, the hierarchical modeling approach is introduced. It is based on set-theory and allows to represent procedural and declarative knowledge as a hierarchy of procedural and declarative models. The hierarchy is multi-dimensional and facilitates to represent a compositional hierarchy as well as an abstraction hierarchy. This allows to model common concepts in the domain of perception, robotics, and automation, as shown.

The benefits of the presented approach are of multiple origin. First, it allows a tight integration between declarative and procedural models, which is not the case for representations with strong focus on declarative models. Moreover, in its core it is a language agnostic approach, where different programming languages can be integrated and used interchangeably for executable procedural models. This reduces engineering effort and facilitates the use of specialized libraries written in different programming languages. Furthermore, model harvesting approaches allow to generate models by taking into account source code and documentation. This allows to reduce the manual modeling effort even further. Reducing manual modeling effort is an important aspect for model-driven design approaches, as there always exists the risk to trade medium implementation efforts with high modeling efforts.

The meta-modeling level facilitates the modeling of procedural knowledge for configuration space exploration. This enables the reuse of exploration approaches especially when targeting problem hierarchies which result from a problem decomposition. For instance, a sub-problem can be solved using a specialized optimizer, while the overall problem hierarchy is addressed via hierarchical planning.

The synthesis of perception systems benefits from a unified modeling approach for declarative and procedural knowledge. Declarative knowledge alone can hardly cover all uncertainty and noise effects, executable procedural knowledge and real operator execution is required to ground abstract models. Executable models furthermore allow the optimization of sub-symbolic system parameters, which occur in large numbers within perception systems. At the same time, an abstraction hierarchy of arbitrary levels can be modeled, which allows to take advantage of abstract models in order to steer the configuration space exploration and can allow to reduce configuration times, especially for runtime purposes. Finally, the automatic generation of models from a heterogeneous set of libraries allows to take advantage of fast paced innovation cycles and reduces general engineering effort.

Chapter 4

Offline Pipeline Synthesis using Structure Priors

After the introduction of the underlying models for perception and configuration space exploration, this chapter is dedicated to pipeline synthesis via optimization. We target the sub problem of generating and parameterizing an appropriate perception pipeline. This step is associated with the highest engineering effort according to an industry survey (Hagelskjær et al., 2018).

The generation and parameterization of perception pipelines has multiple configuration space dimension. First, individual operators have inputs and outputs and a set of operators can be combined to a pipeline. So the first part of the task is to choose operators from an operator library and connect their inputs and outputs such that a pipeline is formed. This pipeline has to be compatible with the task in that it uses provided inputs and produces the desired result. The second part is the selection of appropriate parameters for the employed operators. The output of an operator depends on the choice of parameters given. Therefore, the performance of a pipeline strongly depends on the chosen parameterization of its individual operators. Overall, these two configuration space dimensions already yield a large configuration space, even when a small number of operators is given.

In Fig. 4.1 a high level overview over the optimization procedure and how it embeds in the overall context is given. The approach targets the offline engineering phase, where the system engineer designs a static pipeline for a specific perception task. Among other, a specification of the criteria of the task as an error function is necessary, which is used within the optimization procedure, see Sec. 2.2.2.5. A specialized instantiation step converts the perception task into an optimization problem, while

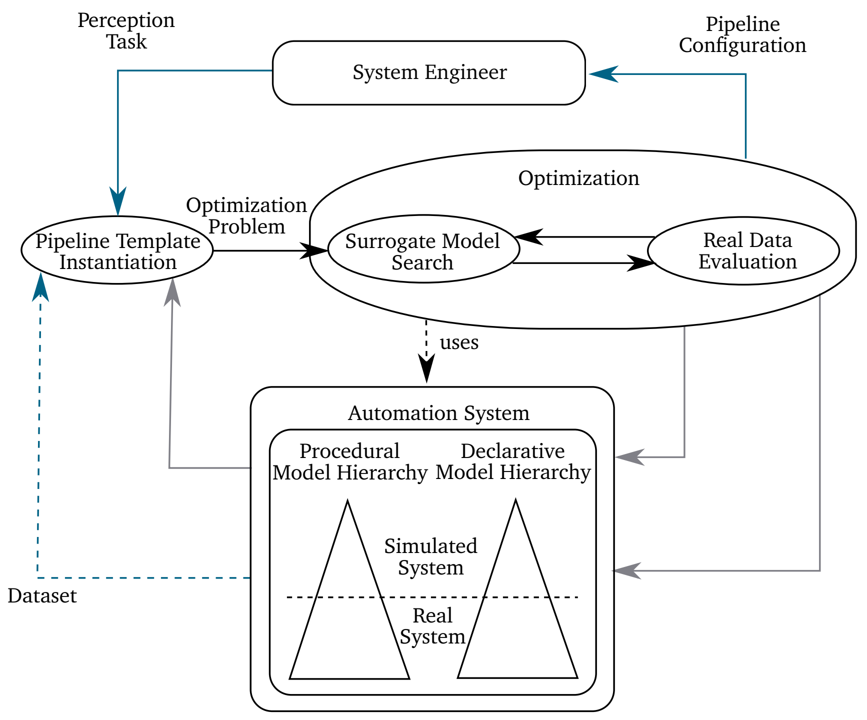


Figure 4.1: *Offline pipeline synthesis via black-box optimization.*

taking into account pipeline templates. Pipeline templates are introduced in Sec. 4.1.3 and employed to reduce the configuration space by restricting the pipeline structure. The optimization procedure uses a surrogate model to determine promising configurations, which are evaluated on real data. The synthesis and the automation system are represented using hierarchical models as introduced in Chapter 3.

This chapter is based on and reuses excerpts of the publication of Dietrich et al. (2019). In addition, a more detailed view on the top-level models for the synthesis is given.

4.1 Pipeline Structuring and Optimization

In this section, the approach and it's individual components are introduced. We start with an overview and proceed with the pipeline structuring and optimization methodology. The descriptions are accompanied with the respective models.

4.1.1 Overview

The approach is displayed with more detail in Fig. 4.2. As inputs, an annotated dataset of the application, an operator library as well as a pipeline structure template is given. The input data represents the application. It contains sensor inputs, which are annotated with ground truth information and expected instances by the application. The operator library is a set of available perception operators. The pipeline structure template enables to encode engineering knowledge and reduce the configuration space. It will be introduced in Sec. 4.1. The output of the automatic configuration is a parameterized perception pipeline.

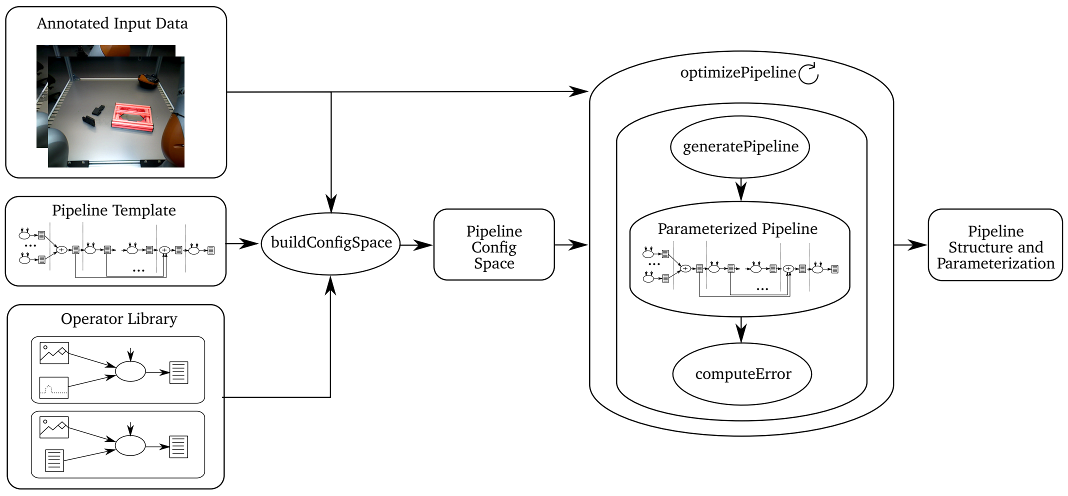


Figure 4.2: Overview of the automatic configuration approach. A pipeline structure template is used to reduce the search space. Operators from an operator library are automatically placed and parameterized within the template within an optimization procedure in order to fulfill the task as specified via an annotated data set.

The synthesis of pipeline structure and parameterization is formulated as the optimization problem:

$$(\mathrm{P}^*, \lambda^*) \in \underset{\mathrm{P}\in\mathbb{P},\lambda\in\Lambda_p}{\operatorname{argmin}} \ \mathrm{e}_\mathrm{D}(\mathrm{P}_\lambda, \mathbb{D}), \tag{4.1}$$

where Λ_P is the parameterization space of pipeline P. Typically, this can only be solved by derivative-free optimization methods due to missing derivative information for most operators and structure adaptations. In this general problem representation, the structural variety is encoded via $\mathbb{P}$. In the following section, we introduce structural elements and structure parameters to model and optimize the set of pipeline structures.

4.1.2 Pipeline Synthesis Meta-Operator

The pipeline synthesis is modeled as a meta-operator, which is shown in Fig. 4.3. This facilitates the reuse of the operator and enables to use it solver for subsets of a decomposed problem hierarchy. It uses the *PipelineStructureTemplate* and *DataSet* model as introduced before. Furthermore, it requires the available operator set, which is given as a list and an operator for the calculation of the error metric. The additional input of prior instances allows to provide background information and pre-trained models for available operators. In the latter experiments the neural networks, which are trained on the dataset as part of a prior step, are given as input for runtime reasons. But it is also conceptually feasible to include the training of, for instance, neural network in the overall optimization. The output of the synthesis is a parameterized perception pipeline together with performance metrics.

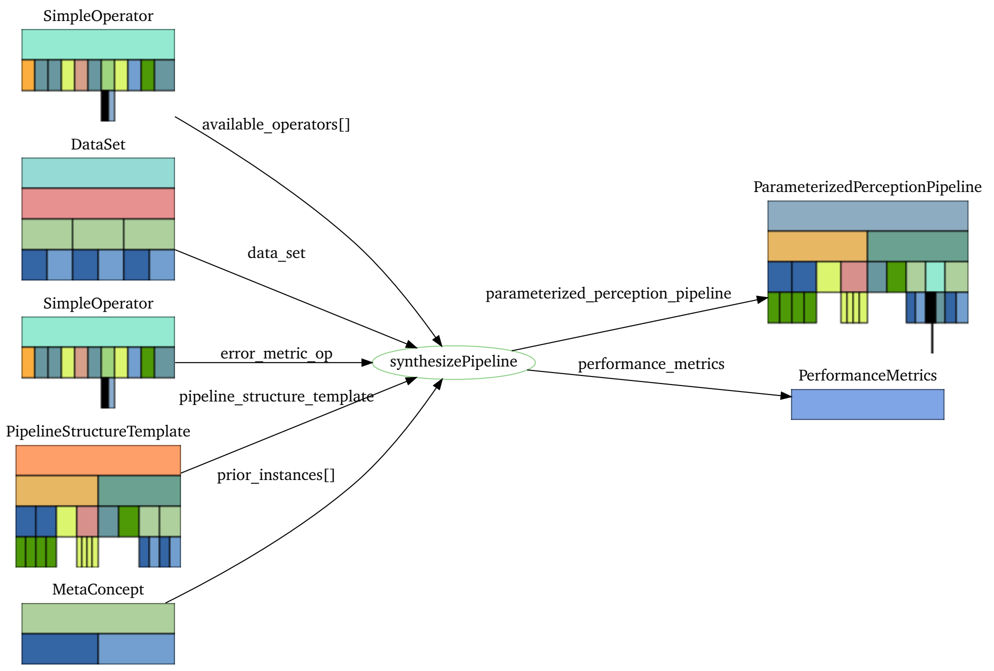


Figure 4.3: Pipeline synthesis operator model

The pipelines synthesis is itself composed of two major steps, the initialization of the configuration space model and a black-box optimization as shown in Fig. 4.4. The black box optimization reuses the meta-operator as introduced in Fig. 3.26. It implements the different strategies, which can be chosen via parameters.

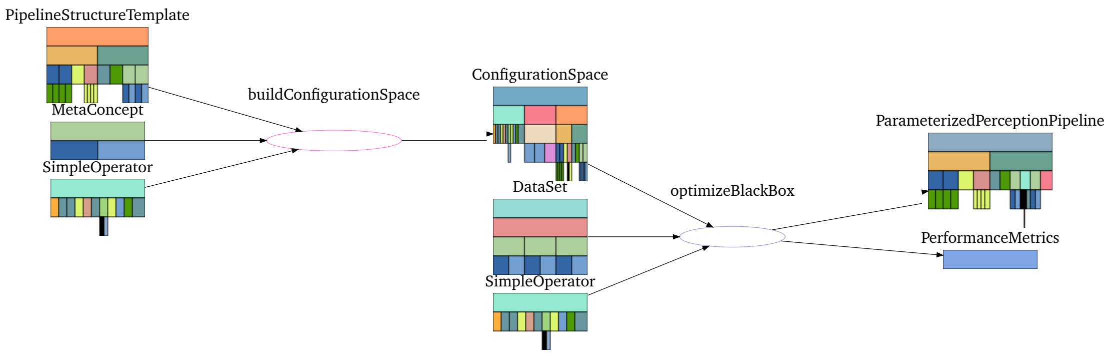


Figure 4.4: Pipeline synthesis operator internal structure

4.1.3 Pipeline Template Model

We employ a pipeline template model, where structural elements of the pipeline are parameterized via structure parameters $\tilde{\psi} \in \tilde{\Psi}$, $\tilde{\Psi} \subseteq \Lambda$ that encode the operator instances and their sequence of execution. The model of the pipeline structure template is visualized in Fig. 4.5. It consists of a list of structural elements as well as a connection graph, which models the interconnections of the structural elements.

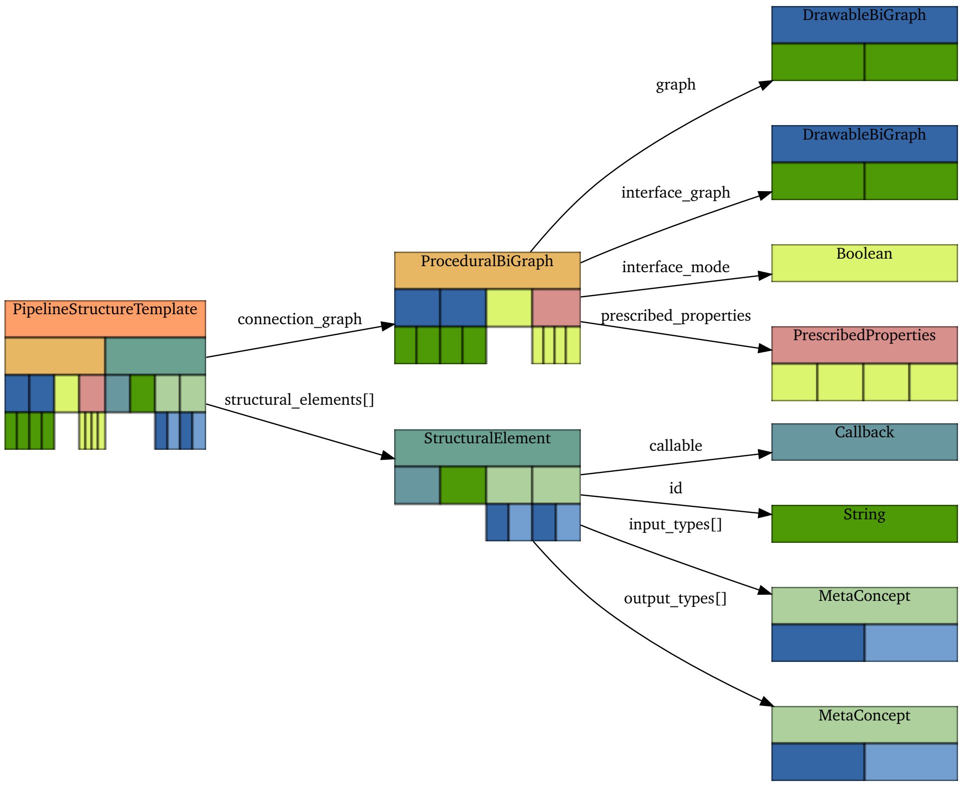


Figure 4.5: Pipeline structure template model

The template we employed as working example is displayed in Fig. 4.6. We structure the pipeline template in three main structural elements: hypothesis generation, hypothesis refinement, and hypothesis scoring. The structural elements are configured via the respective structure parameters $\tilde{\psi}_G$, $\tilde{\psi}_R$, and $\tilde{\psi}_S$.

During hypothesis generation, the resulting hypotheses of different pose estimation pipelines are accumulated to a common list of all initial hypotheses. In the following hypothesis refinement phase, the list of hypotheses is subsequently refined by a parameterized number of operators. Finally, all intermediate hypotheses are gathered and scored. This structure represents an instance of a hypothesize-and-test strategy. The actual operator instantiation of the structural elements is encoded via structure parameters, which represent the employed operators as ordered lists. All different variants are encoded in the set of parameter values, which depends on the available operators $\mathrm{G}, \mathrm{R}, \mathrm{S} \in \mathbb{O}$ for each structural element. For instance, given the two operators R_1 and R_2, the set of parameterizations is

$$\tilde{\Psi}_R(\{R_1, R_2\}) = \{\varnothing, R_1, R_2, R_1R_2, R_2R_1\}, \tag{4.2}$$

with $\tilde{\psi}_R \in \tilde{\Psi}_R(\{R_1, R_2\})$ and $\tilde{\Psi}_R \in \Lambda$.

The pipeline structure template additionally defines conditions on the inputs and outputs of the operators for different structural elements. In the working example, hypothesis generation operators G must produce a hypothesis list as output and may not require a hypothesis list as input. Refinement operators R require a hypothesis list as input and must produce a hypothesis list as output. The hypothesis scoring operators S transform hypotheses to scored hypotheses.

4.1.4 Optimization Strategies

In the following, we introduce two different optimization strategies on top of *sequential model-based optimization* (SMBO) Hutter et al. (2011). In brief, SMBO performs black box optimization effectively by building a computationally efficient performance prediction model, that is used to evaluate the majority of parameter configurations.

With the different strategies we aim to gain insight, whether the joint optimization of all parameters is to be preferred over a strategy where operators are optimized individually as a prior step. We start with a description of common initializations and proceed with the strategies.

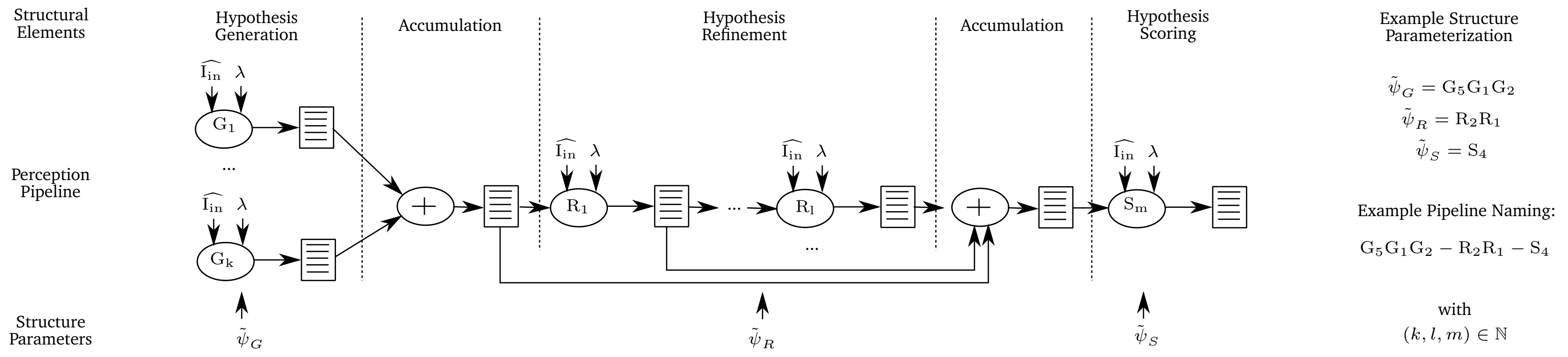


Figure 4.6: Pipeline template that is used as working example. It is separated in three parameterizable structural elements, hypothesis generation, hypothesis refinement, and hypothesis scoring. The hypotheses can be generated by multiple operators. The refinement is a sequential procedure, where each refinement operator gets its input hypotheses from the preceding operator. The structural elements are parameterized via the structural parameters $\tilde{\psi}_G$, $\tilde{\psi}_R$, and $\tilde{\psi}_S$.

First, the available operators from the operator library are matched to the structural elements of the pipeline template, which is performed based on the template conditions for the different structural elements. Additionally, the required input of the operators has to be given within the dataset. The set of values for the structure parameters is initialized by generating the possible sequences of the operators for each structural element. Finally, operator and structure parameters are added to the optimization configuration space. The operator parameters are additionally conditioned on the structure parameters such that they are only considered when the operator is within the currently chosen operator sequence of the structure parameter.

The dataset $\mathbb{D}$ is split into a training set $\mathbb{D}_{\text{train}}$ and a test set $\mathbb{D}_{\text{test}}$. The test set is used as a holdout dataset to assess the generalization of the pipeline structure and parameterization. We furthermore perform the training of data-driven operators, such as neural networks as initialization step. For this step only $\mathbb{D}_{\text{train}}$ as well as additional simulated data can be used.

Joint-Optimization Strategy JointOpt

Within the joint-optimization strategy, structure parameters and operator individual parameters are jointly optimized. Therefore, the responsibility to guide the search process is solely with the optimizer.

Pre-Optimization Strategy PreOpt

For the pre-optimization strategy the assumption is made that individually optimized operators are performing well within the pipeline structure. Therefore, prior to the optimization of structural parameters, the parameters of individual operators are optimized. This requires the following conditions to be fulfilled:

- input instances $\mathrm{I_{in}}$ and ground truth instances $\mathrm{I_{gt}}$ must be available for the input and output of the operator and
- the error metric $\mathrm{e_i}(\mathrm{I_{out}}, \mathrm{I_{exp}})$ needs to be defined for the operator output types.

The operator individual optimization is followed by a joint optimization of the structural parameters and the remaining operator parameters. The previously optimized parameters of the individual operators are fixed.

4.1.5 Test Metric

In order to evaluate the results and stop the optimization, we use a test error $\mathrm{e}_{\mathrm{test}}$ where an acceptance threshold θ is applied on the instance error of the test dataset:

$$\mathrm{e}_{\mathrm{test}}(\mathrm{P}_\lambda, \mathbb{D}_{\mathrm{test}}) = \frac{1}{n} \sum_{j=1}^{n} \begin{cases} 1 & \text{if } \mathrm{e}_{\mathrm{i}}(\mathrm{P}_\lambda(\mathrm{I}_{\mathrm{in}}), \mathrm{I}_{\mathrm{exp}}) > \theta \\ 0 & \text{otherwise} \end{cases}, \tag{4.3}$$

with $\mathrm{d}_j = (\mathrm{I}_{\mathrm{in}}, \mathrm{I}_{\mathrm{gt}}, \mathrm{I}_{\mathrm{exp}}) \in \mathbb{D}_{\mathrm{test}}, n = |\mathbb{D}_{\mathrm{test}}|$. The threshold θ results from the application requirements. The test error reflects the ratio of data instance sets where the application requirements are not fulfilled.

4.2 Evaluation

We evaluate the proposed offline pipeline configuration approach in different experiments. First, we use a subset of the T-LESS Dataset Hodan et al. (2017) for the comparison of the optimization strategies and analysis of different aspects of the overall approach. In the second experiment, the approach is applied to an industrial assembly scenario, where the automatic configuration is used to determine a working pipeline and parameterization for the given assembly task. As the experimental setup is shared to significant parts, common elements are introduced first.

4.2.1 Setup

The set of operators used in the experiments is listed in Tab. 6.1. The operators are chosen such that different approaches for 6D pose estimation and refinement are represented. The shape refinement operator R_{H} and depth adaptation operator R_{D} are custom engineered. The operator G_{MP} is composed of the MaskRCNN operator O_{M} and the Point Pair Feature Matcher O_{P}. Individually, the operators O_{M} and O_{P} are not matched within the pipeline template due to wrong output type and missing input. In these experiments, each operator may only be inserted once in each structural element. Therefore, the initialization of the structure parameters given the set of operators results in 64 distinct pipeline structures.

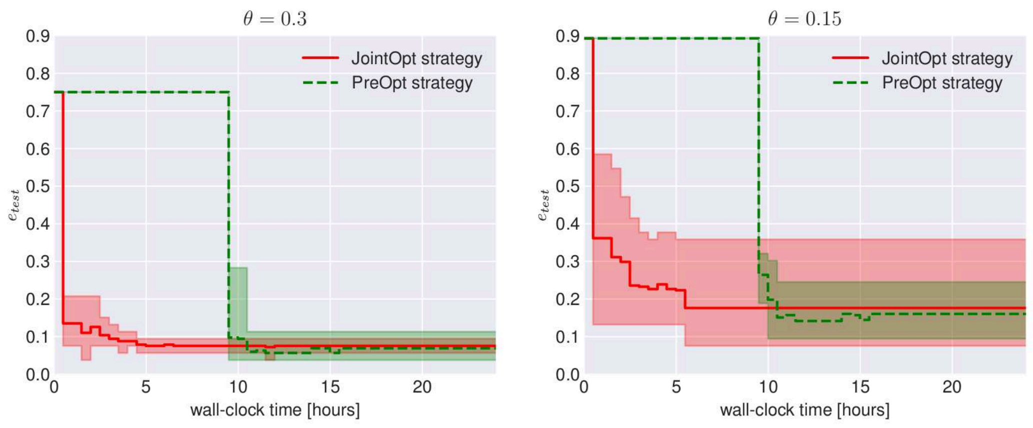


Figure 4.7: *Error trajectories for the strategies* ***JointOpt*** *and* ***PreOpt*** *for different error thresholds. For better visibility the outer hull and mean values of the individual trajectories of the 6 different seeds are displayed. The initial plateau for the* ***PreOpt*** *strategy represents the pre-optimization of individual operators. On average, the performance of both strategies is similar.*

The overall dataset with 100 data points is split into $50\,\%$ training and $50\,\%$ test data. The pre-optimization is only performed on $\mathrm{G_{MP}}$ according to the strategy definition. The parameter default values and ranges are initialized according to recommended values in the documentation, if available.

As error metric e, we use the visual surface discrepancy $\mathrm{e_{VSD}}$ as defined in the BOP benchmark Hodan et al. (2018), with the misalignment tolerance τ. We set $\tau = 0.02\,\mathrm{m}$, in accordance to the BOP benchmark. We evaluate the results for the acceptance thresholds $\theta = 0.3$ and $\theta = 0.15$. For the error calculation only the hypothesis with the highest score computed via $\mathrm{S_R}$ is considered.

The framework is written in Python. As sequential model-based black box optimizer we use PySMAC version 0.10.0 Hutter et al. (2011). The experiments are computed on a Intel Core i7-4810MQ CPU. The training time of the neural networks is not considered in the following diagrams.

4.2.2 Strategy Comparison

In this first experiment, we apply our approach on a perception problem from the T-LESS dataset Hodan et al. (2017) and compare the different optimization strategies. Only the T-LESS scene *scene_09* and object *obj_03* are used for this experiment. The strategy evaluation is performed on a randomly sampled subset of the overall dataset with a size of 100 images, which is kept constant across all experiments.

Strategy	Seed	Pipeline	Test Error $\theta = 0.15$
JointOpt	4	$\mathrm{G_S G_{MP}}$-$\mathrm{R_H R_D}$-$\mathrm{S_R}$	0.08
PreOpt	5	$\mathrm{G_S G_{MP}}$-$\mathrm{R_H R_D R_I}$-$\mathrm{S_R}$	0.09
PreOpt	2	$\mathrm{G_S G_{MP}}$-$\mathrm{R_H R_D}$-$\mathrm{S_R}$	0.11
JointOpt	0	$\mathrm{G_S}$-$\mathrm{R_H R_D}$-$\mathrm{S_R}$	0.13
PreOpt	0	$\mathrm{G_S G_{MP}}$-$\mathrm{R_H R_D R_I}$-$\mathrm{S_R}$	0.13
JointOpt	2	$\mathrm{G_S G_{MP}}$-$\mathrm{R_H R_D}$-$\mathrm{S_R}$	0.15
JointOpt	3	$\mathrm{G_{MP} G_S}$-$\mathrm{R_H R_D}$-$\mathrm{S_R}$	0.15
PreOpt	1	$\mathrm{G_S G_{MP}}$-$\mathrm{R_H}$-$\mathrm{S_R}$	0.17
JointOpt	1	$\mathrm{G_S G_{MP}}$-$\mathrm{R_H R_D R_I}$-$\mathrm{S_R}$	0.19
PreOpt	4	$\mathrm{G_{MP} G_S}$-$\mathrm{R_H R_D}$-$\mathrm{S_R}$	0.21
PreOpt	3	$\mathrm{G_S}$-$\mathrm{R_H R_D}$-$\mathrm{S_R}$	0.24
JointOpt	5	$\mathrm{G_{MP} G_S}$-$\mathrm{R_H}$-$\mathrm{S_R}$	0.36

Table 4.1: *Test error for the best results for 6 different seeds of the* ***PreOpt*** *and* ***JointOpt*** *strategies. Result are sorted by the test error with a threshold* $\theta = 0.15$

The strategies **JointOpt** and **PreOpt** are evaluated for 6 different optimization seeds. The stopping criterion for the optimization is either a zero valued test error or a maximum runtime. For the **JointOpt** strategy the maximum runtime is set to $24\,\mathrm{h}$. The **PreOpt** strategy consists in two optimization phases, the pre-optimization with a maximum runtime of $9\,\mathrm{h}$ and the joint optimization with a maximum runtime of $15\,\mathrm{h}$, such that both strategies run within 24 hours.

In Fig. 4.7, the outer hull and mean of the resulting test error trajectory for the different optimization seeds are displayed for both strategies for a threshold θ of 0.3 and 0.15 respectively. The pre-optimization duration of maximum $9\,\mathrm{h}$ of the **PreOpt** strategy is displayed as default error plateau within the **PreOpt** strategy of the minimal default pipeline $\mathrm{G_S}$-$\mathrm{S_R}$. Additionally, the resulting pipelines for the different strategies and optimization seeds together with their test error are displayed in Table 4.2.

Several characteristics can be observed. First, the resulting minimum absolute test error and error variance is higher for the more difficult case of $\theta = 0.15$, which is the expected outcome. The best performing configuration for each seed is typically found within the first 6 hours of optimization, which leads to a flat error curve.

	Pipeline	Test Error
		$\theta = 0.3$
Origin		
Pre 1	SMP-H-R	0.04
Pre 2	SMP-HD-R	0.04
Default 0	S-DHI-R	0.04
Pre 0	SMP-HDI-R	0.06
Joint 2	SMP-HD-R	0.06
Default 1	S-HDI-R	0.06
Default 2	S-DH-R	0.06
Default 3	MPS-DHI-R	0.06
Joint 0	S-HD-R	0.08
Default 4	SMP-DI-R	0.08
Joint 3	MPS-HD-R	0.08
Joint 1	SMP-HDI-R	0.09
Pre 3	S-HD-R	0.11

Table 4.2: *Test error for the 5 best pipelines of the* ***Default*** *strategy and the best results for 5 different seeds of the* ***PreOpt*** *and* ***JointOpt*** *strategies. Result are sorted by the performance with a threshold of 0.15*

For $\theta = 0.3$ the **PreOpt** strategy has higher variance, whereas for $\theta = 0.15$ the error trajectories of the **JointOpt** strategy display higher variance. The mean error in both settings is slightly better for the **PreOpt** strategy. The **PreOpt** strategy also shows a faster convergence rate, as the parameter search space is smaller once the pre-optimization is finished. In this setup, no clear strategy preference can be deduced. In terms of resulting pipeline structures, there is a strong presence of the refinement operator $\mathrm{R_H}$, especially at the first position. The refinement pipeline $\mathrm{R_H R_D}$ is most frequent within the results.

The resulting pipelines generally are compatible with engineering intuition. The $\mathrm{R_H}$ operator improves the orientation estimate, but remains with erroneous depth estimation due to its RGB only input. The $\mathrm{R_D}$ operator may correct the depth estimation and is mostly placed afterwards. The ICP (operator $\mathrm{R_I}$) is placed last in the top performing pipelines, which can be explained by the required accuracy in the initial guess. Interestingly it is not used in most of the pipelines. This is possible since the ICP may produce wrong hypotheses with good final depth score in the given setup.

Overall, there is a benefit of searching structure and parameter search space in order to improve the perception performance in the given setup. But the sensitivity with respect to the initial seed can be high. But not all configuration space dimensions are explored in an automatic fashion. In the presented scenario the perception engineer still has to decide on structure templates and the hardware setup to be used. Also, the design of entirely new operators and perception algorithms is still out of scope.

4.2.3 Validation in Assembly Scenario

The second experiment is a real-world robotic assembly experiment. The task is to localize a control cabinet part and mount it on a hat rail. The part is a SIMATIC ET 200S terminal module. The manipulators are two seven-axis robots with parallel 2-finger grippers and the sensor is an ASUS Xtion PRO LIVE. The overall setup is displayed in Fig. 4.8. For the manipulation planning and assembly motion generation we use the approaches from Schmitt et al. (2017) and Wirnshofer et al. (2018).

In order to assess the benefit of using the pipeline parameter and structure optimization, we compare the performance of different optimized pipelines based on the assembly success. The used dataset for the pipeline configuration consists of 100 images with multiple object instances that were acquired using a slow but accurate perception pipeline, where additional wrist-mounted cameras are placed close to the objects in order to ensure high accuracy. As configuration strategy the **JointOpt**

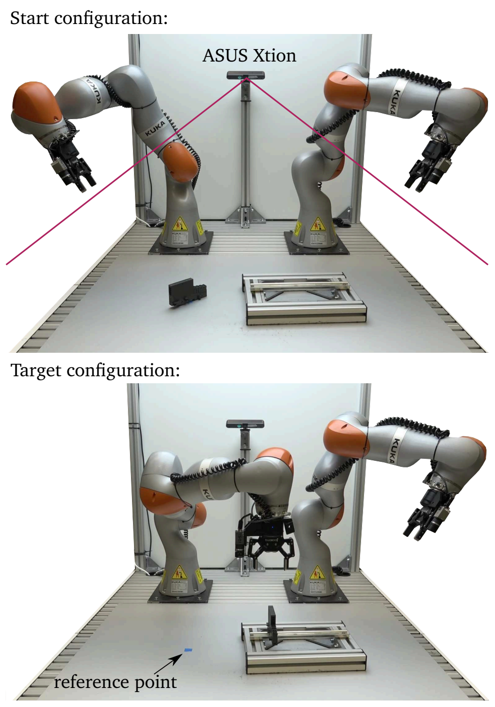


Figure 4.8: Experimental setup for the pipeline evaluation in an assembly scenario. The object is placed in 16 different orientations on a reference point. The goal is to mount the object on the hat rail and success and failure are determined via successful snap-in. The experiment is repeated twice, resulting in 32 trials overall.

strategy is used with a maximum runtime of 12 hours and different seeds. The choice for the **JointOpt** strategy is arbitrary according to the comparison results. The authors prefer it here, as structure and parameters are computed in the same optimization step.

For the actual experiments the part is placed on the reference point, see Fig. 4.8, either lying on the flat side or standing as depicted. These orientations correspond to the acquired training data. Additionally, after each run the object is rotated around the upright axis about 45° with respect to the previous pose. This yields 16 different poses overall. The experiment is performed twice, which results in 32 overall assembly trials. The trial is counted as success, when the part snaps and remains on the hat rail. For the experiments the hat rail is localized with the same pipeline as used for the dataset generation.

The results are shown in Table 4.3. The compared pipelines are the individually optimized basic hypothesis generation pipelines Mask-RCNN + PPF ($\mathrm{G_{MP}}$) and Single Shot Pose ($\mathrm{G_S}$) and two different resulting pipelines from the optimization $\mathrm{G_{MP}G_S}$-$\mathrm{R_H R_D R_I}$-$\mathrm{S_R}$ and $\mathrm{G_S}$-$\mathrm{R_H R_D R_I}$-$\mathrm{S_R}$. The hypothesis refinement pipeline $\mathrm{R_H R_D R_I}$

Pipeline	Assembly Success
$G_{MP}G_{S}$-$R_{H}R_{D}R_{I}$-S_{R}	26 / 32
G_{S}-$R_{H}R_{D}R_{I}$-S_{R}	22 / 32
G_{MP}-S_{R}	16 / 32
G_{S}-S_{R}	5 / 32

Table 4.3: *Pipeline performance in the assembly scenario, sorted by success rate.*

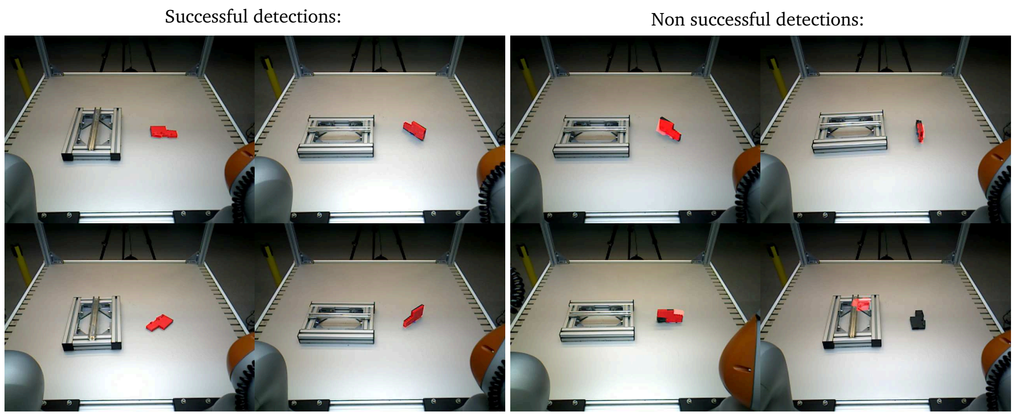


Figure 4.9: Exemplary pose estimation results for successful and non-successful assembly operations. Best viewed in color.

showed to be good in this experimental setting, whereas $R_{H}R_{D}$ performs better in the T-LESS use case.

In summary, it can be deduced, that the joint optimization of pipeline structure and parameterization leads to higher success rates compared to the base operators G_{MP} and G_{S}. Still, there remains a performance gap, due to high process requirements, sensor noise, operator insufficiencies, and the distance to the object. It could be addressed, for instance, by improved generation of training data for the neural networks and a different choice and positioning of the sensor. The integration of these configuration space dimensions is conceptually feasible and within the scope of future work.

4.3 Related Work

The configuration space modeling and exploration approach relates to different fields. AutoML addresses the tuning of hyper-parameters, algorithm choice, and architecture search for machine learning. For the former, approaches like Bayesian Optimization (Falkner et al., 2018), Genetic Programming (Olson et al., 2016), and Sequential Model-Based Optimization (Hutter et al., 2011) have been proposed. Tools such as AutoWEKA (Thornton et al., 2013), Auto-sklearn (Feurer et al., 2015), AutoKeras (Jin et al., 2018), and TPOT (Olson et al., 2016) are designed for specific machine learning libraries, while others such as SMAC (Hutter et al., 2011) and Hyperopt (Bergstra, 2016) are designed for general use. The authors of AutoWEKA (Thornton et al., 2013) introduce the problem of *Combined Algorithm Selection and Hyper-parameter Optimization* (CASH) for machine learning operators. We address a related problem for a mixed set of perception operators. Neural Architecture Search (NAS) addresses the adaptation of network architectures and is a promising field. For a review we refer to (Elsken et al., 2019). In this publication we target mixed pipelines of classic algorithms and neural networks, which is not in the scope of NAS.

AutoML techniques have been applied to data pre-processing pipelines (Quemy, 2019), parameter tuning, and algorithm selection for classification (Feurer et al., 2015) and SAT solver parameter tuning (Hutter et al., 2011), among others. In this work AutoML techniques and tools are applied in the concrete application domain of perception pipeline parameter and structure configuration for a diverse set of operators.

Another relevant field of research is the automated design of perception and sensor fusion software systems, which is typically solved by some sort of design space exploration. One line of research is the use of semantic models in order to describe the task and generate appropriate perception pipelines, such as (Niemann et al., 1990), (Koenderink-Ketelaars, 2010), (Fritze et al., 2017). Beetz et al. (2015a) use a query-answering approach and semantic models in order to generate perception pipelines at run-time. Hochgeschwender et al. (2015) use the Robot Perception Specification Language (Hochgeschwender et al., 2014) in order to describe and select perception pipelines and choose the pipeline parameterization from a pre-configured set depending on the current environment state.

Other approaches directly target the online and offline adaptation of perception pipeline parameters. For instance, Sakar Sarkar and Chavali (2000) model the parameters within a Bayesian parameter dependence network in order to cope with the search space complexity and dependencies. The authors of (Hu and Kantor,

2017) propose an approach to automatically tune the system parameterization online without expert supervision. Durner et al. (2017) use logged execution data in order to optimize the parameterization of different perception pipelines. We present an approach for the joint configuration of perception pipeline structure and parameterization in contrast to previous work where both are regarded as separate problems.

4.4 Summary and Discussion

In this chapter, we present an approach to perform automatic configuration of perception pipelines based on structure templates and sequential model-based optimization. The approach allows to determine suitable parameters as well as a suitable pipeline structure in order to adapt to the specific task setting.

The pipeline structuring approach allows to reduce the configuration space to a manageable size. It is currently instantiated manually and can be seen as part of the functional architecture. The black box optimization facilitates the optimization of pipelines which are composed of classical model-based operators as well as learning-based operators such as neural networks. The structuring as well as the configuration space exploration are represented as declarative and procedural models. Therefore, both can be easily be reused in different contexts.

In experiments on the T-LESS dataset as well as a real-world robotic assembly scenario we could demonstrate that substantial performance improvements can be achieved when both pipeline structure and parameterization are jointly configured.

However, no explicit model of uncertainty is used and the surrogate model is encoded within the optimizer. These points are addressed in the following chapters.

Chapter 5

Perception planning under Consideration of Geometric Uncertainty

In Chapter 2 we highlighted that geometric uncertainty is prevalent in industrial automation systems and has to be taken into account when designing and setting up a perception system. An automation system has different frames with varying uncertainty in its pose description. Observations, measurements and known coordinate frames form a graph of relative poses with associated uncertainties.

In this chapter, we address how geometric uncertainty can be taken into account in an automated fashion by using planning. In Fig. 5.1 a general overview of central components is given. The planning procedure can be applied in the offline engineering as well as the online adaptation phase. This depends on the actual setup, as the planning times have to fulfill the runtime requirements. Therefore, we employ an abstract model of geometric uncertainty based on factor graphs, which provides a good compromise between expressiveness and computational efficiency. During the planning procedure perceptual actions, such as viewpoint adaptations and the execution of algorithms, as well as the structure of the factor graph are explored. Furthermore, inference on the factor graph is used to estimate the resulting uncertainty of the target pose given the planned perceptual actions.

We address multiple sub-problems, which can be derived from the problem of automated synthesis of perception systems. First, an approach is required in order to handle geometric uncertainties across different coordinate frames. This means, that uncertainty needs to be represented and more importantly, the uncertainty between

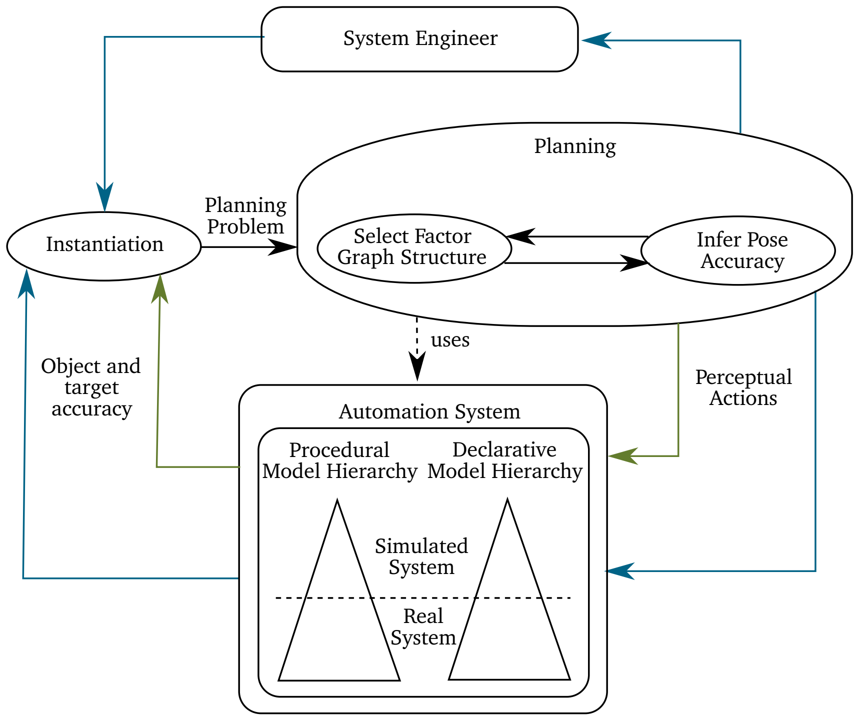


Figure 5.1: *Overview of the synthesis system for perception planning under geometric uncertainty*

the frames of interest needs to be estimated. To this end we present an approach based on factor graphs, which allows to jointly represent different measurements and priors. It thus serves as an approach for probabilistic multi sensor fusion.

Furthermore, we want to investigate whether the aforementioned approach allows to support the synthesis of perception systems. Therefore, we address a pose estimation task in a industrial environment with a static and a dynamic sensor, as shown in Fig. 5.2. The configuration space not only entails the choice of sensor and the choice of the data-processing, but also the movement of the robot. This means that a different sub-set of the configuration space is addressed than in the prior chapter, where the pipeline structure and parameterization where synthesized in an offline fashion.

The automated synthesis requires consistent models and a suitable configuration space formalization. Therefore, all required models are represented using the hierarchical modeling formalism as introduced in Chapter 3. As configuration space exploration approach, single level planning is used.

The overall interplay of components is visualized in Fig. 5.2. Using the semantic hierarchical models as presented in Chapter 3 a perception task in an assembly scenario is represented. The goal is to reduce the relative uncertainty between the

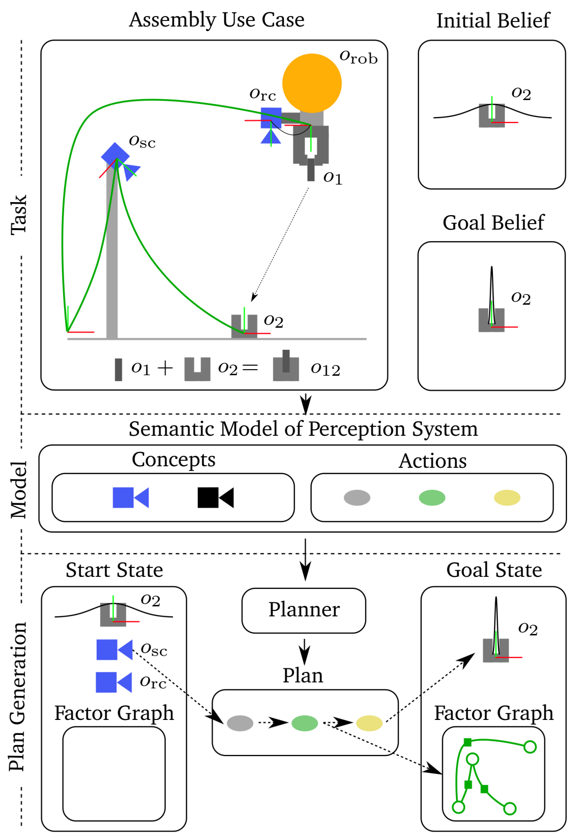


Figure 5.2: Overview of the configuration approach for an exemplary assembly use case, where a robot o_{rob} should assemble the part o_1 into o_2. Therefore, part o_2 needs to be localized precisely. The task and description is grounded in a semantic model, which comprises concepts such as a RGB camera and actions such as an image based object detection. Using the semantic description of the problem, the planner determines a sequence of actions to satisfy the goal. The plan includes the automatic generation of a factor graph which is used to estimate the object pose.

robot end-effector and a target part in the workspace. This uncertainty depends on a graph of poses and observations, as given in the automation system. Therefore, a planning problem can be instantiated, which is solved using a planner from the meta domain.

In summary, this chapter mainly addresses **Contrib. C3**. In the following we introduce the employed uncertainty model as well as the integration into the modeling formalism. Furthermore, we show how design synthesis can be performed based on the model using planning.

This chapter is based on and contains excerpts of the publication of Dietrich et al. (2018). In addition, factor graphs and procedural models for their handling are introduced.

5.1 Factor Graphs for Pose Estimation

In this section we introduce basic theory of factor graphs and how they can be leveraged for the estimation of poses and their uncertainty. Additionally, we present the actual model how factor graphs were modeled and integrated in order to be used within the automatic synthesis of perception systems.

5.1.1 Factor Graph Representation

In general, a factor graph represents the factorization of a function in the form of a bipartite graph. Let's consider the function $g(x_1, x_2, x_3, x_4, x_5)$ that can be represented as the product :

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1, x_2) f_B(x_2) f_C(x_2, x_3, x_4, x_5) f_D(x_4) f_E(x_5) \tag{5.1}$$

The function g can be represented as the factor graph shown in Fig. 5.3. The variables $x_1, \ldots, x_5$ are represented as circles and the functions $f_A, \ldots, f_E$ as black squares. The edges of the graph connect variables and the functions which take the variable as inputs. Formally, a factor graph F_{G} is composed of two types of nodes, *factors* $\phi_i \in \mathcal{U}$ and *variables* $x_j \in \mathcal{V}$, (Dellaert et al., 2017). Edges $e_{ij} \in \mathcal{E}$ always connect factors to variables. Therefore, the factor graph has a bipartite structure, where no factor is connected to another factor and no variable is connected to another variable. Overall, the factor graph can be written as $F_{\mathrm{G}} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$. The factorization represented by the factor graph can be defined as:

$$\phi(X) = \prod_i \phi_i(x_i), \tag{5.2}$$

with X being the state vector which contains the individual state variables.

The factor graph is a general formulation, that is beneficial when applied to uncertainty modeling and handling. Perception and state estimation problems in robotics can be formulated as a factor graph, where variables represent, e.g., poses of objects or landmarks and factors represent observations and constraints. In the pose estimation target domain, the individual state variables represent poses and the factors represent observations and prior knowledge. Uncertainty in the observations is included in individual factors and acts for instance as a weighting, such that uncertain observations have less impact on the overall estimation result.

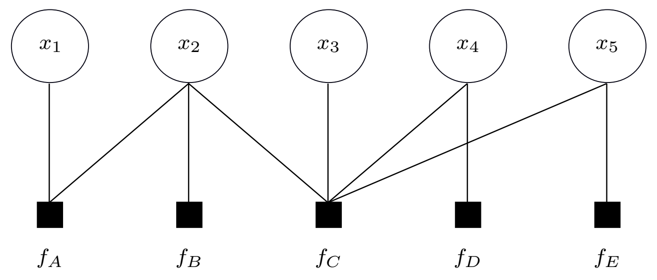


Figure 5.3: *Graphical visualization of a factor graph with 5 variables $x_1, \dots, x_5$ and the functions $f_A, \dots, f_E$, which represent relations between variables.*

5.1.2 Graph Construction

When working with probabilistic graphical models in an automated fashion, one step is to create the graphical model. Therefore, procedural knowledge is required, that is presented in the following.

The graph construction is represented as individual operators that add factors and nodes to a world model factor graph. This process is subject to constraints that will be discussed. A general overview is shown in Fig. 5.4. Initially, the world model graph is empty (PGM-Graph 0). For the graph construction we distinguish between two different types of operators:

1. **Prior addition operator**: Adds prior knowledge to the factor graph, for instance an initial pose distribution of an object in a scene.
2. **Observation addition operator**: Adds the result of incoming observations to the factor graph.

In the exemplary sequence, an operator of type 1, named *PGM_AddPrior*, adds prior knowledge to the world model graph, which is converted to a factor which connects two variables in the factor graph (PGM-Graph 1). In the subsequent step an operator of type 2 is executed, which takes an observation as input. The observation is equally converted into a factor and added to the graph (PGM-Graph 2).

The concrete operator being used at the lowest level of abstraction is illustrated in Fig. 5.5. The operator requires that the pose of the input object is associated with an uncertainty representation, such as a covariance matrix. Internally, it is assured first, that the exact same prior information is not already available within the graph. Otherwise the operator fails immediately. Next, the given pose is converted into a factor which represents the 6D constraint weighted by the associated uncertainty. Furthermore, the pose variables are added to the world model graph. Finally, the factor is added to the underlying factor graph. In general, a prior can also be given

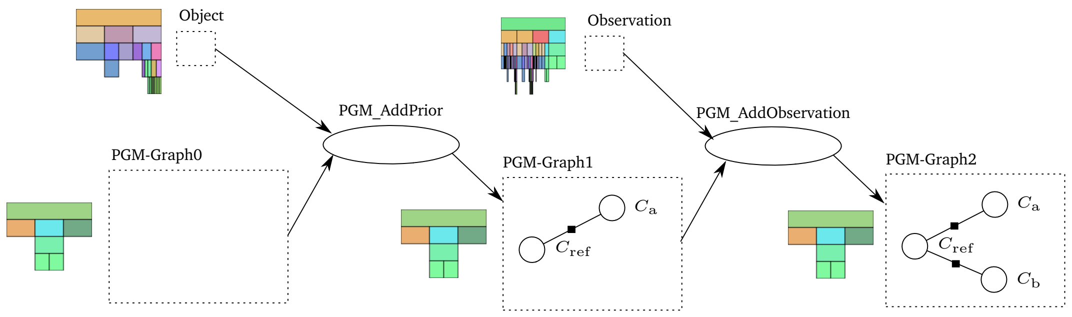


Figure 5.4: Exemplary multi-step factor graph build-up, consisting of an addition of a prior and an observation.

as a value without uncertainty. In this case the value can be used as starting value for the optimization based inference as introduced in the next section. This represents a weak prior that may improve the convergence characteristic of the inference step, but not necessarily the result.

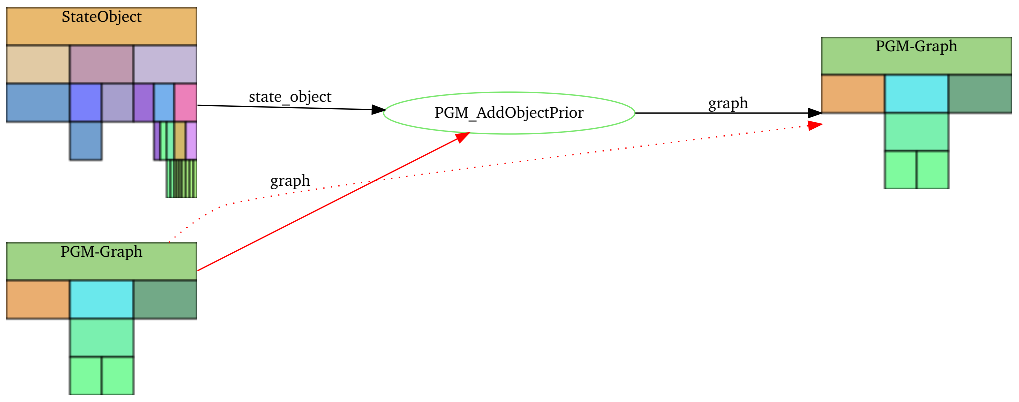


Figure 5.5: Model for the addition of an object prior with uncertainty to the probabilistic graphical model.

In Fig. 5.6 a concrete operator for 6D pose observations is depicted. Internally it is mostly equivalent to the operator *PGM_AddObjectPrior*.

Different types of observations require different types of factors to be added to the factor graph. This is realized by distinct operators. In Fig. 5.7, for instance, an operator is depicted that can handle 2D observations, such as object bounding boxes. Instead of generating and adding a 6D pose constraint, a 2D Point constraint is added to the factor graph. This operator internally requires a camera with associated intrinsics to be present in the world model graph. This information is added by a different variant of the *PGM_AddPrior* operator acting on objects of type camera.

Overall, there are different reasons for this graph construction approach. Here, there are individual operators for the graph construction which allows the overall synthesis

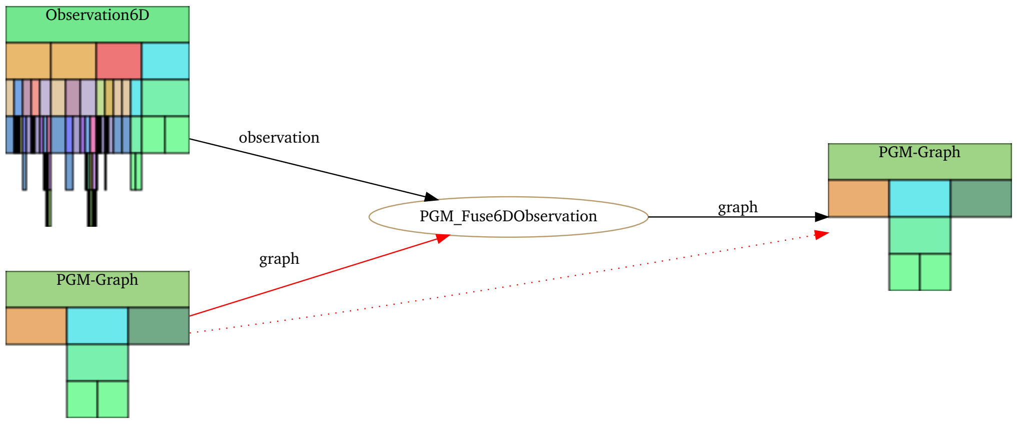


Figure 5.6: Model for the addition of a 6D Pose observation

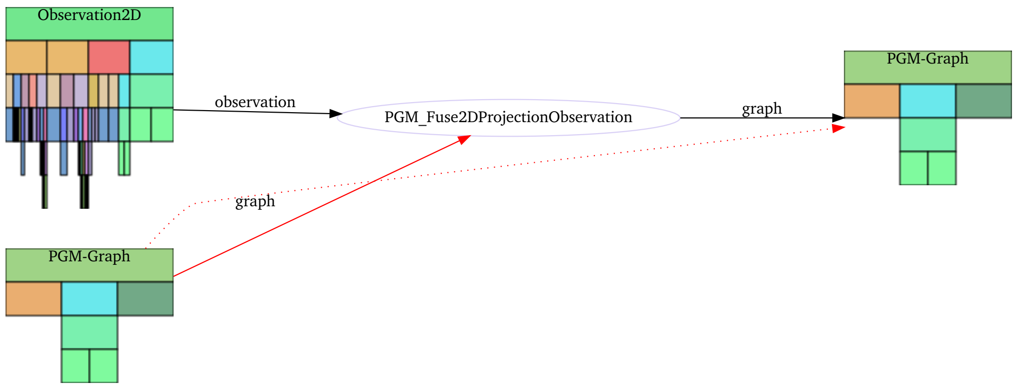


Figure 5.7: Model for the addition of a 2D observation

to leave out specific information in order to speed up or improve the inference. Similarly, a perception engineer chooses the type and amount of information and representation thereof. Therefore it is crucial to give an automated synthesis system similar freedom of decision on this low level. A common alternative would be an approach where every given piece of information is added per default. But this may lead to long inference times and it is not a generally valid approach for computationally demanding algorithms and time constraints as present in robotic applications.

5.1.3 Inference

There are two major activities with respect to uncertainty handling, quantification and reduction. The inference on the factor graph falls under the first category. In this context, *inference* does not refer to logical reasoning but to probabilistic reasoning in a Bayesian sense. More specifically, the goal is to derive information about the probability distribution of a state vector X given a set of observation Z. Written as probability density function, the inference can be denoted as:

$$p(X|Z). \tag{5.3}$$

The factorized description as given by the factor graph can be used to initialize a minimization problem. The following notation is based on (Kümmerle et al., 2011b). The general formulation of the minimization problem is given by the following equation:

$$x^* = \operatorname*{argmin}_x \sum_{k \in \mathcal{C}} e_k(x_k, z_k)^T \Omega_k e_k(x_k, z_k), \tag{5.4}$$

which denotes the minimization of an error sum across all available factors. Here, x describes a vector of individual variables $x = (x_1^T, \ldots, x_n^T)^T$. The output of the optimization is the value x^*, which best fulfills the constraints posed by the factors. The term k denotes the current factor from the set of all factors. For a more detailed introduction and especially handling of non-Euclidean spaces, as present in the rotation representation, we refer to (Kümmerle et al., 2011b).

The presented procedural knowledge is encoded in the operator depicted in Fig. 5.8. As input, an instance of a world model graph as well as a query is given. The query includes the coordinate systems, whose relative transform is required. As output, an instance of a 6D Pose is produced which describes the maximum a-posteriori pose estimate given all factors, as well as its associated uncertainty as Gaussian approximation. Internally, the presented minimization problem is initialized and solved.

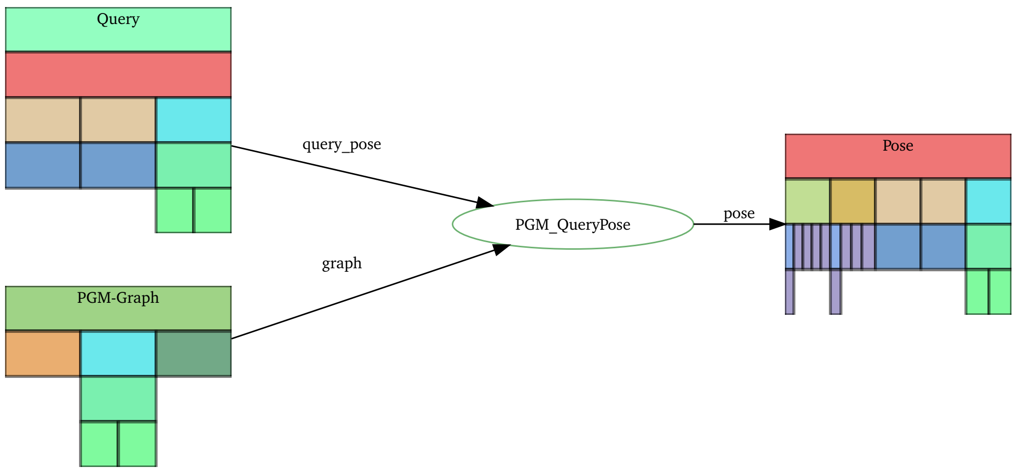


Figure 5.8: Model for the query of a 6D pose from the factor graph.

5.2 Planning problem

We consider the problem of planning a sequence of data gathering, data linkage, and inference steps to obtain a required set of information about the state of the robot and its surroundings. The information about the unknown state x of our system is represented as a probability distribution or belief $\mathrm{bel}\left(x\right)$. The focus in this chapter is on geometric uncertainty as it plays an important role in many domains, such as industrial assembly. As introduced, factor graphs are a well-suited representation for distributions involving geometric uncertainty and allow efficient inference using optimization techniques (Dellaert, 2012). Therefore, we encode the belief as a factor graph F_{G}.

A state within our planning problem is specified by the tuple $s = (F, F_{\mathrm{G}})$. In this tuple, F denotes a set of facts that hold in the current state. These facts encode currently available variables, such as encoder positions, as well as previous measurements, such as outcomes of a object detection procedure or the robot's encoder values when this measurement was taken. The factor graph F_{G} that encodes the belief $\mathrm{bel}\left(x\right)$ is constructed using these facts. At all times the system can choose from three types of actions:

- *Data Gathering*: By performing (simulated) perception actions, the planner gathers new data, which corresponds to adding new facts to F. An example is an object localization routine that produces a fact about the localization outcome, e.g. a relative pose between camera and object.
- *Data Linkage*: This action type sets facts into relation with each other. In the presented scenario, it is used to construct the factor graph F_{G}. Following the example of an object localization routine, the set of edges E of F_{G} is extended by the uncertain measurement. Depending on the already existing vertices in V, new vertices are added, e.g. for the camera pose or the object.
- *Inference*: Actions of this type target the generation of new facts by inference. In the localization example, an inference action may extract for instance a new object pose estimate by optimizing the variables within the factor graph F_{G}.

5.2.1 Formalization

Formally, an action $a \in \mathcal{A}$, with $\mathcal{A}$ being the set of all available actions, may only be applied if the current state s fulfills its preconditions: $s \in \mathrm{Pre}(a)$. After action a is applied we obtain a new state $s' = \mathrm{Post}(s, a)$. With this we can define valid plans (of length $k \in \mathbb{N}$) as sequences of states $\{s_i\}_{i \leq k}$ and actions $\{a_i\}_{i \leq k-1}$:

Definition 13 - Valid Plan

A plan $\{s_i\}_{i \leq k}$, $\{a_i\}_{i \leq k-1}$ is valid iff

1. $s_i \in \mathrm{Pre}(a_i)$ for $i \in 1...k-1$ and
2. $s_{i+1} = \mathrm{Post}(s_i, a_i)$ for $i \in 1...k-1$.

Starting from an initial state s_{start} the aim of the planner is now to reach a desired state of information within the set $\mathcal{S}_{goal}$. With this we can define feasible plans:

Definition 14 - Feasible Plan

A plan $\{s_i\}_{i \leq k}$, $\{a_i\}_{i \leq k-1}$ is feasible iff it is valid and

1. $s_1 = s_{start}$ and
2. $s_k \in \mathcal{S}_{goal}$.

An important assumption that we make during planning is that our system only produces maximum likelihood measurements as proposed in (Platt Jr et al., 2010). This has three key advantages:

- Deterministic planning domain: As our system's actions do not increase uncertainty, the belief state evolves as follows:

$$\mathrm{bel}_{t+1}(x) = \gamma\, p(z \mid x)\, \mathrm{bel}_t(x), \tag{5.5}$$

  where z is a measurement and γ a normalization variable. As the measurement z is random, the evolution of the belief is random as well. Assuming maximum likelihood measurements yields a deterministic evolution of the belief and a deterministic planning domain.

- Decoupling of perception planning and construction of the factor graph: As measurements always correspond to the mode of the measurement model, the generation of new facts during planning does not require the evaluation of the belief represented by the factor graph. This decouples planning of perception steps and sensor data fusion.
- No repeated fusion of the same measurement: Our measurements are assumed to be deterministic during planning. Therefore, perceiving objects with the same algorithm from the same robot configuration results in the same measurement. Thus the resulting fact is not added repeatedly to the set F. This prevents adding the same or similar data repeatedly into the factor graph.

For the experimental evaluation, we use a basic breadth-first search that performs all valid actions $a \in \mathcal{A}$ on s, where $s \in \mathrm{Pre}(a)$ holds.

5.2.2 Modeling

We use the generic graph-based modeling approach as introduced in Chapter 3 to describe the perception domain. Every fact $f \in F$ is an instance of a concept, which models its properties. For instance, the RGB camera concept has an intrinsics concept, which itself models the parameters of a pinhole camera model. Actions are modeled via operators. The model does not contain an explicit formulation of pre- and post conditions. The actions are implemented such that they fail upon execution if the input facts do not fulfill the pre-conditions. An example is the $move\text{-}close$ action as introduced in Sec. 5.3.1.2, which requires to know the object position with certain accuracy. Furthermore, actions may only be executed if actual facts of all modeled input concepts are available in F.

5.2.3 Exemplary Planning Sequence

For the sake of comprehensibility, we use this section to visualize and describe the mode of operation of our approach in a basic example with the help of Fig. 5.9. We assume a setting where an object o_p is visible to a camera o_c and has to be located with respect to a reference coordinate system $\mathcal{F}_\mathrm{ref}$. The goal is given by $x_{\mathrm{ref,p,-}}$, specifying a maximum positional uncertainty allowed for the task at hand. The

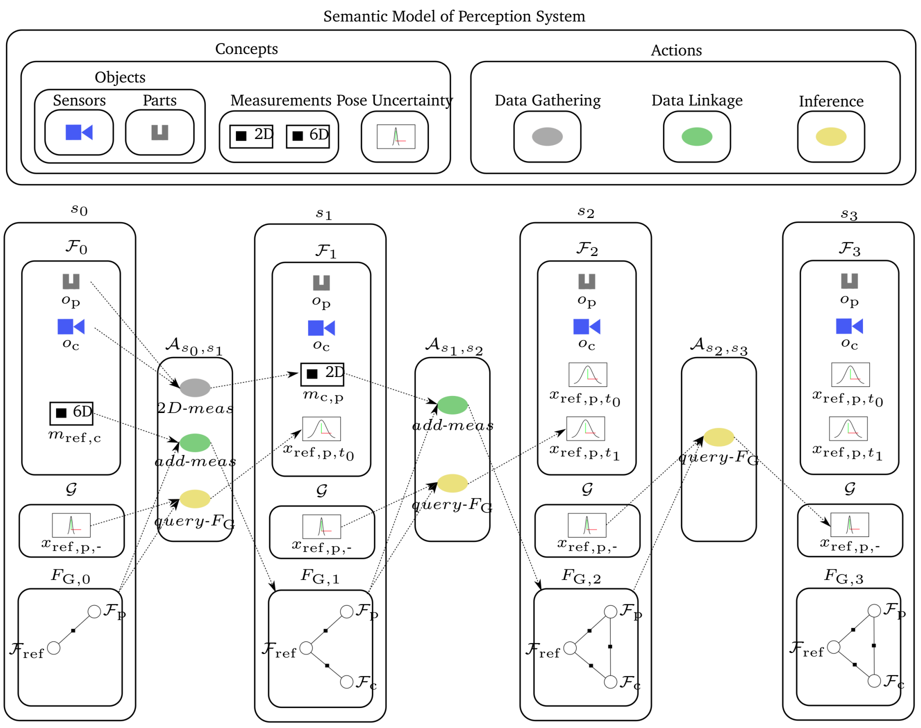


Figure 5.9: Exemplary visualization of the planning procedure for a basic setting with one object and one camera. The goal is to reduce the belief uncertainty over the object pose under a value specified within the goal set. In the upper part an exemplary model of the perception domain is outlined. In the lower part, the different system states from the start set s_0 to the final state set s_3, where the goal could be satisfied, are shown. A more detailed description of this figure can be found in Sec. 5.2.3.

goal $x_{\text{ref,p,-}}$ does not pose requirements on the time stamp of the result. A prior pose estimate between $\mathcal{F}_{\text{ref}}$ and the coordinate system $\mathcal{F}_{\text{p}}$ of o_{p} is given in form of a factor in the factor graph $F_{\text{G},0}$. Additionally, the extrinsic calibration $m_{\text{ref,c}}$ of o_{c} is given in form of a fact. This initial state is contained within s_0. Note that all facts and goals are grounded in a semantic model as indicated in the top part of Fig. 5.9. The visualized model contains only the required concepts and actions, but is in no way limited to these.

In the following, one feasible plan is presented in form of a possible sequence of actions to reach the goal. In the first step, 3 independent actions specified in $\mathcal{A}_{s_0,s_1}$ are executed: $2D$-$meas$, add-$meas$, and $query$-F_{G}. The action $2D$-$meas$ simulates a perception algorithm that may provide an estimate of an object position in a 2D image, e.g. the deep learning approach called *YOLO* (Redmon et al., 2016). As input, $2D$-$meas$ requires a RGB camera and, if successful, produces a 2D measurement. In the given example $2D$-$meas$ acts on o_{p} and o_{c} and produces $m_{\text{c,p}}$. It belongs to the introduced subset of actions that gathers new facts from existing facts. The second action $add\text{-}meas$ belongs to the subset of actions that link information in a graph representation. In the example it inserts the measurement $m_{\mathrm{ref,c}}$ into the factor graph $F_{\mathrm{G},1}$ of s_1. Finally, the third action $query\text{-}F_{\mathrm{G}}$ performs an optimization of the factor graph with respect to the reference frame of the goal input. As output, a pose with uncertainty, specifically $x_{\mathrm{ref,p},t_0}$ is produced. This pose, representing the pose prior over o_{p}, does not fulfill the goal requirements. Therefore the action sequence continues.

The next step, the transition between the states s_1 and s_2 is achieved by executing the action set $\mathcal{A}_{s_1,s_2}$. The action set contains two actions $add\text{-}meas$ and $query\text{-}F_{\mathrm{G}}$. Again, a measurement is added to the factor graph, in this case the previously produced 2D projection $m_{\mathrm{c,p}}$. Note here that using factor graphs, we are able to handle quite different pose constraints like a noisy 6D pose or a noisy 2D image projection. The general procedure and effect of the action $query\text{-}F_{\mathrm{G}}$ has been introduced in the previous paragraph and is similar for this state transition. As can be seen in the factor graph, the previously added extrinsic calibration $m_{\mathrm{ref,c}}$ of the camera has no effect on the pose estimate between the reference frame $\mathcal{F}_{\mathrm{ref}}$ and the target object o_{p} coordinate frame $\mathcal{F}_{\mathrm{p}}$. Therefore the output of $query\text{-}F_{\mathrm{G}}$ is an equally uncertain pose estimate $x_{\mathrm{ref,p},t_1}$ as $x_{\mathrm{ref,p},t_0}$.

Finally, in the last step of the exemplary plan another factor graph optimization is performed. This last action $query\text{-}F_{\mathrm{G}}$ of $\mathcal{A}_{s_2,s_3}$ acts on the factor graph $F_{\mathrm{G},2}$ that contains the measurement between target object and camera. The resulting pose uncertainty is significantly lower and satisfies in this example the requirements encoded in the goal set $\mathcal{G}$.

5.3 Evaluation

We chose an assembly use case, where geometric uncertainties have significant effect on the process success. After an introduction of the use case, we demonstrate the capability of our approach to find plans that achieve the accuracy required by the assembly process in minimal execution time.

The modeling and planning environment is a self-developed system that we intend to use in larger extent for machine knowledge management and robot autonomy. For the factor graph representation and optimization we build upon the open source library GTSAM (Dellaert, 2012).

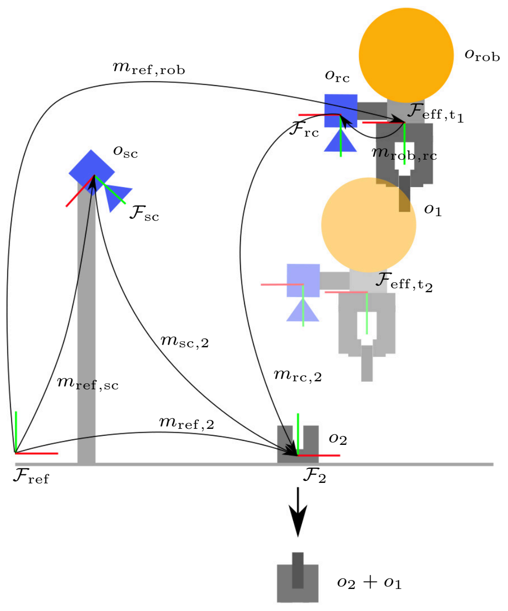


Figure 5.10: Overview of the assembly use case. The symbol o denotes objects, $\mathcal{F}$ denotes coordinate systems and m denotes pose constraints due to measurement or calibration. The graph of black arrows visualizes the factor graph in its full extent.

5.3.1 Experiment

Setup

Industrial assembly is a domain with high requirements on geometric uncertainty quantification as many assembly processes require tight positioning tolerances. In our specific use case an object has to be grasped for a follow-up assembly process as depicted in Fig. 5.10.

The perception goal consists of localizing the target object o_2 with respect to the robot end effector coordinate system $\mathcal{F}_{\mathrm{eff}}$. There is a static camera o_{sc} as well as an end effector camera o_{rc} mounted on the robot o_{rob}. Moreover, there exists an arbitrary chosen reference coordinate system $\mathcal{F}_{\mathrm{ref}}$. The overall application is to assemble o_1 into o_2, where the former already resides within the robot gripper. For the sake of brevity, we assume that o_2 has been localized and grasped with perfect accuracy. Furthermore, to show that our approach can cope with different viewpoints, an important parameter in perception tasks, we let the system decide to take a close view of the object. The action *move-close* that performs this operation requires the object to be known with a certain accuracy that is not given by the prior belief over the object pose. If the relative uncertainty specified by the goal is met, the assembly process can be successfully executed. The uncertainty introduced by the relative movement of the robot for the assembly process is not considered in this example.

Action Description

In this use case we allow the following actions:

- Data Gathering
  - $2D\text{-}meas$: Simulates an object detection algorithm working on RGB images that outputs the object center as a 2D point an the image screen. This action is similar to the 2D projection action described in Sec. 5.2.3. The chosen standard deviation in pixel σ_{px} depending on the object distance d is $\sigma_{px}(d) = 5 + 5 * d$.
  - $6D\text{-}meas$: Simulates a 6D pose estimation algorithm based on RGB images. For instance, it could represent the approach to estimate the 8 object bounding box corners using convolutional neural networks (*BB8*) as described in (Rad and Lepetit, 2017). The chosen positional standard deviation in meter σ_{pos} depending on the object distance d is $\sigma_{pos}(d) = 0.0005 + 0.0045 * d$. And the chosen rotational standard deviation in degrees σ_{rot} depending on the object distance d is $\sigma_{rot}(d) = 2 + 4 * d$. Simple linear approximations of the uncertainty are sufficient to evaluate the proposed configuration system.
  - $move\text{-}close$: Moves the robot o_{rob} such that the end-effector camera o_{rc} has a closer and centered view on the object at a distance of $d = 0.4\,\mathrm{m}$. The pre-condition for this action is that the standard deviation of the positional uncertainty σ_1 of the target object o_2 is lower than a threshold $\sigma_{\mathrm{thres}} = 0.1\,\mathrm{m}$. This is motivated by the fact that a robot cannot take a close look on an object whose position is not known.
- Data Linkage
  - $add\text{-}meas$: This action inserts a measurement to the factor graph. See also Sec. 5.2.3.
- Inference
  - $query\text{-}F_{\mathrm{G}}$: The graph query has a pose as input that specifies reference frame and target frame. The action optimizes the factor graph with respect

to the reference frame and returns the pose and uncertainty between reference frame and target frame. See also Sec. 5.2.3.

5.3.2 Results

In the following, results of the experiment will be discussed from two different viewpoints. First we analyze the reduction of uncertainty in the belief. Subsequently, an analysis of the temporal characteristics of the generated plans is given.

Uncertainty Reduction

We only consider geometric uncertainty, more specifically positional and rotational uncertainty. For the assembly use case we are interested in the maximum standard deviation. Therefore, we define the standard deviations σ in the following as the maximum of the principal components of the separate 3x3 covariance matrices for position and rotation. The joint distribution of position and rotation is currently not considered.

In Fig. 5.11 we visualize the positional and rotational uncertainty of all pose estimates generated by the planning system. Note here again, that the pose estimate between the robot end-effector coordinate system $\mathcal{F}_{\text{eff}}$ and the target object coordinate system $\mathcal{F}_2$ is considered. This has important implications for the following analysis. Hereafter, we will discuss specific poses denoted by p and marked in the figure:

- p_0: The pose p_0 marks the initial belief over the target object o_2. Positional and rotational uncertainty are both relatively large.
- p_1: The pose p_1 denotes the belief after a 2D measurement between the static camera and the target object $m_{\text{sc},2}$. Due to the nature of the 2D measurement, the rotational uncertainty is not reduced with respect to p_0.
- p_2: The pose p_2 denotes the belief after a 2D measurement between the end-effector camera and the target object $m_{\text{rc},2}$. The positional uncertainty is clearly lower than p_1. This is due to the application setting. The end-effector camera is calibrated relative to the robot end-effector, with the calibration denoted by $m_{\text{rob,rc}}$. The uncertainty of p_2 is most heavily influenced by the measurement chain of camera calibration $m_{\text{rob,rc}}$ and measurement $m_{\text{rc},2}$, which is a subset of

the full factor graph as depicted in Fig. 5.10. Contrarily, for the uncertainty of p_1 the measurement chain of robot positioning uncertainty $m_{\mathrm{ref,rob}}$, static camera calibration $m_{\mathrm{ref,sc}}$, and static camera observation $m_{\mathrm{sc},2}$ is decisive. Obviously, this longer measurement chain including the robot positioning uncertainty induces larger uncertainty. The influence of the prior $m_{\mathrm{ref},2}$ on p_1 and p_2 is identical for both cases. As we model the belief as a factor graph, all measurement chains are jointly considered and the correct determination of the uncertainties is automatically handled by our approach.

- p_3: The pose p_3 represents the belief after a 6D measurement with the static camera. The uncertainty is much lower than solely using 2D measurements.
- p_4: This pose marks the lowest achievable belief uncertainty for this exemplary planning setting. It is based on the fusion of all 2D and 6D measurements for all cameras and robot configurations, namely the static camera o_{sc}, the end-effector camera $\mathcal{F}_{\mathrm{rc}}$ in the initial pose of the robot and the end-effector camera $\mathcal{F}_{\mathrm{rc}}$ in the close view pose of the robot.

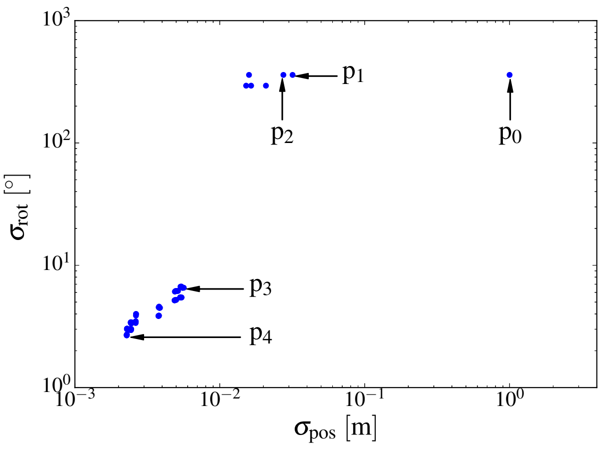


Figure 5.11: Visualization of resulting pose uncertainties generated via the proposed configuration system for the assembly use case. Note that due to the axis configuration this figure reads from right to left and top to bottom as opposed to Fig. 5.12. A detailed description can be found in Sec. 5.3.2.1. The corresponding experimental setup is visualized in Fig. 5.10.

Runtime

In this section we analyze the temporal extent of different plans. We therefore assume that an image is directly available whenever a perception action is applied.

In Fig. 5.12 all belief poses are visualized by time stamp and positional uncertainty. Contrary to the previous section we will additionally address sets of poses denoted by P. Furthermore, state transitions allowed in this planning problem are visualized via gray arrows.

The different actions have different runtime characteristics, which we set to reference values in order to gain insight about the planning system. For the simulated experiment we define the following values motivated by a real use case:

- $2D\text{-}meas$: $30\,\mathrm{ms}$
- $6D\text{-}meas$: $100\,\mathrm{ms}$
- $move\text{-}close$: $500\,\mathrm{ms}$
- $add\text{-}meas$: $1\,\mathrm{ms}$
- $query\text{-}F_{\mathrm{G}}$: $10\,\mathrm{ms}$

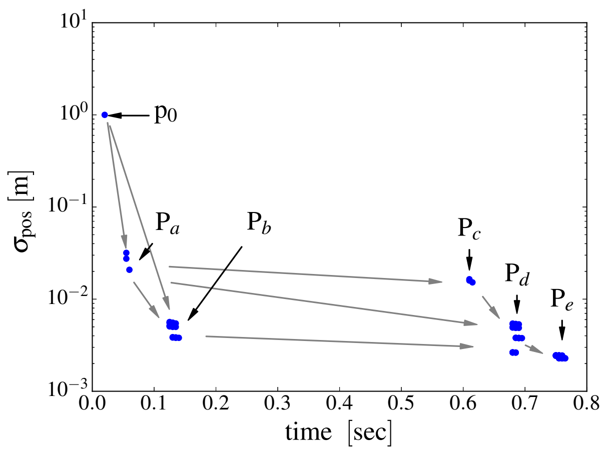


Figure 5.12: Visualization of resulting pose uncertainties and the required execution time in a real system to achieve the belief state quality. A detailed description can be found in Sec. 5.3.2.2. The corresponding experimental setup is visualized in Fig. 5.10.

The achievable accuracy depends on the invested time. Therefore, the characteristic of the different pose sets is discussed in the following.

- p_0: The initial belief.
- P_a: Pose set P_a contains all poses that are estimated using fast 2D measurements without moving the robot.
- P_b: Pose set P_b contains all poses that are estimated using at least one 6D measurement without moving the robot.

- P_c: To reach this set, the robot view pose is changed and further 2D measurements are considered from the new robot pose. The motion of the robot takes a large amount of time, and the uncertainty is not decreased with respect to the pose set P_b. From application view it does not make sense to reach this belief.
- P_d: Pose set P_d is similarly reached via robot motion. Further it requires at least one 6D measurement from the new view pose.
- P_e: Pose set P_e is the set with the lowest reachable belief uncertainty in the given use case. It is reached by changing the view pose and fusing up to all possible measurements.

Summary

Concluding, it can be seen that time can easily be incorporated into the approach. Depending on the application requirements, the planning system generates different action sets. For instance, in an application where a fast position estimate of an object is required, the actions leading to P_a are sufficient. In a less demanding assembly task, where the full pose needs to be known, the action sequence leading to P_b is chosen. Finally, in an application which requires very high accuracy, moving the robot to a better view pose is required and an action sequence leading to P_e is used. The key is that the configuration system, including the planner, can autonomously decide the right steps to take depending on the task.

The presented system represents an initial version of the general approach. To achieve autonomous configuration for a broad range of applications, the action set needs to be further extended and more advanced planning techniques evaluated to efficiently cope with the complexity of the task.

5.4 Related Work

Several approaches have been proposed in the past for the problem of automatic configuration of perception systems. An approach based on unstructured information management is implemented in the *RoboSherlock* framework by Beetz et al. (2015a). In combination with the knowledge bases *KnowRob* (Tenorth and Beetz, 2013) and *OpenEASE* (Beetz et al., 2015b) and the *Semantic Robot Description Language*

SRDL (Kunze et al., 2011), successful perception pipelines can be determined according to situation requirements. The authors focus on semantic reasoning grounded in *OWL* (Martin et al., 2004) ontologies, but also show how probabilistic reasoning can be leveraged for perception, specifically for object classification (Nyga et al., 2014). Geometric uncertainties induced by perception actions and calibration errors are not in the main focus of the framework.

Another approach for modeling and situation aware adaption of perception actions can be found in the work of Hochgeschwender et al. (2015). The authors introduced and use the *Robot Perception Specification Language RPSL* (Hochgeschwender et al., 2013) to describe the perception task and employ a reasoning mechanism to find a suitable perception plan.

The research area of world modeling for autonomous systems does also provide solutions for adaptive perception. Elfring et al. (2013) present an approach to keep a consistent probabilistic world model based on probabilistic multiple hypothesis anchoring. The probabilistic world model is further updated with strategies that maximize information gain and allow for a basic task dependency (Elfring et al., 2015).

Another relevant area of research is the field of active perception (Bajcsy, 1988). Research ranges from view selection (Eidenberger and Scharinger, 2010) over adaptive parameter tuning (Hu and Kantor, 2017) to concepts of a framework which enables autonomous configuration of perception and sensor fusion (Govindaraj et al., 2017). The latter work focuses on the underlying software framework and anticipates the autonomous configuration as future work.

Handling geometric uncertainty, for instance in assembly applications does have a long history. In early work from Su and Lee (1992) geometric relations including uncertainty are modeled within a directed graph of transformations with covariance matrices. Using uncertainty propagation and sensor fusion with Kalman filter, the covariance and pose between coordinate systems is determined. Furthermore, the authors provide an approach based on backward propagation to determine the admissible set of actions as well as required perception actions. The approach for handling geometric uncertainty is still commonly used, for example in the work of Blumenthal et al. (2013). For applications such as simultaneous localization and mapping, handling of uncertainty with factor graphs has become the dominant approach (Cadena et al., 2016).

The most common way to express planning tasks is the *Planning Domain Definition Language PDDL* (Fox and Long, 2003) subsuming, for example, the problems addressable with the *Stanford Research Institute Problem Solver STRIPS* language (Fikes and

Nilsson, 1971) and the *Action Description Language ADL* (Pednault, 1994). Two standard approaches to solve planning tasks are classical planning (Helmert, 2006), (Hoffmann and Nebel, 2001), and *Hierarchical Task Network (HTN)* planning (Erol et al., 1994), (Nau et al., 2003). Planning represents an important step in the automatic configuration of perception systems, but the performance depends on the model and the specific task.

The handling of uncertainties is a wide field that is required in many domains, from finance to biology. Therefore the literature review in this section is focused on handling of uncertainties in the context of the synthesis of perception systems as well as perception problems in robotic applications. Furthermore, the handling of uncertainties is often closely linked with the representation of uncertainty. Hence, the explanation of techniques for uncertainty handling may require to explain underlying declarative representations.

A fundamental approach for handling uncertainties in the perception domain are based on the *Bayes' theorem*, which is a mathematical theorem about conditional probabilities, (Bayes, 1763). It allows to infer a probability distribution of a (state) variable conditioned on given measurements and their probability distributions. This kind of inference falls under the term *state estimation* which is a common task of a perception system. The Bayes' theorem allows to derive the *Bayes' filter* which is a recursive probabilistic method for state estimation, (Thrun et al., 2005). It allows to estimate state and uncertainty of a variable in a continuous manner based on the last estimate and a new measurement. Important variants of the Bayes' filter are the *Kalman filter*, (Kalman, 1960), and the *particle filter*, (Del Moral, 1996). The Kalman filter internally uses normal distributed variables and only produces normal distributions as uncertainty estimate. The particle filter works with a particle based representation of uncertainty, which allows the handling of more complex distributions with the downside of higher computational demand. The choice of the appropriate filter as part of the perception system synthesis for an application at hand again depends on the type of application, the environment and the prevalent dynamics.

Graphical models are a powerful formalism to model and perform inference for a large set of probabilistic estimation problems, (Koller and Friedman, 2009), (Murphy, 2001). In fact, Bayes' filters such as particle filters and the Kalman filter, as well as problems such as decoding and genome analysis can be represented as graphical models and computed based on generic algorithms acting on the graphical model, (Kschischang et al., 2001), (Frey et al., 2005). The core idea is to represent variables and their dependencies in form of a graph. On the one hand this allows to visualize the structure of a problem in a clear and easily understandable manner. On the other hand, the graph structure can be exploited in order to efficiently perform inference on the model, (Bishop, 2006). Prominent types of graphical models are *Bayes networks*, *Markov networks*, and *factor graphs*. In this chapter we concentrate on the factor graph representation.

Factor graphs are widely used for uncertainty handling in various perception tasks, (Dellaert et al., 2017). The most common application is simultaneous localization and mapping (SLAM), where the task is to build a map of the environment and determine the sensors location within the map, (Grisetti et al., 2010). A graph-based formulation of the SLAM problem was introduced in 1997 by (Lu and Milios, 1997). Other applications include calibration, (Kümmerle et al., 2011a), occupancy-grid mapping, (Dhiman et al., 2014), and the connection of language to robot perception , (Walter et al., 2014), as noted in (Dellaert et al., 2017). Factor graphs have also been applied in the example domain of this thesis: pose estimation. (Desingh et al., 2019) use a factor graph representation and a variant of a message passing algorithm called Pull Message Passing algorithm for Nonparametric Belief Propagation in order to estimate the pose and state of articulated objects such as a drawer given noisy point clouds.

Uncertainty handling plays an important role for successful operation of robotic systems in real world applications and various approaches have been proposed. In the early work of (Su and Lee, 1992) a directed graph of transformations between different coordinated systems, such as robot end-effector and object, is build and associated with a covariance matrix to represent uncertainty. The tree allows to propagate uncertainties along kinematic chains. In addition, the Kalman filter is used for the fusion of different measurements. This underlying uncertainty handling based on a normal distributed variables is used to perform forward and backward propagation of actions to determine their applicability and success. This allows to take uncertainty into account for the planning of action sequences for manipulation. The approach does address pose uncertainty for gaussian uncertainty models, object classification errors are not covered. The approach for handling geometric uncertainty is still used, for instance in the work of Blumenthal et al. (2013).

Nyga et al. (2014) use a graphical model, Markov logic networks, in order to handle the uncertainty in object classification in a household robot scenario. An ensemble of expert algorithms is applied in order to symbolically annotate properties of different objects in a scene. A Markov logic network is trained based on a set of 50 ground truth annotated scenes to learn the dependency between object class and the symbolic annotations. At runtime, the Markov logic network can be used to infer the object class probability distribution based on the present symbolic annotations. Lutz et al. (2013) address the handling of classification uncertainty by fusing the results of multiple classification algorithms and by active repositioning of the sensor, see also (Stampfer et al., 2012) and (Lutz et al., 2012). Here, the uncertainty model is more basic than in Nyga et al. (2014), as it only contains a discrete object class probability distribution for each perception result and does not need to handle the diverse set of semantic annotations. But the active perception step addresses a different dimension of the perception system configuration with large potential for acquisition of reliable perception results.

Eidenberger and Scharinger (2010) handle pose uncertainty and classification uncertainty in a unified manner. They represent the inherent uncertainty using multivariate gaussian distributions and realize the active uncertainty reduction using a *partially observable markov decision process* (POMDP) as underlying model.

In summary, the proposed approach stands out with respect to the state-of-the-art by combining planning and factor-graph based uncertainty handling for the design synthesis of perception systems.

5.5 Summary and Discussion

In this chapter we introduced a novel method to model and plan the configuration of perception systems while taking into account geometric uncertainties across different coordinate frames. The main contribution of this chapter is **Contrib. C3**. On the uncertainty handling side, we introduce an approach to represent and infer geometric uncertainties across different coordinate frames using factor graphs. The approach allows to fuse multiple noisy measurements and take into account uncertainty of given poses, such as the extrinsic calibration of the camera as well as the positioning accuracy of a robot end-effector.

The factor graph construction and inference is represented using the hierarchical modeling formalism, which is introduced in Chapter 3. This shows on the one hand, that it is possible to represent the given problem sub-domain within the formalism. On the other hand it allows to use meta-models for configuration space exploration, such as planners.

In order to evaluate the uncertainty modeling, handling and configuration space exploration, an online perception system synthesis problem is formulated. Here, the configuration space is comprised of perception operators, robot movement as well as data fusion and inference. The synthesis addresses the selection of sensor input and perception algorithms, perception planning, and sensor fusion in an integrated yet modular fashion. We validated the approach in an industrial assembly scenario, where our planner successfully employs different sensors, data processing steps and view poses to localize the target part with sufficient accuracy, while keeping the required time as low as possible.

The benefits of the presented uncertainty modeling and perception system synthesis are of multiple origin. The factor graph representation allows to model and consider constraints of different types. For instance, the 2D bounding box observations can be seamlessly integrated, although they only pose a 2D constraint on the higher dimensional 3D pose representation. In addition, the factor graph can be composed individually depending on the task, as no static rule set is defined for its composition. It is rather up to the meta-operators for configurations space exploration to decide on a suitable factor graph composition. This enables automated adaptation to different settings and requirements.

The planning of elementary operations, such as the building of the factor graph comes at a cost: it increases the search space. Basic breath-first planning for configuration space exploration does not scale well with problems of increasing complexity. Therefore, we investigate hierarchical planning and the use of surrogate models within the model hierarchy as approach to improve the configuration space exploration in the following chapter. Furthermore, it is desirable to be able to arbitrarily switch between different uncertainty representations, such as particles and mixture models, while keeping a single factor graph. This is conceptually feasible, but requires further investigation.

Chapter 6

Hierarchical Planning for Perception System Synthesis

In the previous chapters, we showed how the modeling approach together with configuration space exploration operators from the meta-domain allows to synthesize perception system subsets. Parameter optimization allows to improve the system performance and find suitable pipeline structures and parameterizations, as shown in Chapter 4. However, the optimization is very time consuming and can only be applied for offline engineering. In Chapter 5 planning via breath-first search is employed as configuration space exploration method. Unfortunately, such a flat planning approach does not scale with problem complexity and does not provide means to look at a problem from different abstraction levels. Therefore, it can only be used for online adaptation if the planning domain is sufficiently small and well structured.

In this chapter we address online synthesis of perception systems by means of hierarchical planning. Here, a surrogate model of the perception system is encoded within the abstraction layers of a hierarchical model. A hierarchical planner allows to explore the configuration space in an efficient manner, which allows for online synthesis. This furthermore facilitates a model-predictive control scheme for perception system operation (Kast et al., 2019c). The system model is represented via the abstract layers of the model hierarchy and adaptive control is realized via regular goal comparison and backtracking.

A high level overview of the system is given in Fig. 6.1. The system is targeted primarily towards the online adaptation phase, where the perception pipeline and sensor

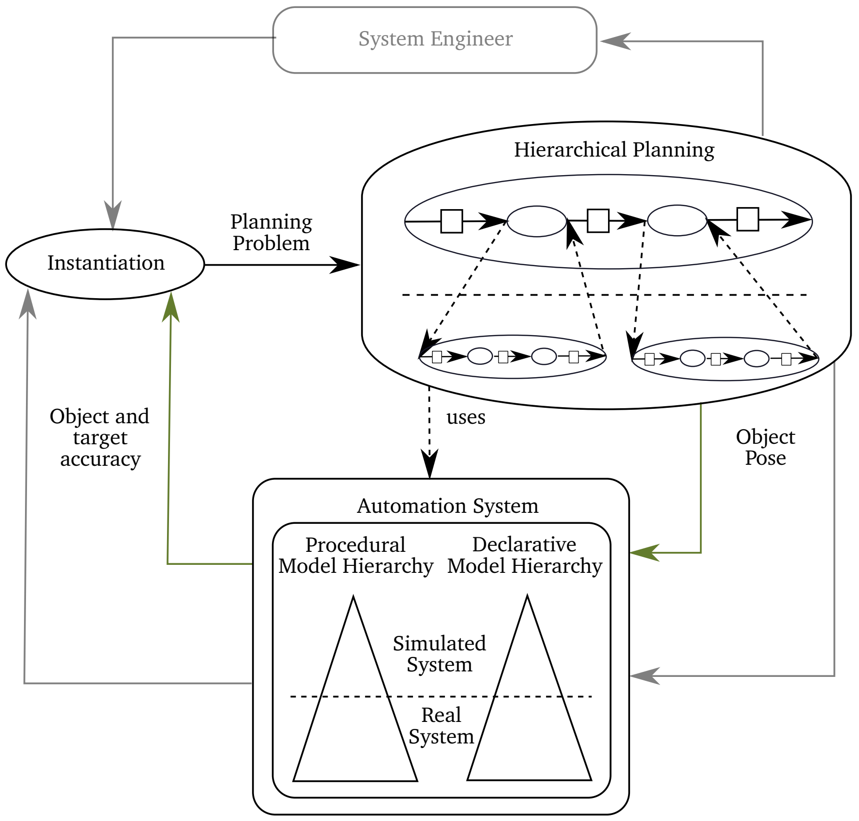


Figure 6.1: *Overview of the synthesis system for online adaptation via hierarchical planning*

viewpoint can be changed at runtime. However, the hierarchical model can be reused for offline engineering. Exploration and evaluation within the configuration space are handled by a hierarchical planner, which uses abstract and computationally cheap upper layers of the hierarchy in order to find promising solutions. Evaluation refers to the real execution of perceptual actions, which happens at the plan refinement at the lowest level of the hierarchy. A direct interaction with the real robotic system is possible.

The core contribution of this chapter is **Contrib. C4**. On the modeling side, we introduce a hierarchical metric model for perception data types, which enables the online quality estimation on different levels of abstraction. Furthermore, on the exploration side, we show how the hierarchical planner, an operator from the meta-domain, can be leveraged to provide fast configuration space exploration, which is suitable for online use.

The hierarchical approach is motivated by the manual procedure of a perception engineer. A perception engineer uses a combination of knowledge and experience in order to steer a trail-and-error procedure to find a suitable configuration. For instance, different algorithms have distinct output characteristics that additionally depend on the quality of the input data. The engineer implicitly knows these characteristics and uses this knowledge to choose promising pipelines. Additionally, a human engineer can perform a hierarchical task decomposition and start with high level decisions such as the sensor choice, while roughly estimating the expected performance given the sensor's data quality.

An overview of the hierarchical synthesis procedure is given in Fig. 6.2. The core is a hierarchical planning system which acts on a 2-layer hierarchical model with a knowledge layer and an execution layer. The knowledge layer needs model calibration which is performed with a ground truth annotated dataset of pipeline executions. This planning system is able to determine a pipeline that converts a given belief to a belief with a target quality specified by a given goal.

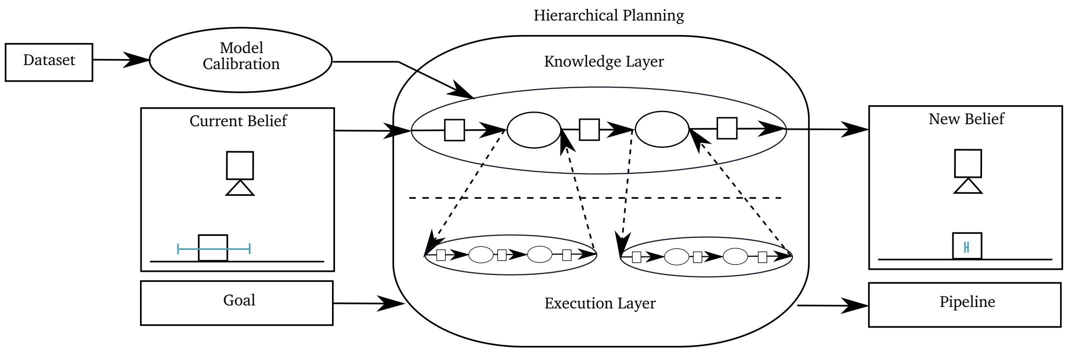


Figure 6.2: Schematic overview of the hierarchical perception system synthesis.

First, the hierarchical model is introduced, which is followed by an introduction of the model calibration and an exemplary planning sequence. Finally, the experimental setup and evaluations are presented and discussed.

This chapter is based on and integrates excerpts of the publication of Dietrich et al. (2020). It is extended by an analysis of the runtime in comparison to a non-hierarchical approach.

6.1 Hierarchical Task Formalization

Our approach is based on a hierarchical model combined with hierarchical planning and model calibration as depicted in Fig. 6.2. The hierarchical model is based on prior publications (Kast et al., 2019b) (Kast et al., 2019c) (Dietrich et al., 2019) and allows to model algorithms, denoted as *operators*, and instance classes, denoted as *concepts*, within the perception domain at different abstraction layers. This allows the hierarchical planner to search for solutions on cheap abstract domains and verify or refine the plans up to a real execution (Kast et al., 2019c). The model contains concepts such as pose, belief, point cloud and depth image. In this chapter we employ only two different layers, the *knowledge layer* and the *execution layer*. The former encodes symbolic knowledge and experience available for the domain. The latter provides the ability to actually execute the available perception algorithms on simulated or on real data.

6.1.1 Metric model

The key aspect is the systematic use and calibration of so-called *metrics* $m \in \mathbb{M}$. The metrics allow to assess properties, especially the quality of instances in a compact manner. E.g., when a perception engineer looks at a point cloud he indirectly assesses its quality and determines by experience which algorithm he should try next in order to get the desired result. The operators on the knowledge layer can be parameterized such that they encode this experience and model the relationship of the input metrics and output metrics. We summarize the metric model as follows:

- A metric is single valued
- A metric can be associated with a distribution
- Multiple metrics can be associated with an instance
- We differentiate between *computable* and *estimated* metrics.

Computable metrics can be calculated at runtime based on the available data, as for instance the size of a point cloud. Estimated metrics cannot be determined in an exact manner at runtime and need to be approximated. E.g., the positional distance with respect to ground truth is not known at pipeline runtime. Still, it has to be estimated if the planning goal is formulated as positional distance. The set of input and output metrics of an operator are denoted as $\mathbb{M}_{\mathrm{IN}}$ and $\mathbb{M}_{\mathrm{OUT}}$ respectively.

In order to work within the hierarchical modeling and planning approach, the metrics require representation on different levels of abstraction. The role of the metrics within the hierarchy is therefore shown exemplary in Fig. 6.3. A pose estimation operator on the knowledge as well as the execution layer is displayed alongside with input and output instances. Each instance is annotated with different metrics that describe the quality of the instance. On the knowledge layer these are mostly estimated metrics that approximate the actual value. On the execution layer actual values can be calculated for the some of the metrics. Due to uncertainties in data and execution the operators on the execution layer not always yield results as predicted on the knowledge layer. A *subset* relationship is defined for all concepts, which is used during planning in order to identify such deviations and perform backtracking if the relationship does not hold. Therefore, model inaccuracies at the knowledge layer can be handled.

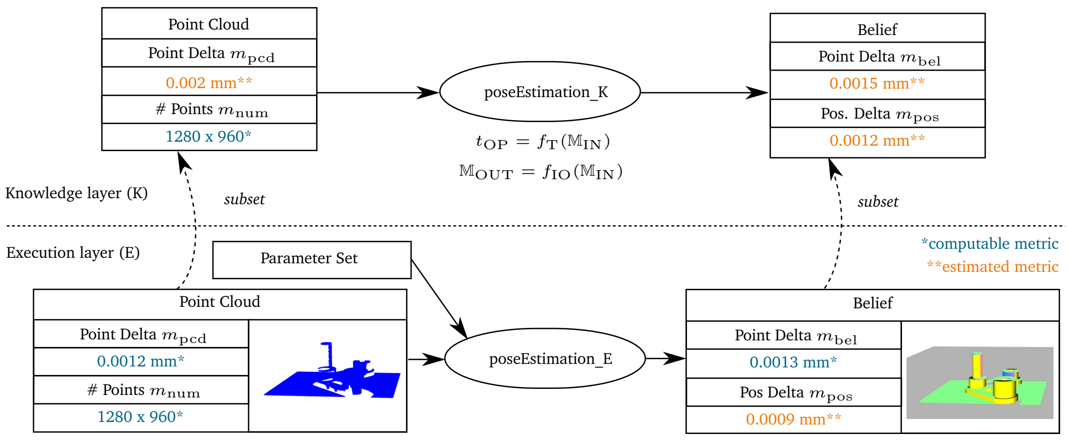

Point Cloud
Point Delta m_{pcd}
0.002 mm**
# Points m_{num}
1280 x 960*
poseEstimation_K
$t_{\text{OP}} = f_{\text{T}}(\mathbb{M}_{\text{IN}})$
$\mathbb{M}_{\text{OUT}} = f_{\text{IO}}(\mathbb{M}_{\text{IN}})$
Belief
Point Delta m_{bel}
0.0015 mm**
Pos. Delta m_{pos}
0.0012 mm**
Knowledge layer (K)
subset
subset
Execution layer (E)
Parameter Set
*computable metric
**estimated metric
Point Cloud
Point Delta m_{pcd}
0.0012 mm*
# Points m_{num}
1280 x 960*
poseEstimation_E
Belief
Point Delta m_{bel}
0.0013 mm*
Pos Delta m_{pos}
0.0009 mm**

Figure 6.3: Hierarchical view on a pose estimation operator on knowledge layer and execution layer. Detailed explanation in Sec. 6.1.

The approximations on the knowledge layer and for estimated metric on the execution layer is encoded in the functions f_{T} and f_{IO} that model the runtime and the relationship between inputs and outputs of the operator, see Fig. 6.3. Both can be arbitrary functions that require parameterization. This step is performed initially based on a dataset of already executed perception pipelines, cf. Fig. 6.2. In this work we use linear models for the transfer functions and linear regression for the calibration.

For the hierarchical planning we use the planner presented in (Kast et al., 2019c). The framework is designed such that different planning or search algorithms can be used at different layers. We use a scheduler at the knowledge level in order to find the fastest pipeline that fulfills the application requirements. On the execution layer breadth first search is used. On the knowledge layer no distinction between different operator parameterizations is done. The parameterization is chosen in the sub-problem planned within the execution layer. Additionally, only metrics are used during planning that can be calculated or estimated during runtime. Therefore, the presented system could also be used out of the box for online decision taking and active perception. However, this is not the focus in this chapter.

6.2 Evaluation

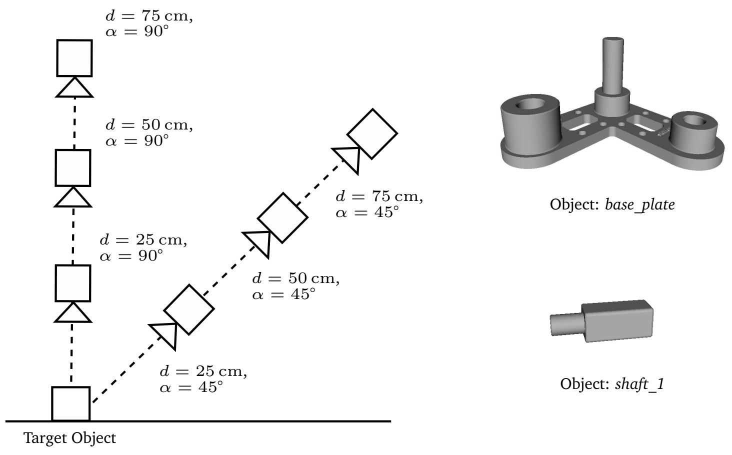

$d = 75$ cm, $\alpha = 90°$
$d = 50$ cm, $\alpha = 90°$
$d = 25$ cm, $\alpha = 90°$
$d = 75$ cm, $\alpha = 45°$
$d = 50$ cm, $\alpha = 45°$
$d = 25$ cm, $\alpha = 45°$
Target Object
Object: *base_plate*
Object: *shaft_1*

Figure 6.4: Left: Application setup with 6 different view poses. Right: Target objects from the Siemens Robot Learning Challenge.

We address two core questions regarding the presented approach:

- Is the model and the calibration approach suitable for the given application?
- Is the planning system capable of synthesizing working pipelines in a reasonable amount of time?

6.2.1 Setup

The target application is an assembly scenario, where the robot perception system should give accurate 6D poses of the objects *base_plate* and *shaft_1* from the Siemens Robot Learning Challenge[1], see Fig. 6.4. The setup is simulated and two different depth cameras are used, a Intel D435 consumer sensor equivalent and a Photoneo PhoXi M industrial sensor equivalent. Distance dependent longitudinal noise is added based on camera characteristics according to manufacturer data [2]. Additionally, the sensors are placed at six different poses in order to cover a range of different hardware setups, see Fig. 6.4.

We used the following computable metrics on the execution layer:

[1]https://new.siemens.com/us/en/company/fairs-events/robot-learning.html

[2]https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/BKMs_Tuning_RealSense_D4xx_Cam.pdf and http://wiki.photoneo.com/index.php/PhoXi_3D_scanners_family

Table 6.1: Overview of used operators and their model calibration.[3]

Operator	**Comment**	**# Param.**	**Sets**	f_{T}	f_{IO}
getPointCloud*	point cloud rendering	0	-	0.030 s	$d \rightarrow m_{\mathrm{pcd}}$
prepareModel	create object point cloud	1	1	0.050 s	-
removePlane	plane estimation and removal	3	1	0.013 s	$m_{\mathrm{num}} \rightarrow m_{\mathrm{num}}$
smoothTSDF	using Signed-Distance Functions	2	1	0.288 s	$\begin{pmatrix} m_{\mathrm{num}} \\ m_{\mathrm{pcd}} \end{pmatrix} \rightarrow \begin{pmatrix} m_{\mathrm{num}} \\ m_{\mathrm{pcd}} \end{pmatrix}$
poseEstimationFPFH	using Fast Point Feature Histograms	8	2	$m_{\mathrm{num}} \rightarrow t$	$m_{\mathrm{pcd}} \rightarrow m_{\mathrm{bel}}$
refineICP	Iterative Closest Point	2	2	$m_{\mathrm{num}} \rightarrow t$	$m_{\mathrm{bel}} \rightarrow m_{\mathrm{bel}}$
refinePhysics*	achieve physical plausibility with plane	6	1	2.060 s	$m_{\mathrm{bel}} \rightarrow m_{\mathrm{bel}}$

- m_{num}: Number of points of the point cloud
- m_{pcd}: average distance between points that can be a associated with known objects in the scene, such as a plane, and those objects
- m_{bel}: average distance between a depth rendering of the object estimate and the point cloud

The position distance m_{pos} between belief and ground truth cannot be computed at runtime and has to be estimated. The mapping between m_{bel} and m_{pos} is therefore as well approximated with a linear model during the model calibration. The available operator set within the experiment is summarized in Tab. 6.1.

The model calibration is performed on a dataset of an exhaustive exploration of pipelines on 2 scenes, 2 cameras and 6 viewpoints of the object *base_plate* . The result of the calibration are parameterized models of the runtime behavior f_{T} and the input to output relation f_{IO}. In this work we use single-input, single-output linear models, but the approach supports models of arbitrary complexity and arbitrary number of inputs and outputs. The choice of input and output types for the models has been made by manual examination of the correlation. The relationship between given and ground truth metrics is calibrated in addition to the operator behavior. In Tab. 6.1 the calibrated input-output relationships for the different operators are listed. Approximately constant values are directly shown. In the first row of Fig. 6.5 actual linear approximations for runtime and input-output relationship are displayed.

[3]Operators marked with * are implemented using Bullet (`https://pybullet.org/`), otherwise using Open3D (`http://www.open3d.org/`)

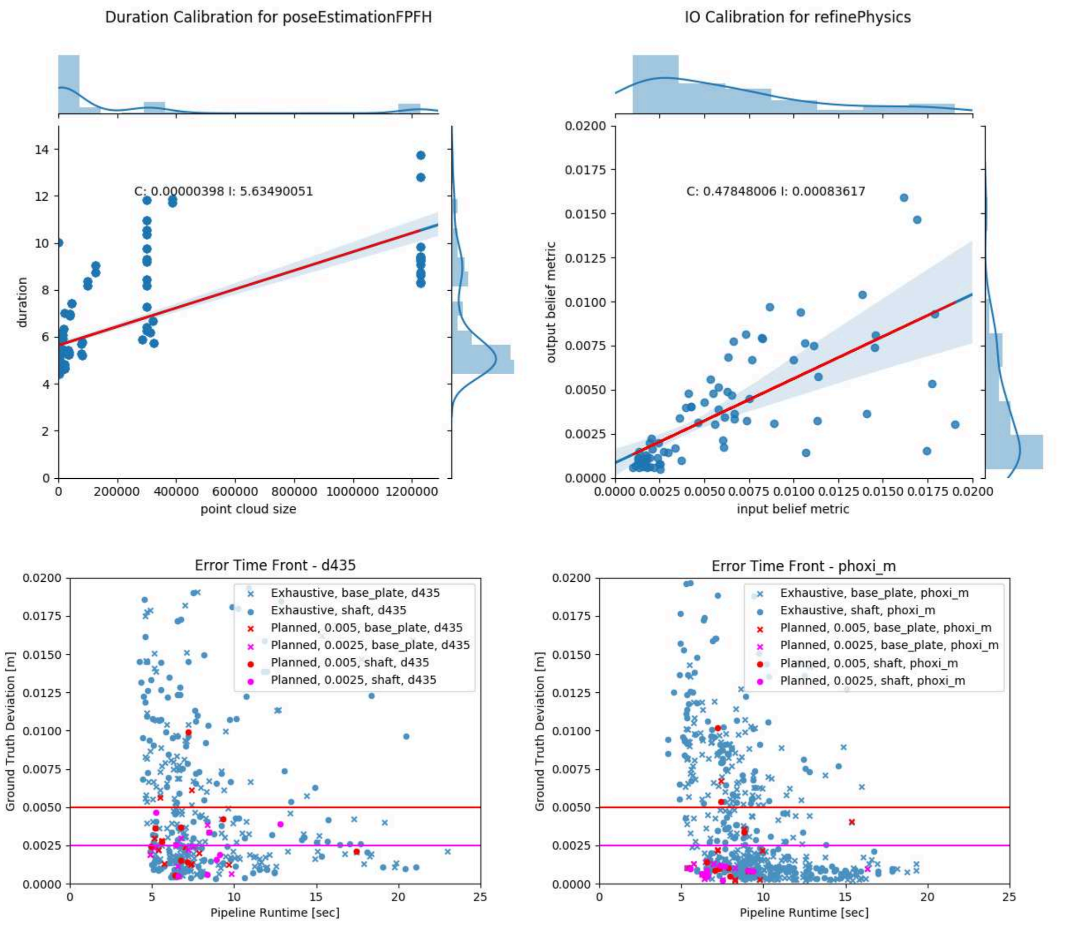


Figure 6.5: Top left: Linear model fit for runtime depending on input cloud size for poseEstimationFPFH. Top right: Linear model fit for the output m_{bel} depending on the input m_{bel}. Bottom left: Comparison between baseline and planned pipelines for sensor D435. Bottom right: Comparison between baseline and planned pipelines for sensor Phoxi M.

6.2.2 Results

In order to assess the approach we compare the planning results with the results of a greedy search, as shown in the bottom row of Fig. 6.5. The calibration has been performed only on *base_plate* and results are displayed for both objects. Furthermore, the planning was performed with two different goal settings: $0.005\,\mathrm{m}$ and $0.0025\,\mathrm{m}$ as indicated by the colors and the horizontal lines.

We can make a few observations. First, longer runtimes lead to lower errors, which is coherent with engineering intuition. Although this is not a general rule. For instance, the removal of the plane can lead to higher accuracy and shorter runtime due to a lower number of points for the pose estimation. Furthermore, we can observe that the ground truth of the majority of the planned results fulfills the goal requirements.

During planning only runtime metrics are used, which can only approximate the ground truth. Therefore, the result may sometimes deviate from the ground truth due to uncertainty and insufficient calibration. The goal is reached with high success rates for the calibration object as well as the unseen object, which indicates that the model generalizes and is sufficiently complex. Most of the outliers are associated with the unseen and non calibrated *shaft_1* object, which conforms with our expectations. A further observation regards the runtime of the planned pipelines. On the knowledge level a scheduler is used in order to find the fastest pipelines that reach the goal. The experimental data shows that the planning system is actually capable of identifying a suitable compromise between speed and accuracy. This is achieved with a median planning time of about $51\,\mathrm{s}$. Finally, we compare the results between the consumer camera and the industrial camera. The more expensive industrial camera can achieve higher accuracy overall. But while there are no outliers for $2.5\,\mathrm{mm}$ the model still leaves room for improvement as the outliers for a goal of $5\,\mathrm{mm}$ imply.

In summary, both questions formulated at the beginning of this section can be generally answered positively. However, there is still potential within further investigation to improve the model accuracy and decrease the synthesis times for actual use for active perception.

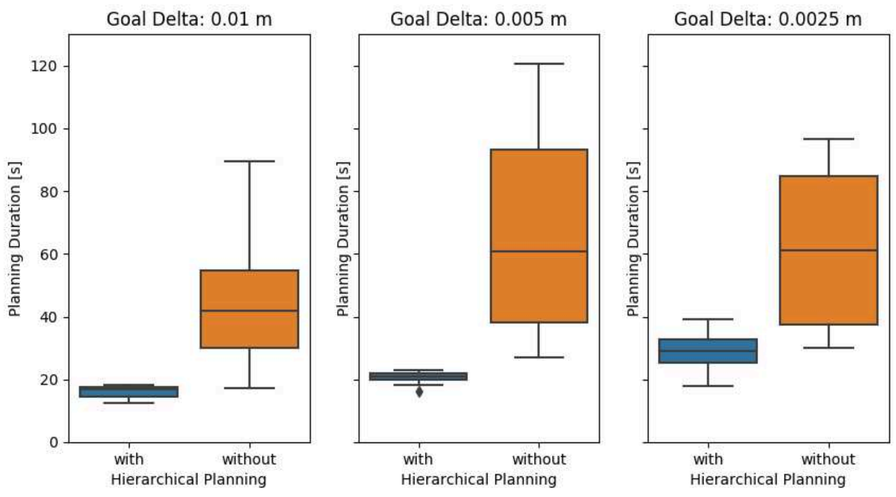


Figure 6.6: *Runtime comparison between hierarchical and non-hierarchical planning for different planning goals generated with the sensor Phoxi M and the object base_plate . The hierarchical approach clearly outperforms the non-hierarchical planning being more than twice as fast. It can also be seen that the planning time generally raises with a more difficult planning goal.*

We furthermore compare the hierarchical approach with non-hierarchical planning. In Fig. 6.6 a comparison of the planning duration for different target accuracies is given. The planning problem is a subset of the overall setup with the object *base_plate* and the sensor Phoxi M at a distance of $25\,\mathrm{cm}$. The more constrained setup allows to isolate and compare the effects of the planning algorithm. The planning is performed with 5 different seeds for each target accuracy.

Clearly, the hierarchical planning outperforms the planning without a knowledge layer. The average planning duration is less the half the time of the non-hierarchical planning. Furthermore, the planning time is increasing with higher accuracy requirements. This is mainly due to the fact that the planning includes the execution of the real operators on real data within the execution layer. Higher accuracy requires longer pipelines in the application setup and therefore a higher overall duration.

6.3 Related Work

In the computer vision community, the problem of automatic configuration or synthesis has a long standing history. For instance, Radig et al. (1992) proposed a toolbox in 1992 for automated design of image understanding systems that uses the *FIGURE* system for knowledge based synthesis of pipelines by Messer (1992). The approach is based in logical rules and descriptions, which are used to infer suitable pipelines. The more recent work of Nagato and Koezuka (2016) applies genetic programming and hierarchical program structuring in order to quickly adapt an image processing pipeline to a changing production environment. The approach was successfully demonstrated in a real production environment. Irgenfried et al. (2017) address the automation of the design of entire inspection systems using accurate sensor simulation and uncertainty quantification. Another notable approach is the *RoboSherlock* framework by Beetz et al. (2015a) which leverages unstructured information management and ontologies in order to generate perception pipelines based on semantic queries. Our own prior work includes automatic configuration of perception systems using single level planning and factor graphs for uncertainty representation (Dietrich et al., 2018) and the joint optimization of pipelines and parameters (Dietrich et al., 2019).

This work is grounded in robotics research and builds upon a hierarchical modeling and planning system that has successfully been demonstrated for task and motion planning (Kast et al., 2019c). This allows a tight integration between task planning and perception planning. Additionally, the target domain is not restricted to computer vision. General sensor fusion and state estimation algorithms, for instance using physics simulation, are targeted as well. Furthermore, a condensed model is trained that encodes experience and can be used for relatively fast planning of pipelines.

6.4 Summary and Discussion

In this chapter, we presented an approach to synthesize perception pipelines using hierarchical planning, which allows to find a sequence of operators with sufficient accuracy for the task at hand. The hierarchical model consists of a knowledge layer, which encodes engineering experience from data and steers the search procedure, and an execution layer where actual operators are executed on the given data.

Scheduling on the top level ensures that pipelines with short runtime are preferred. Experiments in simulation with different objects, sensor types, and sensor placements show promising results. The knowledge layer is calibrated for one object and performs well for a unseen test object. The duration of the synthesis process, including execution, is mostly less than a minute and therefore allows rapid adaptation, e.g., for a product change.

The core contribution of this chapter is **Contrib. C4**. The modeling approach for data types using metrics allows to build a model hierarchy, which can be exploited using hierarchical planning. This allows to factorize the search space and provide more efficient configuration space exploration than a flat planning approach. The approach is validated in an industrial assembly scenario for multiple objects and sensors.

Different aspects have to be discussed critically. First, the presented abstract models are only basic and should be interpreted as an example of how to approach the task. However, more complex models can easily be incorporated. Also, the model calibration represents a general issue, as the quality depends on the available data and its distribution. Errors can occur due to wrong interpolations as well as extrapolations of existing data points. Here, systematic simulated data generation and distribution are a promising avenue of future work. Furthermore, only a single abstraction layer is addressed in the current example. However, the approach conceptually handles well problem formulations with higher numbers of layers, which could be investigated in future work.

Chapter 7

Conclusion

In this thesis we automated substantial parts of the synthesis of perception systems for engineering and runtime adaptation. The difficulty stems, among others, from the large space of system configurations, the task variability, immanent noise, and a high sensitivity to small deviations in task and setup. In addition, the failure tolerance is very low in industrial robotic applications, as errors can lead to defective products or even collisions. As foundation we introduced a hierarchical model as well as different configuration space exploration methods, which are acting on the hierarchical model. The modeling approach allows to represent the task, procedural models, and declarative models at different levels of abstraction. Based on this common model, we showed, that pipeline structure and parameterization can be jointly optimized using pipeline structure templates. The approach is successfully validated in a real-world assembly setup and allows to remove a large portion of the engineering effort, when adapting an automation system to a new task. Furthermore, and again based on the common model, we demonstrate how explicit uncertainty representations can be leveraged for the synthesis process and that uncertainty models can be constructed in an automated manner. Finally, we exploit the abstraction hierarchy in order to provide online adaptation of perception systems to a new task at runtime, which addresses the need for flexible automation system.

In the following, the individual contributions to the automation of the engineering design process are summarized and discussed. In Chapter 3 we introduce a hierarchical modeling formalism and apply it to the domain of perception synthesis, as summarized in **Contribution C1**. It allows to define procedural and declarative knowledge in a programming language agnostic way and on different levels of abstraction. The hierarchical modeling formalism is grounded in set-theory and allows to apply hierarchical planning as configuration space exploration method. Furthermore, additional methods for the exploration of the configuration space, such as optimization and planning can be represented as meta-operators within the presented formalism. This allows to apply suitable exploration methods depending on domain, task, and engineering phase. Finally, methods to automatically generate models from different origin are briefly highlighted, including an approach to use natural language processing for the interpretation of existing code documentation.

Once a common model of the space of system configurations is given, it is required to search the configuration space in order to satisfy the needs of the application. This is part of the offline engineering phase, where the system is still under development and time is available in order to evaluate the different options. Therefore, we introduce in Chapter 4 an automated design synthesis approach based on pipeline templates and sequential model-based optimization which allows to jointly optimize the structure and parameterization of perception pipelines for the offline design synthesis of perception system, as noted in **Contribution C2**. Prior engineering knowledge about pipeline structures can be encoded in pipeline templates, which allows to substantially reduce the search space. Model-based operators, e.g., using point-pair-features, and data-based operators, using neural networks, are automatically combined. This allows to improve the performance with respect to individual operators. A state-of-the-art deep learning approach for instance did not perform sufficiently well when not combined with suitable operators for the refinement of object hypotheses. The design synthesis algorithm is represented in the common modeling formalism as a meta-operator, which facilitates the reuse and combination with different methods. The design synthesis is successfully applied in a flexible assembly setup, where the perception system performance is substantially improved with respect to a baseline of typical pipeline structures with optimized parameters.

As uncertainty and noise is immanent in the data generation and processing of perception systems, explicit modeling and handling thereof is beneficial. A perception engineer for instance knows uncertainty characteristics and takes them into account in the design synthesis process. Especially geometric uncertainties play a crucial role for assembly tasks. Therefore, in Chapter 5 we introduce an approach to model and estimate geometric uncertainties across different observations and coordinate systems by constructing and inferring over a factor graph, as highlighted in **Contribution C3**. The construction and inference of this probabilistic graphical model is formulated in a planning problem, which allows to jointly consider perceptual actions and data fusion. The formulation as a planning problem is facilitated by using the introduced modeling formalism. The approach is validated in a simulated assembly scenario, where the goal is to satisfy the requirements on the geometric localization accuracy.

It is shown that the automated planning allows to find suitable perception and data fusion pipelines using the proposed uncertainty model. The formalization as planning problem allows for general adaptability to new tasks, may however induce additional overhead.

Flexible and autonomous assembly systems need to adapt quickly to new products and even new situations. A static perception system design often does not work sufficiently well for such dynamic requirements. In order to overcome this, we introduce in Chapter 6 an approach to leverage the presented hierarchical models for online design synthesis based on hierarchical planning, see also **Contribution C4**. Here an abstract layer that approximates the behavior of real perception operators is used as a surrogate in order to search for promising pipeline candidates. Once a candidate is found, reduced planning problems are formulated using the real operators to fulfill the intermediate goals of the top-level problem. This hierarchical approach allows to reduce the planning time and copes with the curse of dimensionality. The parameterization of the abstract model layer is calibrated using a log of real operator execution. The approach is validated in a simulated assembly scenario, with different cameras, camera placements, and objects. We could show that a successful pipeline generation is possible within reasonable amounts of time, suitable for online adaptation. Furthermore, the hierarchical approach clearly outperforms a non-hierarchical baseline in terms of planning times.

However, the overall task of automating the entire design of perception systems without human intervention is still subject to challenges. The presented approaches focused primarily on the pipeline design and parameterization for online and offline use. Here, successful design synthesis could be demonstrated, but not all dimensions of the configuration space were open to change. Additional layers in the hierarchical configuration system, for instance, are required in order to cover different aspects such as system cost, hardware integration and deployment. This is conceptually feasible in the presented modeling approach, but requires further research towards individual model structure and scalability, especially due to the increased dimensionality of the configuration space. Furthermore, the availability and integration effort of perception operators and models remains a challenge. Every existing operator and sensor should provide standardized interfaces, abstract and detailed execution models to approximate and test its behavior in different settings. Ideally, these should be available globally, which requires suitable and scalable platforms. The presented approaches for operator harvesting demonstrate that automation is possible, but still require a manual support in understanding the domain. Additionally, a knowledge base is required to store the experiences and different perception system configurations. Here, the transfer of knowledge of the design synthesis of a single perception system to another application is an interesting avenue of future work. Finally, the use of photo-realistic simulation for the data generation is a strong enabler for automated design synthesis and we use simulation operators within the design synthesis process. However, simulation represents an abstract model which approximates reality and there is still ongoing research demand on bridging the simulation-reality gap, depending on the use case.

In summary, we showed that an automation of typical engineering tasks involved with the design of perception systems is possible and made substantial contributions to different sub tasks, especially in the area of pipeline design and parameterization for online and offline use. The contributions include a common model in a hierarchical modeling formalism, uncertainty representation and handling, offline optimization of pipeline structure and parameterization, and online system adaptation via hierarchical planning.

Glossary

concept Basic declarative model, see definition in Sec. 3.1.1.

configuration space The space of different system configurations, see also Def. 5 and Fig. 1.9

configuration space exploration Process of exploring different configurations in the configurations space in order to solve given requirements. See also Sec. 3.4

declarative model See definition in Sec. 2.1.1

design synthesis See Def. 4 and Sec. 2.2.3

instance Actual element within the set of a concept, see also Sec. 3.1.1.

meta domain Top level models, that contains high level concepts such as operator.

offline engineering phase Phase in the design process, where the target system is not yet in operation and time can be spent to evaluate different design choices. See also Sec. 4.

online adaptation phase Phase in the design process, where the target system is already in operation and has to adapt in short time to a new product, task or situation. See also Sec. 2.2.3.6.

operator Basic procedural model, see definition in Sec. 3.1.2.

procedural model See definition in Sec. 2.1.1

RGB Denotes the color spectrum of a camera or image, which is comprised of the colors red, green and blue.

RGBD Denotes the spectrum of a camera or image, which is comprised of the colors red, green and blue and a depth channel. See also RGB.

Bibliography

W. Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. `https://github.com/matterport/Mask_RCNN`, 2017. [Online; accessed 03-March-2021].

M. Aeberhard and N. Kaempchen. High-level sensor data fusion architecture for vehicle surround environment perception. In *Workshop Int. Conf. on Intelligent Transportation Systems*. IEEE, 2011.

F. Bacchus and Q. Yang. The downward refinement property. In *Int. Joint Conf. on Artificial Intelligence*, pages 286–293, 1991.

F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. In *Artificial Intelligence*, volume 71, pages 43–100. Elsevier, 1994.

R. Bajcsy. Active perception. In *Proc. of the IEEE*, volume 76, pages 966–1005. IEEE, 1988.

A. Balazs. International vocabulary of metrology - basic and general concepts and associated terms. In *Chemistry Int.*, 2008.

F. Bálint-Benczédi, J.-H. Worch, D. Nyga, N. Blodow, P. Mania, Z.-C. Márton, and M. Beetz. Robosherlock: Cognition-enabled robot perception for everyday manipulation tasks. In *arXiv preprint arXiv:1911.10079*, 2019.

T. D. Barfoot. *State estimation for robotics*. Cambridge University Press, 2017.

T. Bayes. An essay towards solving a problem in the doctrine of chances. In *Philosophical Trans. of the Royal Society of London*, number 53, pages 370–418. The Royal Society London, 1763.

D. M. Beazley et al. Swig: An easy to use tool for integrating scripting languages with c and c++. In *Tcl/Tk Workshop*, volume 43, page 74, 1996.

P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal. HiPOP: Hierarchical partial-order planning. In *Europ. Starting AI Researcher Symposium*, pages 51–60, 2014.

M. Beetz, F. Bálint-Benczédi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z.-C. Márton. RoboSherlock: Unstructured information processing for robot perception. In *Int. Conf. on Robotics and Automation*, pages 1549–1556. IEEE, 2015a.

M. Beetz, M. Tenorth, and J. Winkler. Open-EASE. In *Int. Conf. on Robotics and Automation*, pages 1983–1990. IEEE, 2015b.

M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, and G. Bartels. Know rob 2.0—a 2nd generation knowledge processing framework for cognition-enabled robotic agents. In *Int. Conf. on Robotics and Automation*. IEEE, 2018.

P. Bercher, D. Höller, G. Behnke, and S. Biundo. More than a name? on implications of preconditions and effects of compound htn planning tasks. In *Europ. Conf. on Artificial Intelligence*, pages 225–233, 2016.

J. Bergstra. Hyperopt: distributed asynchronous hyper-parameter optimization. `https://github.com/hyperopt/hyperopt`, 2016. [Online; accessed 03-March-2021].

C. M. Bishop. *Pattern recognition and machine learning*. Springer Science+ Business Media, 2006.

T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Int. MODELICA Conf.*, number 076. Linköping University Electronic Press, 2012.

S. Blumenthal, H. Bruyninckx, W. Nowak, and E. Prassler. A scene graph based shared 3d world model for robotic applications. In *Int. Conf. on Robotics and Automation*, pages 453–460. IEEE, 2013.

Boston Consulting Group. The Shifting Economics of Global Manufacturing. `https://de.slideshare.net/TheBostonConsultingGroup/robotics-in-manufacturing`, 2015. [Online; accessed 20-February-2020].

T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, et al. Extensible markup language (xml) 1.0, 2000.

C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping:

Toward the robust-perception age. In *Trans. on Robotics*, volume 32, pages 1309–1332. IEEE, 2016.

M. Cashmore, M. Fox, D. Long, and D. Magazzeni. A compilation of the full PDDL+ language into SMT. In *AAAI Workshop: Planning for Hybrid Systems*, 2016.

L. Castillo, J. Fernández-Olivares, and A. Gonzalez. Integrating hierarchical and conditional planning techniques into a software design process for automated manufacturing. In *Int. Conf. on Automated Planning and Scheduling*. AAAI, 2003.

L. Castillo, J. Fernández-Olivares, O. Garcia-Perez, and F. Palao. Efficiently handling temporal knowledge in an htn planner. In *Int. Conf. on Automated Planning and Scheduling*, pages 63–72. AAAI, 2006.

D. Chen, V. Dietrich, and G. von Wichert. Precision grasping based on probabilistic models of unknown objects. In *Int. Conf. on Robotics and Automation*. IEEE, 2016.

D. Chen, V. Dietrich, Z. Liu, and G. Von Wichert. A probabilistic framework for uncertainty-aware high-accuracy precision grasping of unknown objects. In *J. of Intelligent & Robotic Systems*, volume 90. Springer, 2018.

P. Corke. *Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised*, volume 118. Springer, 2017.

E. Courteille, D. Deblaise, and P. Maurine. Design optimization of a delta-like parallel robot through global stiffness performance evaluation. In *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2009.

G. Csurka. A comprehensive survey on domain adaptation for visual applications. In *Domain adaptation in computer vision applications*, pages 1–35. Springer, 2017.

R. Davis, H. Shrobe, and P. Szolovits. What is a knowledge representation? In *AI Magazine*, volume 14, pages 17–17, 1993.

P. Del Moral. Non-linear filtering: interacting particle resolution. In *Markov Processes and Related Fields*, volume 2, pages 555–581. Citeseer, 1996.

F. Dellaert. Factor graphs and GTSAM: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

F. Dellaert, M. Kaess, et al. Factor graphs for robot perception. In *Foundations and Trends in Robotics*, volume 6, pages 1–139. Now Publishers, Inc., 2017.

K. Desingh, S. Lu, A. Opipari, and O. C. Jenkins. Factored pose estimation of articulated objects using efficient nonparametric belief propagation. In *Int. Conf. on Robotics and Automation*, pages 7221–7227. IEEE, 2019.

V. Dhiman, A. Kundu, F. Dellaert, and J. J. Corso. Modern map inference methods for accurate and fast occupancy grid mapping on higher order factor graphs. In *Int. Conf. on Robotics and Automation*, pages 2037–2044. IEEE, 2014.

M. Diab, A. Akbari, M. Ud Din, and J. Rosell. PMK—A knowledge processing framework for autonomous robotics perception and manipulation. In *Sensors*, volume 19, page 1166. Multidisciplinary Digital Publishing Institute, 2019.

V. Dietrich, D. Chen, K. M. Wurm, G. v. Wichert, and P. Ennen. Probabilistic multi-sensor fusion based on signed distance functions. In *Int. Conf. on Robotics and Automation*. IEEE, 2016.

V. Dietrich, B. Kast, P. Schmitt, S. Albrecht, M. Fiegert, W. Feiten, and M. Beetz. Configuration of perception systems via planning over factor graphs. In *Int. Conf. on Robotics and Automation*, pages 1–7. IEEE, 2018.

V. Dietrich, B. Kast, M. Fiegert, S. Albrecht, and M. Beetz. Automatic configuration of the structure and parameterization of perception pipelines. In *Int. Conf. on Advanced Robotics*. IEEE, 2019.

V. Dietrich, B. Kast, S. Albrecht, and M. Beetz. Data-driven synthesis of perception pipelines via hierarchical planning. In *Int. Conf. on Robotics in Alpe-Adria Danube Region*. Springer, 2020.

R. Drath, A. Luder, J. Peschke, and L. Hundt. Automationml - the glue for seamless automation engineering. In *Int. Conf. on Emerging Technologies and Factory Automation*, pages 616–623. IEEE, 2008.

M. Durner, S. Kriegel, S. Riedel, M. Brucker, Z.-C. Márton, F. Bálint-Benczédi, and R. Triebel. Experience-based optimization of robotic perception. In *Int. Conf. on Advanced Robotics*, pages 32–39. IEEE, 2017.

R. Eidenberger and J. Scharinger. Active perception and scene modeling by planning with probabilistic 6d object poses. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1036–1043. IEEE, 2010.

J. Elfring, S. van den Dries, M. Van De Molengraft, and M. Steinbuch. Semantic world modeling using probabilistic multiple hypothesis anchoring. In *Robotics and Autonomous Systems*, volume 61, pages 95–105. Elsevier, 2013.

J. Elfring, R. van de Molengraft, and M. Steinbuch. Semi-task-dependent and uncertainty-driven world model maintenance. In *Autonomous Robots*, volume 38, pages 1–15. Springer, 2015.

T. Elsken, J. H. Metzen, F. Hutter, et al. Neural architecture search: A survey. In *J. of Machine Learning Research*, volume 20, pages 1–21, 2019.

EMVA. GenICam. `https://www.emva.org/standards-technology/genicam/`, 2021. [Online; accessed 03-March-2021].

K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, volume 94, pages 1123–1128, 1994.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Int. Conf. on Machine Learning*, pages 1437–1446. PMLR, 2018.

W. Feiten, P. Atwal, R. Eidenberger, and T. Grundmann. 6d pose uncertainty in robotic perception. In *Advances in Robotics Research*, pages 89–98. Springer, 2009.

D. Ferrucci and A. Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. In *Natural Language Engineering*, volume 10, pages 327–348. Cambridge University Press, 2004.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970. 2015.

R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Artificial Intelligence*, volume 2, pages 189–208. Elsevier, 1971.

S. R. Fiorini, J. Bermejo-Alonso, P. Gonçalves, E. P. de Freitas, A. O. Alarcos, J. I. Olszewska, E. Prestes, C. Schlenoff, S. V. Ragavan, S. Redfield, et al. A suite of ontologies for robotics and automation. In *Robotics & Automation Magazine*, volume 24, pages 8–11. IEEE, 2017.

M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Communications of the ACM*, volume 24, pages 381–395. ACM New York, 1981.

T. Foote. tf: The transform library. In *Conf. on Technologies for Practical Robot Applications*, pages 1–6. IEEE, 2013.

M. Fox and D. Long. Pddl+: Modeling continuous time dependent effects. In *Int. NASA Workshop on Planning and Scheduling for Space*, volume 4, 2002.

M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. In *Artificial Intelligence Research*, volume 20, pages 61–124, 2003.

B. J. Frey, N. Mohammad, Q. D. Morris, W. Zhang, M. D. Robinson, S. Mnaimneh, R. Chang, Q. Pan, E. Sat, J. Rossant, et al. Genome-wide analysis of mouse transcripts using exon microarrays and factor graphs. In *Nature genetics*, volume 37, page 991. Nature Publishing Group, 2005.

A. Fritze, U. Mönks, C.-A. Holst, and V. Lohweg. An approach to automated fusion system design and adaptation. In *Sensors*, volume 17, page 601. Multidisciplinary Digital Publishing Institute, 2017.

J. Fuchs, D. Söhnlein, and B. Weber. Projektion des Erwerbspersonenpotenzials bis 2060: Arbeitskräfteangebot sinkt auch bei hoher Zuwanderung. Technical report, IAB-Kurzbericht, 2017.

M. Galanis, V. Dietrich, B. Kast, and M. Fiegert. Rtfm: Towards understanding source code using natural language processing. In *Int. Conf. on Informatics in Control, Automation and Robotics*, 2020.

D. Gallup, J.-M. Frahm, P. Mordohai, and M. Pollefeys. Variable baseline/resolution stereo. In *Conf. on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

T. Gateau, C. Lesire, and M. Barbier. Hidden: Cooperative plan execution and repair for heterogeneous robots in dynamic environments. In *Int. Conf. on Intelligent Robots and Systems*, pages 4790–4795. IEEE, 2013.

R. Goldman. Durative planning in HTNs. In *Int. Conf. on Automated Planning and Scheduling*, pages 382–385. AAAI, 2006.

E. M. N. Gonçalves, A. Freitas, and S. Botelho. An automationml based ontology for sensor fusion in industrial plants. In *Sensors*, volume 19, page 1311. Multidisciplinary Digital Publishing Institute, 2019.

S. Govindaraj, J. Gancet, M. Post, R. Dominguez, and F. Souvannavong. *InFuse: A Comprehensive Framework for Data Fusion in Space Robotics*. Infinite Study, 2017.

G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. In *Intelligent Transportation Systems Magazine*, volume 2, pages 31–43. IEEE, 2010.

T. R. Gruber et al. A translation approach to portable ontology specifications. In *Knowledge Acquisition*, volume 5, pages 199–221. Academic Press, 1993.

A. Gupta and S. K. Arora. *Industrial automation and robotics*. Laxmi Publications, 2009.

N. Hafez, V. Dietrich, and M. Zwick. Probabilistic orientation resolution for near symmetrical objects using depth images. In *Int. Conf. on Robotics in Alpe-Adria Danube Region*, pages 479–487. Springer, 2019.

N. Hafez, V. Dietrich, and S. Roehrl. Analysis of different methods to close the reality gap for instance segmentation in a flexible assembly cell. In *Int. Conf. on Robotics in Alpe-Adria Danube Region*, pages 479–487. Springer, 2020.

F. Hagelskjær, A. G. Buch, and N. Krüger. Does vision work well enough for industry? In *Int. Joint Conf. on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pages 198–205, 2018.

A. Haidu, D. Beßler, A. Bozcuoglu, and M. Beetz. KNOWROB-SIM: game engine-enabled knowledge processing for cognition-enabled robot control. In *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2018.

D. Hall, F. Dayoub, J. Skinner, H. Zhang, D. Miller, P. Corke, G. Carneiro, A. Angelova, and N. Sünderhauf. Probabilistic object detection: Definition and evaluation. In *Winter Conf. on Applications of Computer Vision*. IEEE, 2020.

R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

M. Hause et al. The sysml modelling language. In *Europ. Systems Engineering Conference*, volume 9, pages 1–12, 2006.

M. Helmert. The fast downward planning system. In *J. of Artificial Intelligence Research*, volume 26, pages 191–246, 2006.

S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *Int. Conf. on Computer Vision*, pages 858–865. IEEE, 2011.

S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conf. on Computer Vision*. Springer, 2012.

S. Hinterstoisser, O. Pauly, H. Heibel, M. Martina, and M. Bokeloh. An annotation saved is an annotation earned: Using fully synthetic training for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

N. Hochgeschwender, S. Schneider, H. Voos, and G. K. Kraetzschmar. Towards a robot perception specification language. In *4th Int. Workshop on Domain-Specific Languages and Models for Robotic systems*, 2013.

N. Hochgeschwender, S. Schneider, H. Voos, and G. K. Kraetzschmar. Declarative specification of robot perception architectures. In *Int. Conf. on Simulation, Modeling, and Programming for Autonomous Robots*, pages 291–302. Springer, 2014.

N. Hochgeschwender, M. A. Olivares-Mendez, H. Voos, and G. K. Kraetzschmar. Context-based selection and execution of robot perception graphs. In *Conf. on Emerging Technologies & Factory Automation*, pages 1–4. IEEE, 2015.

T. Hodaň, J. Matas, and Š. Obdržálek. On evaluation of 6d object pose estimation. In *Europ. Conf. on Computer Vision*. Springer, 2016.

T. Hodan, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis. T-LESS: An RGB-D dataset for 6d pose estimation of texture-less objects. In *Winter Conf. on Applications of Computer Vision*, pages 880–888. IEEE, 2017.

T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. Glent Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, et al. BOP: benchmark for 6D object pose estimation. In *Europ. Conf. on Computer Vision*, pages 19–34, 2018.

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. In *J. of Artificial Intelligence Research*, volume 14, pages 253–302, 2001.

H. Hu and G. Kantor. Efficient automatic perception system parameter tuning on site without expert supervision. In *Conf. on Robot Learning*, pages 57–66, 2017.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Int. Conf. on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

Intel. Intel® RealSense™ Depth Camera D435. `https://www.intelrealsense.com/depth-camera-d435/`, 2020. [Online; accessed 02-March-2021].

S. Irgenfried, H. Worn, S. Bergmann, M. Mohammadikaji, J. Beyerer, and C. Dachsbacher. Cad based workflow for semi-automatic design of optical inspection systems. In *AT-AUTOMATISIERUNGSTECHNIK*, volume 65, pages 426–439. WALTER DE GRUYTER GMBH, 2017.

H. Jin, Q. Song, and X. Hu. Auto-keras: Efficient neural architecture search with network morphism. In *arXiv:1806.10282*, 2018.

R. Kalman. A new approach to linear filtering and prediction problems. In *Trans. of the ASME-Journal of Basic Engineering*, volume 82, pages 35–45, 1960.

S. Kambhampati, A. Mali, and B. Srivastava. Hybrid planning for partially hierarchical domains. In *Nat. Conf. on Artificial Intelligence*, pages 882–888. AAAI, 1998.

B. Karan and M. Vukobratović. Calibration and accuracy of manipulation robot models—an overview. In *Mechanism and Machine Theory*, volume 29, pages 479–500. Elsevier, 1994.

P. Karkus, X. Ma, D. Hsu, L. P. Kaelbling, W. S. Lee, and T. Lozano-Pérez. Differentiable algorithm networks for composable robot learning. In *arXiv preprint arXiv:1905.11602*, 2019.

B. Kast, S. Albrecht, V. Dietrich, F. Wirnshofer, W. Feiten, and G. von Wichert. Der digitale zwilling in der autonomen robotik. In *atp magazin*, volume 61, pages 74–83, 2019a.

B. Kast, S. Albrecht, W. Feiten, and J. Zhang. Bridging the gap between semantics and control for industry 4.0 and autonomous production. In *Int. Conf. on Automation, Science and Engineering*, 2019b.

B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang. A hierarchical planner based on set-theoretic models: Towards automating the automation for autonomous systems. In *Int. Conf. on Informatics in Control, Automation and Robotics*, 2019c.

B. Kast, V. Dietrich, S. Albrecht, W. Feiten, and J. Zhang. Domain optimization for hierarchical planning based on set-theory. In *Int. Conf. on Informatics in Control, Automation and Robotics*. ScitePress, 2020.

N. J. J. P. Koenderink-Ketelaars. *A Knowledge-Intensive Approach to Computer Vision Systems*. PhD thesis, Delft University of Technology, Netherlands, 2010.

D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

F. R. Kschischang, B. J. Frey, H.-A. Loeliger, et al. Factor graphs and the sum-product algorithm. In *Trans. on Information Theory*, volume 47, pages 498–519. IEEE, 2001.

R. Kümmerle, G. Grisetti, and W. Burgard. Simultaneous calibration, localization, and mapping. In *Int. Conf. on Intelligent Robots and Systems*, pages 3716–3721. IEEE, 2011a.

R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g 2 o: A general framework for graph optimization. In *Int. Conf. on Robotics and Automation*, pages 3607–3613. IEEE, 2011b.

L. Kunze, T. Roehm, and M. Beetz. Towards semantic robot description languages. In *Int. Conf. on Robotics and Automation*, pages 5589–5595. IEEE, 2011.

S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

N. Lazaros, G. C. Sirakoulis, and A. Gasteratos. Review of stereo vision algorithms: from software to hardware. In *Int. J. of Optomechatronics*, volume 2, pages 435–462. Taylor & Francis, 2008.

Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng. On optimization methods for deep learning. 2011.

J.-S. Lee. Digital image smoothing and the sigma filter. In *Computer Vision, Graphics, and Image Processing*, volume 24, pages 255–269. Elsevier, 1983.

S.-H. Leitner and W. Mahnke. Opc ua–service-oriented architecture for industrial applications. In *ABB Corporate Research Center*, volume 48, pages 61–66, 2006.

Lexico. Meaning of middleware in English. `https://www.lexico.com/definition/middleware`, 2021. [Online; accessed 03-March-2021].

Li Yang Ku. Installing Asus Xtion on Ubuntu. `https://computervisionblog.wordpress.com/2012/04/22/installing-asus-xtion-on-ubuntu/`, 2012. [Online; accessed 02-March-2021].

B. Lightsey. Systems engineering fundamentals. Technical report, Defense Acquisition University, Fort Belvoir VA, 2001.

R. J. Linn, D. L. Hall, and J. Llinas. Survey of multisensor data fusion systems. In *Data Structures and Target Classification*, volume 1470. International Society for Optics and Photonics, 1991.

W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In *Europ. Conf. on Computer Vision*, pages 21–37. Springer, 2016.

F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. In *Autonomous Robots*, volume 4, pages 333–349. Springer, 1997.

M. Lutz, D. Stampfer, S. Hochdorfer, and C. Schlegel. Probabilistic fusion of multiple algorithms for object recognition at information level. In *Int. Conf. on Technologies for Practical Robot Applications*, pages 139–144. IEEE, 2012.

M. Lutz, D. Stampfer, and C. Schlegel. Probabilistic object recognition and pose estimation by fusing multiple algorithms. In *Int. Conf. on Robotics and Automation*, pages 4244–4249. IEEE, 2013.

D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic markup for web services, 2004. URL `http://www.w3.org/Submission/OWL-S/`. [Online; accessed 03-March-2021].

D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl - the planning domain definition language. In *The AIPS-98 Planning Competition Comitee*, 1998.

T. Messer. Model-based synthesis of vision routines. In *Advances In Machine Vision: Strategies and Applications*, pages 79–97. World Scientific, 1992.

K. S. Metaxiotis, D. Askounis, and J. Psarras. Expert systems in production planning and scheduling: A state-of-the-art survey. In *J. of Intelligent Manufacturing*, volume 13, pages 253–260. Springer, 2002.

H. B. Mitchell. *Data fusion: concepts and ideas*. Springer Science & Business Media, 2012.

K. Murphy. An introduction to graphical models. In *Rap. tech*, volume 96, pages 1–19, 2001.

T. Nagato and T. Koezuka. Automatic generation of image-processing programs for production lines. In *Fujitsu Sci. Tech. J*, volume 52, pages 27–33, 2016.

D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. In *J. of Artificial Intelligence Research*, volume 20, pages 379–404, 2003.

C. V. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *Int. Conf. on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 524–530. IEEE, 2012.

H. Niemann, G. F. Sagerer, S. Schroder, and F. Kummert. ERNEST: A semantic network system for pattern understanding. In *Trans. on Pattern Analysis and Machine Intelligence*, volume 12, pages 883–905. IEEE, 1990.

D. Nyga, F. Balint-Benczedi, and M. Beetz. PR2 looking at things - Ensemble learning for unstructured information processing with markov logic networks. In *Int. Conf. on Robotics and Automation*, pages 3916–3923. IEEE, 2014.

A. Olivares-Alarcos, D. Beßler, A. Khamis, P. Goncalves, M. K. Habib, J. Bermejo-Alonso, M. Barreto, M. Diab, J. Rosell, J. Quintas, et al. A review and comparison of ontology-based approaches to robot autonomy. In *The Knowledge Engineering Review*, volume 34. Cambridge University Press, 2019.

E. Oliver, R. Chapman, and C. French. Data processing and information technology, 1992.

R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore. Automating biomedical data science through tree-based pipeline optimization. In *Europ. Conf. Applications of Evolutionary Computation*, pages 123–137, 2016.

S. Orts-Escolano, V. Morell, J. García-Rodríguez, and M. Cazorla. Point cloud data filtering and downsampling using growing neural gas. In *Int. Joint Conf. on Neural Networks*. IEEE, 2013.

P. Y. Papalambros and D. J. Wilde. *Principles of optimal design: modeling and computation*. Cambridge university press, 2000.

E. P. Pednault. ADL and the state-transition model of action. In *J. of Logic and Computation*, volume 4, pages 467–512. Dov Gabbay, 1994.

T. Pfeifer, P. Weissig, S. Lange, and P. Protzel. Robust factor graph optimization-a comparison for sensor fusion applications. In *Int. Conf. on Emerging Technologies and Factory Automation*. IEEE, 2016.

W. Piotrowski, M. Fox, D. Long, D. Magazzeni, and F. Mercorio. Heuristic planning for hybrid systems. In *Conf. on Artificial Intelligence*, pages 4254–4255. AAAI Press, 2016.

B. Planche, Z. Wu, K. Ma, S. Sun, S. Kluckner, O. Lehmann, T. Chen, A. Hutter, S. Zakharov, H. Kosch, et al. Depthsynth: Real-time realistic synthetic data generation from cad models for 2.5 d recognition. In *Int. Conf. on 3D Vision*, pages 1–10. IEEE, 2017.

R. Platt Jr, R. Tedrake, L. Kaelbling, and T. Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Robotics: Science and Systems*, 2010.

F. Pomerleau, A. Breitenmoser, M. Liu, F. Colas, and R. Siegwart. Noise characterization of depth sensors for surface inspections. In *Int. Conf. on Applied Robotics for the Power Industry*, pages 16–21. IEEE, 2012.

A. Quemy. Data pipeline selection and optimization. In *Int. Workshop on Design, Optimization, Languages and Analytical Processing of Big Data*, 2019.

M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

M. Rad and V. Lepetit. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *Int. Conf. on Computer Vision*, 2017.

B. Radig, W. Eckstein, K. Klotz, T. Messer, and J. Pauli. Automatization in the design of image understanding systems. In *Int. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 35–45. Springer, 1992.

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Conf. on Computer Vision and Pattern Recognition*, pages 779–788. IEEE, 2016.

J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004. ISBN 0321245628.

R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Int. Conf. on Robotics and Automation*, pages 3212–3217. IEEE, 2009.

S. Sarkar and S. Chavali. Modeling parameter space behavior of vision systems using bayesian networks. In *Computer Vision and Image Understanding*. Elsevier, 2000.

B. Schattenberg. *Hybrid Planning and Scheduling*. PhD thesis, University of Ulm, Germany, 2009.

P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. v. Wichert, and W. Burgard. Optimal, sampling-based manipulation planning. In *Int. Conf. on Robotics and Automation*, pages 3426–3432. IEEE, 2017.

Siemens. Siemens Robot Learning Challenge. `https://new.siemens.com/us/en/company/fairs-events/robot-learning.html`, 2017. [Online; accessed 24-February-2020].

D. Stampfer, M. Lutz, and C. Schlegel. Informed active perception with an eye-in-hand camera for multi modal object recognition. In *IROS Workshop on Active Semantic Perception*. IEEE, 2012.

M. Stenmark and J. Malec. Knowledge-based industrial robotics. In *Scand. Conf. on Artificial Intelligence*, 2013.

R. Studer, V. R. Benjamins, and D. Fensel. Knowledge engineering: principles and methods. In *Data & Knowledge Engineering*, volume 25, pages 161–197. Elsevier, 1998.

S.-F. Su and C. G. Lee. Manipulation and propagation of uncertainty and verification of applicability of actions in assembly tasks. In *Trans. on Systems, Man, and Cybernetics*, volume 22, pages 1376–1389. IEEE, 1992.

B. Tekin, S. N. Sinha, and P. Fua. Real-time seamless single shot 6d object pose prediction. In *Conf. on Computer Vision and Pattern Recognition*, pages 292–301. IEEE, 2018.

M. Tenorth and M. Beetz. Knowrob: A knowledge processing infrastructure for cognition-enabled robots. In *The Int. J. of Robotics Research*, volume 32, pages 566–590. SAGE Publications Sage UK: London, England, 2013.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 847–855. ACM, 2013.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Conf. on Computer Vision*. IEEE, 1998.

J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Conf. on Computer Vision and Pattern Recognition*. IEEE, 2018.

M. Ulrich, C. Wiedemann, and C. Steger. Combining scale-space and similarity-based aspect graphs for fast 3d object recognition. In *Trans. on Pattern Analysis and Machine Intelligence*, volume 34, pages 1902–1914. IEEE, 2011.

E. van der Wal. Plcopen. In *IEEE Industrial Electronics Magazine*, volume 3, page 25, 2009.

D. Vaquero, M. Turk, K. Pulli, M. Tico, and N. Gelfand. A survey of image retargeting techniques. In *Applications of Digital Image Processing XXXIII*, volume 7798. International Society for Optics and Photonics, 2010.

VDMA. OPC Machine Vision. `https://opcua.vdma.org/en/viewer/-/v2article/render/27003893`, 2021. [Online; accessed 03-March-2021].

G. Vijayvargiya, S. Silakari, and R. Pandey. A survey: various techniques of image compression. In *arXiv preprint arXiv:1311.6877*, 2013.

S. v. d. Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. In *Computing in science & engineering*, volume 13, pages 22–30. IEEE Computer Society, 2011.

M. R. Walter, S. Hemachandra, B. Homberg, S. Tellex, and S. Teller. A framework for learning semantic maps from grounded natural language descriptions. In *The Int. J. of Robotics Research*, volume 33, pages 1167–1190. SAGE Publications Sage UK: London, England, 2014.

F. Wirnshofer, P. S. Schmitt, W. Feiten, G. v. Wichert, and W. Burgard. Robust, compliant assembly via optimal belief space planning. In *Int. Conf. on Robotics and Automation*. IEEE, 2018.

R. M. Young, M. Pollack, and J. Moore. Decomposition and causality in partial-order planning. In *Int. Conf. on Artificial Intelligence Planning Systems*, pages 188–194. AAAI, 1994.

Q.-Y. Zhou, J. Park, and V. Koltun. Open3d: A modern library for 3d data processing. In *arXiv preprint arXiv:1801.09847*, 2018.