Semantic robot programming by demonstration and natural language instruction

Shuang Lu

Angaben zur Veröffentlichung / Publication details:

Lu, Shuang. 2026. “Semantic robot programming by demonstration and natural language instruction.” Augsburg: Universität Augsburg.

Universität Augsburg
Fakultät für angewandte Informatik

Semantic Robot Programming by Demonstration and Natural Language Instruction

Dissertation
zur Erlangung des akademischen Grades
Doktor-Ingenieurin (Dr.-Ing.)

an der Fakultät für angewandte Informatik der Universität Augsburg

---

vorgelegt von: Shuang Lu, M.Sc.

eingereicht am: 23.05.2025

Anfertigung am Lehrstuhl: Produktionsinformatik
Fakultät für angewandte Informatik

1. Gutachter: Prof. Dr.-Ing. Johannes Schilp

2. Gutachter: Prof. Dr. Elisabeth André

Tag der mündlichen Prüfung: 30.04.2026

献给我的父母：吕文昌，于秀梅

Vorwort

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftliche Mitarbeiterin am Fraunhofer-Institut für Gießerei-, Composite- und Verarbeitungstechnik IGCV sowie im Rahmen meiner Promotion am Lehrstuhl für Produktionsinformatik der Universität Augsburg.

Mein besonderer Dank gilt meinem Doktorvater, Herrn Prof. Dr.-Ing. Johannes Schilp, für die fachliche Betreuung, die wertvollen Anregungen und das mir entgegengebrachte Vertrauen. Frau Prof. Dr. Elisabeth André danke ich herzlich für die Übernahme der Zweitgutachtung. Ebenso danke ich Herrn Prof. Dr. Jörg Hähner für die Übernahme der Funktion des Drittprüfers im Rahmen der mündlichen Prüfung.

Den Kolleginnen und Kollegen am Fraunhofer IGCV danke ich herzlich für die angenehme Zusammenarbeit und den fachlichen Austausch. Mein besonderer Dank gilt Dr. Julia Berger für das sorgfältige Lesen meiner Arbeit und ihre wertvollen Rückmeldungen. Dr. Albrecht Lottermoser danke ich in besonderem Maße für seine vielseitige Unterstützung und den bereichernden Austausch. Christian Härdtlein und Johannes Fink danke ich für ihre hilfreiche Unterstützung. Nicht zuletzt bin ich dankbar dafür, dass ich durch die Zusammenarbeit mit meinen Kolleginnen und Kollegen meine Deutschkenntnisse im beruflichen und wissenschaftlichen Umfeld deutlich verbessern konnte.

Mein größter Dank gilt meinem Mann, Ziyu Wang, der meinen Wunsch zu promovieren von Beginn an unterstützt und mich während der gesamten Promotionszeit mit Geduld und Verständnis begleitet hat. Von Herzen danke ich meinen Eltern, die mir in meiner Heimat China die Grundlage für meinen Weg gegeben und mich bis zur Promotion in Deutschland stets unterstützt haben. Ohne ihren Rückhalt wäre diese Arbeit in dieser Form nicht möglich gewesen.

Zusammenfassung

Die Entwicklung von Industrierobotern weg von isolierten, sich wiederholenden Aufgaben hin zu vielseitig einsetzbaren Agenten in gemeinsamen menschlichen Umgebungen stellt einen Paradigmenwechsel in der Produktionstechnik dar. Herkömmliche Programmiermethoden für Industrieroboter sind nicht nur zeitaufwändig, sondern erfordern auch ein hohes Maß an Fachwissen, was ihre Skalierbarkeit und Flexibilität in sich schnell verändernden industriellen Umgebungen einschränkt. In dieser Arbeit stellt das neuartige Konzept der semantischen Roboterprogrammierung dar, welches darauf abzielt, traditionelle Robotersysteme in anpassungsfähigere, intelligentere und benutzerfreundlichere Einheiten zu transformieren.

Das vorgeschlagene semantische Roboterprogrammiersystem ermöglicht es Robotern, mehrstufige Pick & Place-Aufgaben durch Anweisungen in natürlicher Sprache und durch menschliche Demonstrationen effizient zu erlernen und auszuführen. Das System ermöglicht es Robotern, ein anpassungsfähiges Aufgabenmodell aus einer einzigen Anweisung und Demonstration zu erlernen. Um dieses Ziel zu erreichen, werden separate Ansätze für das Verstehen von Anweisungen und visuellen Demonstrationen entwickelt. Anschließend wird ein modulares Software-Framework vorgeschlagen, um die multimodalen Eingaben in ein einheitliches Aufgabenmodell zu integrieren. Für das Verstehen natürlicher Sprache wird ein hybrider Ansatz entwickelt, der ein auf Deep Learning basierendes Modell mit einem regelbasierten Parser kombiniert, um Handlungsabläufe, Zielobjekte und deren räumliche Beziehungen zu extrahieren. Der hybride Ansatz ermöglicht es, die Methode der Informationsextraktion für neue Domänenaufgaben zu nutzen. Um die visuelle Programmierung durch Demonstration zu verstehen, wird eine Handerkennungsmethode verwendet, um die Handtrajektorien zu erfassen, die für die Abbildung der Handbewe-

gung auf den Roboterendeffektor entscheidend sind. Darüber hinaus wird eine Objekterkennungsstrategie implementiert, bei der Trainingsdaten in einer Simulationsumgebung automatisch generiert werden, so dass die Modelle verschiedene Szenarien erlernen und dieses Wissen anschließend auf reale Daten anwenden können. Darüber hinaus wird eine Bewegungssegmentierungsmethode entwickelt, um die Gesamtaufgabe in Pick & Place Aktionen für einzelne Objekte zu zerlegen, die dann weiter in Fähigkeiten segmentiert werden, die durch dynamische Bewegungsprimitive repräsentiert werden. Die erlernten Fähigkeiten können an unterschiedliche Start- und Zielpositionen angepasst werden. Schließlich werden die vorgeschlagenen Methoden in sechs Softwaremodulen und einer relationalen Datenbank zusammengefasst, die jeweils zur effizienten Darstellung, Planung und Ausführung von Roboteraufgaben beitragen. Das modulare Framework erlaubt eine einfache Erweiterung um zusätzliche Semantiken.

Das vorgeschlagene System wird anhand von mehrstufigen Pick & Place-Aufgaben evaluiert. Die Evaluierung zeigt die Fähigkeit des Systems, multimodale Eingaben zu interpretieren und Aufgabenmodelle zu erstellen. Durch die Integration von semantischem Verstehen und fortgeschrittenen Lernalgorithmen reduziert das vorgeschlagene System nicht nur die Abhängigkeit von Expertenwissen, sondern verbessert auch die Fähigkeit des Roboters, komplexe und vielfältige Aufgaben auf benutzerfreundliche Weise auszuführen.

Abstract

The evolution of industrial robots from isolated, repetitive task performers to versatile, multi-purpose agents in shared human environments represents a paradigm shift in manufacturing technologies. Traditional programming techniques for industrial robots are not only time-consuming but also require a high level of expertise, which limits their scalability and flexibility in rapidly changing industrial environments. This thesis proposes the novel concept of semantic robot programming to transform traditional robotic systems into more adaptive, intelligent, and user-friendly entities.

The proposed semantic robot programming system enables robots to efficiently learn and execute multi-step pick & place tasks through natural language instructions and human demonstrations. The system allows robots to learn an adaptable task model from a single instruction and demonstration. To achieve this goal, separate approaches are developed to understand the instructions and visual demonstrations. Subsequently, a modular software framework is proposed to integrate the multimodal inputs into a unified task model. For natural language understanding, a hybrid approach combining a deep learning-based model and a rule-based parser is developed to extract the action sequences, target objects, and their spatial relations. The hybrid approach enables the use of the information extraction method to be applied for new domain tasks. To understand visual programming by demonstration, a hand detection method is utilized to capture the hand trajectories, which is crucial for mapping the hand motion to robot end-effector. Additionally, an object detection strategy is implemented, in which training data is automatically generated in a simulation environment, enabling the models to learn diverse scenarios and subsequently apply this knowledge to real-world data. Furthermore, a motion segmentation method is developed to break the complete task into pick & place actions

for individual objects, which are then further segmented into skills which are represented by dynamic movement primitives. The learned skills are adaptable to different start and target positions. Finally, the proposed methods are grouped into six software modules and a relational database, each contributing to the efficient representation, planning and execution of robotic tasks. The modular framework allows for easy extension with additional semantics.

The proposed system is evaluated on multi-step pick & place tasks. The evaluation demonstrates the system's ability to interpret the multimodal inputs and generate task models. By incorporating semantic understanding and advanced learning algorithms, the proposed system not only reduces dependence on expert knowledge but also enhances the robot's ability to perform complex, varied tasks in a user-friendly manner, significantly improving operational adaptability in modern manufacturing environments.

Declaration

The results discussed in this dissertation have been published in the following articles:

1. Shuang Lu, Julia Berger, and Johannes Schilp. An integrated approach for hand motion segmentation and robot skills representation. In *MHI Colloquium*, pages 291–301. Springer, 2022a. doi: 10.1007/978-3-031-10071-0_24
2. Shuang Lu, Julia Berger, and Johannes Schilp. System of robot learning from multi-modal demonstration and natural language instruction. *Procedia CIRP*, 107:914–919, 2022b. ISSN 2212-8271. doi: 10.1016/j.procir.2022.05.084
3. Shuang Lu, Julia Berger, and Johannes Schilp. Extracting robotic task plan from natural language instruction using BERT and syntactic dependency parser. In *2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1794–1799, 2023. doi: 10.1109/RO-MAN57019.2023.10309598
4. Shuang Lu, Christian Härdtlein, and Johannes Schilp. Visual imitation learning from one-shot demonstration for multi-step robot pick and place tasks. *Scientific Reports*, 2025. doi: 10.1038/s41598-025-30938-x

Shuang Lu, M.Sc.
23.05.2025

Table of contents

List of figures

List of tables

List of abbreviations

AP	Average Precision
Bbox	Bounding Box
BERT	Bidirectional Encoder Representations from Transformers
CAD	Computer Aided Design
CNN	Convolutional Neural Network
Cobot	Collaborative robot
COCO	Common Objects in Context
DAO	Data Access Object
DMP	Dynamic Movement Primitive
DOF	Degree of Freedom
DRM	Design Research Methodology
DS-I	Descriptive Study I
DS-II	Descriptive Study II
FN	False Negative
FP	False Positive
GloVe	Global Vectors for Word Representation

GMM	Gaussian Mixture Model
GMR	Gaussian Mixture Regression
GPT	Generative Pre-trained Transformer
GRU	Gated Recurrent Unit
GUI	Graphical User Interface
IE	Information Extraction
IFR	International Federation of Robotics
IoU	Intersection over Union
LLM	Large Language Model
LSTM	Long Short-Term Memory
mAP	Mean Average Precision
NE	Named Entity
NER	Named Entity Recognition
NLP	Natural Language Processing
NLU	Natural Language Understanding
NN	Neural Network
OBB	Oriented Bounding Box
PbD	Programming by Demonstration
POS	Part-of-Speech
PS	Prescriptive Study
RC	Research Clarification
RCNN	Region-based Convolutional Neural Network

RE	Relation Extraction
RGB-D	Red, Green, Blue, and Depth
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SpERT	Span-based Entity and Relation Transformer
TCP	Tool Center Point
TN	True Negative
ToF	Time-of-Flight
TP	True Positive
YOLO	You Only Look Once

List of definitions

AprilTag

: According to Olson (2011), AprilTags are "a type of fiducial marker system designed for robust visual identification and localization in computer vision applications."

Collaborative Robot

: According to Berg et al. (2019), collaborative robots are "lightweight industrial robots up to 30 kg that can operate alongside humans in a shared workspace. They are specifically designed to have built-in safety sensors that prevent harm during human-robot collaboration."

Corpus

: Summarized from McEnery and Gabrielatos (2006), a corpus is a large and structured set of machine-readable authentic texts (including transcripts of spoken data) that is sampled to be representative of a particular natural language or language variety. Corpora play an essential role in Natural Language Processing (NLP) research. They provide a material basis and a test bed for building NLP systems.

Dynamic Movement Primitive

: As defined by Schaal (2006), Dynamic Movement Primitives (DMPs) are "a framework used in robotics and motor control to represent and generate complex movements. They provide a mathematical way to model both discrete and rhythmic movements, allowing for flexibility, adaptability and generalization."

End-effector

> "An end-effector is a device specifically designed for attachment to the mechanical interface to enable the robot to perform its task. Examples include grippers, welding guns, and spray guns." (ISO 8373:2013)

Framework

> Based on Collett et al. (2005), this thesis defines a framework as a collection of software tools and libraries that provide a structured and standardized approach to robot program development.

Industrial Robot

> As defined in ISO 8373:2021, an industrial robot is "an automatically controlled, reprogrammable multipurpose manipulator. It is programmable in three or more axes, which can be either fixed in place or fixed to a mobile platform for use in automation applications in an industrial environment."

Information Extraction

> According to Sarawagi (2008), Information Extraction (IE) refers to "the automatic extraction of structured information such as entities, relationships between entities, and attributes describing entities from unstructured sources."

Information Grounding

> Based on Nyga et al. (2018) and Lewis (2008), this thesis defines information grounding as the process of establishing a clear and shared understanding of information or knowledge between individuals or within a system. In practical terms, this involves "enabling a robot to associate words or phrases with the corresponding objects, scenes, or actions it sees" (Vanzo et al., 2020c).

Natural Language Instruction

> Based on She et al. (2014), Venkatesh et al. (2021) and Vanzo et al. (2020c), this thesis defines natural language instruction as the ability to provide commands to a robot using human language, spoken or written.

One-Shot Imitation Learning

As described by Vitiello et al. (2023), one-shot imitation learning is "a subset of imitation learning where a robot or agent learns to perform a task by observing it just once."

Programming by Demonstration

According to Billard et al. (2008), "Programming by Demonstration (PbD), also known as learning from demonstration, imitation learning, or apprenticeship learning, enables robots to autonomously perform new tasks by observing and learning from humans."

Semantic Robot Programming

This thesis defines semantic robot programming as an approach that enables robots to be programmed using high-level semantic representations. It integrates natural language and visual demonstrations, allowing robots to understand both what a task is and how to perform it flexibly.

Skill

Cândido and Barata (2007) define the skills as "the ability to perform actions that are then needed to support the manufacturing process." This thesis applies the definition by Pantano et al. (2024) in the context of robot programming: "a skill is a predefined robot's capability that can be parameterized to solve a specific goal."

System

Bertalanffy (1969) defines a system as "a set of interacting or interdependent components that form an integrated whole." In the context of robot programming, according to Siciliano and Khatib (2016), "a system refers to the complete set of hardware and software components that work together to perform specific tasks. This includes the robotic hardware, sensors, actuators, control algorithms and software framework that executes these algorithms."

Task

> "A task is defined as an ordered ensemble of different skills that represents steps in a workflow to solve a specific goal." (Pedersen et al., 2016; Pantano et al., 2022)

Teach Pendant

> As defined in ISO 8373:2021, a teach pendant is "a hand-held unit linked to the control system with which a robot can be programmed or moved."

Teach Programming

> According to ISO 8373:2021, it refers to "the programming of the task performed by a) manually moving the robot to desired positions; b) using a teach pendant to move the robot through the desired positions; c) using a teach pendant to program without causing motion; or d) using algorithms with sensor data."

Tool Center Point

> According to Siciliano and Khatib (2016), the Tool Center Point (TCP) refers to "the specific point on an end-effector or tool where the robotic system considers it as the reference point for positioning and control." The TCP is typically defined as a coordinate in three-dimensional space, specified by its X, Y, and Z coordinates.

Chapter 1

Introduction

1.1 Motivation

Recent changes in market trends, including increased emphasis on product differentiation, personalization, and high-mix, low-volume production, have highlighted the importance of flexibility in the manufacturing industry. This has been a significant focus of research in recent years (Beibl et al., 2023). The global pandemic between 2020 and 2022 has further enhanced the need to deal with the uncertainty. Collaborative robots (Cobots) have emerged as valuable tools for flexible manufacturing (Matheson et al., 2019). They are lightweight industrial robots that can work alongside human workers without the need for physical barriers. They are easy to move, and can be quickly reconfigured for various tasks. This reconfiguration is achieved through a combination of software adaptations and modular hardware changes (Romiti et al., 2022). The software typically features an intuitive programming interface (Steinmetz et al., 2019), while the hardware includes modular components such as quick-change mechanical units for the end-effector (Iqbal et al., 2021). Recent statistics from International Federation of Robotics (IFR) indicate a significant growth in Cobot installations worldwide, with Cobots accounting for 10% of newly installed robots (Müller, 2023). By deploying Cobots, manufacturing companies can increase their flexibility, manage labor shortages, and minimize potential business losses due to unfilled orders (Du et al., 2022).

Cobots are commonly used for handling tasks such as assembly (Queirós

et al., 2022), logistics (Yumbla et al., 2022), and maintenance (Koch et al., 2017). Fig. 1.1 presents a graphical representation of the distribution of industrial robots across various applications in 2023. The handling category accounts for approximately half of all industrial robots, which are employed for the execution of pick & place tasks. The diverse range of products necessitates the use of flexible pick & place systems, which can accommodate varying objects and layouts. This adaptability enables manufacturers to respond promptly to changing production demands. However, the traditional approach of reprogramming robots for each new factory request is not feasible for flexible pick & place systems, as it requires expert knowledge and is time-consuming (Steinmetz et al., 2019; Berg, 2020).

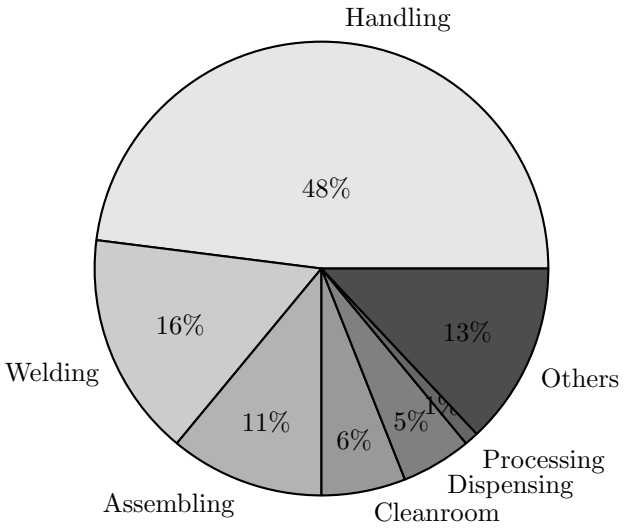


Fig. 1.1: Distribution of industrial robots by application (IFR 2023)

Programming by Demonstration (PbD) is a promising paradigm for intuitive programming by non-expert users (Saveriano, 2017). In PbD, a user demonstrates how to perform a certain task, and the robot leverages this demonstration to acquire and then reproduce the task. The demonstration can be performed in a kinesthetic manner, where the user physically guides the robot, or visually, where the robot observes the task being performed. However, a significant limitation of PbD is that the robot typically lacks an understanding of the underlying semantics of the task. This deficiency restricts its ability to generalize and adapt to new situations or variations of the task. Without semantic understanding, the robot can reproduce only the exact actions demonstrated, failing to adapt

to minor changes in the environment or the task requirements. This constraint hinders the application of PbD in dynamic or unpredictable environments where flexibility and adaptability are crucial.

1.2 Research objectives

The objective of this thesis is to develop a programming system with integrated semantics that allows the robot to understand not only how to perform certain tasks, but also what the tasks are. Such semantic understanding enhances the robot's capability to adapt flexibly to changing task requirements, which is particularly valuable in industrial manufacturing scenarios characterized by high-mix, low-volume production.

Human language naturally embodies semantics that describe the task requirements. Additionally, visual PbD captures rich meanings in both the actions performed and the context in which they are executed. This thesis leverages these two inputs to enable non-expert users to program the robot. The primary focus is on pick & place tasks, as they are widely used in industrial settings and typically involve multiple steps with sequences of actions performed on several objects (Pedersen et al., 2016). Enhancing the efficiency and adaptability of the pick & place process significantly contributes to increasing the flexibility of manufacturing operations.

The concept of the programming system is illustrated in fig. 1.2. The instruction *pick the cuboid and cylinder and place them in the gray box* along with a video demonstration are inputs to program the task. The developed system extracts relevant information from these inputs to create a task plan. The task plan can then be executed by the robot to perform the pick & place tasks in a step-by-step manner. When the task instruction changes, the system can generate an adapted task program for execution, ensuring flexibility and adaptability in task performance.

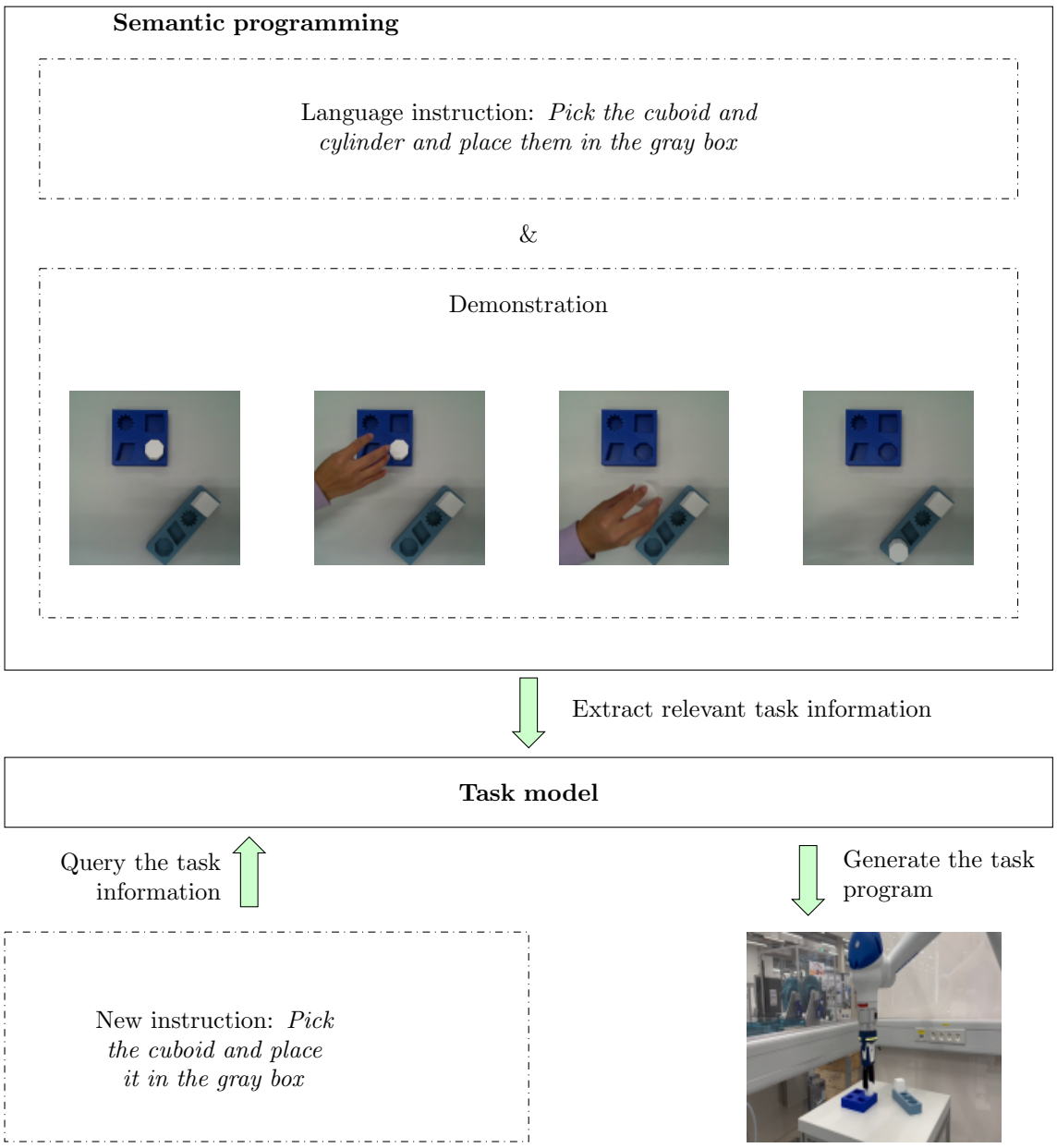


Fig. 1.2: Overview of the concept of the semantic programming system

1.3 Research methodology and thesis structure

This thesis employs the Design Research Methodology (DRM) as outlined by Blessing and Chakrabarti (2009) as the foundational framework to scientifically develop, implement, and evaluate the proposed programming system. In this context, the term *design* refers to "those activities that actually generate and develop a product from a need, product idea or technology to the full documen-

tation needed to realize the product to fulfill the perceived needs of the user and other stakeholders" (Blessing and Chakrabarti, 2009, p. 1). The overall aim of *design research* is to make design more effective and efficient (Blessing and Chakrabarti, 2009, p. 12). Given that this thesis aims to design and realize a semantic programming system, the DRM has been selected as the underlying research methodology of this thesis.

The DRM consists of the following four stages: Research Clarification (RC), Descriptive Study I (DS-I), Prescriptive Study (PS), and Descriptive Study II (DS-II). This methodology is a flexible and iterative process that can adapt to the needs of various research projects by allowing multiple iterations and parallel execution within and between stages. In the RC stage, the literature is reviewed to find evidence to support the assumptions and to define a realistic research goal. Based on the findings, an initial description of the existing situation is developed, as well as a description of the desired situation. The DS-I stage helps to complete the existing situations. Depending on the research goal, DS-I will be limited to a review-based study involving a detailed review of the literature, or a comprehensive study that includes a detailed literature review and an empirical study. The PS stage focuses on leveraging the insights from DS-I or DS-II to develop a design support system. This support can take various forms, including knowledge frameworks, guidelines, checklists, methods, or tools. In the DS-II stage, researchers evaluate the impact of the support tool on achieving the desired outcomes through empirical studies. Fig. 1.3 illustrates the application of the DRM framework within the structure of the thesis. This thesis is composed of ten chapters. A brief overview of each chapter in the context of DRM is outlined below.

The outcomes from the Research Clarification (RC) stage are discussed in chapter 1. It motivates the research and defines the initial research objectives.

The Descriptive Study I (DS-I) stage includes chapter 2 and chapter 3, both based on a literature analysis. This research leverages machine learning, including both classical algorithms and deep learning, to extract information from language and visual demonstrations for robot programming. Due to its multidisciplinary nature, relying solely on external books would require readers to consult multiple sources, reducing reading efficiency. To address this, the essential fundamentals

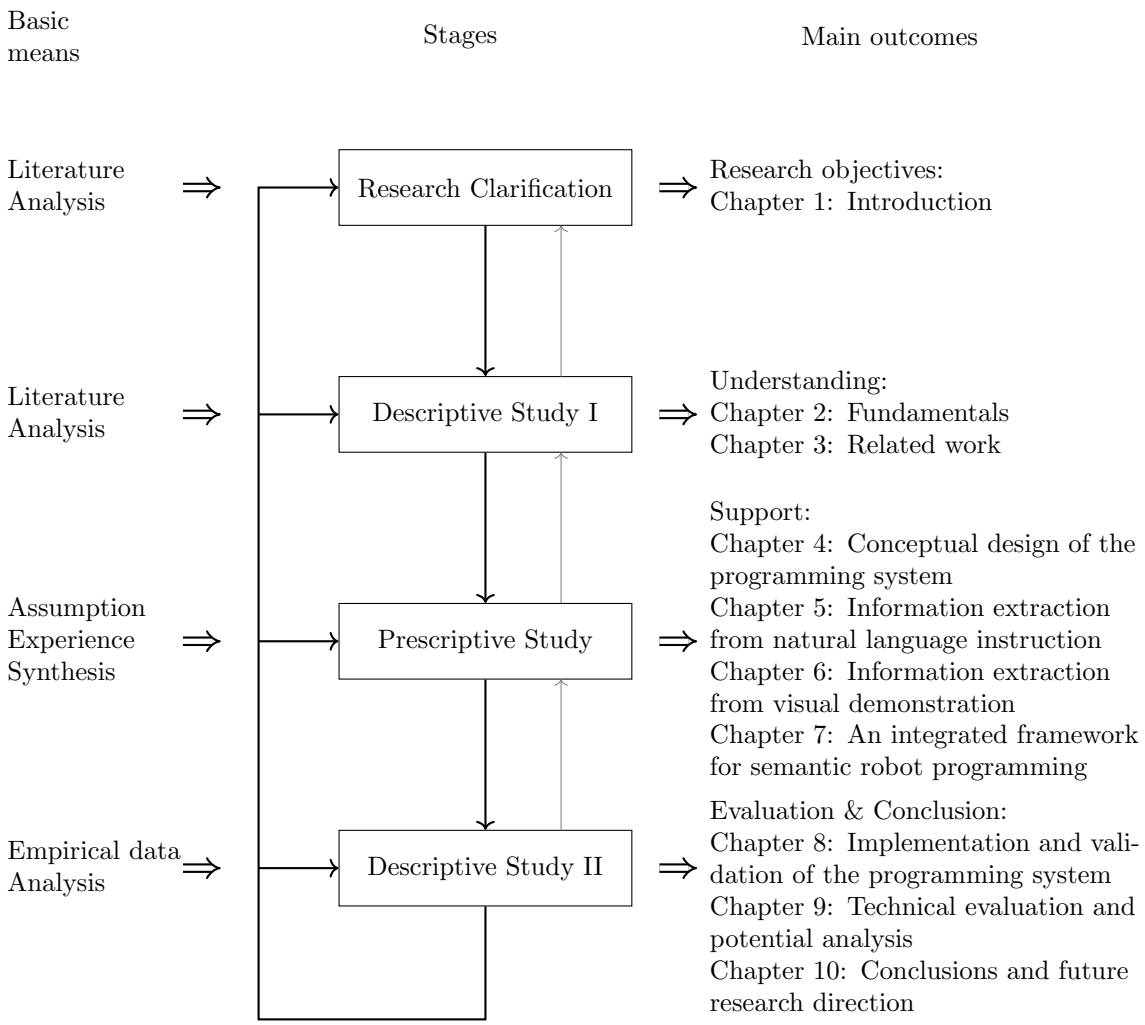


Fig. 1.3: Applied design research methodology and the resulting thesis structure

are summarized in chapter 2. This will help the reader to understand the related work in chapter 3 and the methodologies developed in subsequent chapters. Specifically, the first section in chapter 2 introduces machine learning principles. Basic concepts of deep learning are presented in the second section, including neural networks and learning paradigms. Computer vision applications are described in the third section, with a focus on object detection. The fourth section focuses on techniques in natural language processing, including linguistic structures and common tasks. Finally, the fundamentals of robot programming are explained in the fifth section, focusing on classical programming methods. A review of existing literature and previous work is conducted in chapter 3.

This chapter is organized into six main sections. In line with the motivation to make robot programming accessible to users without specialized expertise, the first section provides an overview of intuitive and user-friendly programming approaches. Two approaches are identified from this section to be used in this thesis. The second section details one such approach, robot PbD, which has been widely studied over the last few decades due to its ability to enable robots to learn tasks from human demonstrations without requiring explicit coding. The third section explores natural language-based robot programming, another intuitive approach that leverages the inherent semantic structure of language to facilitate robot instruction. The fourth section reviews existing efforts to integrate multimodal input for robot programming, aligning with this thesis's focus on leveraging both language and visual input. The related work is evaluated in the fifth section, assessing strengths and limitations to identify research gaps. Finally, the research objectives are defined in the sixth section, highlighting its innovations and advancements in the field.

The Prescriptive Study (PS) phase spans the next four chapters. Based on the findings from DS-I stage, chapter 4 presents the conceptual design of the proposed programming system. It begins with an overview of the developed methodological approach in the first section, providing the overall framework for integrating semantic information into robot programming. The subsequent sections describe each step of this approach in detail.

Chapter 5 presents methods for extracting information from natural language instructions. It begins with an initial analysis of existing corpora to identify underlying linguistic patterns relevant to task descriptions. This is followed by an analysis of natural language instructions specific to manufacturing scenarios. Based on these analyses, an approach for information extraction is developed and described in the third section. Finally, a subset of the existing corpora is annotated and used to evaluate the effectiveness of the proposed approach.

The approaches for extracting information from visual Programming by Demonstration (PbD) are presented in chapter 6, which focuses on mapping the hand motion to the robot end-effector. The first section presents an overview of the proposed approach. In the second section, the methods for hand and object detection are introduced. The third section addresses the use of language input

to support visual information extraction. Following this, the process of mapping motion from the human hand to the robot end-effector is discussed in the fourth section. Lastly, the proposed methods are summarized in the fifth section.

Chapter 7 presents a framework for integrating multimodal inputs to support robot programming and execution. The framework consists of six software modules. An overview is provided in the first section. It is designed to support the integration of new semantics without redeveloping the entire system. The software modules organize and integrate the proposed methods in chapter 5 and chapter 6 into the Language Module, the Environment Module and the Motion Module. The Task Module, discussed in the second section, contains a relational database along with methods for storing and retrieving data. The remaining modules, the Programming Module and the Execution Module, support the programming and execution processes and are described in detail in the subsequent sections.

The Descriptive Study II (DS-II) is documented in the next three chapters. Empirical data analysis is the focus of chapter 8. Concretely, demonstration videos and instructions are collected to evaluate each software module. The collected data serves as test cases to validate the system's functionality and performance. The first section describes the experimental environment and the setup used to validate the programming system. The second section evaluates how effective the language module processes and interprets the natural language instructions. The third section covers the assessment of the methods for visual understanding in environment module. The fourth section details the validation of the motion module. The testing of the programming module and execution module is discussed in the fifth section. Chapter 9 presents a comprehensive technical evaluation and potential analysis of the developed system. It begins with an assessment of the technical performance of the individual system components and their integration. This is followed by an evaluation of how well the research objectives defined in section 3.6 have been achieved. Finally, a potential analysis is conducted to identify the system's benefits and limitations. Chapter 10 provides conclusions on the research contributions presented in this thesis. Additionally, it outlines directions for further research to enhance and extend the proposed programming system.

Chapter 2

Fundamentals

This chapter presents the basic concepts relevant to this research, which covers classical machine learning, deep learning, computer vision, natural language processing, and robot programming. The first section introduces machine learning, covering supervised, unsupervised, and reinforcement learning. The second section focuses on deep learning, explaining aspects necessary for understanding its applications in this research. Various layer types and model architectures are discussed to illustrate how different network structures process data. Transfer learning is also addressed, as it allows the use of pre-trained models for efficient adaptation to new tasks, reducing the need for large datasets. Additionally, performance evaluation metrics are described for assessing model effectiveness. The third section focuses on computer vision, particularly object detection. The fourth section outlines natural language processing, including linguistic structures and common tasks. Finally, the fifth section presents the fundamentals of robot programming.

2.1 Machine learning

Machine learning involves the development of algorithms to learn patterns and make decisions from data. The task can be divided into the following main categories: supervised learning, unsupervised learning, and reinforcement learning (Qiu et al., 2016). The main difference lies in the existence of ground

truth, a set of data samples with desired targets. When this is known, the task is called supervised; otherwise, it is called unsupervised. Reinforcement learning, on the other hand, is distinct as it involves learning from interaction with an environment to achieve a goal. This section provides a brief introduction to these three topics. More detailed explanations can be found in Bishop (2006) and Sutton and Barto (1998).

In the context of supervised learning, "annotated data" refers to a dataset that includes input-output pairs, where each input is associated with a correct output. The goal of supervised learning is to learn a mapping from inputs to outputs that can be used to predict the output for new, unseen inputs. The fundamental tasks in supervised learning are classification and regression. Classification assigns discrete labels or categories to the inputs, while regression predicts numerical values based on input. The objective of both tasks is to minimize a loss function that measures the difference between the model output and the ground truth labels. Some common classical regression techniques include linear regression, ridge regression and support vector machines (Bishop, 2006).

Unsupervised learning, on the other hand, deals with data that does not have labeled outputs. The goal is to infer the natural structure present within a set of data points. Clustering is a common task in unsupervised learning, and one widely used method for clustering is the Gaussian Mixture Model (GMM). It is assumed that the data is generated from a mixture of several Gaussian distributions, each representing a cluster. The algorithm is initialized with predefined parameters for these Gaussian distributions. Iterative steps are then performed, starting with the Expectation (E-step), in which the probability of each data point belonging to a specific Gaussian distribution is calculated. This is followed by the Maximization (M-step), where the parameters are updated to maximize the likelihood of the data given the current assignment probabilities. These steps are repeated until convergence of the parameters is achieved (Bishop, 2006).

The aforementioned classical machine learning techniques are designed to work effectively with small amounts of data. In contrast, modern machine learning approaches, particularly deep learning, are capable of handling large-

scale datasets and complex patterns, leveraging massive amounts of data and computational power to achieve high performance.

Reinforcement Learning (RL) extends the capabilities of machine learning by focusing on how agents should take actions in an environment to maximize cumulative rewards. Unlike supervised learning, which relies on annotated data, RL involves learning through direct interaction with the environment. The agent makes decisions, observes the outcomes, and adjusts its strategies based on the rewards received (Sutton and Barto, 1998).

2.2 Deep learning

Deep learning is a subfield of machine learning that trains an artificial neural network to learn and make predictions from large amounts of data (Goodfellow et al., 2016). This section describes the basic functionality, architecture, and the training process of a deep learning model. Its goal is to provide readers with the foundational understanding needed to follow the subsequent discussions on deep learning applications in computer vision and natural language processing.

Neural Networks (NNs) use mathematical functions to emulate neurons, which are represented as nodes in a graph. Each node represents a linear function followed by an activation function, as illustrated in fig. 2.1. The linear function calculates a weighted sum of the inputs and applies a bias term, as shown in Equation 2.1. To introduce non-linearity to the node, an activation function f is used. Common activation functions include sigmoid, tanh, and Recitified Linear Unit (ReLU) (Goodfellow et al., 2016). The goal of the training process is to identify the weights w_i and bias b that best represent the relationship between the input x and the output y. Optimization algorithms like gradient descent are often used in the training process to minimize the loss function. Key hyperparameters such as number of epochs, batch size, and learning rate are tuned to control model convergence and overall performance.

$$\begin{aligned} z &= \sum_i w_i x_i + b \\ y &= f(z) \end{aligned} \tag{2.1}$$

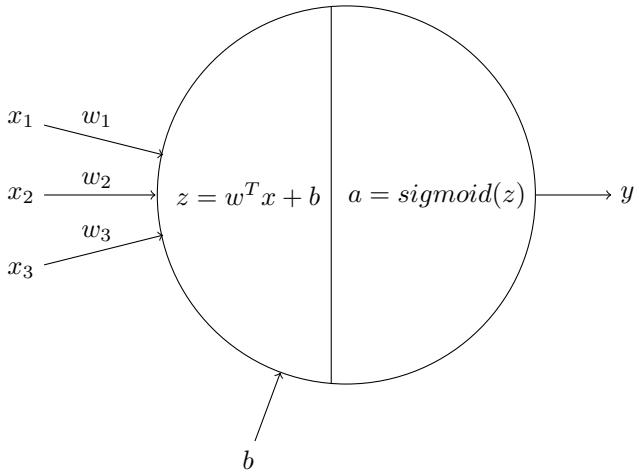


Fig. 2.1: Single neuron representation

A NN usually consists of many neurons that are connected to one another by edges, which symbolize the information flow within the network (Goodfellow et al., 2016). The individual neurons are arranged in layers for NN representation. Each NN consists of an input layer, an output layer, and hidden layers between them. The input layer receives a numerical representation of the data, and the output layer produces results by applying mathematical functions. NN architectures with multiple hidden layers are considered a deep neural network or a deep learning model (Goodfellow et al., 2016; Paszke et al., 2019).

2.2.1 Layer types

In deep learning, various types of layers are used to construct NNs. Some common types include:

- A fully connected layer, also known as a dense layer, is a common type of layer where neurons from two adjacent layers are completely pairwise connected, but share no connections within the layer.
- Convolutional layers, primarily used in Convolutional Neural Networks (CNNs) for processing grid-like data such as images. They apply convolution operations to the input using learnable filters, which helps in detecting local patterns.
- Pooling layers, often used alongside convolutional layers in CNNs. They reduce the spatial dimensions of the input volume while retaining important

information.

- Recurrent layers process sequences of data by maintaining an internal state that captures information about previous elements in the sequence. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are variants of recurrent layers designed to address the vanishing gradient problem.

More details about the layers can be found in Goodfellow et al. (2016).

2.2.2 Model architectures

Model architectures refer to the way how the different layers are organized and interconnected. The architecture determines how the model processes and transforms the input data to produce the desired output. The choice of model architecture depends on the type of problem being addressed and the nature of the data. For example, CNNs are commonly used for image-related tasks due to their ability to capture spatial hierarchies of features. Fig. 2.2 shows a model architecture that includes convolutional and fully connected layers (Sindagi and Patel, 2018). Recently widely used models have more complex architectures than the one shown in the figure. As the field of deep learning continues to evolve, new model architectures are proposed to solve different tasks, which will be discussed in the upcoming sections.

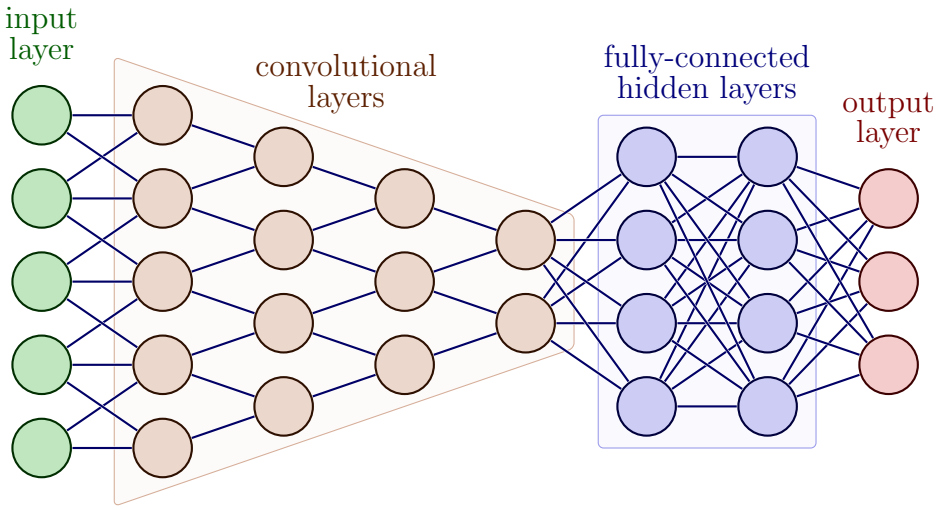


Fig. 2.2: An example of a deep learning model architecture

2.2.3 Transfer learning

This thesis employs supervised deep learning techniques to train deep neural networks using annotated data. Deep learning frameworks like PyTorch (Paszke et al., 2019) can support the training process, resulting in a model with a specific configuration and set of parameters. This model represents the learned knowledge or the problem at hand. It can be viewed as the result of training a particular architecture with specific data. The repeating of the training process of a neural network model is defined as transfer learning (Alzubaidi et al., 2021). Transfer learning allows models to use knowledge gained from large datasets or complex tasks and apply it to new or smaller datasets or tasks. Key concepts of transfer learning include:

- Pre-trained model: A model that has already been trained on a large dataset and can be used as a starting point. It leverages the patterns and features during initial training, thereby reducing the amount of data and time required for training on the new task (Salehi et al., 2023).
- Foundational model: These models, which are pre-trained on large-scale internet data, are known as foundation models. They are trained for general purpose tasks, such as Generative Pre-trained Transformer (GPT)-4. Traditional deep learning models in robotics are trained on small datasets tailored for specific tasks, which limits their ability across diverse applications. According to Firoozi et al. (2023), foundation models demonstrate an emergent ability to find zero-shot solutions to problems that are not present in the training data. Therefore, foundation models have the potential to enhance various components of the robot autonomy stack, from perception to decision-making and control. In their survey, various applications of foundational models in robotics are explored.
- Fine-tuning: It involves adjusting a pre-trained model for a specific new task. This is typically done by freezing certain layers of the pre-trained model to preserve learned features while updating the weights of the remaining layers to adapt the model to the new task. This method helps in customizing the model efficiently without overfitting (Salehi et al., 2023).
- Feature extraction: This approach involves using a pre-trained model

to extract meaningful features from new data. These features, which encapsulate critical information learned from the original dataset, can then be used to train a new model or improve the performance of an existing model on the new task. Feature extraction helps leverage the representational power of pre-trained models without the need for extensive retraining (Salehi et al., 2023).

2.2.4 Performance evaluation

The applications of deep learning in this thesis are primarily focused on classification tasks. To evaluate the performance of these tasks, the following classification metrics are introduced. In order to maintain clarity, the metrics for binary classification are initially addressed, followed by those for multi-class tasks. For binary classification tasks, the four possible classification outcomes are:

- True Positive (TP). The amount of positive samples that were correctly identified as such.
- False Positive (FP). The amount of negative samples that were mistakenly identified as positive.
- True Negative (TN). The amount of negative samples that were correctly identified as such.
- False Negative (FN). The amount of positive samples that were mistakenly identified as negative.

These classification outcomes are typically organized in a confusion matrix, which summarizes the relationship between predicted and actual class labels, as shown in table 2.1.

Table 2.1: Confusion matrix for binary classification

		Predicted Classes	
		Positive	Negative
Actual Classes	Positive	TP	FN
	Negative	FP	TN

To assess the performance of a trained classifier, the following evaluation metrics are typically employed:

- Precision is a metric in deep learning to evaluate the quality of a model (Rainio et al., 2024). It measures the proportion of correctly predicted positive instances out of all instances predicted as positive.

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.2}$$

- Recall, also known as sensitivity or true positive rate, is a performance metric that measures the fraction of correctly predicted positive instances out of the total actual positive instances (Rainio et al., 2024).

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.3}$$

- The F1-score is a commonly used metric for evaluating the performance of classification models, particularly when dealing with imbalanced datasets or when both precision and recall are important. It is calculated as the harmonic mean of precision and recall (Emmert-Streib et al., 2023).

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{2.4}$$

- Accuracy indicates the proportion of correct predictions made by the model out of all predictions (Rainio et al., 2024).

$$Accuracy = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.5}$$

- The precision-recall curve shows the trade-off between precision and recall for different threshold values (Miao and Zhu, 2022). The threshold is a value between 0 and 1. For certain classification tasks, if the value is greater than the defined threshold, the class is considered to be one of the TP. Fig. 2.3 shows an example curve with 10 threshold values. A high area under the curve represents both high recall and high precision, where high precision is associated with a low FP rate, and high recall relates to a low FN rate.

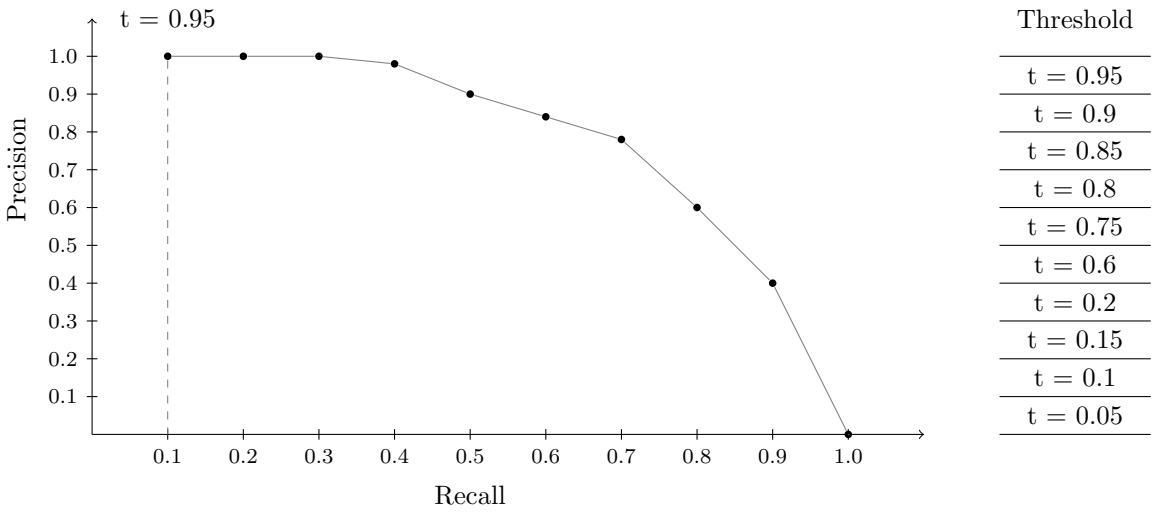


Fig. 2.3: Precision-recall curve with threshold values

- Average Precision (AP) is a metric used to summarize the precision-recall curve for a specific class in an object detection task (Padilla et al., 2020). It represents the average of precision values across different thresholds. For each precision-recall pair, the area under the precision-recall curve can be found by approximating the curve using rectangles, as shown in the equation below, where P_n and R_n are the precision and recall at the n-th threshold.

$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n \tag{2.6}$$

- For multi-class tasks, the Mean Average Precision (mAP) is calculated by taking the average of AP across all the classes under consideration (Padilla et al., 2020).

$$\text{mAP} = \frac{1}{k} \sum_i^k \text{AP}_i \tag{2.7}$$

2.3 Computer vision

Computer vision is a field of artificial intelligence that focuses on enabling computers to interpret and understand visual information from the world (BJ et al., 2024). Key tasks of computer vision include image processing, object detection, image segmentation and so on (Szeliski, 2022). Visual understanding

in robotics is a critical component that enables robots to perceive, interpret, and interact with their surroundings. It involves algorithms for acquiring, processing, analyzing, and understanding images or videos (Van Eden and Rosman, 2019). RGB-D data is normally utilized in robotics, which consists of two key components, an RGB (Red, Green, Blue) color frame and a depth frame (Lopes et al., 2022). The RGB color frame provides visual information in standard color channels. Each channel has values ranging from 0 to 255, which represents the intensity of the color (Gonzalez, 2009). The depth frame offers spatial details by indicating the distance of objects from the camera. The value range of a depth frame depends on the specific technology. The depth frame can be aligned with the color frame to obtain the depth value from the specified pixel coordinates.

In this thesis, the focus is set on object detection from RGB images, which combines classification and regression tasks. The classification aspect of the object detection identifies the categories of objects present in an image, while the regression aspect locates each object within the image. Classical approaches for object detection typically involve developing a feature representation and training classical machine learning algorithms, such as support vector machines (Kienzle et al., 2004). Deep learning-based approaches have significantly improved object detection performance in recent years. The availability of large annotated datasets and the development of advanced model architectures have played a critical role in improving object detection accuracy. The annotations for object detection are normally represented by class labels and 2D bounding boxes with pixel coordinates at top left (x_1, y_1) and right bottom (x_2, y_2), as illustrated in fig. 2.4.

Faster-RCNN (Region-based Convolutional Neural Network) (Ren et al., 2015a) and YOLO (You Only Look Once) (Redmon et al., 2016) are two popular architectures for object detection. Several pre-trained models are available as open source and can be used directly or fine-tuned for new objects. After identifying the location of an object in the RGB color frame, the depth value can be calculated from the corresponding depth frame. This resulting 3D coordinates can then be used to provide the necessary spatial information for a robot to perform actions.

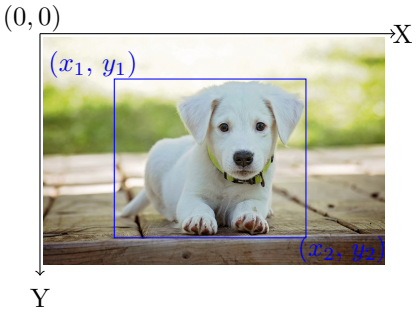


Fig. 2.4: Illustration of object detection output on an RGB image

2.4 Natural language processing

According to Hirschberg and Manning (2015), Natural Language Processing (NLP) is a subfield of computer science that uses computational techniques to learn, understand, and produce human language content. NLP systems include several stages in processing linguistic data, including pre-processing, feature extraction, model building, and response generation. Natural Language Understanding (NLU) is a specialized area in NLP that focuses on comprehending linguistic input, including tasks like intent recognition, named entity recognition, or semantic analysis. This section provides an introduction of common NLP tasks for better understanding their applications in robotics.

2.4.1 Word tokenization

A tokenizer is a crucial component in NLP. This process involves splitting a text into individual sentences, followed by breaking each sentence down into smaller units called tokens. Sentence segmentation is a critical step in text processing, relying primarily on punctuation cues such as periods, question marks, and exclamation points to identify sentence boundaries. Once the text is segmented into sentences, each sentence is further divided into tokens, which are typically individual words or meaningful elements (Jurafsky and Martin, 2024).

2.4.2 Part-of-speech tagger

A Part-of-Speech (POS) tagger assigns parts of speech, such as nouns, verbs, adjectives, adverbs, etc., to each word in a given text. This process is a classic technique in NLP for understanding the grammatical structure and meaning of sentences. One of the key challenges in POS tagging is the inherent ambiguity of natural language. Many words can function as different parts of speech depending on the context. For example, the word *run* can be a noun ("a morning run") or a verb ("to run fast"). Modern POS taggers utilize contextual information to disambiguate such cases.

2.4.3 Syntactic dependency parser

A syntactic dependency parser is a tool to analyze the grammatical structure of a sentence and establish the relationships between "head" words and words which modify those words. An example is shown in fig. 2.5. The word "pick" is dependent on the word "block". The edge "dobj" indicates that the type of dependency is "direct object". The edge "amod" stands for "adjective modifier", which modifies the meaning of noun phrase like the word "block" in the example. The edge "det" refers to "determiner", which is the relation between the head of noun phrase and its determiner. More labels representing syntactic dependencies can be found at https://universaldependencies.org/.

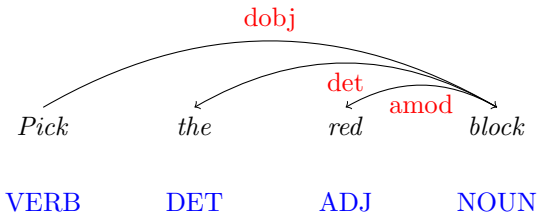


Fig. 2.5: Results of part-of-speech tagger (blue) and syntactic dependency parser (red)

2.4.4 Coreference resolution

Coreference resolution in NLP refers to the task of determining when different expressions in a text refer to the same entity (Sapena Masip et al., 2008). This

typically involves identifying pronouns, noun phrases, or other references that refer to the same person, place, object, or concept within a document. For example, in the sentence "John went home because he was tired", the pronoun "he" refers to "John". This task allows for more accurate extraction of information from text. Various methods are used to achieve this, ranging from rule-based systems to advanced machine learning and deep learning models (Liu et al., 2023b).

2.4.5 Named entity recognition and relation extraction

Named Entity Recognition (NER) is a task of Information Extraction (IE) to identify and classify some types of information elements, called Named Entities (NEs) (Marrero et al., 2013). They are typically noun phrases and comprise one to a few tokens in the unstructured text, such as locations and organizations. Relationships are defined over two or more entities related in a pre-defined way, such as "location of organization" relationship (Sarawagi, 2008). Both NER and Relation Extraction (RE) are typically considered classification tasks, which are solved by training deep learning models, such as Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018). It is pre-trained on a large corpus of text and can be fine-tuned for specific tasks.

2.4.6 Word sense disambiguation

The same word can have multiple meanings depending on the context. For example, "mouse" can refer to either a small rodent or a computer input device. There are two main approaches for word sense understanding: using databases like WordNet and using word embeddings. WordNet is a comprehensive lexical database and available in multiple languages. It captures these various meanings by organizing words into sets of synonyms known as "synsets". Each synset is equipped with concise definitions and usage examples, and it documents various semantic relationships among these sets. For instance, the noun "meal" is semantically subordinate to "lunch". Detailed insights into these relationships are further discussed in the book by Jurafsky and Martin (2024). By leveraging the semantic connections and synonym sets provided by WordNet, algorithms

are equipped to identify the intended meanings of words based on their usage in specific contexts.

Word vectors, also known as word embeddings, are a way to represent words using real number vectors, which capture their meanings and relationships in a way that computers can understand. The most common method for creating word vectors is based on the distributional hypothesis, which suggests that words that occur in similar contexts tend to have similar meanings. Techniques such as Word2Vec (Rong, 2014) and Global Vectors for Word Representation (GloVe) (Pennington et al., 2014) can generate these representations.

Word2Vec models are shallow, two-layer neural networks used to reconstruct the linguistic contexts of words. They take a large corpus of text as input and produce a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. They generate word embeddings by looking at a local context window. The models are particularly known for their ability to capture semantic relationships between words. For example, it is well-known for its capability to perform arithmetic operations with word embeddings, such as subtracting “Man” from “King” and adding “Woman” to get an approximation of “Queen”. Instead of using a local context window, GloVe constructs an explicit word-context or word-co-occurrence matrix using statistics across the whole corpus.

With the emergence of deep learning models such as BERT (Devlin et al., 2018) and GPT (Yenduri et al., 2023), the field has shifted towards contextual embeddings. This approach allows for a word’s representation to change based on the sentence it appears in, addressing the limitations of traditional word vectors by incorporating a deeper understanding of language context. Deep learning models, especially those pre-trained on large datasets, often outperform methods based on hand-crafted linguistic features.

2.4.7 Learning language models

Learning language models is a fundamental task in NLP. Language models are probabilistic models that predict the likelihood of a sequence of words. They form the backbone of many NLP applications such as machine translation,

speech recognition, and text generation. Some commonly used architectures are discussed as follows.

Recurrent Neural Networkss (RNNs) are a class of artificial NNs designed to recognize patterns in sequences of data, such as time series, speech or text. Each step updates a hidden state that stores information about previous steps, allowing the network to maintain a memory of past information. It can be applied for tasks such as POS and NER to capture the dependencies in data (Jurafsky and Martin, 2024).

LSTM is a specialized type of RNN designed to address some limitations of standard RNNs, particularly in capturing long-term dependencies. GRU is a variation of LSTM to solve the issue of vanishing gradients.

Recently, transformer architecture based Large Language Models (LLMs) have demonstrated exceptional capabilities in various NLP tasks. The models have been trained on large amounts of text data to understand and generate human-like language, which typically have millions to billions parameters. BERT (Devlin et al., 2018) is a transformer model pre-trained on a large corpus of English data in a self-supervised fashion. One of its pre-training objectives is Masked Language Modeling (MLM), where 15% of the words in a sentence are randomly masked. The model then predicts the masked words by processing the entire masked sentence. This pre-training method enables BERT to understand language contextually from both directions (left and right of a token). The model is primarily designed to be fine-tuned for specific downstream tasks like text classification, sentiment analysis, and question-answering. For pre-training, BERT used BookCorpus (containing $11,038$ unpublished books) and English Wikipedia (excluding lists, tables, and headers). Other popular LLMs include GPT, Jurassic-1 Jumo (Lieber et al., 2021) and Megatron-Turing NLG (Smith et al., 2022).

Fine-tuning and prompting are two methods used to adapt LLMs for specific tasks. Fine-tuning was discussed in section 2.2, which involves training the pre-trained LLM on a specific dataset for a specific task, and updating the model's weights to optimize performance for that task. Prompting involves crafting specific input prompts to guide the pre-trained model to generate desired responses, without changing the model's weights (Liu et al., 2023a). The

prompting process requires skill in designing effective prompts.

2.5 Robot programming

Classical robot programming follows procedural programming paradigms. It requires creating a sequence of instructions that includes initialization, motion sequences, decision logic, and error handling to form a task plan. At each motion sequence, the robot must move the end-effector to the desired position. This can be accomplished by commanding the joints or the end-effector.

In the kinematics of robotics, a body refers to a link of a robot. Each link between joints is a distinct body that moves in relation to other parts. Given a frame with an origin and three mutually orthogonal basis vectors, a body can be represented with position and orientation in this frame. In robotics, the term pose refers to the combination of position and orientation in space. To define a body's pose, a relative frame is required. Fig. 2.6 shows a rigid body with frame i relative to frame j and k. Given the pose in one relative frame, it can be transformed to another frame using appropriate coordinate transformations. The position of the origin of the frame i in frame j or k can be denoted by the vector $\mathcal{P} = (x, y, z)$. In particular, the coordinates of $\mathcal{P}$ vary depending on the reference frame chosen. Similarly, the orientation of the body can be described using $\mathcal{O} = (\alpha, \beta, \gamma)$ or $\mathcal{O} = (R_x, R_y, R_z)$, commonly known as roll-pitch-yaw angles or Euler angles. They refer to the angles in a sequence of rotations about axes fixed in the space frame. Other common representations of orientation, along with a more in-depth discussion on robotics kinematics can be found in Lynch and Park (2017).

A robot can be moved in configuration or task space. Configuration space refers to the space of all possible configurations of joint coordinates. $\Theta = (\theta_1, \theta_2, \ldots, \theta_i)$, where i represents the number of joints and θ represents the joint angle or position. 6- or 7-axis robot arms are commonly used in cobots. To move the robot in configuration space, the desired joint angles have to be specified. The resulting pose of robot's end-effector in world coordinates is determined using the robot's forward kinematics. This is useful for precise control over the robot's movements. Task space, also known as Cartesian space, is a space in which the

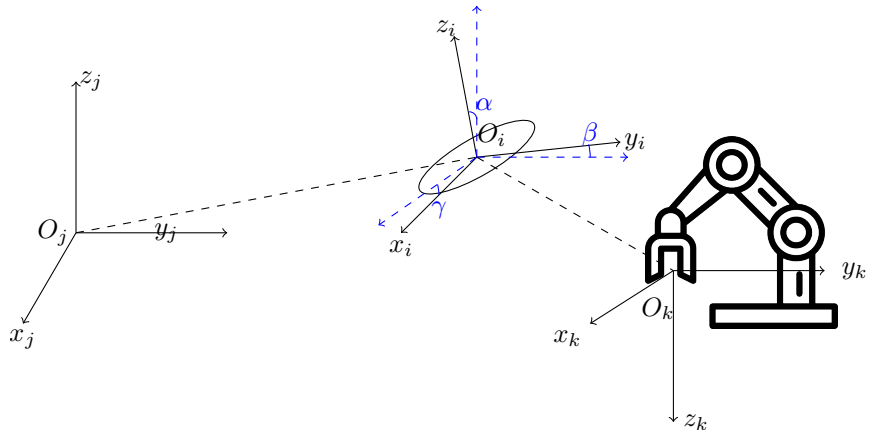


Fig. 2.6: Pose representation of rigid body O_i in Euclidean space with respect to two reference frames O_j and O_k, using XYZ roll-pitch-yaw angles

robot's task can be naturally expressed, such as the position and orientation of a frame attached to the robot's end-effector. To move the robot in task space, the desired position and orientation of the robot's end-effector are specified. The robot's inverse kinematics can then be used to calculate the corresponding joint angles or positions required to achieve that end-effector pose (Lynch and Park, 2017).

An ordered set of poses forms a path, as defined in ISO 1838:2021. Creating a path involves defining a sequence of waypoints in either configuration space or task space. The robot controller provides basic path elements, such as linear or circular paths. These waypoints represent intermediate poses that the robot must follow to reach a desired target. A trajectory is a path in time that is generated by considering the speed and acceleration between the waypoints.

Online and offline programming represent two fundamental approaches in robotics (Haun, 2013). Online programming involves direct interaction with the robot, where programming is performed in the actual work environment using the robot's Teach Pendant, providing immediate feedback but potentially interrupting production. An illustration of online robot programming is shown in fig. 2.7.

Fig. 2.7: Illustration of online robot programming with a Teach Pendant

In contrast, offline programming (OLP) refers to the method where the task program is defined on devices separate from the robot for later entry into the robot controller (ISO1383:2021). It is typically conducted in a virtual environment using specialized software, allowing for the detailed planning, simulation, and optimization of the robot's tasks without interfering with its operational environment (Pan et al., 2012). This method is particularly effective for complex tasks, such as an assembly line with more than ten robots. Each robot is assigned to a single step for an extended period. In the final phase of OLP, it usually requires some online interaction for final implementation (Heimann and Guhl, 2020). This combination of offline planning with online refinement ensures that the programmed robot performs optimally and safely in its intended setting, balancing the efficiency of simulation with the precision of real-world calibration. The choice between these methods depends on the specific needs of the task. The focus of this work is online robot programming.

Robot programming is typically associated with the manufacturer of the robot, with major companies such as ABB, KUKA, and YASKAWA each offering their own proprietary programming languages and tools. ABB uses RAPID language and RobotStudio software, KUKA uses KUKA Robot Language (KRL) along with KUKA.Sim and KUKA.OfficeLite for simulation and programming. The YASKAWA's Motoman robots are programmed using INFORM language, supported by MotoSim EG-VRC for virtual programming. Specialized systems are designed to provide precise control and high efficiency for their respective robots. However, they come with a learning curve and are typically optimized for compatibility with their own brand's hardware. This supplier-specific approach to robot programming further stresses the significance of specialized knowledge and skills in industrial robots.

Chapter 3

Related work

This chapter presents the literature related to this work and the contributions of this thesis. The related work is divided into four main sections. The first section provides a general overview of non-expert robot programming. This is followed by a detailed examination of selected non-expert programming methods in sections 3.2 to 3.4. Specifically, the second section presents work on robotic programming by demonstration. The third section introduces state-of-the-art approaches for robot programming by natural language. The fourth section introduces frameworks that integrate multimodal input for robot programming. The related works are then summarized and discussed in the fifth section. Finally, the last section summarizes the innovative contributions of this work.

3.1 Non-expert robot programming

Non-expert robot programming is the process of making robot programming accessible to individuals without a specialized background in robotics. This aspect is crucial for expanding the use of robotics across various tasks in the manufacturing scenario. This dissertation focuses on integrating semantics into PbD. Before going into details, the general approaches for non-expert robot programming are discussed in this section. The concept of increasing the abstraction level, simplifying interaction and the application of RL techniques are three main research directions. These approaches are not mutually exclusive and can complement each other in making robot programming intuitive and

accessible. Specific works will be discussed in section 3.2 to 3.4.

3.1.1 Increasing the level of abstraction in robot programming

Abstracting common functionalities into components can reduce programming time and enhance the reusability of existing programs, particularly in dynamic environments. Werling (1993) classifies these abstractions into five abstract levels: joint level, arm level, robot level, object level and task level. The most basic, the joint level, focuses on the individual joints of a robot. The arm level extends this by considering the motion of the robot's entire arm. At the robot level, movements are described through explicit robot commands, such as "move to point $\mathcal{P}1$". The object level specifies actions in relation to objects the robot manipulates. At the highest level, task-oriented abstractions use natural language to describe complex tasks in an intuitive way.

The main contribution of this subsection is the modification of abstraction levels to align with the recent developments in robotics research. Specifically, two key changes to the approach by Werling (1993) are introduced as follows. Firstly, the robot and object levels are merged into a unified skill level, such as "pick an object" (Hammerstingl and Reinhart, 2018). By introducing the skill-level, task-level programming can be realized by combining different skills into workflows to achieve specific goals (Heuss et al., 2022). For instance, a task may involve assembling a product by defining steps such as picking up a base component, attaching another component to it, and then placing the assembled product on a conveyor belt. Additionally, the goal level has gained attention recently due to the development of computer vision, where users demonstrate the goal scene through images, see Zeng et al. (2018). The modified approach is summarized in table 3.1. Among these abstraction levels, programming at the joint and arm levels requires expertise in robotics and is not considered as non-expert programming.

3.1.2 Simplifying interaction

Several methods have been developed to simplify human-robot interaction and make robot programming more accessible, particularly for non-expert users

Table 3.1: Modified abstract levels for robot programming. Adapted from Lu et al. (2025), licensed under CC BY 4.0.

Abstract level	**Description**	**Example**
5: Goal level	User demonstrates goal scene in the form of an image with two objects inside a box	
4: Task level	Abstract description of tasks	Task: Assemble a product Steps: 1. *Pick* up the base component from the bin 2. *Attach* component A to the base 3. *Place* the assembled product onto the conveyor belt
3: Skill level	Specification of the program flow in the form of explicit robot commands	*Move* to object at position $\mathcal{P}$
2: Arm level	Specification of motion with a sequence of Cartesian path points	X, Y, Z, R_x, R_y, R_z
1: Joint level	Specification of motion with a sequence of joint positions	$\Theta_1, \Theta_2, \Theta_3, \Theta_4, \Theta_5, \Theta_6$

(Laird et al., 2017). These methods include Graphical User Interfaces (GUIs), physical guidance, and the integration of advanced perception capabilities such as computer vision and natural language processing.

Recent research has focused on developing GUIs that simplify the programming process (Steinmetz et al., 2018; Berg, 2020). The GUIs enhance accessibility and intuitiveness, particularly in task-level programming where specific skills are programmed with adjustable parameters. The research results have been quickly transferred to the industry. An increasing number of cobot manufacturers are

offering touchscreen tablet teach pendants, such as Fanuc programming tablet illustrated in fig. 3.1.

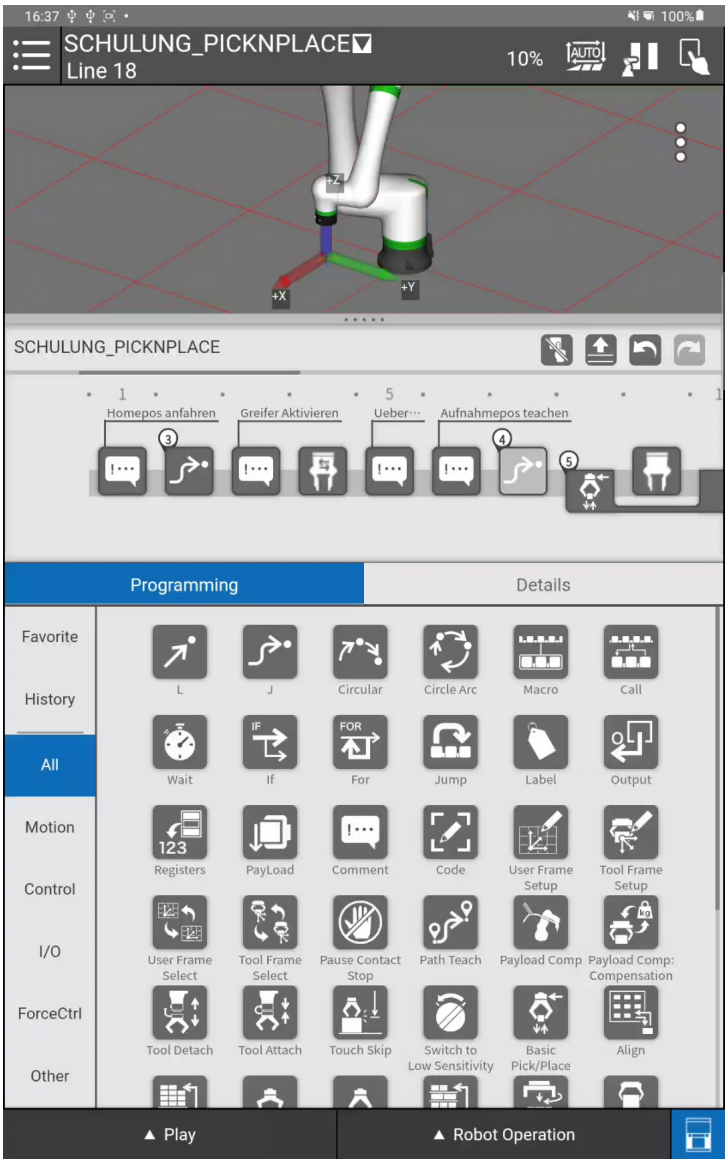


Fig. 3.1: Fanuc programming tablet

Another user-friendly interface for cobots is hand guiding. This interface allows users to physically guide the robot through desired motions or tasks, which the robot then replicates. This interface is an essential enabler for Programming by Demonstration (PbD), which is particularly useful in industrial settings where non-expert users need to quickly and efficiently teach robots new tasks (Saveriano, 2017).

Furthermore, the integration of cameras into robot programming system has significantly advanced the field. Computer vision algorithms enable robots to detect and recognize objects in their environment, facilitating interaction based on visual cues (Finn et al., 2017; Xin et al., 2023). In addition, robots can perceive and understand human gestures, expressions, and actions (Berg and Lu, 2020). By leveraging computer vision, robots can understand their spatial context and interact with objects and people more effectively (Zeng et al., 2018). Meanwhile, natural language processing enables robots to comprehend and respond to human language commands by analyzing syntax and semantics (Matuszek et al., 2013). This allows users to communicate with robots naturally, making interactions more intuitive and accessible.

Overall, these advancements in robotics, including GUIs, hand guiding, computer vision, and natural language processing, are enhancing human-robot interaction and making robot programming more accessible and efficient.

3.1.3 Reinforcement learning

Reinforcement Learning (RL) is another technique for non-expert robot programming. It is a type of machine learning in which an agent learns to interact with its environment with the goal of maximizing the rewards it receives in the long run (Elguea-Aguinaco et al., 2023). The RL technique allows robots to learn tasks and behaviors through trial and error. Some RL approaches integrate user feedback directly into the learning process. For example, a non-expert user can provide corrective feedback or rewards to the robot, which the robot uses to adjust its behavior. This form of learning is known as Interactive RL (Lin et al., 2020). It can significantly ease the burden of non-experts in programming robots. The role of environment modeling in RL is crucial because it influences the robot's ability to learn effectively. Both simulation and real-world environments can be used to train RL algorithms.

Zhou et al. (2023) collected training data using a real-world setup, which consists of a table-mounted 7-DOF robotic arm, a parallel gripper and two Intel RGB-D cameras. Four table-top manipulation tasks are considered: *reach*, *lift* and *pick & place*. To perceive the object to be manipulated, AprilTags were

attached to the object to detect its position. They are visual fiducial markers that encode a unique ID and can be robustly detected by a camera to estimate an object's position and orientation. The dataset consists of $6,500$ trajectories, requiring 270 hours of human work. Experimental studies were performed in this work to evaluate the learned policy with changing environments. The result has demonstrated the advantages of using real-world data to train the RL models.

Nevertheless, the majority of research in the field of robotics employs a simulation environment to train the algorithms, which are referred to as offline learning. The process of building up these training simulations is a labor-intensive process that demands substantial engineering expertise. Once the foundational offline learning is complete, the algorithms can be incrementally improved through online learning in real-world scenarios. However, ensuring the safety of RL deployments in real-world scenarios is another significant challenge. Ibarz et al. (2021) summarized the challenges for training process of different RL algorithms, noting the complexities and technical demands of each stage of the training process.

3.1.4 Summary

This section explored key approaches to non-expert robot programming, focusing on three primary research directions: increasing the abstraction level, simplifying interaction, and applying RL techniques. To refine the classification of abstraction levels in alignment with modern robotics research, this thesis modifies Werling's approach by combining the robot and object levels into a skill level and introducing an additional goal level. Increasing levels of abstraction in robotic programming lead to deeper integration of semantics. As abstraction levels rise, robots must not only execute tasks but also understand commands, make context-aware decisions, and adapt to dynamic environments. This shift underscores the importance of semantics in enabling robots to bridge the gap between mechanical execution and intelligent behavior. In the subsequent review of related works, each approach will be analyzed in terms of its level of abstraction.

Simplifying interaction enables semantic integration by intuitive human-

robot interaction. Advancements in various interaction technologies such as GUI, hand guiding, NLP, and computer vision enable users to define tasks at higher levels of abstraction (e.g., skill or task level) rather than focusing on low-level execution details. This shift allows robots to interpret user intent and understand contextual information.

While RL techniques allow robots to learn behaviors through trial and error, their reliance on extensive training data and simulation environments makes them less practical for rapidly changing industrial settings. Increasing the abstraction level and simplifying interaction offer more effective solutions, as they facilitate quicker adaptation and reduce setup complexity. The following sections provide a detailed discussion of related work within these research directions.

3.2 Robot programming by demonstration

Robot PbD, also known as learning from demonstration, is a technique used in robotics where robots learn to perform tasks from human demonstrations of those tasks. This approach was developed with the motivation to simplify the interaction. Initially, the research field focused on physical interaction, where humans guided robots directly, manipulating their arms and tools to perform specific tasks. Over time, the field expanded to include observational learning, where robots learn from visual data. The related work of robot PbD is categorized into kinesthetic and visual PbD and discussed below.

3.2.1 Kinesthetic programming by demonstration

Kinesthetic PbD is a framework that involves teaching a robot how to perform a task by physically guiding the robot itself or an attached device to the flange. The learning process can be divided into three steps, as shown in fig. 3.2. In the first step, users can demonstrate the desired task through direct interaction, such as physically moving a robot or manipulating an input device. During the demonstration, the user can specify whether joint or Cartesian coordinates should be recorded. In the second step, a representation of the recorded trajectories

is developed to generate the learned skills in dynamic environments. The development of motion representation methods from kinesthetic PbD is one of the key research topics in the last decades. The raw data from the demonstrations are first processed into trajectories that the robot can follow. This involves smoothing the motion, filling in gaps, and sometimes optimizing the paths for efficiency and safety. The next step is to represent the trajectories at the skill or task level. During execution, the learned representations or algorithms should be adaptable to new task requirements, such as initial position and target position. The representations are discussed below at the skill and task levels, respectively.

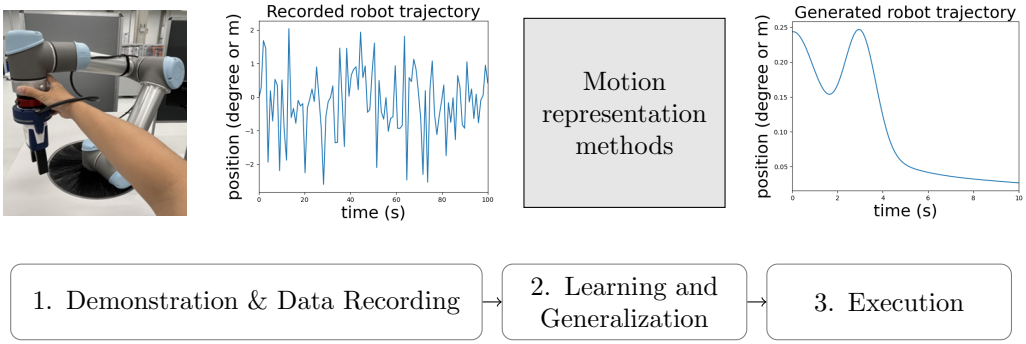


Fig. 3.2: Illustration of kinesthetic programming by demonstration framework

Mülling et al. (2013) proposed an approach to learn striking motions in table tennis at skill level. These movements are demonstrated 25 times and all started from the same initial position of a Barret WAM arm with seven axes. The movements are represented as Dynamic Movement Primitives (DMPs) in joint space, which are formalized as stable nonlinear attractor system allowing final velocities. The parameters of DMPs are learned by locally weighted regression. It is possible to learn DMPs from a single demonstration (Kyrarini et al., 2019). However, table tennis is a complex motor task that requires multiple demonstrations to overcome singular events. During execution, the mixture of motor primitives (MoMP) is proposed. MoMP computes parameters including the temporal and spatial interception points of the ball and the racket, as well as the velocity and orientation of the racket at the moment of hitting the ball. The robot's motion in the form of DMPs is then generated based on these computed

parameters. The approach enables the robot to adapt its behavior dynamically to the specifics of the game situation.

Calinon et al. (2007) presented an approach to represent the demonstrations as Gaussian Mixture Model (GMM). The demonstration was recorded using the arms and torso of a humanoid robot. The robot was taught by the demonstrator moving its two arms and the torso, which has 11 joints. During demonstration, the human was standing behind the robot. The movements of joint angles were recorded at a rate of 1000 Hz. The robot was taught the task by an expert user, who demonstrated it four to seven times. Based on this, the robot learned GMMs to represent the manipulation tasks. Gaussian Mixture Regression (GMR) was then applied to generate motion during the execution. The evaluation showed that the GMM-GMR approach was effective in modeling the shape of the demonstration trajectories. However, these results cannot be generalized to new starting and ending positions. To address this challenge, Pervez and Lee (2018) proposed a method for learning a task parameterized DMP with GMM. This approach enables the learned trajectories to be generalized to new task settings.

Meyer (2011) developed an approach to program the industrial robots for welding tasks. The robot is guided through a haptic interface using force control. Additional safety sensors were integrated into the programming system. The operator grasps the guiding handle and moves it in the desired direction. The forces applied by the operator are measured by the force-torque sensor and translated into movements. The end-effector trajectories of the robot were recorded, and the local geometry of the working pieces was acquired using a laser scanner. The programmed trajectories were generated from the optimized inputs of both end-effector trajectories and the laser scanner. The experiments show that the programmers could reduce the required programming time by up to 40% using the developed programming system.

One drawback of kinesthetic PbD is its dependence on the specific type of robot used. Transferring skills learned on one robotic platform, such as the Barrett WAM arm with 7 axes, to another with different specifications, like a six-axis UR robot, can be complex. This complexity arises because the kinematic and dynamic properties of the robots may differ significantly, affecting how movements and skills are executed. For example, a robot with fewer degrees

of freedom may not precisely replicate the movements of a more flexible robot, requiring adaptations or modifications in the learned skills.

The task level learning in kinesthetic PbD is concerned with understanding and generalizing multi-step manipulation tasks. A task is normally recorded in one continuous session. This approach saves time spent on demonstrations and more closely resembles the fluid sequence of actions typically employed by humans. To facilitate task-level learning, different types of symbolic plans are employed to encode the knowledge gained from demonstrations. These plans serve as a framework to guide the robot's actions in dynamic environments, where they are typically represented using symbolic notation to outline when each action should start and end relative to other actions. The research focuses on developing such a framework for learning the symbols, with an emphasis on developing representation and segmentation methods to divide the continuous demonstration into distinct actions or subtasks.

Kyrarini et al. (2019) developed a PbD framework for a two-arm robot to learn the sequence of actions for an assembly task. Each arm is a UR10 six-axis manipulator, which can be controlled by the gravity compensation controllers. The framework consists of two learning modules: high-level task learning of the action sequences and low-level trajectory learning. During demonstration, the joint angles and the end-effector pose of both robotic arms and the status of the grippers were recorded. In high-level task learning, tasks are divided into subtasks. Each subtask is then segmented into actions. The subtask refers to the process of handling a single object, which consists of a sequence of actions, including *start arm moving*, *grasp object*, *release object*, and *stop arm moving*. The GMM-GMR approach is then applied to learn and generate the trajectories for each action. However, the work did not provide an outline of the segmentation algorithm.

Steinmetz et al. (2019) introduced a system for task-level PbD, which extracts a sequence of skills. Users demonstrate tasks by kinesthetic PbD, and parameterized skill sequences are automatically identified online with a semantic skill recognition algorithm. The algorithms compare the changes in the world with a pre-defined model to identify the skills. The resulting skill sequences are sent to an interface, where a task plan for execution is formulated. The

evaluation shows that the approach reduces the programming time while at the same time being more intuitive, reducing mental load, and achieving the same or even better skill sequences.

Eiband et al. (2023) proposed a framework that recognizes skills online during a kinesthetic PbD by means of position and force-torque sensing. The recognized skill sequence constitutes a task representation that lets the user intuitively understand what the robot has learned. The skills are defined as logic or classified. Logic skills are identified by contact or gripper signal. Classified skills, such as *move* and *process* are recognized by a trained support vector machine classifier. The approach was evaluated with approximately 100 demonstrations collected for each skill. The average recognition accuracy is 92.2%.

The continuous development in combining task planning with PbD represents a significant advancement in robotic programming methods. Task-level PbD is intuitive and increases abstract level. However, it can be physically demanding for operators, especially if they need to handle the robot frequently for extended periods. Moreover, the performance depends on the type of robot and the specific application.

Beyond research advancements, kinesthetic PbD has also been adopted in commercial applications. One such example is the TracePen™. It consists of two components: one component is attached to a robot's flange, and the other component enables users to move the first component and record the end-effector trajectories (Pantano et al., 2024). The recorded trajectories are subsequently processed for application on robots. Its primary objective is to replicate specific tasks. However, current research efforts are focused on developing more effective methods for representing motion, with the aim of improving the generalizability of recorded movements across a range of task contexts.

3.2.2 Visual programming by demonstration

Visual PbD refers to the process where a robot learns to perform tasks by observing and analyzing activities within its environment. This process employs visual inputs captured by cameras or other sensory systems to understand both human-performed and autonomously occurring actions. In the end, the robot

programs are generated based on the interpretation of the recordings. This section provides a summary of recent literature on robot learning from visual observation. Same as kinesthetic PbD, the abstract levels for related works are divided into skill level and task level. Given the advantages introduced by visual context understanding, the abstract level is extended to include the goal level. The three abstract levels are discussed separately below.

At the skill level, the focus is on the acquisition of individual motor skills or actions for basic manipulations. Finn et al. (2017) presented a visual imitation learning method that enables a robot to learn new skills such as *push* and *place* from raw pixel input. The demonstration was collected from the simulation environment OpenAI Gym (Brockman et al., 2016) with the MuJuCo physics engine (Todorov et al., 2012) by pre-defined robot trajectories. The policy is represented by CNN. The policy observation includes both the RGB image and the robot's joint angles and end-effector pose. A policy π is learned to map observations to robot actions. The approach integrates meta-learning with imitation learning, allowing a robot to reuse experience and quickly learn new skills from a single demonstration. In the evaluation, the robot was able to place a new object into a new container in new task settings. This is achieved through a two-level learning process: the inner loop makes task-specific adjustments, while the outer loop updates meta-parameters across tasks. However, the two-level training process is complex, and the learned model is adapting poorly to environment changes.

Dong et al. (2022) proposed an approach for teaching humanoid robots bi-manual tasks using human demonstrations captured with a motion capture suit and virtual reality trackers. The key component of this system is a learnable graph attention network model, which is trained with human hand trajectories and target object poses. This model serves as a trajectory generator that outputs Cartesian trajectories for the robot's end-effectors based on their initial poses or the initial pose of the object. The method aims to prioritize efficiency and adaptability, with the robot learning new skills in the domestic environment. Experiments were conducted on both synthetic data and real-world tests using UBTech Walker humanoid robot. The tests included tasks like hand waving, box holding, and box lifting. The results demonstrated the system's ability to

learn and reproduce both single-armed and bi-manual tasks effectively. However, motion capture suits are not practical for industrial applications. They impose additional requirements on human workers who need to demonstrate a new task.

Xin et al. (2023) developed a so-called IRMT-Net to predict the interaction region and motion trajectory from RGB-D images. The input of the developed model consists of motion trajectories, motion category and cropped object images. To generate motion trajectory from RGB-D images, the hand detection method proposed by Shan et al. (2020) is applied to extract bounding box from each RGB frame. The center coordinate of the bounding box and its depth value are taken as 3D hand coordinates. Faster-RCNN (Ren et al., 2015a) is used to detect objects. The cropped image for the detected object is obtained based on the bounding box with the highest detection score. A dataset is created by labeling the motion trajectories for videos in the Epic-kitchens dataset (Nagarajan et al., 2019) with $9,236$ videos. In the end, skills such as *pull open drawer* and *take cup* in kitchen scenarios are learned. The annotated dataset contributes to the further development of visual PbD for kitchen scenarios.

Wen et al. (2022) introduced a framework for learning task trajectories using a single demonstration video, captured by a statically mounted Photoneo 3D camera. This camera records at 10 Hz. It provides grayscale and depth images, allowing for detailed observation of the working environment. This approach follows a two-step process: retargeting and last-inch manipulation. In the retargeting step, the framework tracks virtual target objects to generate trajectories, which are then re-projected onto objects within the same category by establishing dense correspondence between the demonstrated virtual object and the real-world object. This allows the calculation of trajectories for the real-world object through relative transformation. The object tracker is trained with Computer Aided Design (CAD) files and applied to RGB-D images. The last-inch manipulation step uses a path planner to fine-tune the robot's movements, ensuring precise task execution. This work demonstrates the potential of using CAD files to reduce data collection efforts.

Just like kinesthetic PbD at the task level, research in visual PbD follows a similar approach, where the user demonstrates the whole task as a continuous process. The developed programming system is required to understand and

interpret the entire sequence of actions. Specifically, segmentation approaches are developed to divide the demonstration into various segments, each representing different skills required to accomplish the overall task. Additionally, the system is required to recognize these skills. Qiu et al. (2020b) developed such a system with observing human demonstrations by an ASUS RGB-D camera. The raw data are hand trajectories. During demonstration, a human worker performs an object handling task wearing a colored hand glove. This hand glove improves the accuracy and robustness for hand detection. The hand pose is estimated based on a deep learning model trained by 3D input data (Li and Lee, 2019). The human demonstration is then segmented and recognized by Hidden Markov Models (HMMs). They include *pick up*, *place* and *locate*. Notably, there is no consensus for defining the semantics of skills in the existing works. Qiu et al. (2020b) consider *pick up*, *place* and *locate* as skills. However, Kyrarini et al. (2019) define them as *start arm moving*, *object grasp*, *object release*. This discrepancy in definitions presents a challenge when comparing the performance of different approaches.

In addition to the understanding of action sequences, the availability of visual context allows for the consideration of spatial relationships between task-relevant objects. Ding et al. (2020) developed a learning strategy for pick & place tasks, in which the continuous human hand movement is tracked by Kinect V2. The human hand centroid trajectory in XYZ positions are first segmented into the approach-pick phase, the arrive-place phase and the withdraw phase by calculating the hand velocities. GMM-GMR is then applied to generate optimal demonstration trajectory. To understand spatial relationships, the sequence of pre-action and post-action images are recorded and processed. Classical computer vision algorithms are applied to match features to extract pixel centers of relevant objects. During execution, the algorithm finds the best match between the recorded data and the current scene. The resulted task model is a combination of the learned skills and the spatial configuration. This work illustrates the potential for the integration of additional semantics through the application of computer vision algorithms in the field of visual PbD.

At goal level PbD, the system observes demonstrations with the aim of identifying the ultimate objectives of tasks rather than the exact procedures

used. The key is to infer the reason behind actions, allowing the robot to modify its approach based on its capabilities or the environmental conditions. Zeng et al. (2018) proposed an approach to enable robots to understand and execute tasks by interpreting the goals from demonstrations provided by the users. The initial and goal scene are represented by RGB-D images. A method called Discriminatively-Informed Generative Estimation of Scenes and Transforms (DIGEST) was developed to generate scene graph from demonstrated images. The first step in the DIGEST method involves detecting bounding boxes in the RGB images. Once objects are detected, their pose is estimated using the depth information from the RGB-D images and existing object mesh models. The final step is to generate a scene graph. This graph represents the spatial and relational structure of the scene. It is built using inter-object relations *exist*, *clear*, *in* and *on* by calculating the object poses. Given the observation of the goal state of the world, the robot estimates the goal scene graph, and stores the desired inter-object relations by PDDL, which stands for the Planning Domain Definition Language (Aeronautiques et al., 1998). It is a formal language used for expressing planning problems and domains. The task planner gives a sequence of high-level pick & place actions. To pick an object, the robot receives a number of pre-computed grasping positions for the object, and uses Moveit![1] to determine which of these positions can generate a collision-free path. The learned task concepts are reusable for other robots. However, a new scene should be demonstrated with slightly different spatial relation.

In summary, visual PbD leverages the latest advancements in computer vision and machine learning to improve robot programming at multiple levels. These levels include the execution of basic skills, the understanding of complex task sequences, and the achievement of task goals. The task models learned by visual PbD are independent of the robot's kinematic and dynamic parameters, so they can be applied to different robots. Two primary methods are typically used to record the human demonstration: simulation-based and real-world approaches. In the simulation-based approach, robot joint or Cartesian end-effector trajectories are recorded alongside visual scene data in a simulation environment. In contrast, the real-world approach captures trajectories by observing actual hand

[1]https://moveit.ros.org/

movements or object motions using a camera directly positioned in the physical task environment. Both methods share common challenges, such as the need for precise goal definition, the variability of environmental conditions, and the availability of sufficient training data. Addressing these challenges is crucial for the effective implementation of visual PbD in manufacturing scenarios.

3.3 Robot programming by natural language

The application of natural language understanding to robot programming has been an active research area since the 1970s (Stenmark and Nugues, 2013). With the advancement of intelligent robots and NLP technology, there have been continuous research results and developments in this field. These developments are aimed at making robot programming more accessible. The field of research starts with the understanding of routing instructions for mobile robots and later for manipulation tasks. It typically works at the task level, i.e. the programming allows the robot to understand high-level commands that describe a task to be performed. Some research focuses on developing methods for composing task plans from atomic actions, whereas the majority of research focuses on developing methods for extracting information from natural language input.

To compose task plan from atomic actions, Wölfel and Henrich (2018) proposed an approach for defining complex robot tasks based on sensor input. The approach systematically links action verbs to robot motions, while taking into account available tools and highlighting the combinations of motion primitives that make up complex motions. For instance, the verb *cutting* is associated with the tool *knife* and requires the underlying motion primitives of *pressing* and *dragging* in a parallel combination. Similarly, using a cup to scoop involves a sequence of *lowering*, *rotating*, *rotating*, and *lifting*. This type of definition is useful for designing robotic systems that must interpret verbal instructions and translate them into physical actions involving tools. The approach was evaluated by a user study. In the first experiment, participants observed a simulated robot arm performing a set of tasks. After each task, they were asked to annotate all the verbs they would use to describe each robot motion they saw. In the second experiment, participants were given nine verbs and five motion primitives:

rotating, *dragging*, *lifting*, *lowering*, and *pressing*. They were instructed to draw each verb by combining different motion primitives. The results indicate that the mapping corresponds to common sense regarding similarities of motion primitive combinations. Although the method is theoretically scalable, in practice, scaling it to handle a vast array of tasks across different domains can be challenging. As the number of verbs, tools, and motion primitives grows, the complexity of managing and accurately linking these elements increases significantly.

Tenorth et al. (2010) proposed an approach for generating plans for household robots. This work focuses on mobile manipulation tasks such as navigating spaces and manipulating objects. The natural language instructions are transformed into formal, logic-based task plans as below:

```
(methodForAction
(set_table1 table1)
(actionSequence
(TheList action1 action2 action3)))
```

Each step `action` is an instance of an action concept like *PuttingSomething-Somewhere* with parameters. To extract the task plan, the process starts with instructions being converted into a tree structure. The tree has leaves, which are words with labels, POS tags, and synsets. This structure is then enriched with semantic insights drawn from two key sources: the WordNet lexical database and the Cyc ontology. The WordNet was discussed in section 2.4, and the Cyc ontology was detailed in Fellbaum (1998) and Matuszek et al. (2006). In WordNet, each word can have multiple senses, a word sense disambiguation method was developed to select one of them. With the Cyc ontological concepts resolved, the sequence of actions can be derived from the input. The evaluation results with data from ehow.com and wikihow.com demonstrate the viability of the proposed approach, with a successful translation rate of about 80% for 150 instructions analyzed. This approach effectively generates task plans by identifying the most likely pairs of `action` and `object`, along with their spatial relationships, to accurately represent each step of the task. This demonstrates the feasibility of applying logic-based representations to comprehend and execute tasks described in natural language. However, the approach relies on pre-defined

relationships to create structured task plans, which presents a challenge for application in new domains. Furthermore, the authors have indicated that the task plan is not sufficient for successful execution, and that the missing action parameters should be inferred.

Matuszek et al. (2013) learns a parser based on example pairs of English commands and corresponding control language expression for indoor navigation, such as "Go left to the end of the hall". The control structures are modeled in a logic-based Robot Control Language (RCL). 418 instructions for different maps were collected in the experiments. The parsing is performed using an extended version of the Unified Based Learner (UBL) (Kwiatkowksi et al., 2010). The cross-validation test of the extended UBL parser has a F1-score of 71.8%. A detailed explanation of the F1-score can be found in section 2.2.4.

To resolve vagueness and ambiguity in natural language, Nyga (2017) developed an interpreter for natural language instructions which is able to infer missing information pieces that are required to render an instruction executable by a robot. The application scenarios are assistive household and chemical laboratory tasks. The approach formulates the problem of instruction interpretation as a reasoning problem in first-order probabilistic knowledge bases. The output of the approach is a parameterized robot action plan. For instance, the instructions "Pour milk into a bowl" and "Pour the batter into the pan" can be recognized as the structured task plan "pour-from-container" as shown below:

```
pour-from-container(from :default (an object
                    (type container.n.01)
                    (contains (Pouring Source))
amount :default (Pouring Quantity)
to :default (an object
                    (type container.n.01)
                    (contains (Pouring Destination)))))
```

The data from wikihow.com were annotated and used to evaluate the proposed framework. The experimental results demonstrate that the framework can outperform state-of-the-art classification systems.

The research works discussed above have applied rule- or template-based

methods to extract the procedural knowledge for robotic task planning. The advances in Neural Networks (NNs) have motivated researchers to acquire the knowledge directly from natural language sources. Zhou et al. (2019) applied the LSTM networks to encode the task titles, step gists and step explanations for the household tasks from wikihow.com. The gist is a brief and concise summary of the step, such as "purchase packing supplies". The corresponding explanation gives additional contexts and details to the gist. The trained models have 91.1% for step relevance and 75.2% for step ranking accuracy respectively. However, the resulting steps can not be executed by robot directly.

Recently, the LLMs demonstrated promising performance in the generation of program code designed to solve a given task. Singh et al. (2023) introduced a prompting scheme that utilizes Pythonic program structures with comments to represent robotic task plans. The plan functions consist of API calls to action primitives, comments to summarize actions, and assertions for tracking execution. A dataset with 70 household tasks were created to evaluate the approach. The robot is able to generate task plans in Python following prompt engineering. However, the action may not be valid for each step. The authors propose that human verification is necessary to ensure the validity of the action.

It is important to note that the availability of datasets to facilitate the understanding of natural language instructions is a crucial aspect of the development of robot programming systems. The challenge of dataset availability in domains beyond household tasks represents a significant obstacle. While sources such as wikihow.com offer comprehensive, structured data for household scenarios, similar high-quality, accessible datasets are often lacking in other specialized fields. Some work focuses on the collection of datasets for research purposes. Scalise et al. (2018b) introduced a dataset of natural language instructions for object reference in robot manipulation scenarios. A total of 1582 individual written instructions were collected by online crowdsourcing, which are used to specify objects of interest or target objects in a collaboration tabletop manipulation setting. The sentences show that people often use color, geometry and spatial relations to describe objects.

In summary, natural language provides a powerful and intuitive way to integrate semantics into robot programming, especially at the task level. A

majority of related work represent a task plan through sequences of actions and objects and their associated context. The development of NLU for robotic task plan generation consists of three main components: 1) Data collection; 2) Knowledge representation; 3) Development of NLU framework to extract the representation. With advancements in deep learning, the information extraction methods have evolved from rule-based to LLMs or embedding-based methods. Due to the limited availability of datasets, existing research work is limited to general purpose tasks and therefore not easily adaptable to new domains such as pick & place tasks in manufacturing scenario.

3.4 Robot programming by multimodal demonstration

Recently, the trend in robotics research shows an increasing focus on developing frameworks that integrate multiple input modalities, such as language and vision. Researchers aim to create more intelligent and adaptable robotic systems. Existing frameworks are classified into end-to-end and modular approaches, discussed as follows. Generally, these frameworks require multimodal input to capture the task context and human actions. The end-to-end approach processes all inputs together in a unified manner, while the modular approach processes each input independently before combining the information into a single task representation.

3.4.1 End-to-end approach

Gubbi et al. (2021) proposed a two-stage pipelined architecture to perform spatial reasoning on the text and image input. For instance, given the text instruction “line up card 3 squarely above card 6”, the robot needs to use this along with the visual input from the camera to infer the *start* coordinate from where the robot must pick up the object and the *end* coordinate where the object must be placed. To address the research question, a language network Lang-UNet was proposed to predict the *start* (x_s, y_s) and end position (x_e, y_e). The input of the model includes a natural language instruction, the object

position, and sizes from the object detector. The dataset created by Bisk et al. (2016) and a synthetic dataset created by the authors were used to evaluate the proposed framework. The approach demonstrates the advantages to manipulate objects based on natural language instructions together with visual input.

Shao et al. (2021) developed a framework which allows a robot to learn manipulation concepts from human visual demonstrations and natural language instructions. By manipulation concepts they mean for instance "put [something] behind/into/in front of [something]". The model's inputs are natural language instruction and an RGB image of the initial scene. The outputs are the parameters of a motion trajectory to accomplish the task in the given environment. Task policies are trained by integrated reinforcement and supervised learning algorithm. Instead of classifying all possible actions in video demonstration, the focus of this work is to extract motion trajectories from each video.

Lynch et al. (2023) presented a framework for robot learning from language and demonstration. In the first step, video demonstrations (RGB) and trajectories are collected by teleoperation. In the next step, the videos are relabeled with language instruction like "push the red start to the top center of the board" by defining the start and end frame in the demonstrated video. Robot policy is then learned with both video and language input with transformer-based architecture LAVA (Language Attends to Vision to Act) to generate action. In the execution phase, robot can solve goals with real time language instruction to achieve a high-level goal such as "put all the blocks in a vertical line on the right of the board".

Kalithasan et al. (2023) proposed an approach to train a model to learn a manipulation program from a natural language instruction and an input scene. The resulting manipulation program is a high-level task plan. The end-effector trajectory is generated by a motion planner. The dataset is collected in a PyBullet tabletop environment using a simulated Franka Emika Panda robot arm. The workspace consists of a tabletop with blocks of different shapes and colors. Each datapoint consists of an initial scene paired with a language instruction and the expected resulting final scene. For training, close to $5,000$ synthetic scenes and language instructions were sampled with three to five blocks of varying colors and shapes and placed at randomized orientations on the table.

Wang et al. (2023) proposed a multisensory perception approach to tackle the task of natural language instruction understanding for robotic manipulation, in which the robot coordinates its visual, tactile and auditory perception to fully understand the instruction and then executes the manipulation task.

3.4.2 Modular approach

While end-to-end approaches often require multiple demonstrations to learn a task model, modular approaches focus on executing task plans from a single demonstration by incorporating domain knowledge. Daruna et al. (2021) introduced a task generalization module for generalization of the demonstrated task plans to new execution environments. They focus on generalizing object-oriented primitive actions and assume that the actions fail for 3 types of reasons: 1) object not being found in the demonstrated location, 2) object not being present in the environment to perform action and 3) the environment lacking any object that can be used to perform action. They define three levels of reasoning about the task plan constituents to generalize the task plan: spatial, object and action reasoning. The approach was evaluated for household scenarios. The experimental results demonstrate that the integration of the domain knowledge can achieve one-shot task execution and improve the generalization ability of the demonstrated task plans.

3.5 Summary and evaluation of related work

This section presents a summary and evaluation of the related work presented. The first section presented three main research directions for non-expert robot programming: 1) Increase of the abstract level; 2) Simplifying interaction; 3) Application of RL techniques. Raising the level of abstraction supports the integration of semantic information by allowing users to describe tasks using high-level concepts rather than low-level programming instructions. This approach improves the accessibility of robot programming, as users can specify goals and actions in a manner closer to human understanding without the need for technical expertise. To simplify the interaction, efforts were made specifically

by integrating various interaction modalities, such as hand guiding, GUI, visual perception and natural language instruction. Additionally, the RL techniques have been discussed. They allow robots to learn from trial and error. In this work, both increasing the level of abstraction and simplifying interaction are considered in developing a semantic robot programming system.

The second section presented related works of PbD in detail. They were discussed in terms of different levels of abstraction for kinesthetic and visual PbD separately. Kinesthetic PbD involves physically guiding the robot to demonstrate a task, the resulting task model depends on robot kinematics and dynamics, leading to limited generalization across different robots. Moreover, multi-step pick & place tasks in kinesthetic PbD can be physically exhausting for human demonstrators. Visual PbD gains attention recently thanks to the development of computer vision technology. This approach employs visual data to facilitate programming, offering enhanced flexibility and abstraction compared to kinesthetic PbD. However, the approach often requires a large amount of training data. The majority of current research focuses on household scenarios, with limited attention given to manufacturing environments, where publicly available data is limited. The collection of demonstration data for each new task is not practical due to the significant time and effort required, which limits the scalability and real-world applicability of visual PbD in manufacturing scenarios.

The research on robot programming by natural language was discussed in the third section. The related work demonstrated the effectiveness of integrating semantics through natural language instructions, enabling users to interact with and program robots in a more intuitive and accessible manner. Typically performed at the task level, it enables users to command robots to execute predefined action sequences, integrating essential information about objects and task contexts. This method simplifies the programming process, making it accessible to non-experts and facilitating the efficient execution of complex tasks. Similarly to visual PbD, the household scenarios remain the primary focus. The data collected, and the models trained for these environments aren't directly applicable to multi-step pick & place tasks in manufacturing scenarios, mainly due to differences in the types of variability of objects involved.

Integrating multimodal input for robot programming gains attention recently,

as discussed in the fourth section. The majority of research in this field develops frameworks that train a deep learning model with both language input and visual demonstration input. The end-to-end systems integrate all processing into a single model, which makes them difficult to modify and interpret. The modular approach demonstrates the flexibility to integrate domain knowledge and to achieve one-shot task execution.

In summary, current research on integrating semantics with PbD reveals several challenges. The availability of training data and the ability to generalize across different environments and tasks remain significant hurdles. Despite advances in natural language and visual PbD, complex models remain difficult to interpret and modify, posing challenges for scalability and adaptation. In addition, a focus on household scenarios has led to task specialization, limiting solutions for diverse applications, particularly in manufacturing.

3.6 Focus of this thesis

As introduced in chapter 1, the objective of this thesis is to develop a semantic robot programming system to handle pick & place tasks in industrial settings. This system integrates visual PbD and natural language instructions to generate task programs for execution. The term semantic robot programming is selected to emphasize that the approach focuses on understanding and leveraging the meanings behind inputs, both at the task level and the goal level. This makes the robot capable of adjusting its actions based on the context during task execution.

Considering the research deficits outlined in section 3.5 and the specific challenges for application in manufacturing scenarios, the following research goals are identified for the development of the semantic robot programming system:

G1 Capability of adjusting its actions based on context during task execution
Following the research trend discussed in section 3.4, this thesis also focuses on integrating the decision of *what to do* with *how to do it* for robot programming. By combining these two aspects, the system can improve its adaptability to changing conditions during task execution.

G2 Reducing data collection effort
Existing approaches in household scenarios require substantial amounts of data. When tasks involve new work environments or unfamiliar products, additional data collection is necessary to retrain the model. However, this process is impractical for manufacturing scenarios.

G3 Minimizing training effort
Most of existing approaches require multiple demonstrations to learn a task model. The number of demonstrations will be reduced in this thesis to enable a more efficient programming process.

G4 Providing flexibility to improve and integrate new capabilities
The fields of NLU and computer vision evolve rapidly. The system will be designed with the flexibility to integrate these innovative approaches, allowing it to improve system performance, such as more robust object detection methods, more accurate language understanding, or advanced spatial reasoning capabilities.

To achieve these goals, the following hypotheses are formulated:

H1 Integrating NLU and visual PbD can enhance a robot's ability to perform complex tasks in dynamic environments.

H2 Utilizing existing data sources or implementing effective data generation methods will reduce the overall effort required for data collection.

H3 Effective approaches for integrating natural language instructions with visual PbD will minimize the training efforts required for the programming system.

H4 A modular framework will allow for flexible integration of new semantics.

Chapter 4

Conceptual design of the programming system

This chapter presents the concepts of the programming system developed in this thesis. It is the first part of the results of the Prescriptive Study (PS) presented in chapter 1. The chapter follows a structured approach, starting with an introduction to the methodological framework. Subsequent sections describe each phase and procedure in detail. This chapter establishes the basic principles and system components necessary for the development of a semantic robot programming system.

4.1 Overview of the methodological framework

The proposed programming system builds upon previous methodologies but extends them to meet the specific requirements of semantic robot programming, particularly the ability to adapt dynamically to changing task conditions.

A widely used reference model for transferring human skills to robotic systems was introduced by Dillmann and Knoop (2007). The approach starts with the recording of human demonstrations using perception sensors, such as cameras or motion capture devices. The recorded movements are then segmented into meaningful action primitives to extract relevant motion patterns. Following this, the system analyzes the demonstrated actions within the given context and background knowledge. The system then abstracts and generalizes the learned actions to improve adaptability to different scenarios. The processed actions are

then mapped to the robot platform for execution. Finally, a simulation phase is applied to evaluate and refine the extracted actions in a simulated environment.

While this approach provides a solid foundation, it has several limitations in the context of semantic robot programming. Specifically, it does not explicitly define how contextual background knowledge is represented or integrated. Moreover, it lacks mechanisms for processing and utilizing natural language input. Finally, the approach does not address the integration of information from both perception and language to generate a semantically informed task model.

To address these challenges, this thesis adopts the methodological approach proposed by Backhaus (2016), which provides a structured framework for defining task knowledge. While the full process is originally designed for assembly tasks, this work focuses specifically on its components for requirement analysis and task modeling.

The developed approach, illustrated in fig. 4.1, enhances the model of Dillmann and Knoop (2007) by integrating two key components from Backhaus (2016): requirements analysis and structured task modeling. Additionally, it introduces mechanisms for recording, processing, and integrating natural language input into the model, further enriching the representation and interpretation of task knowledge. The framework consists of four main phases:

1. Requirements analysis: Based on representative use cases, this phase identifies the essential constraints and objectives of the programming system, forming the foundation for clearly defined task scenarios. It ensures subsequent modeling and execution steps align with these system requirements.
2. Task modeling and representation: Task structures are formally defined, emphasizing adaptability to variations in the task execution environment. The task model captures and organizes the semantic information required for flexible robot programming.
3. Input acquisition and knowledge base integration: In this phase, task-related information is acquired through both visual demonstrations and natural language instructions. Simultaneously, a knowledge base containing domain-specific background information is integrated to support task interpretation and execution.

4. Information processing, task execution, and validation: A central component of the proposed approach is the processing of multimodal input to extract, abstract, and generalize task knowledge. This involves the segmentation and interpretation of both visual and linguistic information to construct a coherent and executable task model. In this process, the natural language input provides contextual and semantic cues that support and facilitate the extraction of relevant information from the visual demonstration. By leveraging these cues, the system reduces the complexity of visual information processing and enhances the accuracy of task interpretation. The resulting task knowledge is mapped onto the robotic platform, ensuring adaptability to different scenarios as defined in the requirements analysis. Finally, a validation phase is conducted to evaluate the generated task program, ensuring its correctness and completeness.

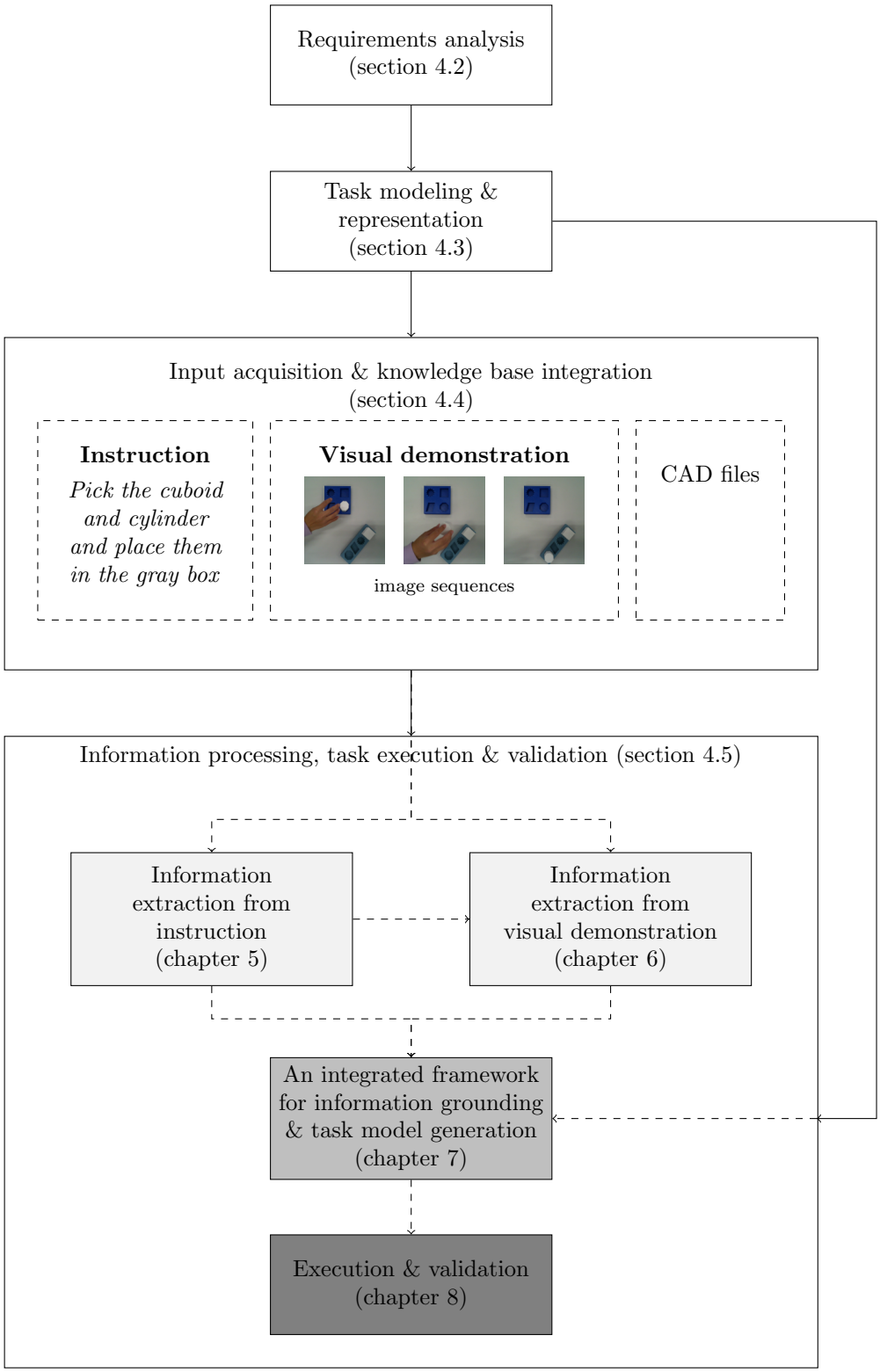


Fig. 4.1: Structured methodological approach for semantic robot programming

4.2 Requirements analysis

The goal of this work is to develop a robot programming system that uses high-level representations of tasks and knowledge. Instead of relying on low-level coding instructions, semantic programming focuses on capturing the meaning and intent behind human behavior. It attempts to bridge the gap between human understanding and robot control by using concepts and abstractions that are aligned with human cognition. To systematically analyze the requirements, this work follows the approach based on the analysis of use cases, which is widely applied to the development of programming systems, as demonstrated by Krug (2012) and Backhaus (2016). While their focus is on analyzing use cases in the context of offline programming, this work examines semantic robot programming from an online perspective.

Building on this foundation, this work introduces a structured three-step methodology. It consists of: 1) defining the scenario and the required robot capabilities, 2) outlining the initial programming phase using natural language and demonstrations, and 3) evaluating the robot's ability to adapt to task modifications without requiring reprogramming. The following two pick & place use cases illustrate the benefits and requirements of integrating NLU and PbD to enhance robotic performance in dynamic environments.

4.2.1 Use case C1: Flexible packaging machine

In the first use case, a robot operates within a flexible packaging system, as illustrated in fig. 4.2. Items labeled "A" and "B" arrive via a conveyor belt at Location 1. The system is designed to fill, seal, and package two different types of products. The robot serves as a critical link between sealing and packaging processes, with the ability to:

- Direct items to different locations based on customer requirements.
- Detect and recognize items "A" and "B" for appropriate handling.
- Adjust to variations in the number and type of items.
- Modify its actions without the need for reprogramming.

During initial programming, the user provides instructions using natural language: "*Pick item A from the conveyor belt, place item A at Location* 2*, pick item*

B from the conveyor belt, place it at Location 3". Then the user demonstrates the required actions to the robot. In the end, the robot learns the high-level task plan and the low-level trajectory to handle each object.

Once trained, the robot can dynamically adapt to new task requirements. If the task changes, such as both items needing to be placed at Location 2, the user inputs: "*Pick item A and item B from the conveyor belt, place them at Location* 2". The system processes this new instruction and updates the resulting task program from initial programming without additional demonstrations.

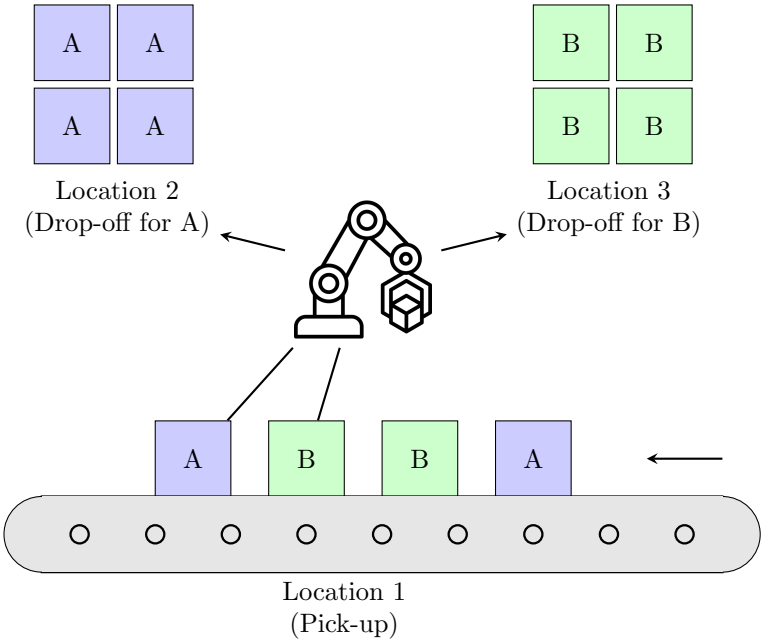


Fig. 4.2: Illustration of pick & place operations in a flexible packaging machine

4.2.2 Use case C2: Training for robot programming

The second use case involves a robotic pick & place task within a training scenario for robot programming, as shown in fig. 4.3. The robot is initially programmed to move four objects from Location 1 to Location 2. This scenario tests the robot's ability to:

- Adapt to changes in task direction, such as moving objects back to Location 1.
- Adjust to variations in the positions of Locations 1 and 2.

- Handle subsets of objects, moving fewer items than initially demonstrated.

During initial programming, the user provides instructions using natural language: "*Pick up the cuboid, the parallelogram, the star, and the octagon from the blue box and place them in the gray box*". Simultaneously, the user demonstrates the required actions to the robot. Similar to the first use case, the robot learns both *what to do* and *how to do it*.

If the task changes, such as one object needing to be placed in the blue box, the user inputs: "*Move the cuboid from the gray box back to the blue box*". The system processes this new instruction and updates the resulted task program from initial programming without additional demonstrations.

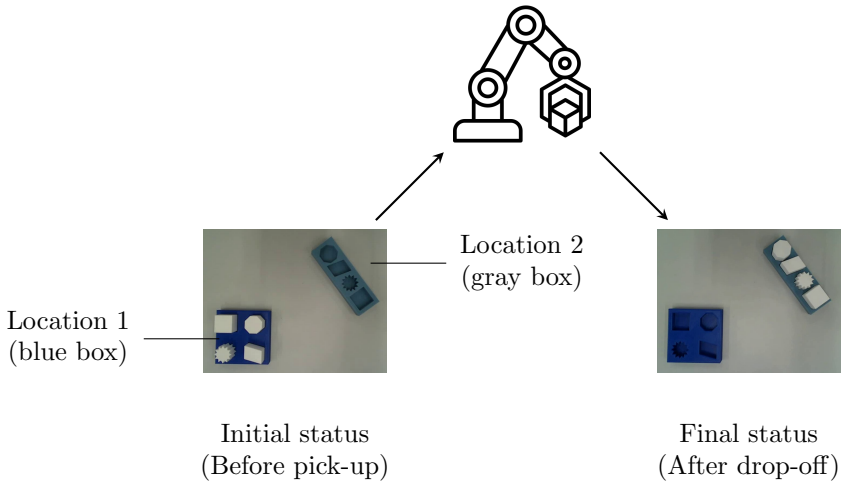


Fig. 4.3: Illustration of pick & place tasks for a robot training scenario

4.2.3 Summary of requirements

The analysis of the two use cases demonstrates the research hypotheses H1 in section 3.6. Integrating NLU and visual PbD can enhance a robot's ability to perform complex tasks in dynamic environments. The robot can be programmed once and can adapt during execution based on new task instructions, such as changes in the number of objects or their positions. The robot is able to generate a task program without the need for reprogramming. Based on the analysis of these two use cases, the following requirements have been identified for developing the robot programming system:

1. Adaptable task model: A generalized representation of tasks allows the robot to adjust to new instructions and environment changes.
2. Natural language understanding: Effective NLU capabilities are required for the robot to understand and act upon verbal instructions.
3. Visual programming by demonstration: The ability to learn from demonstrations enables the robot to imitate the actions and motions to complete tasks.
4. Contextual and environmental awareness: The system should be capable of adjusting to changes in the environment, such as object positions or task plan during execution.
5. User-friendly interface: Simplifying the interaction between the user and the robot through natural language and demonstrations lowers the barrier to programming and allows for task adjustments.
6. Efficient system architecture: The system must effectively process and integrate inputs from NLU and PbD to generate appropriate robot actions.

4.3 Task modeling & representation

This section develops a task representation framework for semantic robot programming. The approach starts with a hierarchical task model which is derived based on the requirements analysis. The model is then extended to integrate both language and demonstration input to provide a grounded representation.

4.3.1 Hierarchical task modeling

Based on the above analysis, three types of variations are identified for the task model: 1) the object is not found in the demonstrated location, 2) the object is not present in the environment to perform the action, and 3) modifications to the steps in a task plan. The first two variations are also defined in the approach of Daruna et al. (2021). The third variation is introduced specifically for the multi-step pick & place tasks in this work. During task execution, the robot is required to understand the modified task plan from the user input and generate the task program accordingly.

To address these variations, this work adopts the six-layer task model developed by Backhaus (2016) for the programming of assembly systems. This layered representation provides a structured framework that supports flexible and adaptive task execution, making it suitable for dynamic environments. Specifically, it enables users to apply a programming approach that accommodates variations within a flexible assembly system. In this work, the primary process and assembly sequence process from Backhaus' model are omitted, allowing a direct focus on the secondary process. This decision is motivated by the need to concentrate on operations that modify object positions, which are pick & place actions in this work.

Based on the requirements summarized above, the task model is reduced to four layers, as illustrated in fig. 4.4. These layers range from high-level task plans to low-level positions. The task plan is a sequence of actions to complete the task, which can be derived from instruction. Each action has a parameter of an object. In this work, the focus is on pick and place actions, which involve changing the position of an object. A *pick* action refers to the act of selecting and lifting an object from a particular location. It involves grasping the object to obtain control over it. To apply the learned task in different situations mentioned above, the *pick* action is further divided into *Reach* and *Grasp* skills. *Reach* is represented by motion from random start to grasp position. *Grasp* is considered as a discrete event, which occurs in single timestamp. It represents the moment when robot's end-effector successfully makes contact with the object and securely holds it. The *place* action is divided into *Move* and *Release* skills. *Move* is represented by motion from grasp to release position with the moving object. *Release* is also considered as a discrete event. It involves opening the gripper. The skills are extracted from visual PbD. The grasp and release position of an object can be estimated from visual information. This structured task model helps the robot to effectively understand and execute the tasks in dynamic environments. The layered structure also allows for effective integration of the knowledge from the instruction and visual demonstration. For instance, if an object's position can be updated through visual perception, the system adjusts the corresponding step in the task plan, addressing the first variation. If an object is absent in the environment, corresponding to the second variation, the

system removes the step from the task plan and proceeds to the next step. For the third variation, where the task plan itself changes, the system reorganizes the sequence based on the updated information.

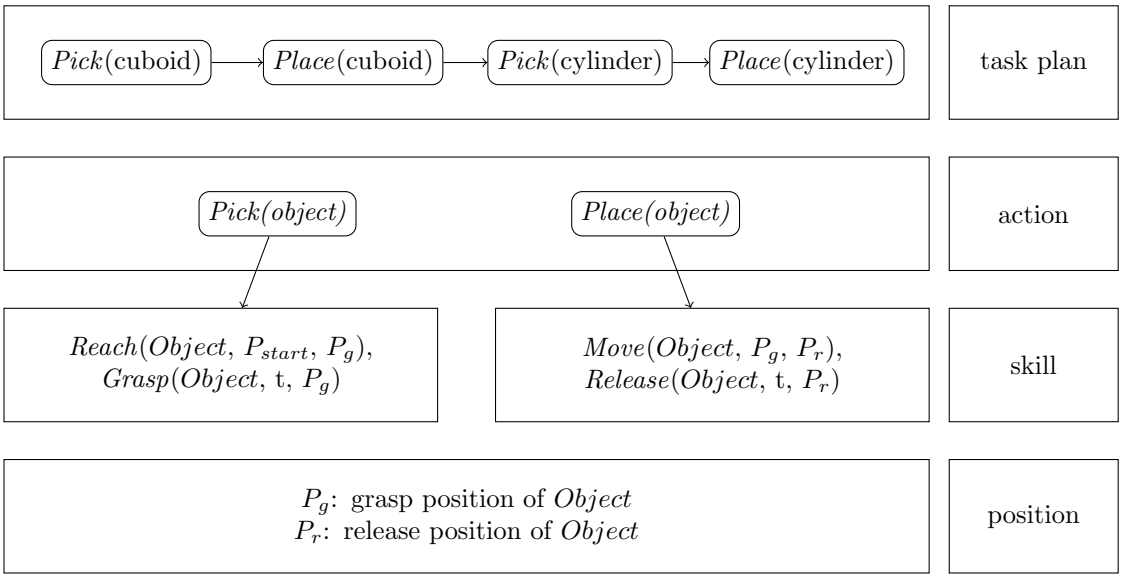


Fig. 4.4: Adapted from Backhaus (2016): Task model for the semantic programming system, illustrating the hierarchical structure from task plan to position.

4.3.2 Semantic task representation

In the context of semantic robot programming, a grounded knowledge representation is essential for generating a task model that integrates both language and visual input. This work builds upon and extends the approach proposed by Vanzo et al. (2020c) to improve the integration and interpretation of semantic information from multiple modalities. The original approach was designed with three levels: platform, domain, and perception. The platform level, which represents activities performed by different robot platforms, is renamed as the task level in this work to maintain consistency with robot programming terminology. The domain level includes real-world objects and their properties, while the perception level encodes the positions of objects along with their symbolic representations.

To enhance this representation and incorporate knowledge from both language and visual demonstration input, this work extends the model by introducing three additional levels: spatial, temporal, and skill levels, as shown in fig. 4.5. The skill level from the hierarchical task model (see fig. 4.4) is integrated to bridge high-level task descriptions with executable motion primitives, ensuring that task execution is both structured and adaptable. Similarly, the position information from the hierarchical task model is incorporated into the perception level, aligning planned execution with real-world object locations.

The temporal level models the sequence of events, associating each timestamp with a corresponding image from the visual demonstration. At time zero, an instruction is provided to the robot. From time one onward, each timestamp is linked to a specific image from the demonstration, along with the corresponding actions and spatial relationships in the scene. By integrating these elements, the system achieves a more comprehensive understanding of the instruction. Notably, while this work follows an approach where the instruction is provided at the first timestamp, language input can also be introduced after the visual demonstration.

By bridging hierarchical task decomposition with semantic grounding, this representation enables the robot to adapt dynamically to variations in its environment. The following chapters will discuss how this framework incorporates natural language processing and visual PbD to further enhance semantic robot programming.

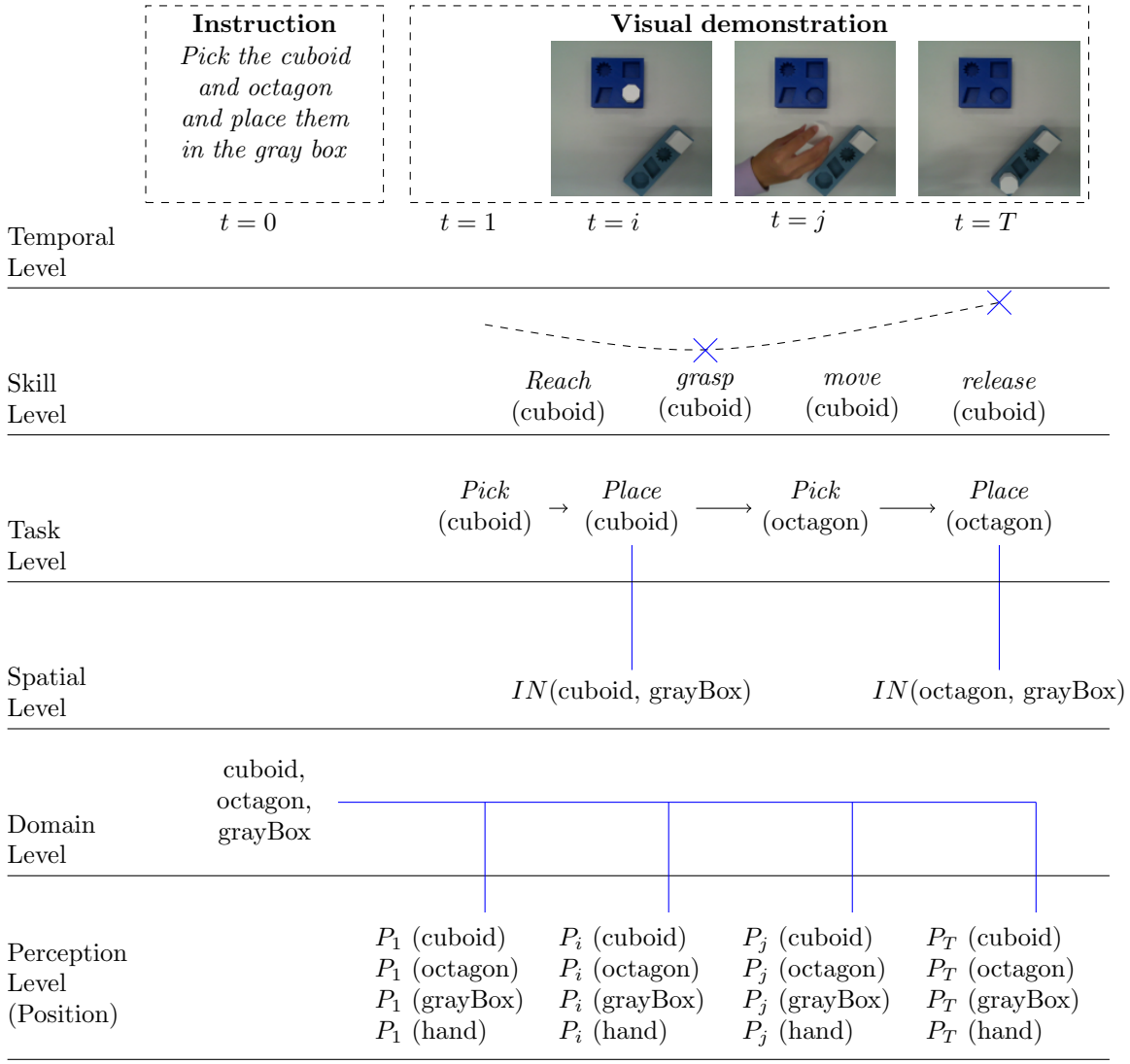


Fig. 4.5: Extended layered representation for semantic task modeling

4.4 Input acquisition & knowledge base integration

The system receives task information through a combination of natural language instructions and human demonstration, which can occur in different sequences depending on the context. Natural language instructions provide high-level information about the task, outlining the intended actions in spoken or textual form. These instructions can serve as an initial guide for structuring the task plan, or they can be used to refine a task learned from demonstration.

In addition to textual input, a human demonstrator performs the task while being recorded in RGB-D format. This demonstration provides a real-world

example of task execution, capturing both static and dynamic aspects, as illustrated in fig. 4.6:

- Static information: This includes the setup, the positioning of objects, and the environment in which the task will be performed. It provides a visual context for the initial and target environments.
- Dynamic information: This is about the sequence of action and interactions. It includes:
  - Sequence of actions: The step-by-step procedure followed by the demonstrator, highlighting the chronological order of tasks.
  - Interaction with objects: The manner in which the demonstrator interacts with various objects during the task, including changing the locations of objects.
  - Latent information: Implicit information, such as human trajectory shape, which can be extracted from the demonstrator's movements. For instance, the demonstrator's motion patterns may reveal best practices for efficiency or safety.

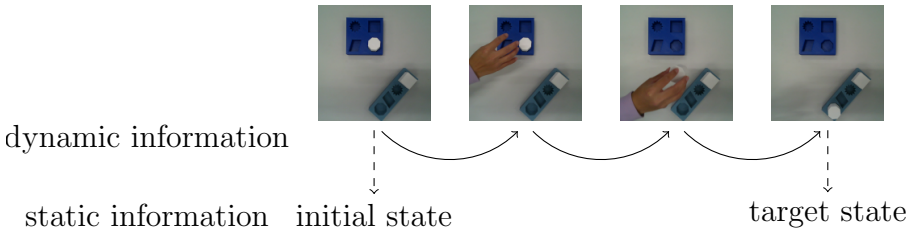


Fig. 4.6: Illustration of static and dynamic information from visual demonstration

In manufacturing scenarios, domain-specific information is often readily available and can be leveraged to enhance task execution. In this work, such information is referred to as the knowledge base, which consists of pre-existing data independent of the specific tasks being programmed. Unlike input data, which must be provided for each individual task, the knowledge base serves as a foundational resource that supports semantic programming.

One key component of the knowledge base is CAD files, which play a crucial role in manufacturing environments. These files contain detailed representations of objects, including their geometries and spatial relationships, making them

highly valuable for robotic perception and manipulation. In this work, CAD files are utilized to generate training data for object detection, enabling the system to recognize and interact with objects.

4.5 Information processing, task execution & validation

This section presents the concept for extracting, integrating, and validating information to facilitate task execution, which corresponds to the final phase in the approach illustrated in fig. 4.1. Following the modular approach of research hypothesis H4, the system is built by processing language and visual input separately before integrating them. Additionally, as stated in hypothesis H3, language input is processed first to guide the extraction of information from visual demonstrations, thereby reducing training efforts. The details of each step are described below.

- Extraction of information from natural language instruction: This is the first step of information processing. The system analyzes the textual instructions to extract key information, such as the goals of the task, specific steps mentioned, objects involved, and any spatial or temporal constraints.
- Extraction of information from visual demonstration: This involves processing the video data to extract meaningful data, such as the movements of the demonstrator, the sequence of actions, and interactions with objects or the environment. The extracted textual information provides guidance by defining relevant objects. Advanced computer vision techniques and deep learning algorithms are used in this step to interpret the visual data.
- Integration & task model generation: The task here is to integrate the information from both the instructional text and the visual demonstration into a unified representation, following the task model concept defined in the second step of the approach. This representation must contain all necessary details to understand and replicate the task. The system generates a task model based on the unified representation.

- Task Execution & validation: The robot follows the generated task model to execute the multi-step pick & place task, performing actions in the correct sequence with accurate movements. The execution is validated in simulation to ensure completeness and correctness.

4.6 Summary

This chapter introduced a structured methodological framework for developing a semantic robot programming system. The framework builds upon and improves established approaches for integrating semantic information extracted from natural language instructions and visual demonstrations. A detailed requirement analysis, guided by two representative pick & place use cases, provided the foundation for task modeling and representation. The developed hierarchical task model enables the robot to adapt flexibly to dynamic environments and varying task constraints. Additionally, a semantic task representation was defined to effectively ground multimodal input within a unified framework.

The chapter further described methods for acquiring input data and integrating domain-specific knowledge into the system. The subsequent processing pipeline involves separately extracting task information from language instructions and visual demonstrations before merging them into a coherent, executable task model. This integrated representation forms the basis for accurate task execution and subsequent validation. Building upon this conceptual design, the following chapters will focus on method development, guided by defined research plans:

P1 Development of information extraction methods for understanding the natural language instructions for pick & place tasks in manufacturing scenarios

P2 Development of information extraction methods for understanding the one-shot visual demonstration

P3 Development of a modular software framework to combine the knowledge extracted from instruction and demonstration

P4 Implementation and validation of the programming system

Chapter 5

Information extraction from natural language instruction

This chapter represents the first step in the research plan defined in section 4.5. It addresses the challenge of extracting relevant information from natural language instructions. Following research hypothesis H2 presented in section 3.6, which states that utilizing existing datasets significantly reduces the effort needed for data collection, this chapter begins by analyzing existing corpora to identify linguistic patterns and common structures within instructions. Additionally, it identifies the characteristics and challenges associated with instructions specific to manufacturing scenarios. Although these scenarios introduce domain-specific complexities, including specialized terminology and sequential multi-step instructions, they share structural similarities with generic pick & place tasks. Leveraging these similarities, and guided by the semantic task representation outlined in section 4.3, a hybrid extraction method combining deep learning with syntactic dependency parsing is proposed. The extraction method is detailed and evaluated in subsequent sections. The main content of this chapter is based on the conference article by Lu et al. (2023).

5.1 Analysis of related corpora

As described in section 1.1 and section 3.6, this thesis focuses on pick & place tasks, which account for approximately half of all industrial robot applications.

The wide variety of products necessitates the use of flexible pick & place systems. Based on this context, an extensive literature research is conducted, leading to the identification of three relevant corpora: SemEval-2014 Task 6, Human Robot Interaction Corpus (HuRIC) and Collaborative Manipulation Corpus (HuRCM). Each corpus is analyzed in terms of the instructions it contains and its corresponding annotation schema. Finally, this section provides a summary of the common patterns and key findings found across the three corpora, which can guide the development of effective information extraction methods.

5.1.1 Semantic Evaluation (SemEval)-2014 Task 6

SemEval[1] is a series of NLP research workshops that focus on advancing the current state of the art in semantic analysis and creating high-quality annotated datasets for various natural language understanding problems. The workshops started in 1998 and cover a diverse range of tasks, such as biomedical inference, emotion analysis and legal texts understanding. Among these, SemEval-2014 Task 6 (Dukes, 2014) is an annotated dataset for semantic parsing of robotic commands. There are 3409 sentences in this corpus. The task focuses on contextual parsing of spatial commands. Five semantic categories are taken into account, which are `actions`, `relations`, `indicators`, `objects` and `colors`. The values of `actions` are the moves used to control the robotic arm. `Relations` indicate the associations between two `objects`. `Indicators` are spatial attributes of an *object* that are specific to a particular domain. The values for each semantic category and its examples from the original work is outlined in table 5.1. A formal Robot Control Language (RCL) is used to annotate the sentences (Dukes, 2014). Here are some example instructions in SemEval-2014 Task 6:

Example 1 *"Pick up the yellow triangle placed on top of the green cube."*

Example 2 *"Place the yellow pyramid on top of the yellow cub."*

Example 3 *"Pick the red pyramid which is on top of yellow brick and place it above the blue block."*

[1]https://semeval.github.io/

Table 5.1: Representative examples of semantic categories in SemEval-2014 Task 6 (Dukes, 2014)

Category	**Values**
Actions	move, take, drop
Relations	left, right, above, below, forward, backward, adjacent, within, between, nearest, near, furthest, far, part
Indicators	left, leftmost, right, rightmost, top, highest, bottom, lowest, front, back, individual, furthest, nearest, center
Objects	cube, prism, corner, board, stack, row, column, edge, tile, robot, region, reference, type-reference
Colors	blue, cyan, red, yellow, green, magenta, gray, white

5.1.2 Human Robot Interaction Corpus (HuRIC)

HuRIC (Vanzo et al., 2020b) is a collection of datasets. The details about the corpus can be found in Vanzo et al. (2020a). The first work was introduced by Bastianelli et al. (2014) and further developed in Bastianelli et al. (2016, 2017). The corpus exploits different situations representing possible commands given to a robot in a household environment. Frame Semantics (Fillmore, 1976) are used to characterize robot actions. The idea of Frame Semantics is that people have in memory an inventory of frames for structuring, classifying and interpreting experiences, and that they have various ways of accessing these schemata and various procedures for performing operations on them (Fillmore, 1976). People understand things by performing mental operations on what they already know. Such knowledge is describable in terms of information packets called frames. Each frame represents a real-world situation, e.g., the actions of `Bringing`, `Taking` and `Placing`. In the instruction "bring the book on the table in the kitchen" the `Bringing` frame is evoked by the word *bring*: [*bring*]$_{\texttt{Bringing}}$ [*the book*]$_{\texttt{THEME}}$ [*on the table in the kitchen*]$_{\texttt{GOAL}}$. The `THEME` frame element refers to the object that is taken in a `Bringing` action, and `GOAL` arguments the target location. Frame and its values in HuRIC are summarized in table 5.2.

Table 5.2: Representative examples of semantic categories in HuRIC (Vanzo et al., 2020a): Actions and Objects

Category	**`Frame`: Values**
Actions	`Bringing`: get, deliver, move, carry, bring, fetch, take `Motion`: reach, move, drive, go, walk `Taking`: grab, get, remove, pick, catch, take `Change_operational_state`: put, open, shut, activate, start, restart, switch, stop, turn `Change_direction`: veer, turn `Locating`: look, search, find `Attaching`: connect, disconnect, attach `Arriving`: reach, enter `Manipulation`: grasp `Releasing`: let, drop, release, leave `Perception_active`: look, watch `Cotheme`: follow, come, go `Being_in`: is `Placing`: hang, put, place `Closure`: close, lower, put, open `Inspecting`: control, check, see, inspect
Objects	robot, box, television, door, bathroom, table, studio, thermostat, book, chair, desk, couch

To make reference to objects in the environment (e.g., the table in the kitchen vs. the table in the lounge), Spatial Semantics (Zlatev, 2007) is applied in HuRIC to represent the different roles and relations involved in spatial referring expressions (Bastianelli et al., 2014). However, the annotation is not included in the final corpus (Vanzo et al., 2020b). A spatial relation is composed by a `trajector`, i.e., the subject of the spatial relation, a `spatial_indicator`, i.e., the part of a sentence holding and characterizing the nature of the whole relation, and a `landmark`, i.e., the reference entity in relation to which the location or the trajectory of motion of the `trajector` is specified. For example, in the sentence "go near the table in the kitchen", the preposition "in" is the `spatial_indicator` of the relation between "table" and "kitchen", respectively a `trajector` and a `landmark`. Here are some example instructions in HuRIC:

Example 4 *"Take the laptop that is on the table on the couch."*

Example 5 *"Grasp the book on the table near the wine glass."*

Example 6 *"Take my phone and place it on the bench in the kitchen."*

5.1.3 Collaborative Manipulation Corpus (HuRCM)

HuRCM (Scalise et al., 2018a) is a dataset of natural language instructions for object reference in collaborative tabletop manipulation setting (Scalise et al., 2018b). It comprises 1582 individual written instructions. The dataset is developed for the field of natural language processing, human-robot interaction, and robotic tabletop manipulation. However, the instructions are not annotated in the original publication. Here are some example instructions in HuRCM:
Example 7 *"Grab the blue block that is closest to you."*

Example 8 *"Pick up the nearest blue block."*

Example 9 *"Pick up the green block that is in the middle, next to the solitary blue block."*

5.1.4 Summary of common patterns from the related corpora

The aforementioned corpora have similar sentence structures in terms of describing actions, target objects and related spatial-temporal information. To extract a task plan from instruction, the crucial information that the robot needs to understand is the `action` and `object` involved. The relation between `action` and `target object` can be denoted by *objectActedOn*. The example instructions demonstrate that each instruction contains at least one verb that describes an `action`. As the focus of this work is the pick & place task, the activity such as `Inspecting`, `Perception_active` and `Arriving` in HuRIC are not taken into account. HuRIC and SemEval-2014 Task 6 have more variance in the set of `actions` compared with HuRCM. The variance and similarity suggests that they can be utilized together to develop an information extraction system. Using multiple corpora can enhance the robustness and generalization capabilities. A task plan is then presented as a sequence of *action* entities, $\mathcal{TP} =$

$\{\texttt{action}_1, \texttt{action}_2, \ldots, \texttt{action}_n\}$, where n is the number of `actions` mentioned in one instruction. The value of n is normally 1 or 2 in the corpora.

In terms of `target objects`, different groups are addressed in the corpora. In HuRIC, the objects are various household items such as chairs, books and couches. On the other hand, the other two corpora employ simple geometric bodies. A common approach for extracting the information is to annotate them as objects. A pre-trained language model is then fine-tuned to classify them. However, these data are domain-specific. Integrating all objects in the extraction approach requires a significant amount of annotation effort.

Besides `actions` and `target objects`, descriptions of the objects are also included in the instructions. People often use color and spatial-temporal relations to provide detailed descriptions of objects. Spatial information involves the position, size, shape, and arrangement of objects in a given space. It is able to help the robot understand the relationship between objects. Temporal information is often provided alongside spatial details when describing objects. For instance, example 1 indicates the presence of a scene where the yellow triangle is positioned on top of the green cub. Example 2 implies that the yellow pyramid is not currently on top of the yellow cub, but it will be achieved after performing the `action`. To ground the information between instruction and demonstration, a spatial-temporal knowledge representation is required. This aspect was taken into account for developing the extended layered representation for semantic task modeling in fig. 4.5.

The semantics discussed above are represented in the entity-relationship diagram shown in fig. 5.1. The "Action" entity represents an operation, such as pick or place, and is linked to the "TargetObject" entity via the *objectActedOn* relationship, indicating that the action is performed on a specific object. The "TargetObject", characterized by the attributes "Color" and "ObjectName", is further connected to the "Object" entity through the *hasSpatialRelationWith* relationship. This relation captures spatial dependencies by linking to "Source-Location" and "TargetLocation", which define the initial and goal positions of the object, respectively. This model provides the foundation for developing the software framework to integrate knowledge from both language and demonstration in chapter 7.

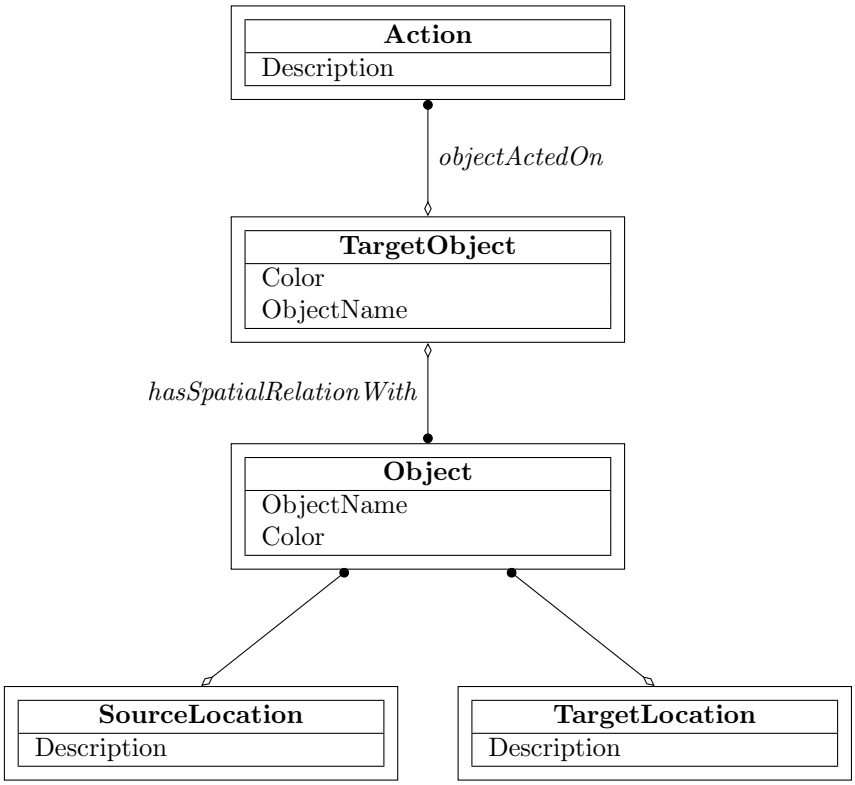


Fig. 5.1: Entity-relationship diagram for representing action and object semantics in instruction

Additionally, instructions also provide implicit information, including the definition of objects of interest and their state, whether static or in motion. In a pick & place task, the target object is moved while the other mentioned objects remain in their positions. This information can be used to understand object trajectories when extracting information from visual demonstrations.

5.2 Analysis of instructions for manufacturing scenarios

In contrast to the previously discussed corpora, which primarily focus on household and generic pick & place tasks, manufacturing environments present unique challenges in natural language instruction processing. The first major challenge is domain-specific terminology, as manufacturing instructions often contain specialized terms such as workpiece, fixture, conveyor, and calibration station,

which are critical for accurate task execution. In addition, different factories may use proprietary terminology and equipment, making it difficult to develop standardized datasets. In particular, no specific corpora for manufacturing scenarios have been identified in the literature, highlighting a gap in domain-specific datasets. The second challenge lies in the sequential and multi-step nature of manufacturing instructions. Unlike simple pick & place tasks, these instructions often require multiple actions to be performed in a strict sequence. Here are some examples for manufacturing scenarios.

Example 10 *"Pick the aluminum workpiece from the feeder, move it to the CNC machine, and position it on the fixture."*

Example 11 *"Lift the metal plate from the stack, rotate it 90 degrees, and place it on the welding station."*

Example 12 *"Remove the finished part from the press, transfer it to the cooling rack, and stack it onto the designated pallet."*

From the examples, we can observe that there are similarities between manufacturing and generic pick & place tasks, particularly in their basic structure and execution patterns. Both involve fundamental manipulation actions such as *picking*, *placing*, *moving*, and *positioning* objects, often follow a sequential order to complete a task. Additionally, they rely on spatial relationships to specify object locations and movements. In both cases, robots must process natural language instructions and translate it into executable actions. Furthermore, these tasks take place in dynamic environments, requiring adaptation to changes in object positions and workspace constraints.

Given these characteristics, a hybrid approach is proposed to extract information, combining deep learning and rule-based methods to enhance accuracy and adaptability in manufacturing scenarios. The details of this approach will be discussed in the following section.

5.3 Extraction approach

In this approach, the first step involves obtaining a domain-independent information extraction model by fine-tuning a pre-trained language model. The next step involves extracting domain-specific information using a syntactic dependency parser. This rule-based method leverages the linguistic structure and dependencies within sentences to identify and extract objects in specific domain. The hybrid approach ensures the system remains domain-independent, meaning it can be applied to new tasks involving different objects without requiring modifications. This flexibility enhances the system's ability in handling various tasks, especially in manufacturing scenarios. The proposed approach aligns with the research goal G3 (minimizing training effort) mentioned in section 3.6. The hybrid approach is illustrated in fig. 5.2 and is described in the following subsections.

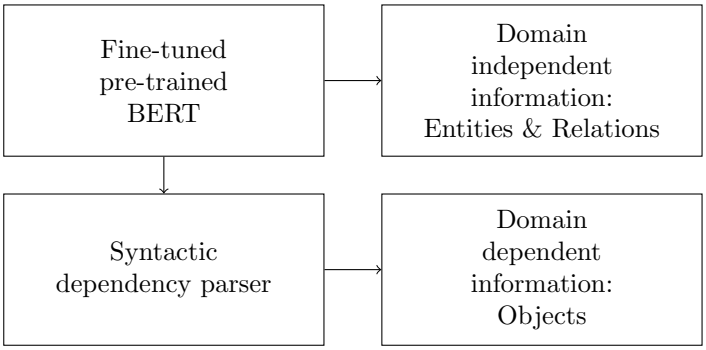


Fig. 5.2: Illustration of the hybrid approach

5.3.1 Data annotation

The extraction procedure for domain-independent information is taken as Named Entity Recognition (NER) and Relation Extraction (RE) tasks. Five entities and two relations are defined to represent the task knowledge, as shown in table 5.3. `Action` refers to verbs that indicate the physical movement causing the position change of an object. `Goal_indicator` represents the location signal that can be reached after performing an `action`. The entities `trajector` and `spatial_indicator` represent the current location, where the `action` will be started in next time stamp. The meanings are slightly different from those in

HuRIC due to the inclusion of temporal information. `Trajector` refers to the object, whose current location is specified by `spatial_indicator`. The `spatial relation` indicates the initial scene, while the `move relation` describes the goal scene. It should be noted that `spatial_indicator` and `goal_indicator` may share the same lexical items, but their semantic roles differ depending on the context of the robot task. A cycle from the initial to the goal represents the completion of a positional change of an object. An annotated sentence in English looks like as follows:

[*Pick*]$_{\texttt{action}}$ *the aluminum* [*workpiece*]$_{\texttt{trajector}}$ [*from*]$_{\texttt{spatial\_indicator}}$ *the feeder*, [*move*]$_{\texttt{action}}$ *it* [*to*]$_{\texttt{goal\_indicator}}$ *the CNC machine.*

Table 5.3: Defined entities and relations

	Types	**Examples**	**Descriptions**
Entities	`action`	pick, place, move, put, take, grab	Physical movement which causes position changes of a target object
	`goal indicator`	on top of, over, onto, in, into	Preposition that specifies the target position
	`color`	red, blue, green, yellow, grey, purple	Adjective describing the color of the object
	`trajector`	block, pyramid, cube, prism	An object with specified location
	`spatial indicator`	on top of, above, on, next to	Preposition characterizing the spatial relation
Relations	`spatial relation`	<block, on top of>	<`trajector`, `spatial indicator`>
	`move relation`	<move, to>	<`action`, `goal indicator`>

5.3.2 Fine-tuning pre-trained Bidirectional Encoder Representations from Transformers (BERT) model

Pre-trained LLMs, such as BERT, can be fine-tuned for specific NER and RE tasks. BERT is preferred for these tasks due to its bidirectional training, which allows it to capture context from both the left and right sides of a word in a sentence (Qiu et al., 2020a). BERT has been released in base and large variations, for cased and uncased input text. The base model has approximately 110 million parameters, while the large model has around 340 million parameters. For new tasks, BERT can be fine-tuned by adding a classification layer on top of its pre-trained model. Eberts and Ulges (2019) proposed a span-based approach called span-based entity and relation transformer (SpERT) to fine-tune the pre-trained BERT model. The span references to any token subsequence. For example, in the sentence "Pick the aluminum workpiece from the feeder", the spans might be "aluminum workpiece" or "feeder". A simplified structure of this approach is shown in fig. 5.3. Span classification is the process of identifying and classifying these spans. The classifier determines where a given span is an entity, such as `action`, `trajector` or `spatial_indicator`. The span filtering step filters out the spans that are not classified as entities, leaving only the spans that are considered entities. The next step is to determine the relations between these entities in the final step. The SpERT model is trained using strong within-sentence negative samples. Details about the model architecture can be found in the original publication (Eberts and Ulges, 2019). The SpERT approach is applied in this work to train a classifier to recognize entities and relations defined in table 5.3.

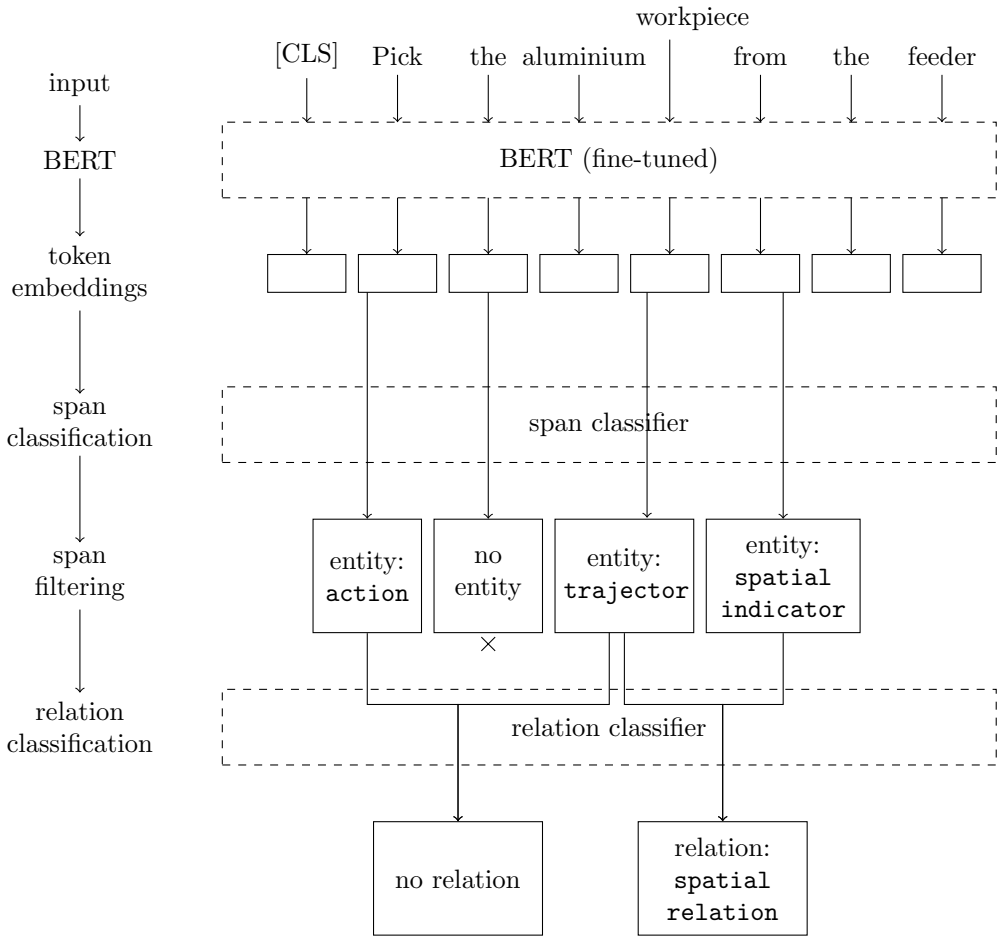


Fig. 5.3: Illustration of the span-based entity and relation transformer (SpERT) approach

5.3.3 Syntactic dependency parser

The syntactic dependency parser plays a key role in the information extraction pipeline, complementing the deep learning classifier by capturing grammatical relationships within instructions. The parsing process begins with pre-processing steps. First, a tokenizer segments each instruction into individual tokens, separating words and punctuation marks. Next, a part-of-speech tagger labels each token according to its grammatical category, such as noun, verb, adjective, or preposition.

After tokenization and tagging, syntactic dependencies are extracted. The dependency parser identifies grammatical relationships between words, which are particularly useful for capturing domain-specific objects and their attributes. Specifically, the following dependency relations are utilized in this work:

- `dobj` (direct object): identifies the noun phrase that directly receives the action expressed by a verb.
- `pobj` (object of preposition): identifies the noun phrase linked to spatial indicators or goal indicators via a preposition.
- `amod` (adjective modifier): identifies adjectives describing the attributes (e.g., color) associated with an object.

The target object is thus recognized as the direct object (`dobj`) associated with an action, while the spatial reference object is identified using the prepositional object (`pobj`). By employing this rule-based syntactic parsing, domain-specific variations in noun phrases or terminology are handled effectively without additional annotation effort.

5.3.4 Proposed hybrid information extraction approach

The proposed hybrid approach for information extraction combines a trained classifier with a syntactic dependency parser to extract relevant task plan information. The extraction process by the trained SpERT classifier is shown in black and the process of syntactic dependency is shown in blue, as illustrated in fig. 5.4.

First, a trained SpERT classifier (section 5.3.2) processes the input instruction to identify and classify domain-independent entities and relations, such as `action`, `trajector`, `spatial_indicator`, `goal_indicator`, and `color`, as defined in table 5.3. Following this, a syntactic dependency parser (section 5.3.3) analyzes the instruction to extract grammatical relationships between identified entities, particularly focusing on domain-specific objects. Specifically, the parser extracts the direct object (`dobj`), which identifies the object receiving the action, as well as the object of a preposition (`pobj`), associated with the spatial or goal indicators. Additionally, adjectival modifiers (`amod`) are employed to identify attributes like color associated with each object.

The process identifies the objects of interest, including the target object (the one to be moved) and other objects (which will remain unchanged). The approach checks if a trajector is part of the instruction. If so, the current location is determined using the spatial indicator and the object. Specifically, if the instruction explicitly mentions a `trajector`, its current location is derived from the identified `spatial indicator` and corresponding object. Otherwise, the initial location is marked as unknown. The intended target location is always determined by analyzing the `goal_indicator` and the related object. Similarly, if the color of the target object is explicitly stated, it is included; otherwise, it remains unspecified.

The overall workflow of this hybrid approach is illustrated in fig. 5.4, and the output is represented semantically as depicted previously in fig. 5.1. This systematic approach ensures adaptability across domains, particularly benefiting manufacturing scenarios, which frequently involve specialized terminology and sequential task structures.

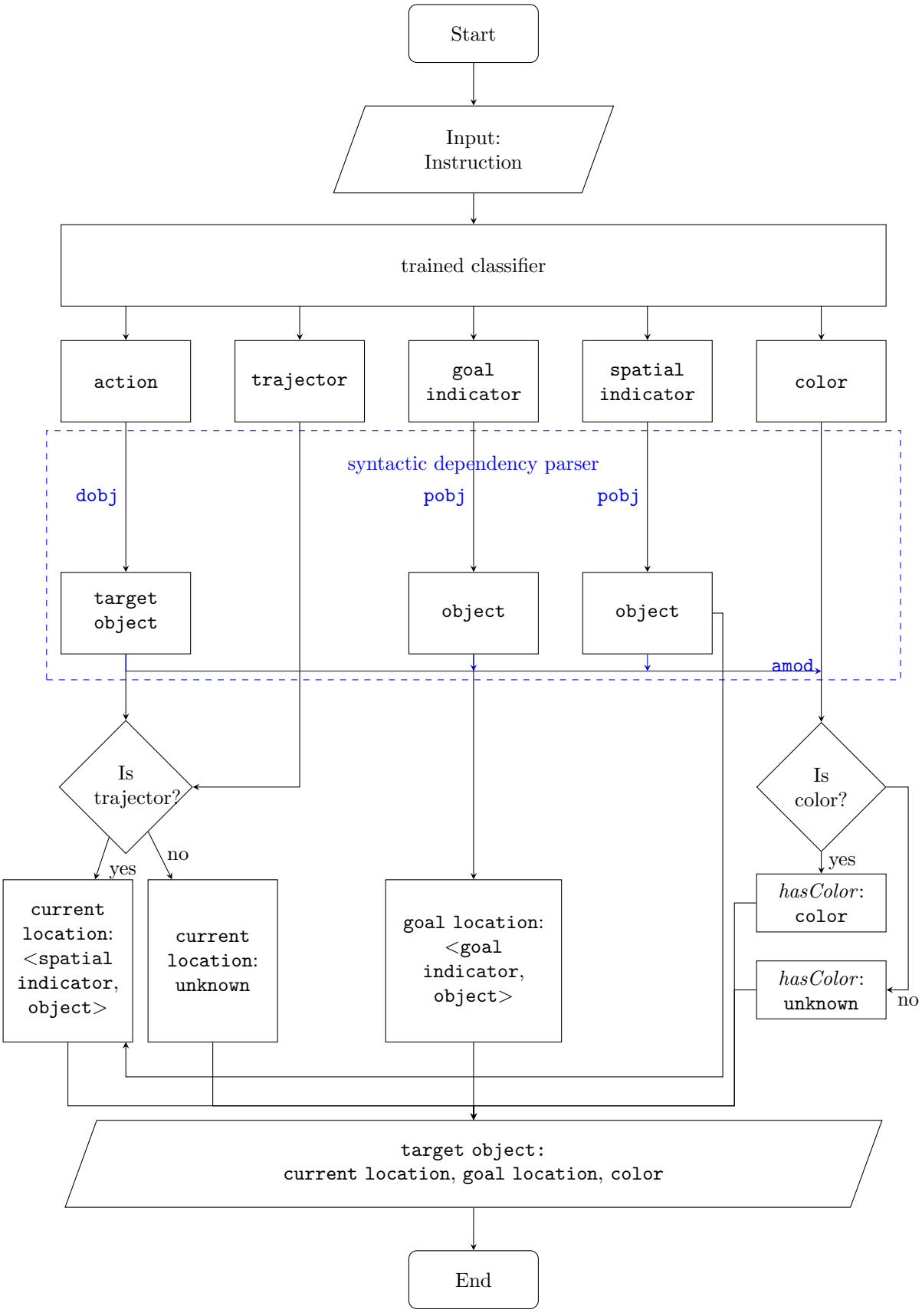


Fig. 5.4: Information flowchart of the information extraction approach

5.4 Experiments for the proposed information extraction approach

5.4.1 Data preparation

To evaluate the proposed information extraction approach, a dataset was created by selecting and annotating instructions involving manipulation tasks from the three corpora discussed in section 5.1. The annotation schema is based on the defined entities and relations summarized previously in table 5.3.

Representative examples of annotated entities from each dataset (SemEval-2014 Task 6, HuRIC, and HuRCM) are presented in tables 5.4, 5.5, and 5.6, respectively. From the SemEval-2014 Task 6 dataset, 552 manipulation instructions were selected due to their high relevance for robotic programming scenarios. Additionally, 242 pick & place instructions were annotated from the HuRIC dataset, which focuses on household scenarios. Lastly, the HuRCM dataset, developed primarily for object referencing, contributed 576 annotated sentences; however, it lacks goal indicators.

As demonstrated in the examples, each dataset has inherent limitations regarding the diversity of vocabulary for entities and relations. Combining these datasets resulted in a more comprehensive corpus, enhancing the generalizability of the proposed approach. The final annotated dataset comprises 1370 instructions, divided into 70% training, 15% validation, and 15% testing. The annotated data are publicly available online as open source[2].

[2]https://github.com/shuanglu2023/TaskPlanUnderstanding

Table 5.4: Examples of entities in annotated Semantic Evaluation (SemEval)-2014 Task 6

action	trajector	spatial indicator	goal indicator	color
pick, *put*, *place*, *move*, *take*, *drop*	*block*, *pyramid*, *cube*, *prism*, *brick*, *tetrahedron*	*above*, *on top of*, *on*, *from*, *in*	*on top of*, *on*, *to the top of*, *over*, *on the top of*	*red*, *blue*, *green*, *yellow*, *grey*

Table 5.5: Examples of entities in annotated Human Robot Interaction Corpus (HuRIC)

action	trajector	spatial indicator	goal indicator	color
take, *bring*, *put*, *grab*, *carry*, *get*, *grasp*	*table*, *book*, *box*, *phone*, *pillow*, *laptop*, *towel*	*on*, *from*, *near*, *in*, *of*, *next to*, *to*	*to*, *on*, *in*, *near*, *into*, *onto*	*gray*, *blue*

Table 5.6: Examples of entities in annotated Collaborative Manipulation Corpus (HuRCM)

action	trajector	spatial indicator	goal indicator	color
pick, *grab*, *get*	*block*, *cube*	*next to*, *above*, *on*, *behind*, *between*	-	*yellow*, *blue*, *green*, *orange*

5.4.2 Training and results

As discussed in section 5.3, the implementation of SpERT (Eberts and Ulges, 2019) was used to train the model. The pre-trained "bert-base-uncased" model[3] was used for fine-tuning. The model was trained for 30 epochs. The learning rate is set to a small value of 0.00005, which updates the weights of the model with smaller steps. This can help in achieving more precise convergence and avoiding overshooting the optimal solution. The warm-up value is set to 0.1, which means that at the beginning of training, the learning rate will gradually increase from a small value to the desired value. This helps the model to overcome the initial instability and reach a more stable state faster. The weight decay is set to 0.01, which adds a penalty term to the loss function to discourage large weights and prevent overfitting.

The evaluation was first conducted separately on the three corpora. The results are summarized in tables D.1, D.2 and D.3. Overall, the evaluation across the three datasets shows that the model performs best in recognizing `action` and `color`, achieving F1-scores above 90%. `Trajector`, `spatial_indicator`, and `goal_indicator` exhibit stable performance above 80%. Spatial relations consistently pose the greatest challenge, with F1-scores ranging from 78-86%.

To further assess the model's ability to generalize across different linguistic descriptions, an additional evaluation was performed on a combined dataset comprising all three corpora. As described in section 5.1, the three corpora differ in their linguistic characteristics. Combining them increases variance, which may improve the model's robustness to different expressions. The precision, recall and F1-score for validation and test on the combined dataset are reported in table 5.7 and 5.8 respectively. Note that the results are slightly different from the publication by Lu et al. (2023), as more data is added and a test split is created. The evaluation results on the combined dataset exhibit similar trends to those observed on the individual corpora. Specifically, the system achieves an accuracy of more than 90% for recognizing of `action` and `color`. The accuracy for `trajector`, `spatial indicator` and `goal indicator` are around 80%. The relations are considered correct if the relation type and the

[3]https://huggingface.co/bert-base-uncased

related entities are predicted correctly. Consequently, the model's accuracy in recognizing spatial relations is approximately 70%, highlighting the inherent complexity and ambiguity in spatial language understanding.

Table 5.7: Validation results on the combined dataset

	Precision	Recall	F1-Score
`action`	99.13%	99.13%	99.13%
`trajector`	86.76%	81.94%	84.29%
`spatial indicator`	82.28%	83.33%	82.80%
`goal indicator`	81.82%	90.00%	85.71%
`color`	94.86%	97.19%	96.01%
`spatial relation`	73.75%	76.62%	75.16%
`move relation`	80.30%	86.89%	83.46%

Table 5.8: Test results on the combined dataset

	Precision	Recall	F1-Score
`action`	99.22%	100.00%	99.61%
`trajector`	78.21%	85.92%	81.88%
`spatial indicator`	75.82%	84.15%	79.77%
`goal indicator`	94.57%	98.06%	96.67%
`color`	96.88%	98.84%	97.85%
`spatial relation`	63.92%	77.50%	70.06%
`move relation`	93.48%	96.63%	95.03%

In summary, the evaluation confirms that the model performs consistently across both individual and combined datasets, with high accuracy in recognizing object-related categories and moderate performance on spatial relations. The combined evaluation further demonstrates the model's capability to generalize across different linguistic variations. Combining the datasets increases linguistic variability, which leads to a slight drop in performance compared to training on the individual corpora. Nevertheless, this approach is beneficial, as it mitigates the risk of overfitting to the specific structure of a single dataset and improves the model's adaptability.

According to the relevant literature (Eberts and Ulges, 2019; Hillebrand et al., 2022), achieving about 80% accuracy for NER and RE tasks is generally considered acceptable. However, for practical use in semantic robot programming, accuracy approaching 100% is highly desirable due to the safety-critical and precise nature of robot actions. To further improve performance, expanding the dataset, integrating multimodal inputs (e.g., visual information), or implementing human verification processes could mitigate existing challenges and reduce potential errors.

5.4.3 Error inspection of syntactic dependency parser

After recognizing entities and relations, spaCy is used for extracting the syntactic dependencies: `dobj`, `pobj` and `amod`, as shown in fig. 5.4. spaCy is a free open-source library for natural language processing. The parser uses a variant of the non-monotonic arc-eager transition-system described by Honnibal and Johnson (2015). The details can be found at the spaCy website[4]. To evaluate the performance, the validation data are investigated. More than 90% of the sentences can be extracted correctly, a correct example is illustrated in fig. 5.5.

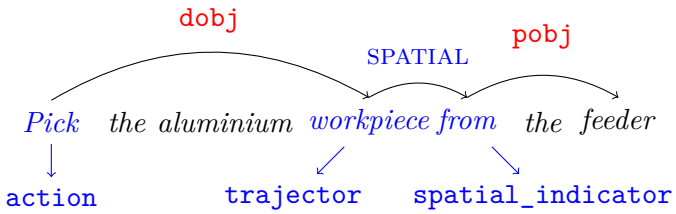


Fig. 5.5: Result of understanding an instruction using SpERT (blue) and syntactic dependency parser (red)

However, the rule does not hold for instructions which consist of the object *parallelepiped*, an error is shown in fig. 5.6. The word has been falsely identified as a verb instead of a noun, which prevents the correct identification of the syntactic structure between the spatial indicator and the spatial object. Such parsing errors typically result from limited representation of rare words in spaCy's general-purpose training corpus. As future work, a possible solution

[4]https://spacy.io/api/dependencyparser

to address these parsing inaccuracies is to perform a secondary validation with LLM. Additionally, incorporating human verification could further enhance parsing accuracy and reliability.

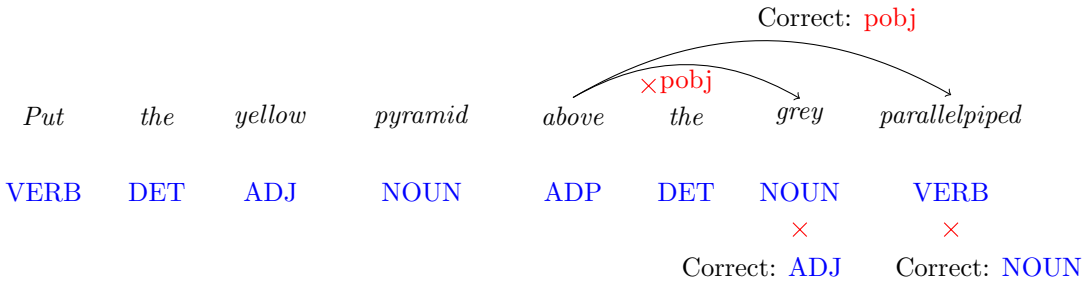


Fig. 5.6: Example of an error made by the syntactic dependency parser

5.5 Summary

This chapter addresses the first concrete step following the conceptual design presented in chapter 4. It focuses on extracting structured information from natural language instructions, specifically targeting robotic manipulation tasks within manufacturing scenarios. Guided by the research hypothesis that leveraging existing datasets significantly reduces data collection effort, three relevant corpora (SemEval-2014 Task 6, HuRIC, and HuRCM) were identified, analyzed, and annotated. A key contribution of this chapter is the development of a unified annotation schema, designed to accommodate diverse pick & place tasks by capturing common linguistic structures across these datasets.

Another major contribution is the development and evaluation of a hybrid extraction method designed explicitly for manufacturing environments, which often involve specialized terminology and sequential instructions. This hybrid method combines the SpERT deep learning model, extracting domain-independent entities such as actions, colors, and spatial indicators, with a syntactic dependency parser that effectively captures grammatical relationships and domain-specific vocabulary. This combination significantly reduces annotation and training efforts, enhancing adaptability to new manufacturing tasks.

Evaluation results demonstrated that the hybrid approach achieved an overall accuracy exceeding 80%, with accuracy surpassing 90% for action and color

recognition. Recognition accuracy for spatial relationships remained around 70%, reflecting the inherent complexity of spatial language. Although this level of accuracy aligns well with benchmarks in related literature, near-perfect accuracy remains essential for practical implementation in semantic robot programming within manufacturing due to precision and safety requirements.

The proposed extraction method will be integrated into the software framework in chapter 7 to generate a complete task model. This integration will be subsequently implemented and evaluated in chapter 8.

Chapter 6

Information extraction from visual demonstration

This chapter presents methods for extracting human hands and relevant objects from the task environment. It represents the second step in the research plan defined in section 4.5. The goal is to extract the movements of human hands and objects and store them in the task model so that the robot can reproduce the task. The main content of this chapter is based on the article by Lu et al. (2025), published in *Scientific Reports* under the Creative Commons Attribution 4.0 International License (CC BY 4.0). Changes were made to adapt the article to the structure and context of this thesis.

6.1 Overview

A structured framework is developed in this section to extract and represent task-relevant information from human demonstrations, which are captured using an RGB-D camera and supplemented with language-based information. The process begins with acquiring RGB and depth images, which serve as inputs for hand detection and object detection. Simultaneously, language-based input is leveraged to define objects of interest and provide contextual information about object states and sequences. The extracted visual and linguistic information is then used to generate hand and object trajectories, which are then mapped to a robot's end-effector motion. These trajectories are further segmented into motion units based on interaction states and object sequences. Finally,

the segmented motion units are learned and represented as skills using DMPs, enabling the robot to reproduce the demonstrated task. An overview of this process is presented in fig. 6.1. The detail of each step are discussed in the following sections.

6.2 Detection of human hands and objects of interest

6.2.1 Hand detection

Detecting human hands is a crucial step in extracting motion information from visual demonstrations, as it enables the system to capture the demonstrator's hand trajectories. Various trained models and frameworks have been developed for hand detection in computer vision applications (Qi et al., 2024). In this work, two widely adopted approaches are selected, each representing a different category of detection methods: a bounding box-based method and a keypoint-based method. Both methods have demonstrated strong performance and computational efficiency in the context of visual PbD (Xin et al., 2023; Lu et al., 2022a). To ensure their applicability for this application, both detection approaches are evaluated on demonstration videos in section 8.3.1.

As discussed in section 3.2.2, Xin et al. (2023) applied a pre-trained Bounding Box (Bbox)-based model to learn hand motion trajectories with promising results. The model was proposed by Shan et al. (2020) to understand hand and hand-object interaction. The system was built on top of a popular object detection system, Faster-RCNN (Ren et al., 2015b). It has shown strong performance in detecting hands. Therefore, this model is chosen in this work. The model has a two-stage architecture, where the first stage proposes regions and the second stage classifies and refines the bounding boxes. Faster R-CNN uses a so-called region proposal network (RPN) that generates candidate regions of interest (ROIs) based on the input image. The models were trained on the datasets proposed in the same publication, the 100DOH dataset and the 100K frames. The 100DOH is a large video dataset containing hands and hand-object interactions. They are 27.3K YouTube videos from 11 categories with nearly 131

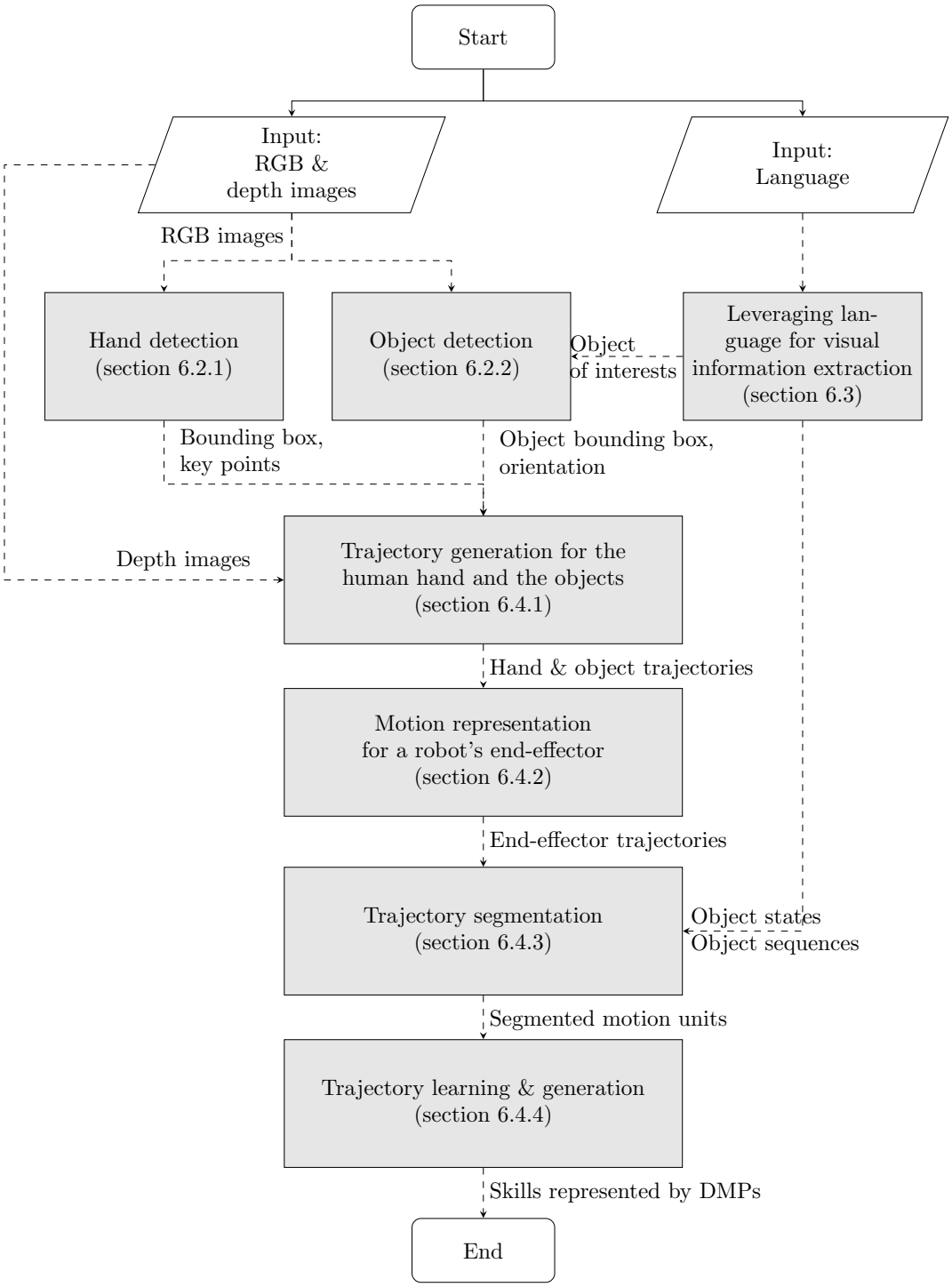


Fig. 6.1: Process and information flowchart for information extraction from visual demonstration

days of footage of everyday interaction. To detect hands on a single frame, the

authors created a new 100K frame-level dataset, which consists of 99 899 frames extracted from the videos in 100DOH and VLOG Dataset (Fouhey et al., 2018). There are 189 426 annotated bounding boxes for hand. Two models trained on 100K and 100K+ego are provided by the authors for hand detection. The 100K+ego consists of 56.4K frame subset of egocentric data from Damen et al. (2018), Li et al. (2015), and Sigurdsson et al. (2018). Both pre-trained models for hand detection achieved approximately 90% average precision[1], demonstrating strong and promising results in the field of computer vision.

Another detection method is based on key points, which is an open-source framework developed by Google called MediaPipe (Lugaresi et al., 2019). This method has also been widely applied for visual imitation learning, as demonstrated in works like Gao et al. (2023) and Zhan and Huang (2022). Thus, this method is also chosen for this work. The two models, Faster-RCNN and MediaPipe, will be compared during the evaluation. The MediaPipe provides a pipeline for hand detection. The hand landmark model in this pipeline for key points detection was trained on around 30K real world images and several rendered synthetic hand models imposed over various backgrounds. The detector runs two stages, where the first stage is palm detection and the next step is key point detection. The output of the hand detector is 21 3D hand-knuckle coordinates inside the detected hand regions. The representation of each key point is composed of x-, y- and z- coordinate, where x and y are pixel coordinates on image and z is the distance to the wrist as origin. A hand trajectory can be created from either method using the center of the hand position. An example of detection from both methods is shown in fig. 6.2.

6.2.2 Object detection

As mentioned in section 3.2.2, object detection in visual PbD is crucial for understanding task environments. Thanks to the availability of large datasets, various model architectures have been proposed and shown to continuously improve performance. Most available datasets are labeled with regular bounding boxes for everyday objects. They serve as benchmarks for comparing different

[1]https://github.com/ddshan/hand_detector.d2

(a) Detected bounding box using model trained on 100K+ego

(b) Detected hand key points with MediaPipe

Fig. 6.2: Results of hand detection on RGB images using different models. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

model architectures. Some well known datasets are ImageNet (Russakovsky et al., 2015), Pascal VOC (Everingham et al., 2010) and COCO (Lin et al., 2014). They provide raw images, their annotations and standardized evaluation protocols. According to Zou et al. (2023), COCO is currently the standard benchmarking dataset for the object detection community. The latest release of COCO 2017 consists of a total of 123 287 images and 896 782 objects, across its validation datasets. Additionally, the testing dataset comprises 40 670 images. They cover 80 object categories, which include everyday objects such as person, bicycle, and car, as well as household items like chair, couch, and dining table. The annotations include the bounding box coordinates, the category level, and other optional attributes. More detailed information about the COCO dataset can be found on the official website[2]. As introduced in section 2.3, the bounding boxes provided in COCO are rectangular boxes aligned with the image axes that enclose the object. They do not account for the object's rotation. However, the orientation is essential for application scenarios such as understanding aerial images. Dataset of object detection in aerial images (DOTA) (Ding et al., 2021) is a large-scale dataset for the application of aerial images. The DOTA dataset consists of a total of 1 793 658 object instances that are annotated with Oriented Bounding Box (OBB) annotations. The illustration of the annotations is shown in fig. 6.3, where θ refers to the rotation of the rectangular box with respect to the image axis. The object instances in DOTA belong to 18 different categories, which include objects like planes, ships, and basketball courts. The

[2]https://cocodataset.org

OBB approach is followed in this work, which allows for improved accuracy and better object differentiation.

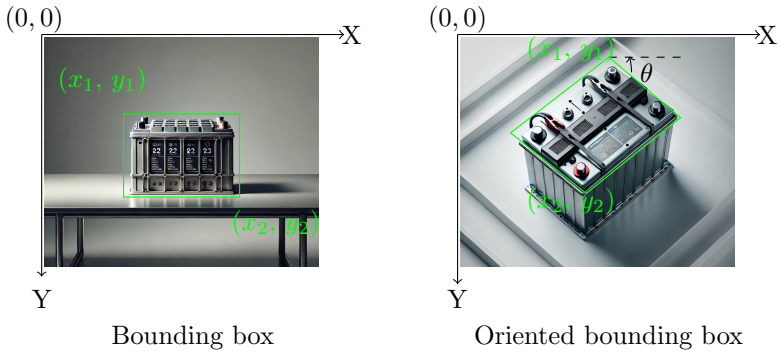


Fig. 6.3: Illustration of the differences between axis-aligned bounding box and oriented bounding box annotation

As discussed above, existing datasets such as COCO and ImageNet primarily cover everyday objects. Limited works focus on industry-relevant datasets, such as T-LESS (Hodaň et al., 2019) and the dataset presented by Wen et al. (2022). However, creating large datasets like COCO for various industry-specific tasks is impractical due to the diversity of objects in manufacturing scenarios. Each task may involve highly specialized items that require tailored data collection and annotation processes, making it infeasible to develop a one-size-fits-all dataset. In such cases, synthetic datasets generated from CAD files offer a valuable solution. By leveraging available CAD models, synthetic training data can be created efficiently, providing accurately labeled datasets at a lower cost (Vanherle et al., 2022). In this thesis, Blender is used to render synthetic data. It is an open-source 3D graphics software that provides rendering capabilities for creating photorealistic images. This software supports ray tracing, physics-based rendering, and procedural material generation.

Photorealistic images are created or rendered to closely resemble real-life objects or scenes. Hodaň et al. (2019) developed an approach to synthesize photorealistic images of 3D models, which are used to train a convolutional neural network for detecting the objects in real images. The approach is implemented in BlenderProc, which is an open source pipeline to render the images (Denninger et al., 2023). The pipeline is a Blender extension with Python API with various

examples. The image synthesis approach[3] for generating LineMOD dataset (Hinterstoisser et al., 2012) in BOP challenge[4] is followed in this work. In this approach, objects are arranged inside an empty room. A random photorealistic material from the CC0 Textures[5] library is assigned to the walls of the room, and light with a random strength and color is emitted from the room ceiling and from a randomly positioned point light source. Realistic object poses are achieved by dropping objects on the ground plane using PyBullet physics engine integrated in Blender. In the pipeline, both the images and their corresponding annotations are produced simultaneously and automatically. The annotations are generated in OBB format to preserve the orientation of the objects.

Faster-RCNN (Ren et al., 2015b) and YOLO (Redmon et al., 2016) are both popular object detection models. YOLO has a one-stage architecture, where it divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell. Since its initial release in 2015, there have been subsequent versions and enhancements, such as YOLOv5 (Jocher, 2020) and YOLOv6 (Li et al., 2022). Ultralytics released YOLOv8 version 8.1 in 2024. They provide pre-trained models trained on large datasets COCO and DOTA for Bbox and OBB detection tasks respectively. It also provides a framework for fine-tuning.

YOLOv8 is utilized in this work to detect objects of interest due to the availability of pre-trained models for OBB detection and its software framework. The model utilizes a convolutional neural network that can be divided into two main parts: backbone and detection head. A simplified architecture of RangeKing (2023) is illustrated in fig. 6.4, highlighting the process of model initialization and adaptation for a new task. The backbone is responsible for extracting features from the input images, where each "Px" corresponds to different feature levels extracted from the image. Details about each part of the architecture can be found in Ultralytics implementation (Jocher et al., 2023). The detection head of YOLOv8 consists of multiple convolutional layers followed by a series of fully connected layers. These layers are responsible for predicting

[3]https://github.com/DLR-RM/BlenderProc/blob/main/examples/datasets/bop_challenge/README.md

[4]https://bop.felk.cvut.cz/home/

[5]https://ambientcg.com/

the oriented bounding boxes and class probabilities for the objects detected in the image. During the fine-tuning process, the parameters of the backbone are used to initialize the model. The detection head is adapted to the number of classes specific to the new task. The blue-edged boxes in fig. 6.4 represent the components that require adjustments in the fine-tuning process.

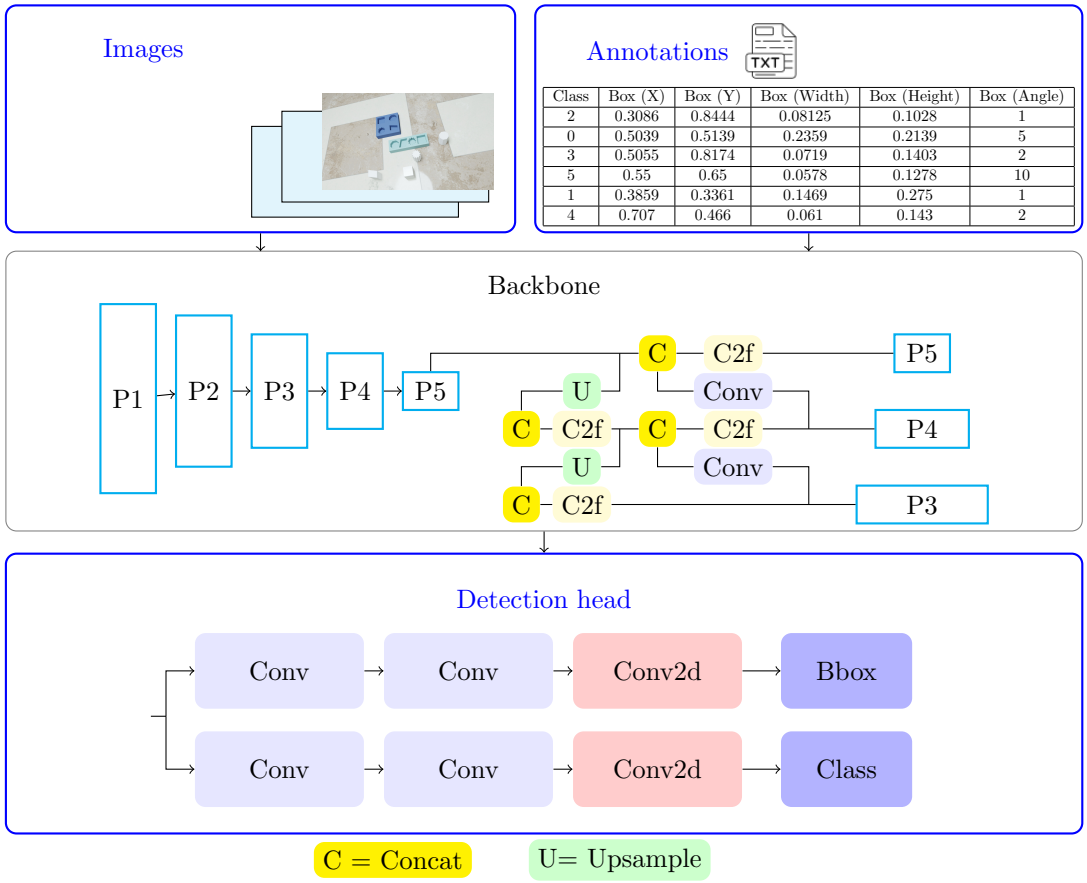


Fig. 6.4: Simplified YOLOv8 architecture of RangeKing (2023) and illustration of the model initialization for fine-tuning on a new object detection task

To evaluate the detection performance, the basic metrics introduced in section 2.2.4 are applied. The specific aspects for object detection with YOLOv8 are discussed as below. These metrics will be applied to evaluate the object detection performance in section 8.3.2.

- Intersection over Union (IoU) is a measure that evaluates the overlap between two bounding boxes. It requires a ground truth bounding box B_{gt}

and a predicted bounding box B_p.

$$IoU = \frac{area(B_p \bigcap B_{gt})}{area(B_p \bigcup B_{gt})} \tag{6.1}$$

- Confidence score in YOLOv8 is a combination of box confidence and class confidence. Box confidence is a measure of how certain the model is that a bounding box contains an object of interest. It combines the objectness score with the IoU between the B_p and B_{gt}. Class confidence is an expression of how certain the model is that the detected object belongs to a particular class.
- Both confidence score and IoU are used as criteria that determine whether a detection is true or false.
- mAP50 refers to the mean average precision at 0.5 IoU. It is a measure of the model's accuracy considering the relative easy detections.
- $mAP50 - 95$ is the average of the mean average precision calculated at varying IoU thresholds, ranging from 0.5 to 0.95. It gives a comprehensive view of the model's performance across different levels of detection difficulty.

6.3 Leveraging language for visual information extraction

In many scenarios, visual information may be unclear or incomplete, especially when objects are occluded. Object detection can be improved by incorporating language input, which provides additional context or descriptions. It provides contextual cues that enhance the visual perception process in three aspects:

- Definition of objects of interests: Language input explicitly identifies which objects are relevant to the task, allowing the system to focus on these objects during detection. This information can be used to automatically generate training data and fine-tune pre-trained models to recognize the specified objects.
- Object states: Language input can also provide latent information about the objects' static and dynamic states, helping the detection system better

interpret the scene. For static objects, this information can be used to refine detection by replacing the predicted OBB with the one that has the highest confidence score. For dynamic objects, language input can inform the system of potential changes in object behavior (e.g., movement or interaction), improving the overall recognition accuracy.

- Object sequences: Language input provides information about the intended order of actions and interactions between objects. This temporal information supports the segmentation of hand and object trajectories, enabling the system to correctly identify and structure the sequence of task-relevant events.

To systematically organize and integrate the multimodal information obtained from language and visual inputs, an entity-relationship diagram is introduced, as illustrated in fig. 6.5. The diagram defines the relationships between the key entities involved in the task demonstration and execution.

Specifically, the *GroundedProducts* entity represents the objects of interest identified from the natural language instruction, linking the recognized objects in the environment to their corresponding categories and names. This allows each object to be associated with its visual pose data. The *ProductPose(ObjectName)* entity records the position, orientation, and skill labels of each object at every timestamp, creating a complete trajectory history. In parallel, the *HandPosition* entity logs the demonstrator's hand positions, which are temporally synchronized with the object poses. By correlating these entities through a common time index, the system can accurately align the demonstrator's actions with object states and movements.

This structured representation forms the basis for constructing the task model introduced in section 4.3, supporting the extraction, interpretation, and reproduction of human demonstrations in the semantic robot programming system.

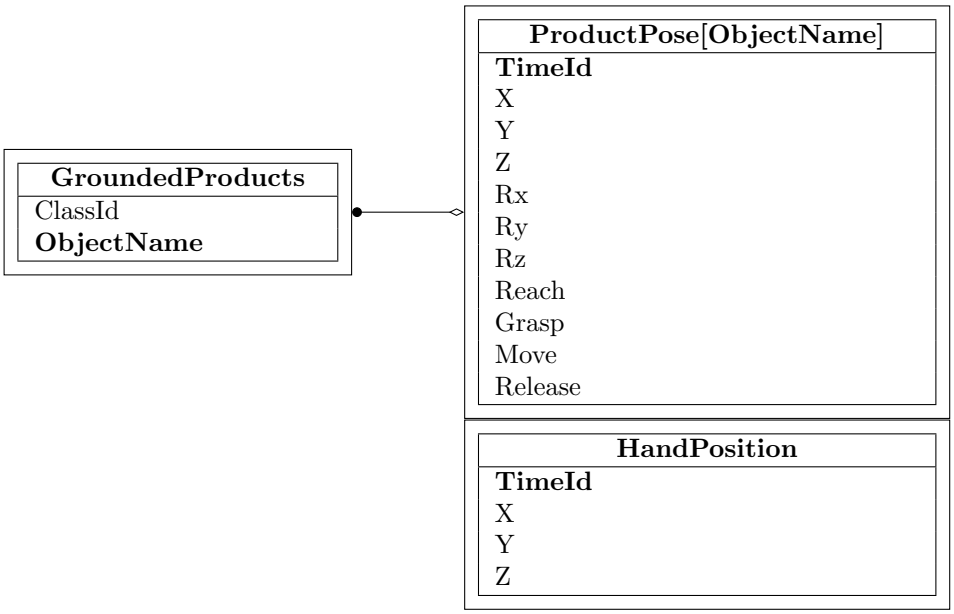


Fig. 6.5: Entity-relationship diagram for representing task knowledge by extracting information from visual programming by demonstration

6.4 Mapping motion from a human hand to a robot end-effector

This section introduces a framework for mapping the motion of a human hand to a robot end-effector. The approach in this work focuses on single-hand demonstrations, where only one hand interacts with the objects throughout the task. This simplification reduces ambiguity when associating hand trajectories with object movements and enables more robust segmentation and skill learning. In contrast, two-hand demonstrations involve coordinated actions between both hands, which introduces additional challenges such as role assignment (e.g., active vs. assisting) and the need to map motions either to a single-arm robot or to a dual-arm system. While certain two-hand actions could still be executed by a single robot arm, others may require a dual-arm setup and more complex coordination, which are beyond the scope of this work.

The framework starts by generating hand and object trajectories from continuous frame-by-frame hand and object detection. These trajectories are then segmented based on the hand and object interaction state and the object state. The resulting hand motion segments *reach* and *move* are represented as skills

with DMPs. Fig. 6.6 shows an example of the *reach* motion mapping. These representations are integrated into the task model described in section 4.3 to reproduce the task.

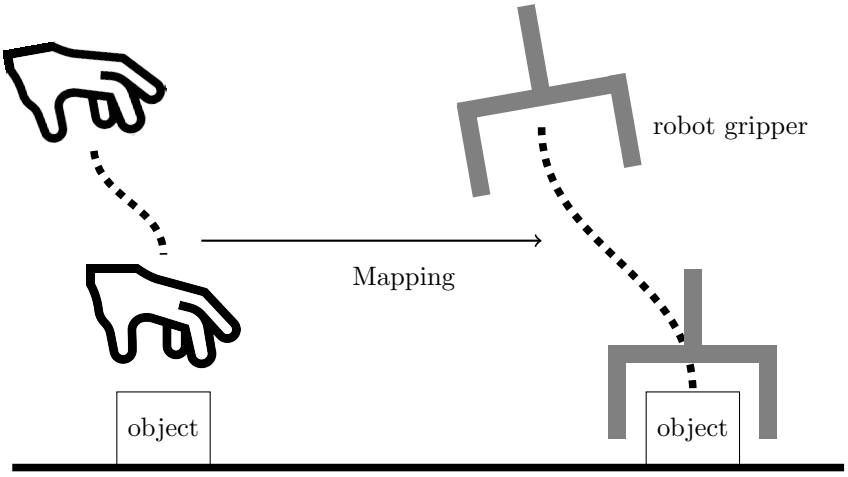


Fig. 6.6: Mapping *reach* motion from a human hand to a robot end-effector. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

6.4.1 Trajectory generation for the human hand and the objects

An RGB-D camera is required to record the visual demonstration. Hand and object detection are performed on the RGB frames. The detected results are bounding boxes at pixel coordinates. The corresponding depth values can be read from the depth frame. The equation 6.2 shows the process. The center of detected bounding boxes from RGB images for the human hand and objects are X_{center} and Y_{center}. The responding depth value Z_{center} can be identified from depth image. Similar as RGB image, the depth image is represented by a two-dimensional matrix, $\mathcal{I} = (y,\ x)$, the coordinate $(x,\ y)$ corresponds to the column and row indices, respectively. The depth value Z_{center} can be identified from depth image $\mathcal{I}(Y_{center},\ X_{center})$. The resulted sequence of $\mathcal{P} = \{X_{center},\ Y_{center},\ Z_{center}\}$ are hand and object trajectories in pixel coordinates as shown in fig. 6.7. In the next step, the position is transformed to real world coordinates with camera as origin. In the equation 6.2, f_x and f_y describe the focal length of the image, and the *ppx* and *ppy* describe the pixel coordinates

of the principal point. They are camera intrinsic parameters to describe the internal characteristics of the camera.

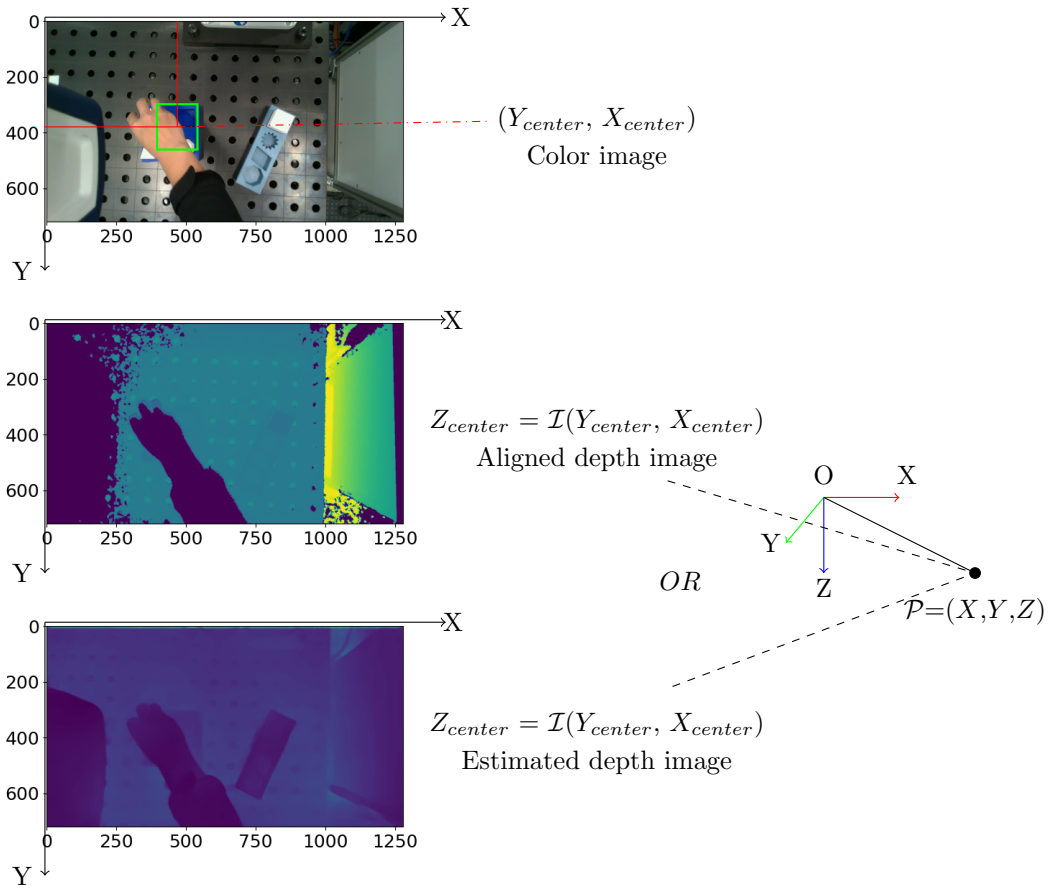


Fig. 6.7: Generating trajectories by transforming hand and object positions from pixel coordinates to world coordinates in the camera frame. Adapted from Lu et al. (2025), licensed under CC BY 4.0.

$$
\begin{aligned}
X &= \frac{X_{center} - ppx}{f_x \cdot Z_{center}} \\
Y &= \frac{Y_{center} - ppy}{f_y \cdot Z_{center}} \\
Z &= Z_{center}
\end{aligned}
\tag{6.2}
$$

According to Ghidoni (2023), several technologies are available for estimating depth information: stereoscope, infrared light and LiDAR. A stereoscopic camera

captures two images simultaneously from different angles to determine distance. Infrared and LiDAR technologies work on a similar principle. They use a light source and a receiver to measure the time it takes the light to reach the sensor. Both of these technologies are referred to as Time-of-Flight (ToF) sensors. Each technique can be affected by lighting conditions and the texture of objects, and have invalid depth values. To overcome bad depth value, an additional estimated depth map from the color image is used in this work. It uses the global-local path networks model for monocular depth estimation (Kim et al., 2022). The model is made available by Hugging Face[6]. In case the depth value is unavailable in aligned depth frame from camera, it will be estimated from color frame, the transformation from pixel coordinates to camera frame is also performed using equation 6.2.

By repeating the aforementioned process for each pair of RGB and depth frames, the hand and object trajectories can be generated in 3D space. Normally, the object detection accuracy cannot reach 100%. If the hand and the object of interest cannot be detected in some frames, the last valid detection is propagated to the next frame. The generated hand and object trajectories are $Traj = (\mathcal{P}_1, \cdots \mathcal{P}_i, \cdots, \mathcal{P}_T)$.

6.4.2 Motion representation for a robot's end-effector

To map hand motions to the robot's end-effector, the hand trajectories are segmented into smaller components, which allows the hand to adapt to the end-effector position, as shown in fig. 4.4. Methods-Time Measurement (MTM) provides a systematic approach for representing human motions. Among several variants, the MTM-1 method for representing hand motion has been widely used in the context of human-robot collaboration, as seen in studies such as Berg (2020) and Riedel et al. (2022). Therefore, the MTM-1 is also applied in this work to analyze hand motion.

In the MTM-1 method, hand motions are categorized into five basic motions: *reach*, *grasp*, *move*, *position*, *release*. Each basic motion is assigned a specific duration based on defined influencing factors, such as distance, complexity, or

[6]https://huggingface.co/vinvino02/glpn-nyu

physical effort (Bokranz and Landau, 2012). As described by Bokranz and Landau (2012), the *reach* movement describes the movement of the hand or finger to a specific target. Its duration can be influenced by factors such as size, shape, weight, surface texture, and stability of the object, as well as the length of the movement. *Grasp* is the basic movement that is used when the primary purpose is to secure sufficient control of one or more objects with the fingers of the hand to enable the performance of the next required basic movement. *Move* represents the purpose of transporting objects. *Position* is the basic element used to align, orient, and engage one object with another when the movements used are so minor as not to warrant classification as other basic elements. The time for position is influenced by the class of fit, symmetry, and ease of handling. *Release* is the basic element for relinquishing control of an object by the fingers or hand.

While human hand motions are continuous in nature, a robot end-effector typically operates in a discrete manner. For the *grasp* and *release* actions specifically, the robot's end-effector can be taken as discrete event and commanded at specific timestamp. The method to identify the timestamp will be discussed in the following subsection. The skills' representation for robot's end-effector during the pick & place process are summarized in table 6.1, where $\mathcal{P}(t)$ represents the position of end-effector at timestamp t.

Table 6.1: Representation of motion as skills for robots

Actions	**Skills**	**Segmented hand trajectories**
Pick	*Reach*	$[\mathcal{P}(t_1), \mathcal{P}(t_2), \ldots, \mathcal{P}(t_{g-1})]$
	Grasp	$\mathcal{G}(\mathcal{P}(t_g), t_g)$, $\mathcal{P}(t_{g-1}) = \mathcal{P}(t_g)$
Place	*Move*	$[\mathcal{P}(t_{g+1}), \mathcal{P}(t_{g+2}), \ldots, \mathcal{P}(t_{r-1})]$
	Release	$\mathcal{R}(\mathcal{P}(t_r), t_r)$, $\mathcal{P}(t_{r-1}) = \mathcal{P}(t_r)$

The action of picking an object involves *reach* and *grasp*. The *reach* skill represents the motion of the end-effector from a starting point to grasping position. Once the end-effector has reached the object, the *grasp* occurs in the next timestamp. After grasping an object, the skill *move* is instantiated to transport the object to the desired location. Instead of separating the two basic

motions *move* and *position* for human hands, transporting an object by robot is taken as a continuous motion. After arriving the desired position, *release* is triggered to release control of the object. An example of the segmented trajectories are illustrated in fig. 6.8.

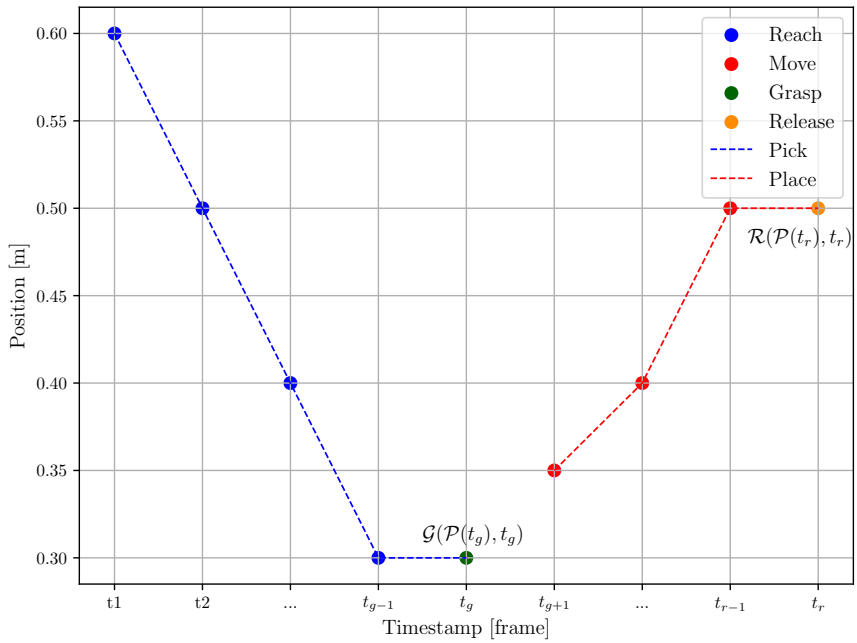


Fig. 6.8: Illustration of the segmented trajectories. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

In the practice of robot programming, after defining the grasp and release positions, additional pre- and post-motion segments are created to ensure a controlled grasp and release (Berg, 2020). These predefined positions facilitate smooth and precise motion, supporting safety mechanisms such as speed reduction between transitions.

The pre-grasp involves approaching the object at reduced speed and corrected orientation. Once the robot reaches the pre-grasp pose, it performs the grasp action to securely hold the object. After successfully grasping the object, the robot moves back to post-grasp pose with the reversed approach trajectory to avoid collision with the environment. The pre- and post-grasp pose are typically the same, ensuring consistency in the robot's pose. Once in the post-grasp pose, the robot transitions to the *move* skill to transport the object to the desired place position. The pre-release involves preparing the robot and the object for release.

It includes adjusting the robot's pose to ensure a controlled release. The speed of pre-release and post-release is usually reduced compared to the transport motion. After releasing the object, the robot returns to the post-release pose, preparing for the next operation. Similar to the grasping process, the pre- and post-release poses are typically identical, ensuring repeatability and precision in motion execution. Fig. 6.9 illustrates the grasp and release process.

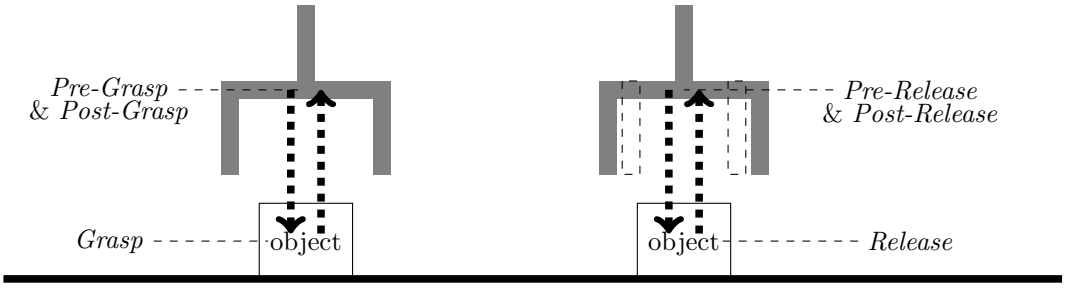


Fig. 6.9: Illustration of pre-grasp, post-grasp, pre-release, post-release phases. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

In summary, to map a human hand trajectory to a robot's end effector, the hand trajectories are first segmented in to *reach*, *grasp*, *move* and *release*, and then augmented with *pre-grasp*, *post-grasp*, *pre-release* and *post-release*. The orientation $\mathcal{O}(t)$ of the objects is determined during the *pre-grasp* and *pre-release* phases based on object detection results and remains constant throughout the *post-grasp* and *post-release* phases.

$$\mathcal{P}_{\textit{pre-grasp}} = \mathcal{P}_{\textit{post-grasp}} = \mathcal{P}(t_g) \pm \Delta p$$
$$\mathcal{O}_{\textit{pre-grasp}} = \mathcal{O}_{\textit{post-grasp}} = \mathcal{O}(\textit{object}, t_g)$$
$$\mathcal{P}_{\textit{pre-release}} = \mathcal{P}_{\textit{post-release}} = \mathcal{P}(t_r) \pm \Delta p$$
$$\mathcal{O}_{\textit{pre-release}} = \mathcal{O}_{\textit{post-release}} = \mathcal{O}(\textit{object}, t_r)$$

6.4.3 Trajectory segmentation

This subsection explains how hand trajectories are segmented into smaller elements to represent the skills of *reach*, *grasp*, *move*, and *release*. This is accomplished by identifying the timestamps for grasping and releasing. The

input data consists of the hand and object trajectories generated in section 6.4.1, as well as the information extracted from the natural language instructions, including the objects of interest and the order of their movements. Segmenting hand trajectories directly can be a challenging task. Because hand movements often transition smoothly from one phase to another without clear and distinct boundaries. It is therefore difficult to identify where one segment ends and another begins. To address this issue, an approach is proposed that leverages the status of hand-object interaction and object status.

Before segmenting the trajectories, missing values caused by detection failure are filled by propagating the last valid value to the next one. Then, a Kalman filter is applied to reduce noise. The hand trajectories are first segmented into pick and place cycles for each object by identifying the release timestamps. They are achieved by segmenting the object trajectories by GMM. It is a statistical model that represents a probability distribution $p(x, y)$ between input x and output y variables as a weighted sum of Gaussian distributions.

$$p(\mathcal{X}, y|\theta) = \sum_{k=1}^{K} p(w_k)p(\mathcal{X}, y|\mu_k, \Sigma_k) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathcal{X}, y|\mu_k, \Sigma_k), \tag{6.3}$$

where each Gaussian component $\mathcal{N}(x, y|\mu_k, \Sigma_k)$ has prior probability $p(w_k) = \pi_k$, mean μ_k, and covariance matrix Σ_k. GMM parameters $\theta = \pi_k, \mu_k, \Sigma_k$ are learned from training data using the expectation-maximization as described in section 2.1. It can be utilized for motion segmentation by modeling the distribution of trajectories over time. In a pick & place task, the object trajectory can be divided into three distinct parts, the static state before picking, the moving state, and the static state after placing. The first timestamp in the state after placing indicating the *release* of the hand. In this work, the object trajectories are segmented by the GMM model with $K = 3$ and $\mathcal{X} = Traj = \{\mathcal{P}_1, \cdots, \mathcal{P}_i, \cdots, \mathcal{P}_T\}$.

In the second step, the *grasp* timestamps are used to segment the pick and place motion into *reach* and *move*. When a human hand and other objects are present in the same visual frame, occlusion can occur, which means the hand might partially or fully block the view of the object. This can result in inaccurate object trajectories that do not accurately represent the grasping

process. However, the object can be correctly detected at the beginning and end of the demonstration, where the human demonstration has not yet sarted or finished. The *grasp* timestamp is then determined by calculating the shortest distance between the hand and the object's initial position. The results are *grasp* and *release* index in the time sequence for each object. The segmentation approach is summarized in fig. 6.10.

```
   /* The sequence of object labels moved by the robot       */
   Input: objectSequence = [3, 5, 2, 4]
   Input: handTraj: hand trajectory sequence
   Input: objTraj: object trajectory sequence
   Output: {objectLabel: [graspIndex, releaseIndex]}
 1 handTraj ← smoothTraj(handTraj);
 2 start, end ← process(handTraj);
 3 foreach objectLabel ∈ objectSequence do
 4 |   objTraj ← objTraj(objectLabel);
 5 |   objTraj ← smoothTraj(objTraj);
 6 |   gmm ← GaussianMixture(n_components = 3);
 7 |   gmm.fit(objTraj);
 8 |   labels ← gmm.predict(objTraj);
 9 |   releaseIndex ← the first frame in the last cluster;
10 |   initialPosition ← mean(objTraj[: start]);
11 |   finalPosition ← mean(objTraj[end :]);
12 |   dists ← empty list;
13 |   foreach x ∈ handTraj[start : releaseIndex] do
14 |   |   dist ← Euclidean distance between x and initialPosition;
15 |   |   Append dist to dists;
16 |   end
17 |   minValue ← minimum value in dists;
18 |   minIndex ← index of minValue in dists;
19 |   graspIndex ← minIndex + start;
20 |   start ← releaseIndex;
21 end
```

Fig. 6.10: Algorithm for segmentation of hand trajectories

The resulted sequence symbols for three objects can be represented as [*Reach*(O_1), *Grasp*(O_1), *Move*(O_1), *Release*(O_1), *Reach*(O_2), *Grasp*(O_2), *Move*(O_2),

Release(O_2), *Reach*(O_3), *Grasp*(O_3), *Move*(O_3), *Release*(O_3)].

6.4.4 Trajectory learning & generation

After segmenting the trajectories into *reach*, *grasp*, *move* and *release*, this section discusses how to represent the *reach* and *move* in world coordinates so that the robot can execute the movements. As discussed in section 3.2, DMPs are capable of learning from a single demonstration and adapting to varying start and goal positions (Kyrarini et al., 2019). This aligns with the research goal G3, which aims to reduce number of demonstrations and G4, which focus on flexibility. DMPs are thus used to model the goal-directed behaviors. The resulting *reach* and *move* segments are represented by DMPs, which are formalized as stable nonlinear attractor systems (Schaal, 2006). There are many variations of DMPs. However, as summarized by Fabisch (2020), they share the following properties:

- they have an internal time variable (phase), which is defined by a so-called canonical system,
- they can be adapted by tuning the weights of a forcing term and
- a transformation system generates goal-directed accelerations.

Based on the formulation of DMPs by Ijspeert et al. (2013), Pastor et al. (2009), and Fabisch (2020), the relevant equations are described as follows. The transformation system is a spring-damper system and generates a goal-directed motion that is modified by a forcing term f.

$$\begin{aligned} \tau\dot{v} &= \alpha(\beta(g - x_0) - v) + f \\ \tau\dot{x} &= v \end{aligned} \tag{6.4}$$

where x and v are position and velocity of the system, x_0 and g are the start and goal position, τ is a temporal scaling factor. α and β are constants that have to be set for critical damping, common values are $\alpha = 25$, $\beta = \alpha/4$. The forcing term f can be chosen hypothetically:

$$f(s) = \frac{\sum_{i=1}^{N} \phi_i(s) w_i}{\sum_{i=1}^{N} \phi_i(s)} s(g - x_0) \tag{6.5}$$

with parameters w controlling the shape of the trajectory. $\phi_i(s) = \exp(-\frac{d_i}{2}(s - c_i)^2)$ are Gaussian basis functions with constant widths d_i and centers c_i. The function f depends on a phase variable s, which monotonically changes from 1 towards 0 during a movement and is obtained by the equation

$$\tau \dot{s} = -\alpha_x s \tag{6.6}$$

where α_x is a pre-defined constant. The equation 6.6 is referred to as canonical system.

During demonstration, a movement $x_{\text{demo}}(t)$ is recorded and its derivatives $v_{\text{demo}}(t)$ and $\dot{v}_{\text{demo}}(t)$ are computed for each time step $t = 0, ..., T$. $s(t)$ is computed for temporal scaling. The equation 6.4 can be rewritten as:

$$f_{\text{target}}(s) = -\alpha\beta(g - x_{\text{demo}}) + \alpha v_{\text{demo}} + \tau \dot{v}_{\text{demo}} \tag{6.7}$$

The task is to adjust the parameters of $f(s)$ such that it is as close as possible to $f_{\text{target}}(s)$. The function $f_{\text{target}}(s)$ is then learned by identifying the parameters w_i by locally weighted regression (Ijspeert et al., 2013), which is done by minimizing the quadratic error,

$$J = \sum_s (f_{\text{target}}(s) - f(s))^2 \tag{6.8}$$

To overcome the uncertainties in the demonstration data, the regularization coefficient λ in the function approximators are also defined by experiments. The parameter is used to prevent overfitting by penalizing larger parameter values.

$$w = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T y \tag{6.9}$$

The choice of coordinates for representing a model can make a big difference in how the generalization of the dynamics systems appears (Ijspeert et al., 2013). The DMPs are represented in the camera frame in both the trajectory learning and generation processes. Fig. 6.11 illustrates the steps for generating robot end-effector trajectories using DMPs. To generate the trajectory for *reach*, the process begins with defining the current position of the end-effector and aims to

reach the pre-grasp position. The learned DMPs from the hand trajectories are updated based on defined initial and goal position, as discussed in the subsection 6.4.2. This process generates a smooth trajectory for the specified duration. Similarly, DMP for *move* is updated by defining the post-grasp position and the pre-release position. The adaptability of DMPs allows for updating with new initial and goal position, ensuring controlled movements.

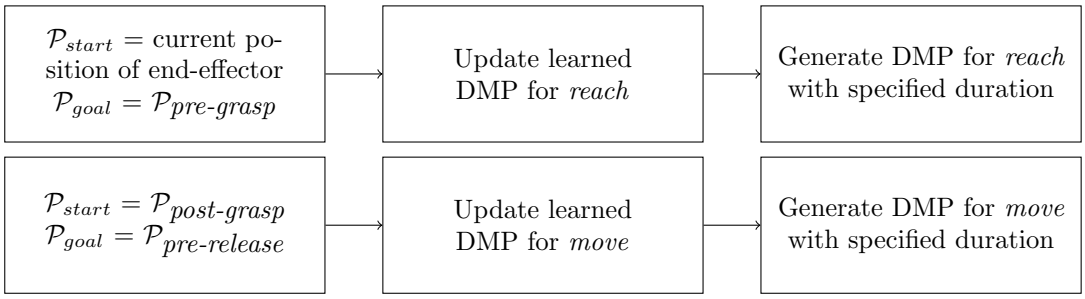


Fig. 6.11: Trajectory generation process. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

6.5 Summary

This chapter introduced a structured, multimodal framework for extracting and analyzing motion data from human demonstrations using an RGB-D camera. By integrating visual, linguistic, and synthetic data generation techniques into a unified pipeline, the approach enables one-shot semantic robot programming, offering a more efficient and flexible alternative to existing multi-demonstration or manually programmed methods.

The chapter first examined hand and object detection from RGB images, complemented by depth values derived from camera intrinsic parameters. Two primary models were explored for hand detection: Faster-RCNN and MediaPipe. For object detection, OBB is preferred over traditional Bbox because it can align with the orientation of an object. To generate training images and annotations, the BlenderProc pipeline was identified to generated photorealistic images and annotations from CAD models. A pre-trained YOLOv8 model was selected for object detection. By detecting hands and objects separately, the integration of pre-trained models becomes feasible. Furthermore, training the model with

photorealistic images significantly reduces the effort involved in data collection. These approaches align with the research hypothesis H2, presented in section 3.6.

The next step is to discuss how language input can be used to improve the extraction of visual information. By incorporating the information extracted from chapter 5, the system can automatically generate training data based on the defined object of interest. Additionally, the object detection results can be improved by leveraging static state information, while trajectory segmentation can be supported by analyzing object sequences. These approaches align with research hypothesis H3.

Finally, the process of mapping motion from a human hand to a robot's end-effector is discussed. To enable semantic understanding and execution of pick & place tasks, demonstrations were segmented into discrete actions using object states and hand-object interaction cues. A GMM was employed to identify motion phases, and DMPs were used to model reusable and adaptable robot skills. This enables the system to generalize motion execution to new situations, supporting one-shot programming.

This chapter established the methodology for integrating multiple information extraction techniques into a cohesive framework. Its performance evaluation and validation will be presented in chapter 8.

Chapter 7

An integrated framework for semantic robot programming

This chapter presents a framework for semantic robot programming that integrates information from both natural language instructions and visual demonstrations. It corresponds to the research plan P3 outlined in section 4.6. The framework supports data acquisition, processing, the generation of grounded knowledge representations, and the synthesis of executable robot programs. A portion of this work was previously summarized in the conference paper by Lu et al. (2022b); the content of that publication, along with subsequent developments, is incorporated into this chapter. In addition to outlining the conceptual design of the framework, the chapter also details the software implementation of each individual module.

7.1 Overview

Developing an integrated framework is essential for creating a unified system that can interpret and act upon the multimodal information in a cohesive manner. This framework bridges the gap between understanding instructions and visual PbD. Chapter 5 and 6 detail text and video processing respectively, providing the necessary background for the development of the integrated framework. This integration is critical for the programming system that can adapt to new environments. It ensures that both descriptive and procedural knowledge are synthesized to form a complete task model.

Based on the research hypothesis H4 in section 3.6, the framework is structured into six distinct software modules. A conceptual architecture and process flow for programming and execution using these six software modules is illustrated in fig. 7.1. During the programming phase, the system ingests language and video inputs, processes them, and stores the extracted knowledge. During the execution phase, it checks again for updated inputs if the task plan or environment changes, and then adapts the stored information to generate robot actions appropriate to the current conditions. The responsibilities of each software module are summarized as follows:

1. Language Module interprets high-level instructions by applying the classifier and syntactic dependency parser introduced in chapter 5.
2. Environment Module perceives and interprets the human demonstration and the robot's environment. It uses camera data to detect objects, locations, and other context. The methods from chapter 6 for generating hand and object trajectories are part of this module.
3. Motion Module handles robot motion learning and generation, producing the trajectories that the robot will execute.
4. Task Module maintains a representation to store the processed input from both Language Module and Environment Module, serving as a structured format that keeps track of all task information. The basic structure is based on the semantic task representation developed in section 4.3.
5. Programming Module records the language input and demonstration video, invokes the above modules to process this data, and ultimately saves a new or updated robot program in the Task Module.
6. Execution Module manages task execution by retrieving the relevant robot program from the Task Module, updating it if necessary based on real-time environmental changes (detected again by Environment Module), and then using Motion Module to generate the final robot actions.

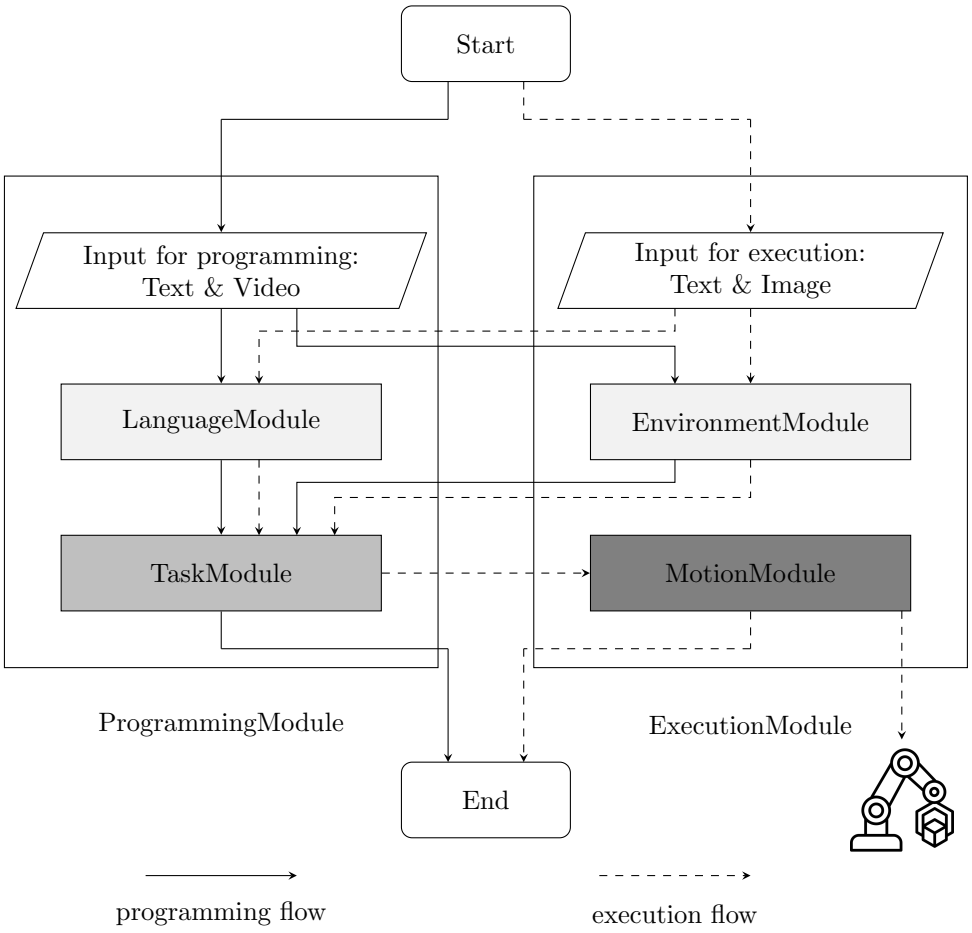


Fig. 7.1: Conceptual architecture and process flow of the six software modules for programming (left) and execution (right)

As mentioned above, the methods for the modules Language Module, Environment Module, and Motion Module have been developed in chapter 5 and 6. Fig. 7.2 shows the class diagrams for the three modules. Each module focuses on a specific task, which provides the flexibility to adapt the system to different requirements or input modalities. This enables various applications, such as training new objects of interest for different tasks. This approach aligns with the research goal G4 in section 3.6. The remaining three modules will be discussed in the following sections in details.

LanguageModule
- train_spert_classifier() - run_spert_classifier() - dependency_parser() - correference_resolution() + process_text()

(a) Class LanguageModule and its methods

EnvironmentModule
+ video2frame() - depth_estimation() + grounding_products() + train_object_detector() + detect_hand() + detect_object() - process_hand_trajectory() - process_object_trajectory() + segmentation_hand_trajectory() + process_video()

(b) Class EnviornmentModule and its methods

MotionModule
+ learn_trajectory() + generate_trajectory()

(c) Class MotionModule and its methods

Fig. 7.2: UML class diagrams of the Language Module, Environment Module, and Motion Module

7.2 Task module

The Task Module establishes a unified database structure that integrates language and visual inputs. Meanwhile, it provides methods for database interaction, including querying, updating, and efficient data management. The knowledge representation was introduced in sections 5.1 and 6.3 respectively, this module focuses on connecting the different entities through the database for a unified task representation. The database consists of seven interrelated table types. The entities defined in the knowledge representation are organized into tables with primary and foreign keys to ensure proper relationships between them.

Four of these tables are dedicated to encapsulating task knowledge extracted from language inputs, as shown in fig. 7.3. They are the implementation of knowledge representation shown in fig. 5.1. Following are the four tables which store information from language:

- The *Products* table: this table represents the entity *object* in fig. 5.1. It stores the objects of interest in a structured manner, with *ObjectId* serving as the primary key, ensuring each product mentioned in the instruction is uniquely identifiable within the database. The *Color* attribute describes the products' color. The attributes *SentenceId* and *TokenId* are used to support the language processing. *ClassId* is a categorization attribute used in conjunction with visual input to generate the unified task model. *SourceLocationId* and *TargetLocationId* are foreign keys that establish relationships with the *SourceLocations* and *TargetLocations* tables.
- The table *SourceLocations* store spatial indicators and their related objects. The objects are represented by *TrajectorObjectId*, which are foreign keys that reference the *ObjectId* in the *Products* table, defining the object's presence at the source locations. This relational structure supports complex queries and operations on the data, allowing the system to track the data during programming and execution.
- The *TargetLocations* table: this table stores goal indicators and their related objects. The objects are presented by *IndicatorObjectId*, which are also foreign keys to reference the *ObjectId* in the *Products* table. They define the object's target location.

- The *TaskModelTargetObjects* table: this table records the sequences of operations related to products as instructed. It stores information solely about the target objects that require movement, forming a task plan. The *TaskModelId* attribute group together all objects that are part of a single tasks, while *TargetObjectId* provides a reference to the *ObjectId* of the product that is to be moved. This approach simplifies task representation within the system by focusing on object relocation rather than action sequencing in fig. 5.1. It simplifies the grounding process with visual input by providing a clear and direct reference to the target objects that are involved in tasks.

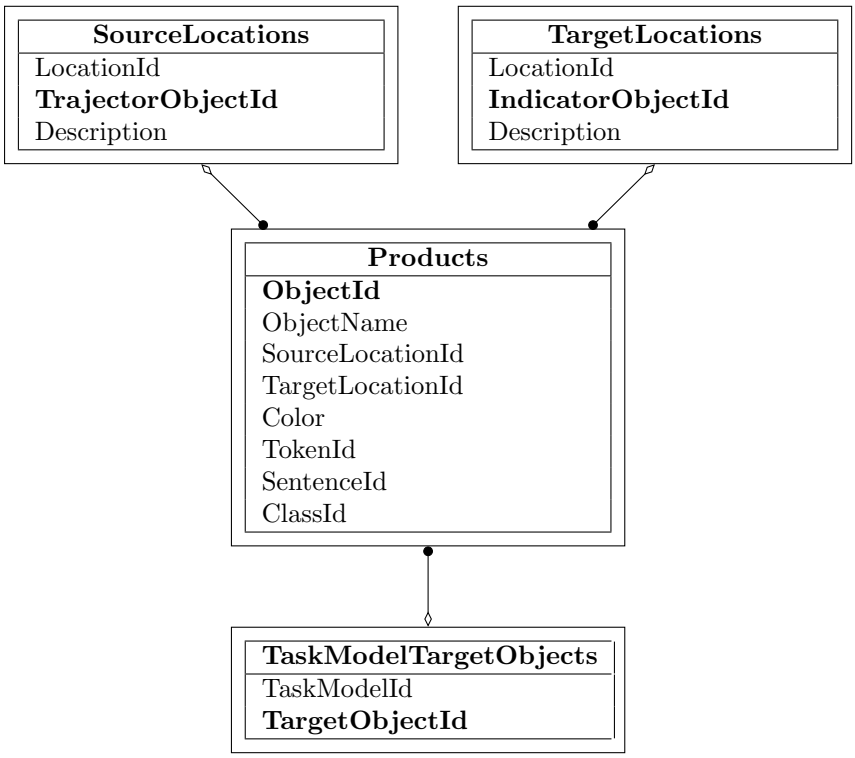


Fig. 7.3: Entity-relationship diagram for representing task knowledge by extracting information from natural language instructions

The remaining three tables store and manage information derived from visual PbD, as shown in fig. 7.4, implementing the representation of fig. 6.5. Following are the three tables which store information from visual input:

- The *GroundedProducts* table: using the results from the *Products* table in fig. 7.3, an object detector can be trained to detect OBBs. The detected

objects by Environment Module are then linked with language input and saved in the new table *GroundedProducts*. The table extends the entity *GroundedProducts* in fig. 6.5 with attributes *SoruceLocationId* and *TargetLocationId*. It stores information about identified objects. By referring to the tables *Products*, *SourceLocation* and *TargetLocation* in fig. 7.3, it contains their classification, name, and where they are located and where they should be moved to.

- The *ProductPose* table: in this table, the trajectories before *grasp* indicates that the object is at the source location, and after *release*, they indicate that the object are placed in target location. The trajectories in between show the object being moved with the hand. This structure ensures a comprehensive record of an object's movement during a task, including its starting and ending static positions and contexts. The language's spatial relation, such as `in` or `on`, can be utilized to establish context between different objects.
- The *HandPosition* table: this table stores the hand trajectories which are synchronized with the table *ProductPose* by *TimeId*. This synchronization enables the segmentation of hand movements into defined motion units: *reach*, *grasp*, *move*, and *release*. By correlating the hand's position with object states and actions over time, the system can interpret manipulation phases and derive meaningful patterns from demonstrations, facilitating the learning and reproduction of tasks by robots.

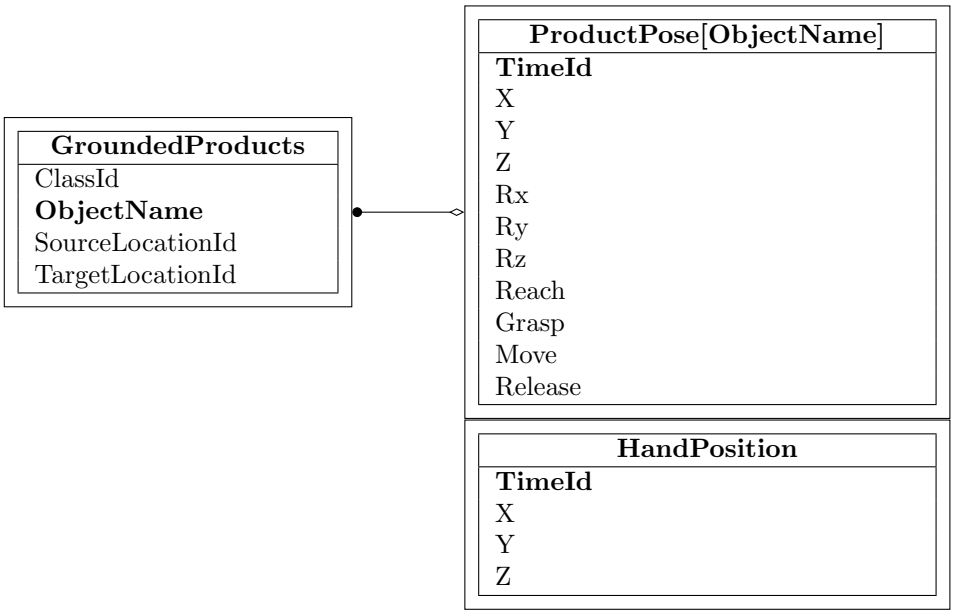


Fig. 7.4: Entity-relationship diagram for integrated task knowledge from visual and language input

As previously stated, methods have been developed to simplify the process of grounding information when interacting with the database. These methods also aid in retrieving grounded information to support task execution. Specifically, the Data Access Object (DAO) design pattern is applied to each table within the Task Module. This pattern centralizes all the database-related code in one place and provides an abstract interface during programming and execution. It consists of methods to *add*, *get*, *update* and *delete* data. To systematically represent the structure and functionality of each software module, a UML class diagram is used. This diagram provides a clear and structured visualization of the module's design, facilitating easier modification and further development of the software framework. The class diagram for Task Module is illustrated in fig. 7.5.

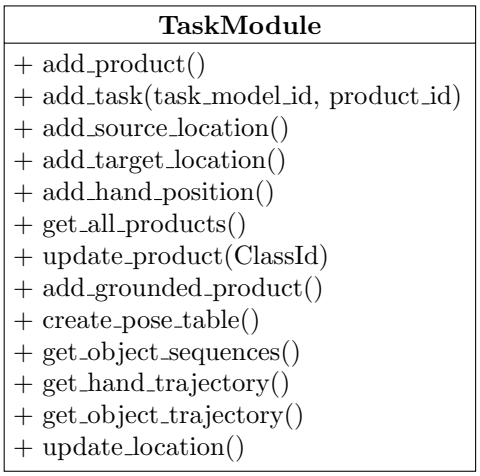


Fig. 7.5: UML class diagram for software module: Task Module

7.3 Programming module

The Programming Module is an essential component of the programming system. It is designed to handle various forms of input to complete information extraction processes. The process of programming a task starts with recording incoming data, followed by processing textual and visual content. The information extraction process is realized by Language Module and Execution Module. Finally, the task representation developed in the Task Module is saved in the database. The class diagram is shown in fig. 7.6.

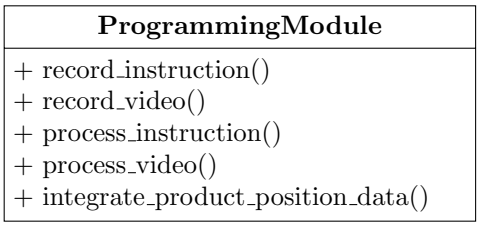


Fig. 7.6: UML class diagram for software module: Programming Module

The processing of the text input is shown in detail in fig. 7.7. The raw input text is first segmented into sentences. Then, coreference resolution is performed to link pronouns and nouns that refer to the same entities. After that, each sentence is tokenized into individual words or phrases. The SpERT classifier trained in section 5.3 is then applied to these tokens to categorize them. The dependency parser is then applied to analyze the grammatical structure of

sentences and identify relationships between words. The extracted information is lastly stored in the database tables shown in fig. 7.3.

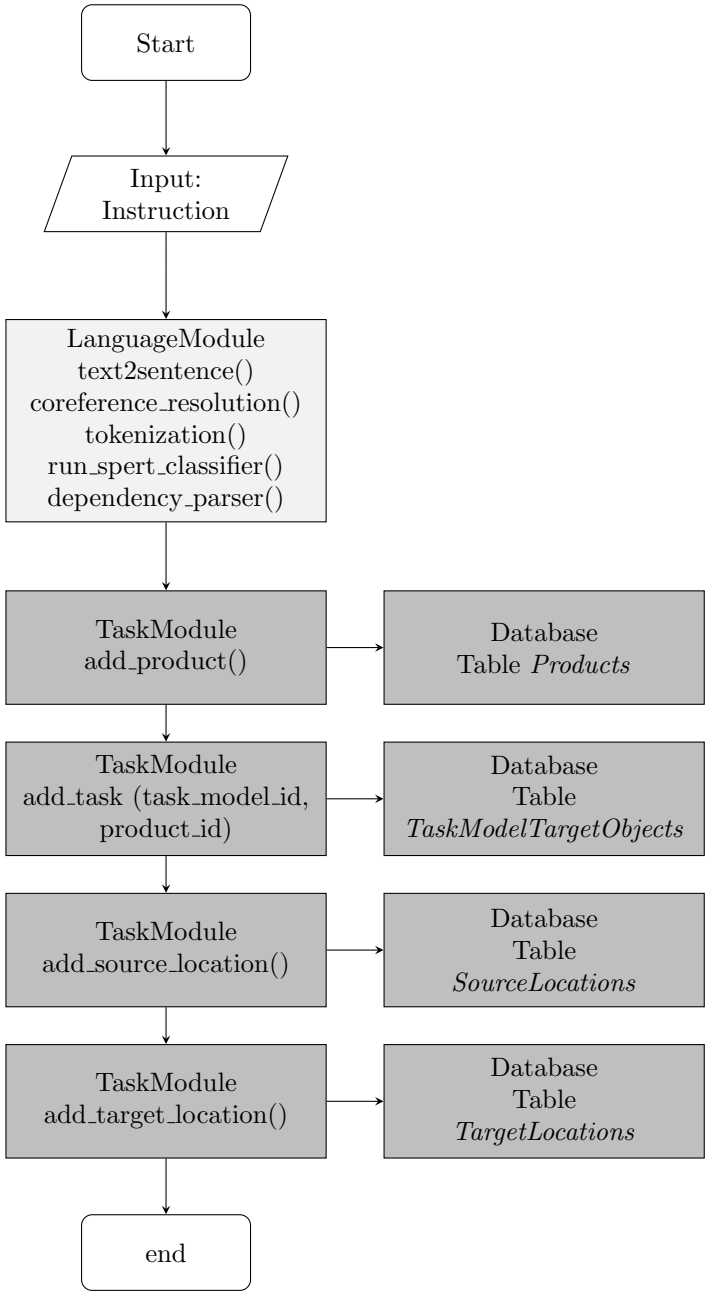


Fig. 7.7: Process flowchart of the text input in Programming Module

Fig. 7.8 shows the flowchart of the video input processing. It starts by converting the video into frames and detecting hands whose positions are stored in the database. In the next step, the Task Module retrieves the complete list of

objects of interest from the *Products* database table. This list is required for the Environment Module to train the object detector, which is then used to identify objects in the recorded demonstration video. The process categorizes products and updates data in the *GroundedProducts* database table, providing a grounded representation between language and visual input. The detected position and orientation for each object in the frame are stored in the table for its category. The hand trajectories are segmented in the next step by identifying the grasp and release timestamps for each object. This process is supported by the sequence of moved objects stored in the *TaskModelTargetObjects* table. Finally, the source location is related to *TimeId* before grasping, and the target location is referred to as *TimeId* after the release event. The *ProductPose* table is updated accordingly. In the next step, the robot can generate task programs and motion for execution using the saved database.

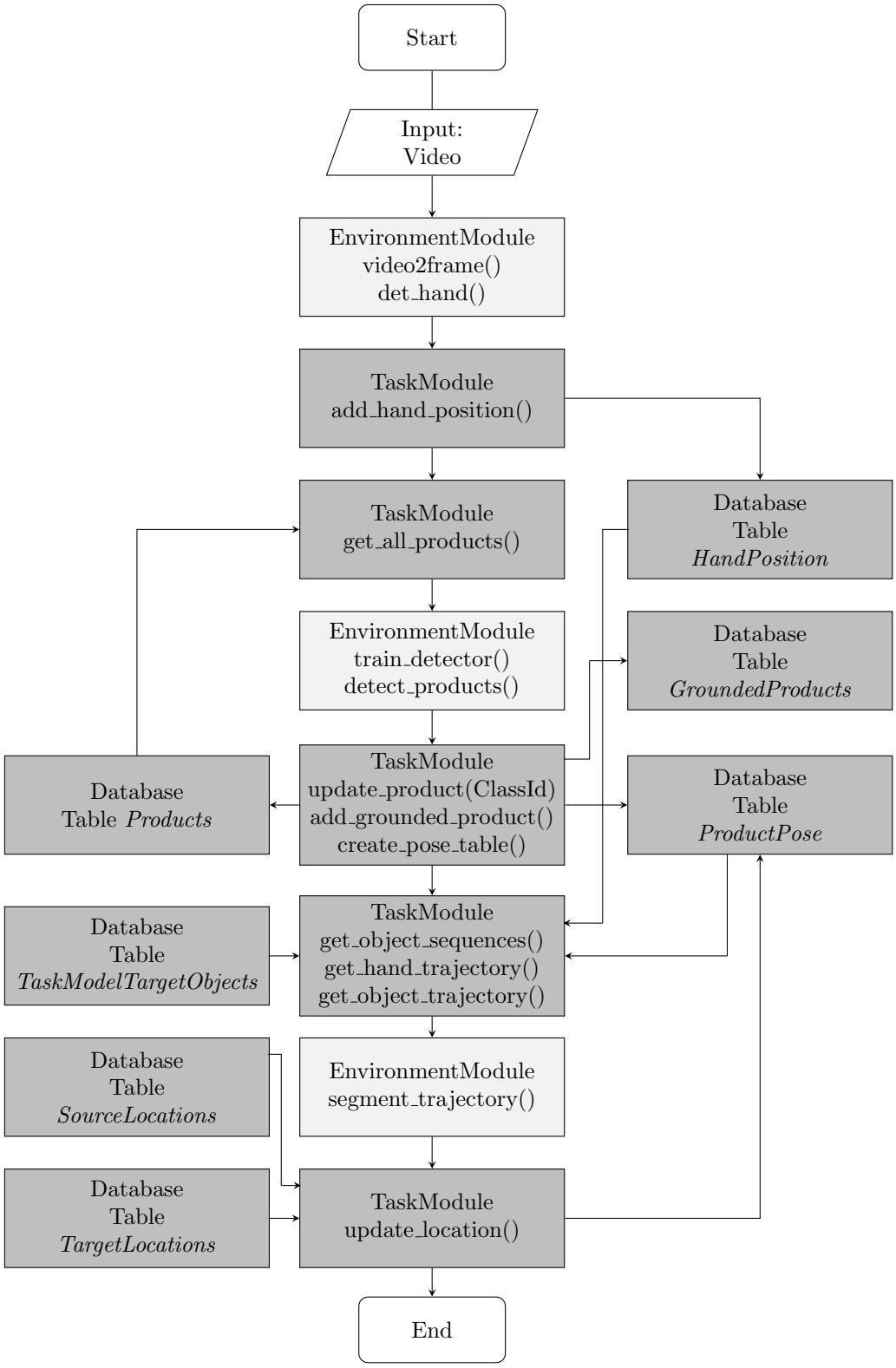


Fig. 7.8: Process flowchart of the video input in Programming Module

7.4 Execution module

The Execution Module is responsible for task execution. It can retrieve and modify existing tasks from the database in Task Module and adapt them to new instructions in the context of the environment. Specifically, it generates a task plan and replicates motions demonstrated during the training phase and accommodates pose changes of target objects, making it adaptable to variable environments. The class diagram is shown in fig. 7.9.

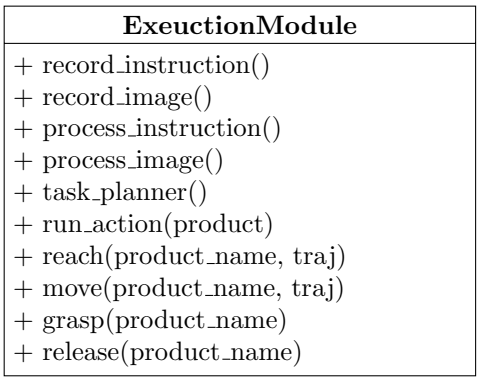


Fig. 7.9: UML class diagram for software module: Execution Module

Fig. 7.10 presents a visual representation of the operation of the proposed programming system's framework, illustrating the sequence of steps and the interaction between various modules during task execution. During the execution phase, the task planning process is initiated by the introduction of a new instruction. The Language Module processes the instruction by extracting specific commands and relevant task knowledge from the database to formulate a coherent task plan. The initial plan settings are captured using an RGB-D camera, enabling detailed environmental perception.

Simultaneously, the Environment Module identifies target objects within the robot's operational environment and interprets the specific requirements of the task based on the visual data gathered. Subsequently, the Motion Module leverages these visual inputs to generate the adapted DMPs for each action outlined in the task plan. These trajectories are then converted into executable movements and communicated to the robot for physical execution.

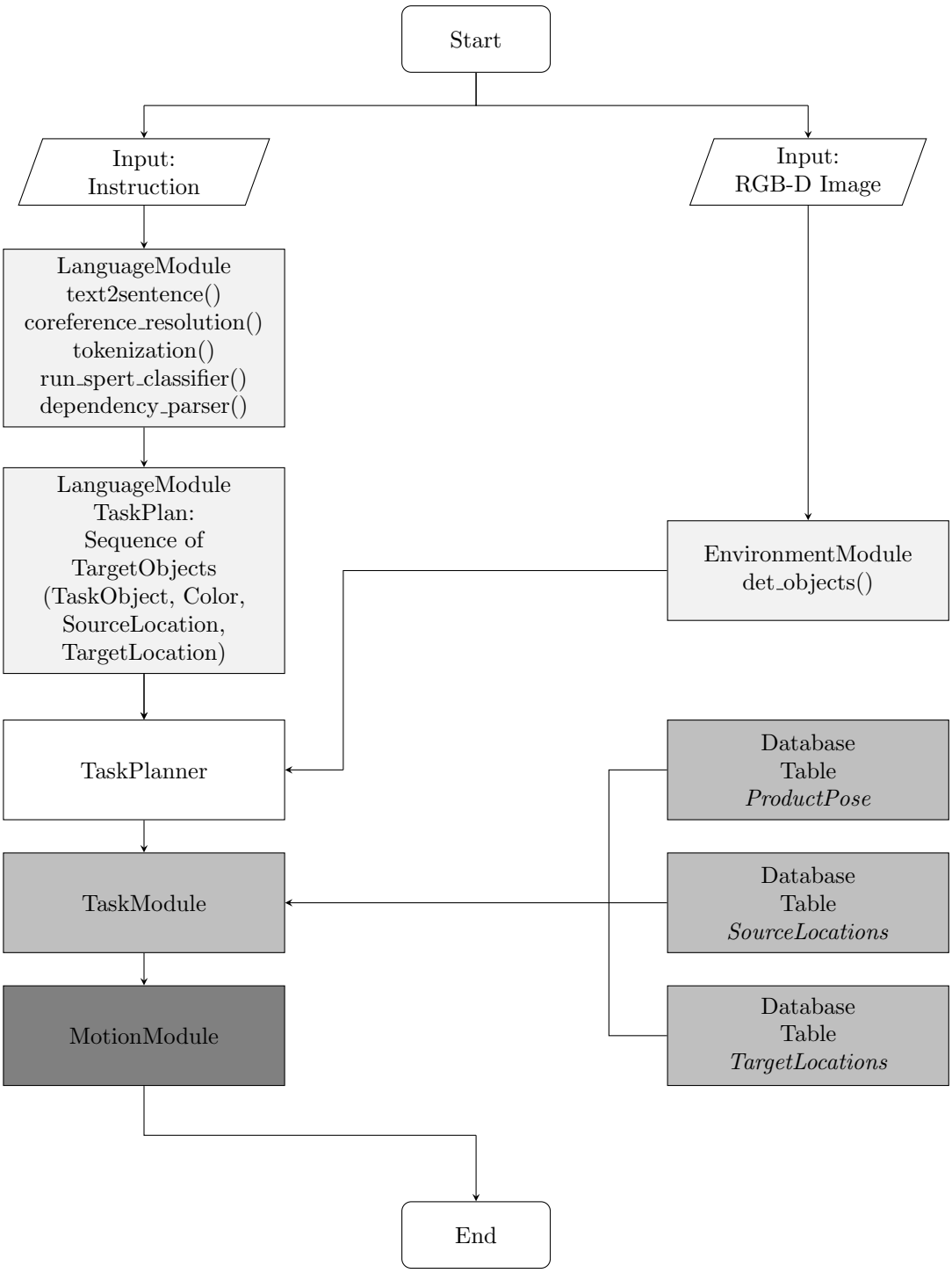


Fig. 7.10: Process flowchart of the execution process

The algorithm in fig. 7.11 describes the details about the planning process for the multi-step pick & place tasks. Based on the sequences extracted from the Language Module, the task planner executes the pick & place task for each target object step by step. At each step, the Environment Module detects the target object, source and target locations. If they exist, the robot will continue to perform the action. As proposed in section 6.4, the pick & place action for one object is divided into four skills: *reach*, *grasp*, *move* and *release*. The trajectories for *reach* and *move* are extracted from the table *ProductPose* and the table *HandPosition* in the database and represented by DMPs. The DMPs are generated based on the interpretation of the new environment. Specifically, the initial pose of the target object is estimated through object detection. When the robot's task is to replicate a human demonstration, it directly extracts the target pose of the object from the demonstration data. If the robot needs to determine the target pose independently, it estimates the target pose based on the object's location and its spatial relationship with nearby entities. For example, in the scenario of a "cuboid in a graybox", the robot first identifies the pose of the graybox using its camera. It then computes the cuboid's relative position and orientation within the graybox. This estimation can be derived from previously recorded demonstration data or from a CAD model that provides detailed spatial information.

```
1  Procedure TaskPlanner(targetObjects)
2      foreach targetObject ∈ targetObjects do
3          sourceLocation ← FindSourceLocation(targetObject);
4          targetLocation ← FindTargetLocation(targetObject);
5          if Exists(sourceLocation) and Exists(targetLocation) then
6              ExecuteAction(object, sourceLocation, targetLocation);
7          else
8              HandleError(object, sourceLocation, targetLocation);
9          end
10     end
11 end
12 Function ExecuteAction(object, sourceLocation, targetLocation)
       /* Task execution by robot                                    */
13     ReachIndices ← GetProductPoseTable(object);
14     MoveIndices ← GetProductPoseTable(object);
15     ReachTrajectory ← GetHandPoseTable(ReachIndices);
16     MoveTrajectory ← GetHandPoseTable(MoveIndices);
       /* Get initial and target pose                                */
17     InitialPose ← ObjectDetection(object);
18     TargetPose ← ObjectPoseEstimation(targetLocation, database);
19     PreGrasp, PostGrasp ← UpdateGraspPose(InitialPose);
20     PreRelease, PostRelease ← UpdateReleasePose(TargetPose);
       /* Motion generation                                          */
21     ReachDMP ← DMP(ReachTrajectory, Start, PreGrasp);
22     MoveDMP ← DMP(MoveTrajectory, PostGrasp, PreRelease);
23 end
```

Fig. 7.11: Algorithm for task planner

7.5 Summary

This chapter introduces a novel integrated framework for semantic robot programming, designed to interpret and execute tasks by combining natural language instructions and visual demonstrations. Compared to state-of-the-art approaches, which typically adopt end-to-end learning pipelines and require a large number of demonstrations for training, this framework distinguishes itself by employing a modular design that reduces data dependency and increases interpretability.

The core concepts of six software modules are outlined in the first section. The modular approach provides flexibility in the system, as mentioned in the research hypothesis H4 in section 3.6. Each module specializes in a particular aspect of the work. Methods in each module can be added, removed, or replaced without affecting the entire system. Considering the complexity of semantics in robot programming, it's essential for the system to be capable of incorporating and processing a wide range of semantic information.

The second section presents the development of a relational database in which information about their tasks is stored. The database consists of seven tables. *SourceLocations* and *TargetLocations* table are used to store identifiers and descriptions for the initial and final spatial relations of objects. The *Products* table is used to store information related to objects. Tasks are linked to their respective target objects through the *TaskModelTargetObjects* table. Grounded information, derived from visual input, is stored in the *GroundedProducts* table. Time-specific poses of objects are captured in the *ProductPose[ObjectName]* table, while the *HandPosition* table is used to track the Cartesian coordinates of the human hand over time. Together, these tables provide a structured foundation through which instructions and environmental data can be translated into executable task plans by the robot.

In the third section, the process of interpreting instructions and visual PbD was described by the Programming Module, and the extracted information was stored in the database. The procedure was initiated by reading the instruction input and recording the demonstration video. Subsequently, the acquired data were processed and saved in the database introduced in the second section.

In the fourth section, the Execution Module is described, which is responsible for translating abstract task plans into concrete actions that can be performed by the robot. The execution of planned tasks generated by the Programming Module is managed by this module, thereby bridging the gap between task planning and physical execution. Key aspects covered in this section include the execution of motion trajectories generated by the Motion Module, the monitoring of task progress, and the handling of unexpected events or errors during execution.

Chapter 8

Implementation and validation of the programming system

In this chapter, the programming system (see fig. 7.1) is implemented and evaluated using a multi-step pick & place task. It involves transporting four geometric shapes from the starting position to the target position. This chapter corresponds to P4 in the research plan outlined in section 4.6. The task instructions and demonstration videos are recorded to guide the evaluation. The system's modules are implemented and evaluated individually to assess their functionality and performance.

8.1 Experimental setup

The aim of this thesis is to extract the necessary information from both a single instruction and a corresponding video. To ensure robustness and avoid potential bias or reliance on singular cases, five sets of instructions and their corresponding videos were recorded and analyzed. An example of the instructions is as follows:

Begin by picking up the cuboid from the blue box and placing it in the gray box. Next, carefully lift the star and position it next to the cuboid in the gray box. Then, move the parallelogram, place it beside the star in the gray box. Finally, pick up the octagon and place it adjacent to the parallelogram in the gray box.

Additional instructions for the test, generated by ChatGPT, can be found in appendix A. The programming system received these instructions in text form.

Fig. 8.1 (a) illustrates the experimental setup to record the videos. The Intel® RealSense™ L515 3D camera is mounted on the robot to record the demonstration process. This setup allows optimal positioning of the camera to capture both the task environment and human movements. The task environment is shown in fig. 8.1 (b). As a LiDAR camera, it projects an infrared laser at 860 nm wavelength as an active light source. 3D data is obtained by evaluating the time required for the projected signal to bounce off the objects in the scene and come back to the camera (Servi et al., 2021). The frame rate is 30 frames per second (FPS). The videos were recorded using Intel® RealSense™ Viewer and saved as bag files, which can be extracted as sequence of color and depth images. The size of color images recorded by the L515 is 1280×720 and the size of depth image is 640×480. The depth frame is re-scaled and aligned[1] to color frame, such that the depth value can be read by pixel coordinate of color image.

Multiple camera positions were used to evaluate the robustness and effectiveness of the proposed approaches from different angles and perspectives. The videos feature the same four-step pick & place tasks, which involve transporting a cuboid, star, parallelogram, and octagon from a blue box to a gray box. At the beginning and end of the recording, the task environment was observed without the presence of hands, ensuring a clear view of the working environment for accurate analysis of the conditions before and after the manipulation was performed. The task was performed using only one hand, as the motion mapping method presented in section 6.4 was developed for single-hand demonstrations. However, the underlying methodological approach introduced in section 4.1 is generally applicable to tasks involving two hands and can be extended to handle more complex, coordinated bi-manual actions in future work.

The software modules developed in chapter 7 were implemented in a Python environment with the objective of evaluating the technical capabilities of the programming system. The evaluation process involved assessing the functionalities of each module. By testing each module, the overall performance of the complete programming system is therefore validated.

1. Language Module: This module is validated by evaluating its ability to accurately interpret and process natural language instructions in a new

[1]https://dev.intelrealsense.com/docs/rs-align-advanced

Fig. 8.1: Overview of the demonstration setup and recorded RGB-D data (a) Demonstration setup; (b) Color image; (c) Aligned depth image Adapted from Lu et al. (2025), licensed under CC BY 4.0.

domain. The analysis and results of this evaluation are presented in section 8.2.

2. Environment Module: The validation of this module focuses on assessing its capability to detect and interpret visual information from PbD inputs. The evaluation process and findings are summarized in section 8.3.
3. Motion Module: This module is validated by evaluating how effectively it segments and learns motion trajectories from demonstration data. The validation methodology and results are described in section 8.4.
4. Programming Module & Task Module: These modules are validated by assessing how well the system translates processed language instructions and environmental data into a structured task model and corresponding action sequences. The validation results are presented in section 8.5.
5. Execution Module & Task Module: The validation of these modules involves testing the system's ability to reproduce and generalize learned tasks. The system is evaluated based on its capability to replicate demonstrated tasks and adapt to new task variations. The evaluation results are discussed in section 8.5.

8.2 Implementation and evaluation of the language module

This section describes the evaluation of the information extraction method developed in section 5.3, which has been implemented in Language Module. As

outlined in section 7.1, this module consists of several key components, each contributing to a distinct aspect of the information extraction process. The main components include pre-processing, the trained SpERT classifier and the synthetic dependency parser. Each component is evaluated individually to determine its accuracy and effectiveness in the information extraction process.

Pre-processing was carried out using spaCy, which is a widely used open-source NLP library in Python. This evaluation was performed using the instructions listed in appendix A, which cover a wide range of linguistic patterns and structures. The input text was correctly segmented into sentences. Each sentence was then processed using a cross-lingual coreference model[2]. This model replaces the pronouns such as "it" with the appropriate referent. The result of the example in section 8.1 is as follows:

Begin by picking up the cuboid from the blue box and placing cuboid in the gray box. Next, carefully lift the star and position the star next to the cuboid in the gray box. Then, move the parallelogram, place the parallelogram beside the star in the gray box. Finally, pick up the octagon and place the octagon adjacent to the parallelogram in the gray box.

Additional results are listed in appendix B. After coreference resolution, the sentences were then further broken down into tokens. An example of tokenization for the first sentence mentioned above is as follows:

{"*tokens*": ["begin","by","picking", "up", "the", "cuboid","from","the","blue", "box", "and", "placing", "cuobid", "tinhe", "the","gray", "box", "."]}

In conclusion, the pre-processing methods were able to effectively process input text into tokens.

The second focus of this evaluation is to test the accuracy of Language Module in processing and extracting relevant information on test data that were not exposed to the system during the training phase. This testing is essential to confirm the strength and adaptability of the proposed approach to real world linguistic variation. Notably, the test instructions in appendix A contain different items compared to those in the training dataset in section 5.1, ensuring an evaluation of the model's ability to generalize. The tokens from the pre-processing step were further classified by the trained SpERT classifier

[2]https://spacy.io/universe/project/crosslingualcoreference

in section 5.3. Table 8.1 summarizes the classification results. The "support" column in the table refers to the number of actual occurrences of the class from the four input sentences. The results show that the model has high precision and recall for most entity classes. However, `spatial relation` has the lowest scores across all metrics, similar as the validation set performance in section 5.4. While the overall accuracy is acceptable for an information extraction system (Eberts and Ulges, 2019; Hillebrand et al., 2022), further improvements are necessary to meet the demands of a semantic programming system.

The syntactic dependency parser achieves complete accuracy in its operations. These results highlight the effectiveness of the proposed information extraction method when applied to new test data from different domains that were not included in the training. However, the results depend on the correctly recognized `spatial_indicator` or `goal_indicator`. The combined error in the two-stage process indicates the need to improve the performance of the SpERT classifier, which could be achieved by incorporating a more extensive dataset.

Table 8.1: Evaluation of the SpERT classifier for the test set

	Precision	Recall	F1-Score	Support
`action`	90.7%	100%	95.12%	39
`trajector`	85.71%	85.71%	85.71%	14
`spatial_indicator`	91.3%	100%	95.45%	21
`goal_indicator`	94.44%	89.47%	91.89%	19
`color`	100%	100%	100%	25
`spatial relation`	60%	64.29%	62.07%	14
`move relation`	77.27%	80.95%	79.07%	21

In the test instructions provided, it should be noted that in some sentences the action for the `pick` procedure is omitted or assumed rather than explicitly stated. It turns out that the proposed `move relation` annotation between the `action` and `goal_indicator` entities is advantageous so that the information grounding with visual PbD at temporal level can be done correctly.

8.3 Implementation and evaluation of the environment module

This section evaluates the methods in the Environment Module based on the five recorded RGB-D videos. Each video was pre-processed to separate color and depth frames. This step is essential for enabling the subsequent analysis of the spatial and temporal dimensions of hand and object movements within the RGB and depth frames. This evaluation aims to assess the effectiveness of the proposed approach in capturing and interpreting complex interactions in dynamic environments.

8.3.1 Evaluation of hand detection methods

In section 6.2.1, two methods for hand detection are discussed. The first method is the bounding box-based approach developed by Shan et al. (2020), which was trained with the Faster-RCNN model. The second method is the keypoints-based approach from Google, known as MediaPipe. Both methods were applied to the RGB image sequences. After detection, the results were further processed by transforming the hand locations into Cartesian coordinates in the camera's coordinate system. This transformation leverages the depth value of the center point of the detected hand, which is crucial for accurately determining the hand's position in three-dimensional space. This step enables the generation of hand trajectories.

The models provided by Shan et al. (2020) are available online[3] as open source. The detection process is integrated into Detectron2[4], which is a framework for computer vision tasks such as object detection and instance segmentation. The model trained on 100K+ego was used to run the test. The score threshold for filtering out low-confidence detection was set at 0.7. Any model output below this threshold is considered as invalid. A low threshold means that the model accepts detections with lower confidence levels, which can significantly increase the number of false positives. The performance of the both methods are evaluated on the five recorded videos, which consist of a total of 3601 RGB

[3]https://github.com/ddshan/hand_detector.d2
[4]https://ai.meta.com/tools/detectron2/

images, where the hand is visible in 2631 images. The results are summarized in the table 8.2. The percentages in the table represent the proportion of images where the methods successfully detected hands when hands were present. It was observed that the model failed when the hand was too small in the image, an example is shown in fig. 8.2.

Fig. 8.2: An instance of missed hand detection by the model Faster-RCNN. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

The keypoints-based method MediaPipe[5] is an open-source framework, which is available for use with different programming languages and devices. The hand landmarks detection solution in Python within this framework is used to evaluate the performance. The parameter static_image_mode was set to True and the maximum number of hands was set to 1. The result in table 8.2 show that the Faster-RCNN is generally more reliable for hand detection tasks than MediaPipe.

Although 100% hand detection accuracy is ideal, it is not a strict requirement for the proposed system. In practice, minor detection failures can be compensated by interpolation techniques and filtering. In this work, missing detections are handled by propagating the last valid hand position forward in time, followed by the application of a Kalman filter to smooth the resulting trajectory, as presented in section 6.4.3. This approach aligns with findings in the literature, where similar methods have been employed to compensate for noise and occasional detection failures in PbD systems, such as Xin et al. (2023) and Shan et al. (2020). Therefore, a detection accuracy of over 90% is considered sufficient.

[5]https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

Table 8.2: Comparison of hand detection accuracy on recorded videos between Faster-RCNN and Mediapipe. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

	Faster-RCNN	Mediapipe
Video 1	100.00%	93.99%
Video 2	95.76%	8.86%
Video 3	99.82%	15.74%
Video 4	92.59%	9.94%
Video 5	91.67%	12.50%

The hand trajectories were computed using the detection results from Faster-RCNN because it has better performance. The approach was presented in section 6.4.1. An example of a hand trajectory is shown in fig. 8.3. From this graph, the hand movement is most variable along the z-axis (depth), as indicated by the green line. This is due to the hand moving towards and away from the camera for the multi-step pick & place task.

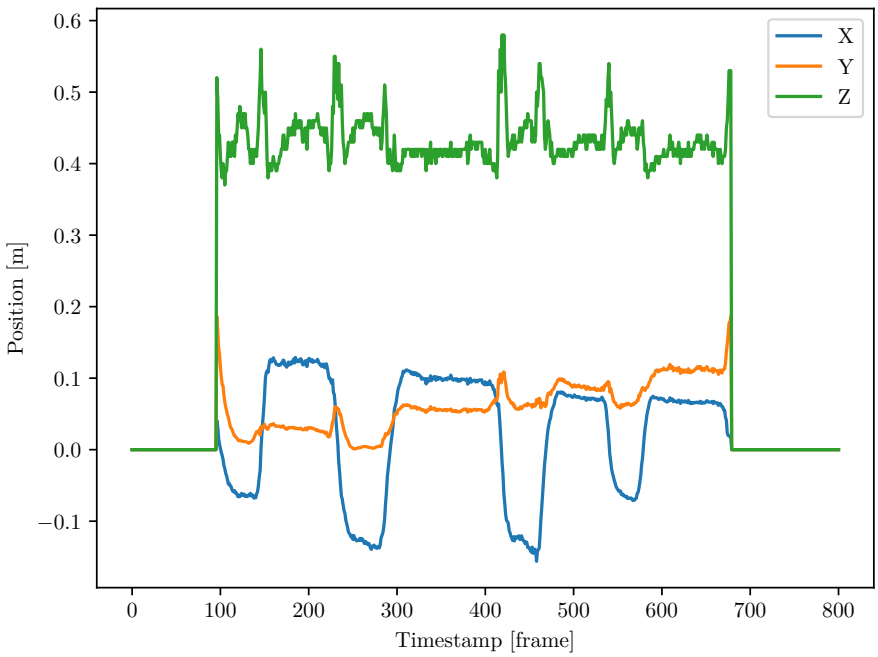


Fig. 8.3: Generated hand trajectory in camera frame with Faster-RCNN. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

8.3.2 Training and evaluation of the object detection model

Unlike hand detection, where pre-trained models and large datasets are available, detecting less common or task-specific objects generally necessitates the generation of a dedicated dataset for model training. Accordingly, a total of 18 750 photorealistic images were generated using the approach discussed in section 6.2.2 to train the YOLOv8 model for the OBB detection task. Out of these, 15 000 images were used for the training set, and 3750 images were allocated for the validation set to evaluate the performance of the model.

As discussed in section 6.2.2, transfer learning approach is used with the pre-trained model YOLOv8 from Ultralytics[6]. During training, the number of epochs was set to 20. The IoU was set as 0.7. The learning rate was set to the default value of 0.01, as this has been demonstrated to be a common value for transfer learning tasks (Cardoza et al., 2022).

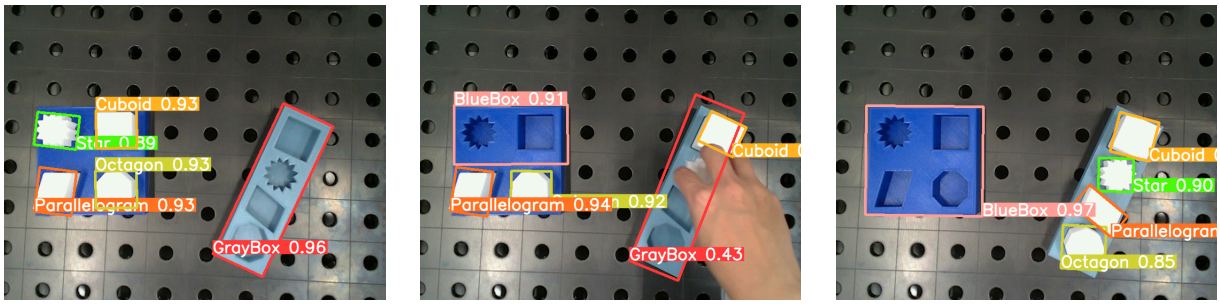
Table 8.3 presents the object detection results for the validation set consisting of 3750 images with a total of 21 821 instances across various classes. The overall precision and recall for detecting all classes are 98.5% and 99% respectively, with a mAP at 50% IoU threshold of 98.7% and mAP at 50-95% IoU of 47.8%. Each class, including GrayBox, BlueBox, Parallelogram, Cuboid, Octagon, and Star, has a high precision and recall, ranging from 96.8% to 99.3% and 97.4% to 99.4% respectively. The mAP50 for these classes varies slightly, between 98.4% and 99%, while the mAP50-95 ranges from 43.4% to 49.6%. The Star class has the highest mAP50-95 at 49.6% and shares the highest precision and recall with the Octagon class at 99.3%. The Cuboid class shows a slightly lower precision compared to others at 96.8%. Overall, the results indicate a high level of accuracy in the object detection task for this validation set.

The object detection model trained on photorealistic images was evaluated on real-world images to assess its performance. Three representative images with the detection results are presented in fig. 8.4. In these images, the model is able to identify and localize the object of interest in the scene. However, the occlusion caused by other objects and the presence of a human hand show poor performance.

[6]https://docs.ultralytics.com/tasks/obb/#models

Table 8.3: Object detection results on the validation set. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

Classes	Images	Instances	Precision	Recall	mAP50	mAP50-95
all	3750	21821	98.5%	99%	98.7%	47.8%
GrayBox	3750	3633	98.8%	99.1%	98.6%	43.4%
BlueBox	3750	3670	98.7%	99.3%	99%	48.4%
Parallelogram	3750	3637	98.3%	97.4%	98.4%	48.3%
Cuboid	3750	3598	96.8%	99.4%	98.4%	48.4%
Octagon	3750	3652	99.1%	99.3%	98.7%	48.6%
Star	3750	3631	99.3%	99.3%	99%	49.6%



Initial scene | Demonstration process | Final scene

Fig. 8.4: Object detection results on real world images. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

To evaluate the performance of the domain adaptation, images without the presence of a human hand were considered. Specifically, 971 images from the 5 recorded videos are extracted. Table 8.4 presents performance for different classes, including the number of images, instances and accuracy. The model achieves impressive performance on the entire dataset, with 100% precision indicating that all detected instances are correct. The recall of 83.3% suggests that the model is capturing a significant fraction of the actual instances. The mAP50 score of 90.5% indicates a high average precision at a threshold of 50. The model performs differently for each class. For instance, the GrayBox and BlueBox class exhibit lower recall values of approximately 50%. This is caused by the occlusion of the objects inside the box. The detection results can be

rectified by utilizing the information that both objects remain stationary during the demonstration. As expected, mAP50-95 scores are lower than mAP50, which is typical in object detection tasks due to the stricter IoU thresholds required for higher precision.

Table 8.4: Object detection results on real-world images. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

Classes	Images	Instances	Precision	Recall	mAP50	mAP50-95
all	971	5826	100%	83.3%	90.5%	72.1%
GrayBox	971	971	100%	50.3%	75.1%	38.5%
BlueBox	971	971	99.9%	49.7%	70.1%	70.1%
Parallelogram	971	971	100%	100%	99.5%	73.5%
Cuboid	971	971	100%	100%	99.5%	71.4%
Octagon	971	971	100%	100%	99.5%	89.9%
Star	971	971	100%	100%	99.5%	89.3%

Before trajectory generation for objects, the detection results are improved by incorporating language input, as described in section 6.2. The Language Module in section 8.2 identifies target objects, including a parallelogram, cuboid, octagon, and star, which are moved during the demonstration. In contrast, GrayBox and BlueBox remain static. The positions of these two static objects are corrected by relying on the object with the highest confidence score.

In summary, similar to the hand detection results, the performance of object detection can be further improved by post-processing techniques, particularly in refining object trajectories. Nevertheless, the achieved mAP50 with over 90% is considered sufficient for the purpose of this work, as the goal is to extract reliable object trajectories for task modeling and motion segmentation. For this purpose, a moderate spatial precision (e.g., correct localization with a 50% overlap) is adequate. Specifically, the performance of GrayBox and BlueBox can be improved by the latent information from language input. However, the lower mAP50–95 scores reflect a limitation in bounding box precision, highlighting the need for improved accuracy if the system were to be extended for object pose estimation.

8.3.3 Evaluation of depth value quality

Depth value is essential to generate the hand and the objects trajectories in Cartesian coordinates. Chen et al. (2020) discusses the common challenge of motion blur in depth sensing technology, particularly in dynamic movements with ToF sensors. Depth is calculated by measuring the time it takes for a light signal to travel to the object and back to the sensor. However, movement of objects can distort the timing measurement, resulting in inaccuracies. The experimental results of calculating the depth value of hand and object show the same pattern, where the depth value of hand trajectories is generally worse than that of object trajectories. Specifically, the study notes that valid depth values for hand trajectories fall below 50%, whereas for objects, they exceed 70%. In case of invalid depth value, the depth is estimated using the method discussed in section 6.4.1. The difference between the depth value estimated by the pre-trained deep learning model and the valid depth value from the camera is less than 1 cm. This evaluation demonstrates the necessity of using depth estimation techniques to ensure accurate trajectory generation and highlights the performance of the applied depth estimation method.

8.3.4 Evaluation of the proposed segmentation approach

The hand trajectories from the recorded videos were segmented using the algorithm in fig. 6.10. They are calculated by the object trajectories and the distance between the hand and the target object. An object trajectory is outlined in fig. 8.5. Each color represents a component generated by GMM. The trajectories were taken as correct segmented by inspecting the image at the identified timestamps for *grasp* and *release*. This method of verification ensures that the segmentation aligns with the actual moments when the object is manipulated. The approach was successful in identifying the *grasp* and *release* events for all target objects in the video.

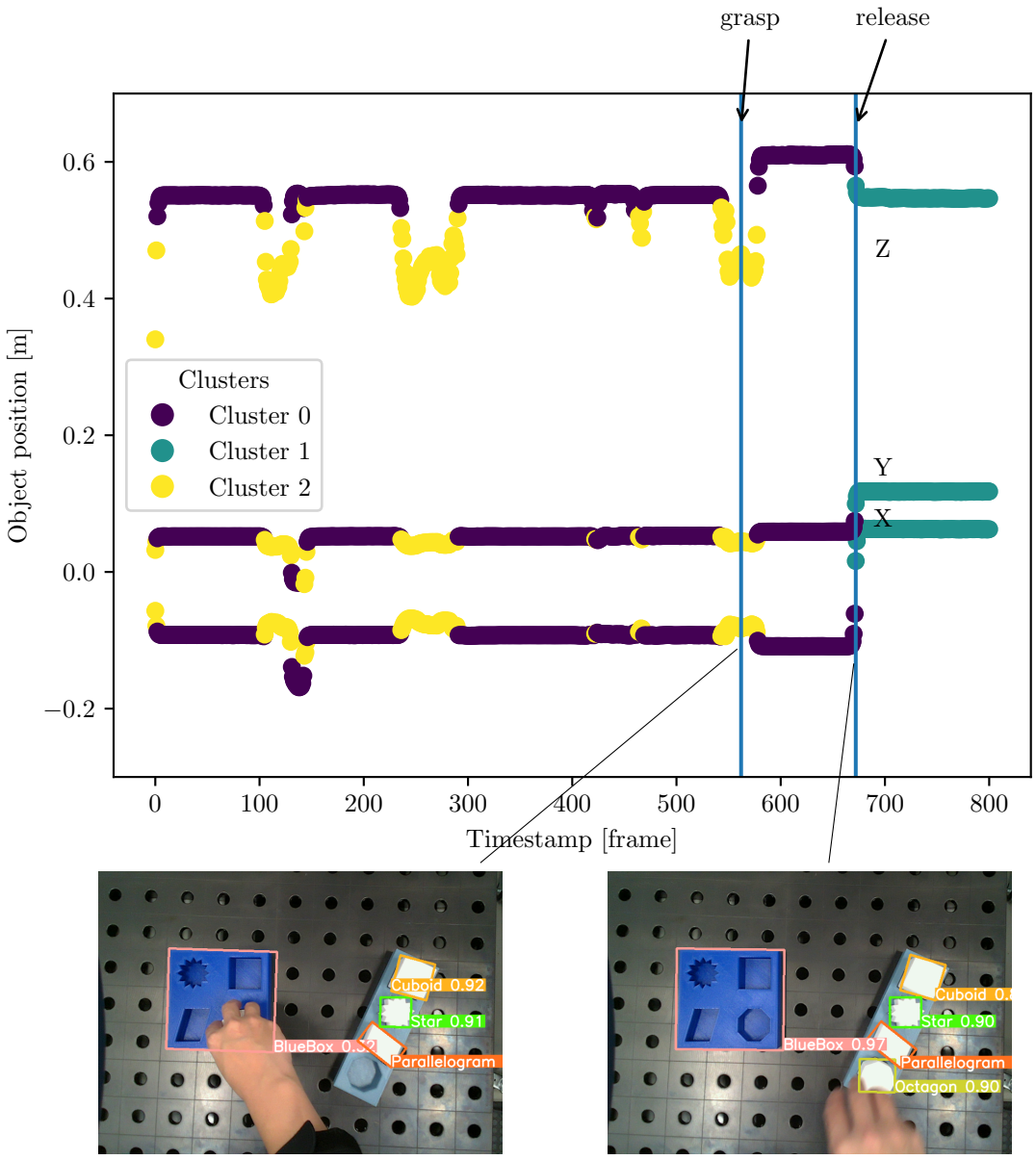


Fig. 8.5: Hand trajectory segmentation results. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

8.4 Implementation and evaluation of the motion module

In Motion Module the segmented trajectories are modeled by DMPs, which represent *reach* and *move* skills. As shown in fig. 8.3, the hand trajectories have irregularities, which illustrates the necessity for DMPs to smooth these trajectories. Moreover, the DMPs should be capable of generalizing to new start and target positions. The implementation of Fabisch (2020) in Python was used to learn the DMPs[7]. Fig. 8.6 illustrates the learned and updated DMP for representing the skill *reach* with regulation coefficient $\lambda = 0$. The number of ϕ functions in equation 6.5 was set to $N = 20$. The "Demonstration" curve represents a segmented trajectory of a hand motion from the starting point to a *pre-grasp* position. The "Reproduction" curve shows how the DMP was learned to imitate the original hand motion by attempting to follow the demonstrated trajectory. To enhance the manipulation accuracy, the DMP was adapted to object's position "New Start" and "New Goal". Since $\lambda = 0$, the noise and irregularities in the hand trajectory are encoded in the DMP, the phase approaching the goal in the figure shows the description. Fig. 8.7 illustrates the improved DMP with $\lambda = 0.1$, the "Reproduction" and "Adaptation" curves show that the influence of noise from the demonstration trajectory is reduced. Fig. 8.8 shows the learned and updated DMP for representing the *move*. In summary, the learned DMPs are able to adapt learned behaviors to new situations while preserving the details of the original demonstration.

[7]https://github.com/dfki-ric/movement_primitives

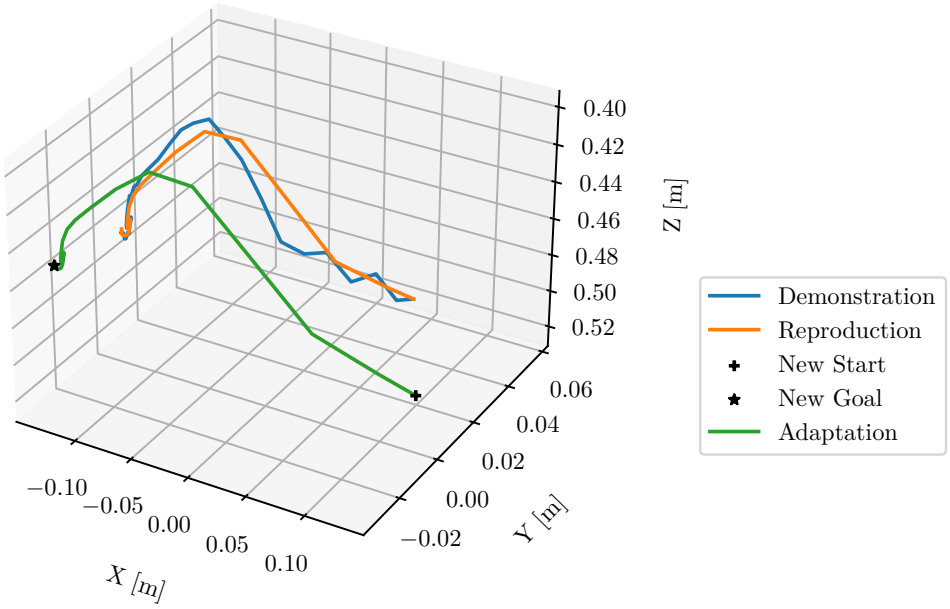


Fig. 8.6: Learned and updated DMP for representing *reach* with $\lambda = 0$ and $N = 20$. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

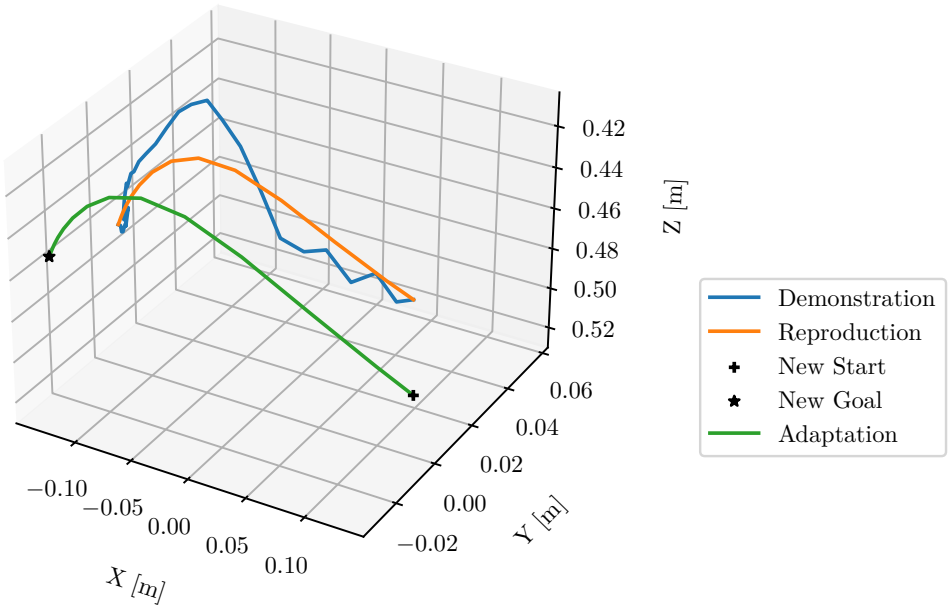


Fig. 8.7: Learned and updated DMP for representing *reach* with $\lambda = 0.1$ and $N = 20$. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

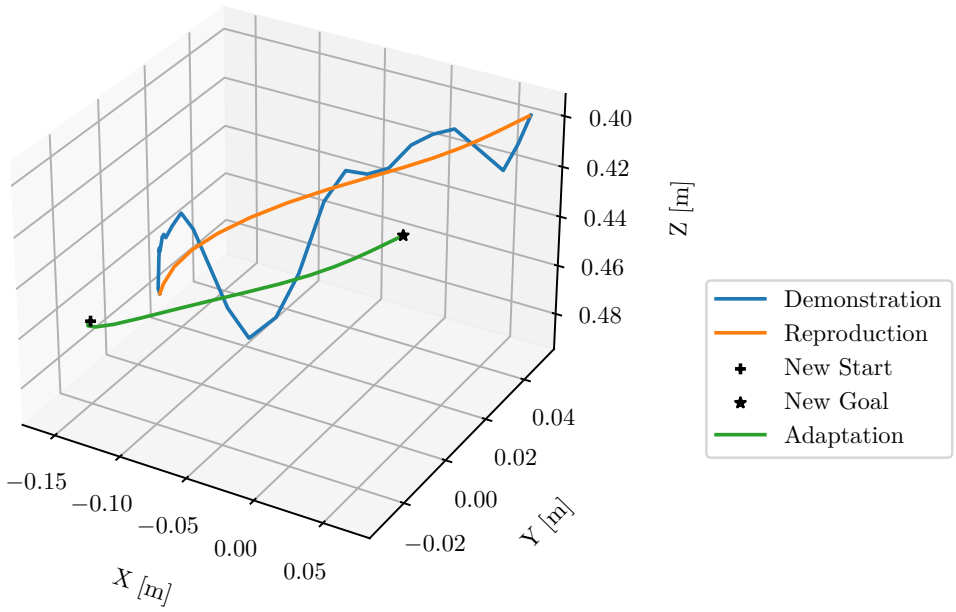


Fig. 8.8: Learned and updated DMP for representing *move* with $\lambda = 0.1$ and $N = 20$. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

8.5 Implementation and validation of the programming, execution & task modules

This section presents the implementation and validation the programming, execution, and task modules described in chapter 7. It focuses on evaluating the effectiveness of these modules in processing language and demonstration video inputs to generate a unified task model for robot execution. Additionally, this section illustrates how the extracted information is structured and stored within the system, providing insight into the practical implementation of the proposed approach.

8.5.1 Programming & task modules

The concept presented in section 7.3 was implemented in Python to validate the programming process with the language and video input. The programming process includes processing the instruction input in the first step and processing the demonstration video in the second step. The resulting data was stored in the

database by interacting with Task Module. The components of Language Module and Environment Module have been evaluated in the above sections. This section focuses on evaluating the effectiveness of the process of using these components to generate the unified task model. The programming process begins with recording the language and demonstration inputs, followed by further processing of these inputs.

The language processing starts with pre-processing, where raw input data is transformed into structured sentences. Next, coreference resolution is performed to ensure that subsequent analysis is based on clear and accurate context. After this step, the processed data is broken down into tokens. Following pre-processing, the system employs the trained SpERT classifier to perform predictions. The final component involves the dependency parser, which is applied to the classifier's results. This parsing step is specifically designed to identify objects of interest. The resulted information from language processing were saved in database in Task Module, as shown in appendix C. It consists of four tables: *Products*, *SourceLocations*, *TargetLocations* and *TargetModelTargetObjects*. The details about the tables were introduced in section 7.2.

The video processing is the second step in the programming module. It was related to the instruction input by the same *TaskModelId*. As shown in the text instructions, an object was mentioned multiple times, but was only present in one instance in the demonstration video. This singular appearance of the object is systematically recorded in the *GroundedProducts* table. After identifying grounded products, a *ProductPose* table was created for each product. It stores the position and orientation in each timestamp. Only the orientation in the z-axis was considered, which was identified by the OBB detection. The hand trajectories were saved in the table *HandPosition*. All the trajectories were aligned with the *TimeId*. Besides the trajectories, the segmented skill label were also stored in the tables. The resulting tables in the database in appendix D demonstrate the effectiveness of the programming module.

Overall, the Programming Module and the Task Module effectively translated language instructions and visual demonstrations into a unified, structured task model. The clear and organized data storage ensures accurate task reproduction and easy adaptation to variations in task complexity or sequence.

8.5.2 Execution & task modules

This section focuses on validating the unified task model generated by the system through execution in a simulation environment using ROS Gazebo, as shown in fig. 8.9. ROS Gazebo is an open-source robotics simulator that facilitates accurate and efficient simulation of robots interacting within complex environments. For this validation, the simulation environment was configured to match the demonstration setup. This includes a table, all relevant objects specified in the task, and a UR5 robot model. Note that the approach proposed in this work is designed to be independent of the specific robot model, the UR5 model was chosen for its common usage in research and industry. The aim was to assess the system's ability to execute tasks derived from language and video inputs within this simulation.

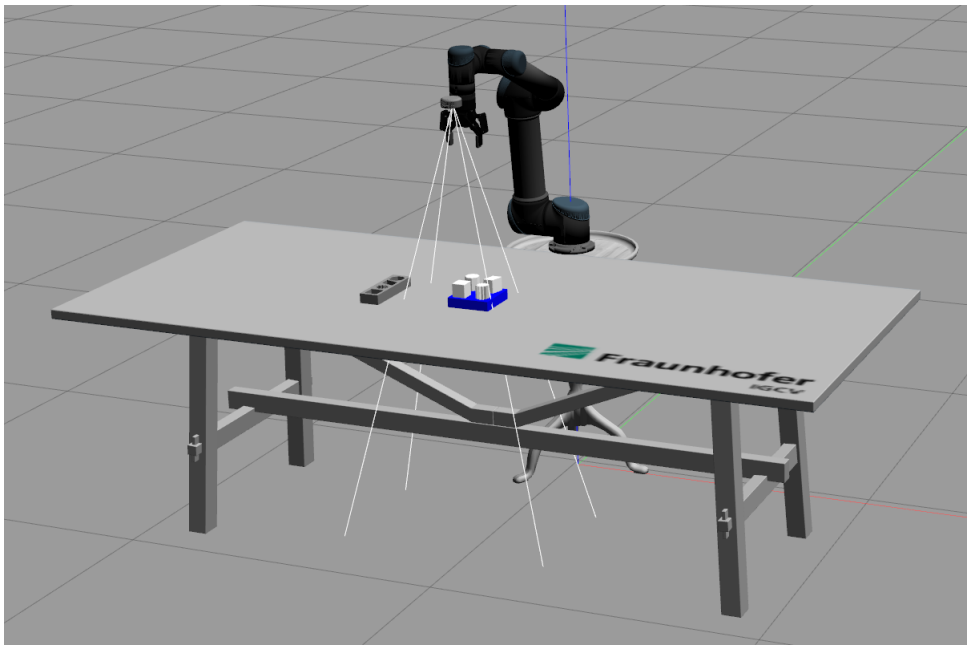


Fig. 8.9: Simulation environment. Reproduced from Lu et al. (2025), licensed under CC BY 4.0.

To validate the learned task model, the task plan is generated based on the sequences of target objects stored in the tables *TaskModelTargetObjects* C.4 and *Products* C.1. The *pick* and *place* actions for each target object are executed in the ROS environment sequentially. To perform the *pick* action, *reach* and *grasp* skills were required. *Reach* represents the trajectory from a random start

pose to the pre-grasp pose. The pre-grasp pose has a predefined distance 2 cm to grasp pose. The grasp pose was defined by the object pose in the camera frame in the simulation environment. The learned DMP for *reach* was modified with the new start and goal poses and sent to the ROS node. The trajectories were transformed to robot joint trajectories in the ROS node and executed. The release pose was also determined by the simulation environment. The DMP for *move* was then modified using the pose-grasp pose and the pre-release pose, which also has a predefined distance of 2 cm to the release position.

The system was tested with the five sets of instructions outlined in appendix A. Additionally, task plans containing fewer objects than the original input were executed to evaluate the system's adaptability. For example, the system successfully processed and executed a simplified instruction such as: "*Pick up the cuboid from the blue box and place it in the gray box*". In these cases, the system dynamically adjusted the task model to focus on the specific objects and actions mentioned.

These validation results confirm that the execution and task modules effectively bridge the gap between semantic instructions, demonstration videos, and robotic execution, ensuring accurate and adaptable task reproduction.

Chapter 9

Technical evaluation and potential analysis

This chapter provides a comprehensive analytical perspective on the developed semantic robot programming system. It complements the validation performed in chapter 8 by critically evaluating technical performance, achievement of research objectives, and practical applicability. A comparative potential analysis is also presented to illustrate the system's efficiency and usability advantages over traditional robot programming approaches.

9.1 Technical evaluation

This thesis addresses the challenge of enabling non-expert users to program robots by integrating semantic understanding from both natural language and visual demonstration. By combining these two modalities, the system can interpret what needs to be done and how to do it. This approach is particularly beneficial for flexible pick & place systems in manufacturing environments, where adaptability and ease of programming are essential to handle varying tasks, objects, and configurations.

The developed system follows a modular approach to process each modality separately and then combine the results into a unified task representation. This approach allows individual modules to be improved or replaced as needed without affecting the overall pipeline. Non-expert users can interact with the system only with language and visual demonstration. Once the task model is created, a

simulation is run that allows the user to visualize the demonstrated results to validate correctness and completeness.

In the Language Module, the system employs a multi-stage information extraction pipeline to process natural language input. Pre-processing methods such as sentence segmentation, coreference resolution, and tokenization are applied to prepare the input for semantic analysis. The core of the proposed hybrid approach combines a fine-tuned SpERT classifier for entity and relation classification with a syntactic dependency parser for structural interpretation. While the SpERT classifier identifies domain-independent temporal and spatial components, the dependency parser is responsible for extracting domain-specific elements, such as object references. If the underlying set of entities and relations remains consistent, this hybrid design supports reusability across tasks and significantly reduces the effort required to retrain or adapt the system for new domains. However, the module still faces some limitations, as discussed in section 8.2. The classification of certain relation types, particularly spatial relations, yields relatively low precision and recall, which can negatively affect the quality of temporal and spatial grounding. Furthermore, the dependency parser's performance is inherently tied to the accuracy of the SpERT classifier; errors in upstream predictions may propagate and result in incorrect or incomplete semantic structures. While the overall accuracy is acceptable for general information extraction tasks, it remains insufficient for deployment in a semantic programming system, where a higher standard is required. For practical deployment in safety-critical environments, near-perfect classification accuracy would be required to ensure reliable grounding and robust program generation.

The Environment Module processes RGB-D video input to extract spatial and temporal information essential for understanding human demonstrations. The implemented pipeline consists of several stages, including the separation of color and depth frames, hand and object detection, depth value refinement, trajectory generation, and motion segmentation. Hand positions are detected using a pre-trained Faster-RCNN model, while object detection is trained on a photorealistically generated synthetic dataset and fine-tuned using YOLOv8 for oriented bounding boxes. This use of transfer learning and synthetic data

significantly reduces the data collection and training effort. Depth values are obtained from the camera, and missing or invalid values are compensated using a depth estimation method. Based on these results, hand and object trajectories are generated and segmented using a GMM-based approach. As demonstrated in section 8.3, the system achieves sufficient accuracy for generating reliable hand and object trajectories, particularly when supported by language input to resolve ambiguities or occlusions. However, for applications requiring more precise pose information, the current accuracy remains a limitation. In such cases, further improvements in detection accuracy or the integration of a dedicated object pose estimation method would be necessary.

The Motion Module models segmented hand trajectories using DMPs to represent the skills *reach* and *move*. These primitives allow the system to generalize demonstrated motions to new start and goal positions while smoothing out noise and irregularities from the original trajectories. Evaluation results in section 8.4 demonstrate that the learned DMPs accurately reproduce the motion patterns and can be flexibly adjusted for new task configurations.

The Programming Module, Execution Module, and Task Module together validate the system's ability to transform language and visual demonstration inputs into an executable robot task. The Programming Module uses methods from the Language Module and Environment Module to build a unified task model, which is saved in the implemented relational database. The Execution Module uses this unified model to perform the task in a ROS Gazebo simulation environment with a UR5 robot. The system successfully demonstrated its capability to execute full and partial task plans across multiple scenarios, showing reusability and semantic alignment between input and robot behavior. However, the system lacks real-time task plan adaptation, meaning it cannot dynamically adjust actions in response to environmental changes during execution.

9.2 Assessment of research goal achievement

The developed system was designed based on the research objectives defined in section 3.6. This section evaluates the extent to which these objectives were achieved, based on the validation in chapter 8 and the evaluation in section 9.1.

The system demonstrated the following capabilities:

G1 Capability of adjusting its actions based on context during task execution

- Integration of language instruction and visual PbD: By integrating visual data from RGB-D videos with language instructions, the system adjusts its actions according to both sources of context. This approach is validated in section 8.5.
- Task plan modification: The system demonstrated flexibility in adjusting the task plan based on specific instructions, as validated in section 8.5.
- Trajectory adjustment with DMPs: The use of DMPs allowed the system to generalize and adjust its movements based on new starting and target positions during task execution, as validated in section 8.4.

G2 Reducing data collection effort

- Leveraging existing corpora for extracting information from natural language input: The existing corpora can be annotated in a domain-independent way to fine-tune a BERT model. In the subsequent step, a syntactic dependency parser identifies domain-specific information. This approach is validated and evaluated in section 8.2.
- Transfer learning with pre-trained models: By leveraging transfer learning with pre-trained models like YOLOv8 and Faster-RCNN, the system reduced the need for extensive data collection. The approach is validated and evaluated in section 8.3.
- Photorealistic image generation: To further reduce the effort in collecting real-world data, the system used photorealistic image generation to create a synthetic dataset for training the object detection model. The approach is validated and evaluated in section 8.3.
- Cross-domain generalization: The system's ability to generalize across domains, particularly from training data to unseen test data, showed that it didn't need huge datasets for each new task. The approach is validated and evaluated in section 8.3.

G3 Minimizing training effort

- Minimal data demonstration: The system extracts task models from a

single demonstration and a single language instruction. This approach is validated in section 8.5.

- Integration of language instruction and visual PbD: This approach allows for definition of objects of interest and improves object detection accuracy by leveraging both static and dynamic information provided by the language input. This approach is validated in section 8.5.

G4 Providing flexibility to improve and integrate new capabilities

- Modularity: The system was designed with a modular architecture, separating language processing, visual information extraction and motion generation. This modularity facilitates the integration of new capabilities or replacement of individual modules without affecting the entire system. This approach is validated in 8.5.

9.3 Potential analysis

The following potential analysis aims to illustrate the impact of the developed semantic programming system on programming effort. Similar to the work of Berg (2020), which addresses robot programming by non-experts, this thesis evaluates the engineering effort involved in implementing semantic robot programming under various practical scenarios. The evaluation quantifies effort in terms of time (measured in hours). The estimated effort was calculated considering the expertise of a software engineer with a background in robotics, computer vision, and machine learning. As a reference, the use case involving four geometric shapes presented in chapter 8 serves to estimate and compare the effort required across different scenarios. Specifically, the following three scenarios are considered:

- Scenario 1: Re-developing the programming system using the proposed methodological approach in section 4.1, assuming that the new task involves a different set of entities and relations. This represents the highest engineering effort, as most components need to be (re-)developed or re-trained.
- Scenario 2: Direct reuse of the developed system without modifications. This scenario assumes that a new task shares the same underlying entities

and relations as the original use case. Only minimal effort is needed for integration and validation.

- Scenario 3: The task has changed slightly, but it is still compatible with the existing task model and system, such as object positions or task plan during execution, as described in section 4.2. The system can be reused with minor configuration or input changes, resulting in minimal additional engineering effort.

Table 9.1 provides a detailed breakdown of the estimated engineering effort, measured in hours, for each scenario compared to a classical programming baseline. Scenario 1 requires the highest effort (19 hours), predominantly due to manual activities such as requirement analysis, task modeling, and implementing the instruction understanding pipeline. It should be noted that information extraction from visual demonstrations, including data generation (10 hours) and model fine-tuning (2 hours), is fully automated through a Blender-based pipeline and scripted routines. Thus, these activities do not require manual intervention and are listed separately in parentheses for completeness but excluded from the total engineering effort calculation.

In contrast, scenarios 2 and 3 significantly reduce engineering effort due to system reusability, requiring only 1 hour and 0.5 hours, respectively. The classical programming baseline refers to the teach programming approach outlined in section 2.5. Although classical robot programming consistently requires 1 hour across all scenarios, this effort does not reflect adaptability to variations. Each task variation, even minor adjustments in object positions or execution sequences, necessitates manual reimplementation and validation. Consequently, classical programming quickly becomes inefficient in dynamic contexts. The semantic robot programming approach offers significant efficiency gains through its inherent adaptability and reusability, which is particularly beneficial for flexible pick & place systems in manufacturing.

Table 9.1: Engineering effort by scenario (in hours) for semantic and classical robot programming

Phase	Scenario 1 [h]	Scenario 2 [h]	Scenario 3 [h]
Requirement analysis	**2**	**0**	**0**
Task modeling & representation	**2**	**0**	**0**
Input acquisition & knowledge base integration	**0.5**	**0.5**	**0**
Information extraction from instruction	**14**	**0**	**0**
Develop annotation scheme	1	0	0
Data annotation	10	0	0
Train SpERT classifier	2	0	0
Develop syntactic dependency parser logic	1	0	0
Information extraction from visual demonstration	**0**	**0**	**0**
Hand detection	0	0	0
Data generation for object detection	(10)	0	0
Fine-tuning model for object detection	(2)	0	0
Trajectory generation	(1)	0	0
Trajectory segmentation	0	0	0
Trajectory learning	0	0	0
Execution & validation	**0.5**	**0.5**	**0.5**
Total effort for semantic robot programming	**19**	**1**	**0.5**
Classical robot programming	**1**	**1**	**1**

9.4 Summary

This chapter evaluated the technical performance and practical applicability of the developed semantic robot programming system. The technical evaluation demonstrated that the system effectively integrates language and visual demonstration inputs to automatically generate executable robot task models. The modular architecture allowed for independent evaluation of the system components, with results showing that each module achieved sufficient accuracy to support task execution. However, some limitations remain, particularly regarding relation classification in the language module and precise pose estimation in the visual module.

The assessment of research goal achievement confirmed that all defined objectives were successfully addressed. The system demonstrated flexibility in contextual task adaptation, significantly reduced data collection and training effort, and provided a modular structure for future extensions.

Furthermore, a potential analysis illustrated the system's practical benefits, showing substantial reductions in engineering effort compared to classical robot programming, particularly when reusing or adapting the system for new tasks.

Overall, the evaluation results highlight the system's ability for improving the efficiency and flexibility of robot programming in dynamic environments.

Chapter 10

Conclusions and future research directions

10.1 Conclusions

The overall objective of this research was to develop a programming system that integrates semantic understanding, allowing robots not only to learn how to perform tasks, but also to understand the meaning and context of multi-step pick & place tasks in industrial scenarios. This was achieved by combining natural language instruction with visual PbD. Driven by the demands for efficiency and flexibility in industrial scenarios, this work presents a system capable of one-shot instruction and demonstration. The system reduces the need for extensive data collection and provides the flexibility to integrate new capabilities, ensuring that it can adapt to varying task requirements and evolving technological advances.

To build such a system, a methodological approach is first developed in section 4.1, which provides the guideline to support the development of semantic robot programming system. This framework structures the development process into four main phases. First, a requirements analysis (section 4.2) identifies the needs and constraints of industrial environments, such as minimizing programming effort, supporting multimodal input, and enabling task generalization. Based on these requirements, a formal task model is introduced (section 4.3) to semantically represent actions, objects, and spatial relationships in a structured format. The input acquisition phase (section 4.4) then brings together three

sources of information: natural language instructions, visual demonstrations recorded as RGB-D sequences, and a knowledge base of CAD models. In the final phase (section 4.5), the system processes the input to generate an executable task plan that is validated by execution in simulation.

To enable robots to understand natural language instructions, the availability of datasets plays an important role. These datasets allow NLP techniques to be developed, ensure that robots can perform the tasks based on the language input. Three corpora have been identified and analyzed in section 5.1. To extract the required information, a hybrid approach for cross-domain applications was developed, which consists of a fine-tuned deep learning model and a syntactic dependency parser. The deep learning model, based on the pre-trained BERT model, was designed to solve a NER and RE task. It is responsible for extracting the domain-independent information such as `action`, `spatial_indicator`. The domain specific products are extracted using the syntactic dependency parser. The hybrid approach allows the application of the trained models in different application scenarios compared to the related work.

To interpret how to perform the task, the motion of a robotic end-effector is mapped from a human hand by visual PbD. This process starts with the recognition of the hand from the RGB image. The evaluation in section 8.3.1 showed that the Bbox-based approach (Shan et al., 2020) had better performance. Therefore, the hand trajectories were generated using the pre-trained Faster RCNN model by Shan et al. (2020). This is followed by object detection, where the objects of interest identified from the instruction are used to build the object detection model. To generate training data, the BlenderProc framework (Denninger et al., 2023) was used to automatically generate the RGB images and their annotations. The YOLOv8 model (Jocher et al., 2023) was fine-tuned with the generated datasets to train the object detector. The evaluation in section 8.3.2 demonstrated the effectiveness and accuracy of training on the photorealistic RGB images and applying the trained model to real-world images. The object detection result was then improved by using language input, where the detection results of static objects were improved by the position with the highest confidence score. Object trajectories were generated from these refined results. The hand trajectories were then represented as skills in section 6.4.2 and

segmented using the approach proposed in section 6.4.3. Finally, DMPs were learned to represent the *reach* and *move* skills. The representation is adaptable to new start and target positions, which ensures the flexibility of the learned tasks. In summary, the approaches proposed in chapter 6 enable robots to learn tasks from one-shot demonstration.

After developing methods to process both language and visual inputs, a framework with six software modules were proposed in chapter 7 to integrate the both inputs and implement the semantic programming system. These modules include the Language Module, the Environment Module, the Motion Module, the Task Module, the Programming Module and the Execution Module. Each module plays a distinct role in facilitating the integration of semantics and PbD. This modular framework makes it easier to integrate improvements or add new functionalities without redeveloping the entire system, which is an advantage comparing to end-to-end approaches. The Language Module groups the methods developed in chapter 5, while the methods developed in chapter 6 were grouped in the Environment Module and the Motion Module. A relational database was developed and integrated in the Task Module to represent the unified task model. It consists of seven type of tables to store the information from instruction and visual demonstration. A task planner was proposed in the Execution Module for online execution. It interprets the unified task model provided by the Task Module, translating into specific commands for the robot. The task planner sequences the actions for target objects to ensure that the robot follows the correct order of operations. Meanwhile, it can adjust the DMP with the changed start and goal positions.

In summary, the implementation and validation in chapter 8 demonstrated the effectiveness of the whole programming system from information recording to robotic execution. Furthermore, the comparative effort analysis in chapter 9 confirmed the system's practical benefit by highlighting its efficiency and adaptability compared to traditional robot programming approaches, particularly in dynamic or variable task settings. However, the system has several limitations that impact its overall performance and flexibility. Possible extensions to address these limitations are discussed in the following section.

10.2 Future research directions

Based on the aforementioned conclusions, several areas require further exploration to enhance the system's capabilities and address existing limitations:

1. Real-time task plan adaptation: Future work should focus on developing real-time task adaptation mechanisms, enabling the robot to dynamically adjust its actions during execution in response to changes in the environment.
2. Improving object detection accuracy: To overcome the limitations of the current object detection module, future research could incorporate more sophisticated 3D object pose estimation techniques.
3. Enhancing spatial reasoning: Improving the system's spatial reasoning capabilities is another important area for future research. The integration of advanced NLP techniques and visual reasoning models would help the robot better interpret and execute tasks involving complex spatial relationships.
4. Extending the framework to bimanual manipulation: While the methodological approach is in principle applicable to bimanual tasks, the current approach to mapping motion from a human hand to a robot end-effector is limited to single-handed demonstrations. This simplification reduces ambiguity in associating hand trajectories with object movements and supports robust segmentation and skill learning. In contrast, bimanual tasks involve coordinated motion between both hands, which introduces additional challenges such as role differentiation (e.g., active vs. assisting hand), synchronization of actions, and increased complexity in task planning and execution. Depending on the nature of the task, two-handed demonstrations can be mapped to either a single robot arm or to a dual-arm system for parallel and coordinated execution. Future research could explore how motion segmentation and real-time adaptation can be extended to support such bimanual interactions.
5. Integration of RL learning techniques: This allows the system to refine its actions based on real-time feedback or additional demonstrations. This would improve the robot's ability to learn more complex tasks with minimal human intervention.

6. User experience: The focus of this thesis was primarily on technical performance. The system can be extended by considering user experience. Specifically, intuitive graphical or voice-based interfaces can be added to allow users to guide and modify robot behavior easily.

References

Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. Pddl| the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8:1–74, 2021.

Julian Christoph Sebastian Backhaus. *Adaptierbares aufgabenorientiertes Programmiersystem für Montagesysteme*. PhD thesis, Technische Universität München, 2016.

Emanuele Bastianelli, Giuseppe Castellucci, Danilo Croce, Luca Iocchi, Roberto Basili, and Daniele Nardi. Huric: a human robot interaction corpus. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4519–4526, 2014.

Emanuele Bastianelli, Danilo Croce, Andrea Vanzo, Roberto Basili, Daniele Nardi, et al. A discriminative approach to grounded spoken language understanding in interactive robotics. In *IJCAI*, pages 2747–2753, 2016.

Emanuele Bastianelli, Giuseppe Castellucci, Danilo Croce, Roberto Basili, and Daniele Nardi. Structured learning for spoken language understanding in human-robot interaction. *The International journal of robotics research*, 36 (5-7):660–683, 2017.

Julia Beibl, Jongsuk Lee, Dieter Krause, and Seung Ki Moon. Flexibility - grand challenge for product design and production: Review and status. *Procedia CIRP*, 119:91–96, 2023. ISSN 2212-8271. doi: 10.1016/j.procir.2023.05.003. The 33rd CIRP Design Conference.

J Berg, L Merhar, A Lottermoser, C Härdtlein, and M Henke. Roboter für den Mittelstand–Praxisleitfaden für Leichtbauroboter in der Produktion. *Fraunhofer-Einrichtung für Gießerei-, Composite-und Verarbeitungstechnik*

IGCV, verfügbar in: http://www. kompetenzzentrum-augsburg-digital. de/leitfaden_robotik, 2019.

Julia Berg and Shuang Lu. Review of interfaces for industrial human-robot interaction. *Current Robotics Reports*, 1:27–34, 2020.

Julia Katharina Berg. *System zur aufgabenorientierten Programmierung für die Mensch-Roboter-Kooperation*. PhD thesis, Technische Universität München, 2020.

Ludwig von Bertalanffy. General system theory: Foundations, development, applications. 1969.

Aude Billard, Sylvain Calinon, Rüdiger Dillmann, and Stefan Schaal. *Robot Programming by Demonstration*, pages 1371–1394. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-30301-5. doi: 10.1007/978-3-540-30301-5_60.

Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer New York, NY, 2006.

Yonatan Bisk, Daniel Marcu, and William Wong. Towards a dataset for human computer communication via grounded language acquisition. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Sowmya BJ, S Seema, S Rohith, et al. A visual computing unified application using deep learning and computer vision techniques. *International Journal of Interactive Mobile Technologies*, 18(1), 2024.

Lucienne TM Blessing and Amaresh Chakrabarti. *DRM: A design reseach methodology*. Springer, 2009.

Rainer Bokranz and Kurt Landau. Handbuch industrial engineering. *Produktivitätsmanagement mit MTM*, 2:2, 2012.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Sylvain Calinon, Florent Guenter, and Aude Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007.

Gonçalo Cândido and José Barata. A multiagent control system for shop floor assembly. In *International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, pages 293–302. Springer, 2007.

Isaac Cardoza, Juan P García-Vázquez, Arnoldo Díaz-Ramírez, and Verónica Quintero-Rosas. Convolutional neural networks hyperparameter tuning for classifying firearms on images. *Applied Artificial Intelligence*, 36(1):2058165, 2022.

Zhuo Chen, Peilin Liu, Fei Wen, Jun Wang, and Rendong Ying. Restoration of motion blur in time-of-flight depth image using data alignment. In *2020 International Conference on 3D Vision (3DV)*, pages 820–828, 2020. doi: 10.1109/3DV50981.2020.00092.

Toby HJ Collett, Bruce A MacDonald, and Brian P Gerkey. Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian conference on robotics and automation (ACRA 2005)*, volume 145. Citeseer Citeseer, 2005.

Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Scaling egocentric vision: The epic-kitchens dataset. In *European Conference on Computer Vision (ECCV)*, 2018.

Angel Daruna, Lakshmi Nair, Weiyu Liu, and Sonia Chernova. Towards robust one-shot task execution using knowledge graph embeddings. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11118–11124. IEEE, 2021.

Maximilian Denninger, Dominik Winkelbauer, Martin Sundermeyer, Wout Boerdijk, Markus Knauer, Klaus H. Strobl, Matthias Humt, and Rudolph Triebel. Blenderproc2: A procedural pipeline for photorealistic rendering. *Journal of Open Source Software*, 8(82):4901, 2023. doi: 10.21105/joss.04901.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

R. Dillmann and S. Knoop. Robotik ii: Programmierung von robotern, 2007. URL https://pasta.place/scraping/Informatik/Robotik_2_%5BHIS%5D/Skript/Skript_SS-15.pdf3. Lecture Script, Summer Semester 2015.

Guanwen Ding, Yubin Liu, Xizhe Zang, Xuehe Zhang, Gangfeng Liu, and Jie Zhao. A task-learning strategy for robotic assembly tasks from human demonstrations. *Sensors*, 20(19):5505, 2020.

Jian Ding, Nan Xue, Gui-Song Xia, Xiang Bai, Wen Yang, Michael Yang, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Object detection in aerial images: A large-scale benchmark and challenges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. doi: 10.1109/TPAMI.2021.3117983.

Zhipeng Dong, Zhihao Li, Yunhui Yan, Sylvain Calinon, and Fei Chen. Passive bimanual skills learning from demonstration with motion graph attention networks. *IEEE Robotics and Automation Letters*, 7(2):4917–4923, 2022.

Yuyang Du, Jian Wang, Zhanxi Wang, Fei Yu, and Chen Zheng. Robotic manufacturing systems: A survey on technologies to improve the cognitive level in hri. *Procedia CIRP*, 107:1497–1502, 2022. ISSN 2212-8271. doi: 10.1016/j.procir.2022.05.181. Leading manufacturing systems transformation – Proceedings of the 55th CIRP Conference on Manufacturing Systems 2022.

Kais Dukes. SemEval-2014 task 6: Supervised semantic parsing of robotic spatial commands. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 45–53, Dublin, Ireland, August 2014. Association for Computational Linguistics. doi: 10.3115/v1/S14-2006.

Markus Eberts and Adrian Ulges. Span-based joint entity and relation extraction with transformer pre-training. *arXiv preprint arXiv:1909.07755*, 2019.

Thomas Eiband, Johanna Liebl, Christoph Willibald, and Dongheui Lee. Online task segmentation by merging symbolic and data-driven skill recognition during kinesthetic teaching. *Robotics and Autonomous Systems*, 162:104367, 2023.

Íñigo Elguea-Aguinaco, Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Ibai Inziarte-Hidalgo, Simon Bøgh, and Nestor Arana-Arexolaleiba. A review on reinforcement learning for contact-rich robotic manipulation tasks. *Robotics and Computer-Integrated Manufacturing*, 81:102517, 2023.

Frank Emmert-Streib, Salissou Moutari, and Matthias Dehmer. *Elements of Data Science, Machine Learning, and Artificial Intelligence Using R*. Springer, 2023.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

Alexander Fabisch. *Learning and generalizing behaviors for robots from human demonstration*. PhD thesis, Universität Bremen, 2020.

Christiane Fellbaum. Wordnet: An electronic lexical resource, 1998.

Charles J Fillmore. Frame semantics and the nature of language. In *Annals of the New York Academy of Sciences: Conference on the origin and development of language and speech*, volume 280, pages 20–32. New York, 1976.

Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pages 357–368. PMLR, 2017.

Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, et al. Foundation models in robotics: Applications, challenges, and the future. *arXiv preprint arXiv:2312.07843*, 2023.

David F. Fouhey, Weicheng Kuo, Alexei A. Efros, and Jitendra Malik. From lifestyle vlogs to everyday interactions. In *CVPR*, 2018.

Jianfeng Gao, Zhi Tao, Noémie Jaquier, and Tamim Asfour. K-vil: Keypoints-based visual imitation learning. *IEEE Transactions on Robotics*, 2023.

Stefano Ghidoni. Performance evaluation of depth completion neural networks for various rgb-d camera technologies. 2023.

Rafael C Gonzalez. *Digital image processing*. Pearson education india, 2009.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Sagar Venkatesh Gubbi, Anirban Biswas, Raviteja Upadrashta, Vikram Srinivasan, Partha Pratim Talukdar, and Bharadwaj S. Amrutur. Spatial reasoning from natural language instructions for robot manipulation. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11196–11202, 2021. URL https://api.semanticscholar.org/CorpusID:229678027.

Veit Hammerstingl and Gunther Reinhart. Skills in assembly, 2018. URL https://mediatum.ub.tum.de/1428286. Version 1.1.

Matthias Haun. *Handbuch Robotik: Programmieren und Einsatz intelligenter Roboter*. Springer-Verlag, 2013.

Oliver Heimann and Jan Guhl. Industrial robot programming methods: A scoping review. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 696–703. IEEE, 2020.

Lisa Heuss, Clemens Gonnermann, and Gunther Reinhart. An extendable framework for intelligent and easily configurable skills-based industrial robot applications. *The International Journal of Advanced Manufacturing Technology*, 120(9):6269–6285, 2022.

Lars Hillebrand, Tobias Deußer, Tim Dilmaghani, Bernd Kliem, Rüdiger Loitz, Christian Bauckhage, and Rafet Sifa. Kpi-bert: A joint named entity recognition and relation extraction model for financial reports. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 606–612. IEEE, 2022.

Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer, 2012.

Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015.

Tomáš Hodaň, Vibhav Vineet, Ran Gal, Emanuel Shalev, Jon Hanzelka, Treb Connell, Pedro Urbina, Sudipta N Sinha, and Brian Guenter. Photorealistic image synthesis for object instance detection. In *2019 IEEE international conference on image processing (ICIP)*, pages 66–70. IEEE, 2019.

Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1162.

Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40 (4-5):698–721, 2021.

Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

Zubair Iqbal, Maria Pozzi, Domenico Prattichizzo, and Gionata Salvietti. Detachable robotic grippers for human-robot collaboration. *Frontiers in Robotics and AI*, 8:644532, 2021.

Glenn Jocher. YOLOv5 by Ultralytics, 2020. URL https://github.com/ultralytics/yolov5.

Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO. https://github.com/ultralytics/ultralytics, 2023.

Daniel Jurafsky and H.James Martin. Speech and language processing, 2024. URL https://web.stanford.edu/~jurafsky/slpdraft/ed3book.pdf.

Namasivayam Kalithasan, Himanshu Singh, Vishal Bindal, Arnav Tuli, Vishwajeet Agrawal, Rahul Jain, Parag Singla, and Rohan Paul. Learning neuro-symbolic programs for language guided robot manipulation. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7973–7980, 2023. URL https://api.semanticscholar.org/CorpusID:253180551.

Wolf Kienzle, Gökhan Bakır, Matthias Franz, and Bernhard Schölkopf. Efficient approximations for support vector machines in object detection. In *Pattern Recognition: 26th DAGM Symposium, Tübingen, Germany, August 30-September 1, 2004. Proceedings 26*, pages 54–61. Springer, 2004.

Doyeon Kim, Woonghyun Ga, Pyunghwan Ahn, Donggyu Joo, Sehwan Chun, and Junmo Kim. Global-local path networks for monocular depth estimation with vertical cutdepth. *CoRR*, abs/2201.07436, 2022. URL https://arxiv.org/abs/2201.07436.

Paul J. Koch, Marike K. van Amstel, Patrycja Debska, Moritz A. Thormann, Adrian J. Tetzlaff, Simon Bøgh, and Dimitrios Chrysostomou. A skill-based robot co-worker for industrial maintenance tasks. *Procedia Manufacturing*, 11: 83–90, 2017. ISSN 2351-9789. doi: https://doi.org/10.1016/j.promfg.2017.07.141. URL https://www.sciencedirect.com/science/article/pii/S2351978917303451. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy.

Stefan Alexander Krug. *Automatische Konfiguration vom Robotersysteme (Plug & Produce)*. PhD thesis, Technische Universität München, 2012.

Tom Kwiatkowksi, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, Cambridge, MA, October 2010. Association for Computational Linguistics. URL https://aclanthology.org/D10-1119.

Maria Kyrarini, Muhammad Abdul Haseeb, Danijela Ristić-Durrant, and Axel Gräser. Robot learning of industrial assembly task via human demonstrations. *Autonomous Robots*, 43(1):239–257, 2019.

John E Laird, Kevin Gluck, John Anderson, Kenneth D Forbus, Odest Chadwicke Jenkins, Christian Lebiere, Dario Salvucci, Matthias Scheutz, Andrea Thomaz, Greg Trafton, et al. Interactive task learning. *IEEE Intelligent Systems*, 32 (4):6–21, 2017.

David Lewis. *Convention: A philosophical study*. John Wiley & Sons, 2008.

Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, et al. Yolov6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*, 2022.

Shile Li and Dongheui Lee. Point-to-pose voting based hand pose estimation using residual permutation equivariant layer. In *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition, pages 11927–11936, 2019.

Yin Li, Zhefan Ye, and James M Rehg. Delving into egocentric actions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 287–295, 2015.

Opher Lieber, Or Sharir, Barak Lenz, and Yoav Shoham. Jurassic-1: Technical details and evaluation. *White Paper. AI21 Labs*, 1:9, 2021.

Jinying Lin, Zhen Ma, Randy Gomez, Keisuke Nakamura, Bo He, and Guangliang Li. A review on interactive reinforcement learning from human social feedback. *IEEE Access*, 8:120757–120765, 2020.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023a.

Ruicheng Liu, Rui Mao, Anh Tuan Luu, and Erik Cambria. A brief survey on recent advances in coreference resolution. *Artificial Intelligence Review*, 56 (12):14439–14481, 2023b.

Alexandre Lopes, Roberto Souza, and Helio Pedrini. A survey on rgb-d datasets. *Computer Vision and Image Understanding*, 222:103489, 2022.

Shuang Lu, Julia Berger, and Johannes Schilp. An integrated approach for hand motion segmentation and robot skills representation. In *MHI Colloquium*, pages 291–301. Springer, 2022a. doi: 10.1007/978-3-031-10071-0_24.

Shuang Lu, Julia Berger, and Johannes Schilp. System of robot learning from multi-modal demonstration and natural language instruction. *Procedia CIRP*, 107:914–919, 2022b. ISSN 2212-8271. doi: 10.1016/j.procir.2022.05.084.

Shuang Lu, Julia Berger, and Johannes Schilp. Extracting robotic task plan from natural language instruction using BERT and syntactic dependency parser. In *2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1794–1799, 2023. doi: 10.1109/RO-MAN57019.2023.10309598.

Shuang Lu, Christian Härdtlein, and Johannes Schilp. Visual imitation learning from one-shot demonstration for multi-step robot pick and place tasks. *Scientific Reports*, 2025. doi: 10.1038/s41598-025-30938-x.

Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.

Corey Lynch, Ayzaan Wahid, Jonathan Tompson, Tianli Ding, James Betker, Robert Baruch, Travis Armstrong, and Pete Florence. Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*, 2023.

Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.

Mónica Marrero, Julián Urbano, Sonia Sánchez-Cuadrado, Jorge Morato, and Juan Miguel Gómez-Berbís. Named entity recognition: fallacies, challenges and opportunities. *Computer Standards & Interfaces*, 35(5):482–489, 2013.

Eloise Matheson, Riccardo Minto, Emanuele GG Zampieri, Maurizio Faccio, and Giulio Rosati. Human–robot collaboration in manufacturing applications: A review. *Robotics*, 8(4):100, 2019.

Cynthia Matuszek, Michael Witbrock, John Cabral, and John DeOliveira. An introduction to the syntax and content of cyc. *UMBC Computer Science and Electrical Engineering Department Collection*, 2006.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *Experimental robotics: the 13th international symposium on experimental robotics*, pages 403–415. Springer, 2013.

Tony McEnery and Costas Gabrielatos. English corpus linguistics. *The handbook of English linguistics*, pages 33–71, 2006.

Christian Meyer. *Aufnahme und Nachbearbeitung von Bahnen bei der Programmierung durch Vormachen von Industrierobotern*. PhD thesis, 2011.

Jiaju Miao and Wei Zhu. Precision–recall curve (prc) classification trees. *Evolutionary intelligence*, 15(3):1545–1569, 2022.

Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

Christopher Müller. *World Robotics - Industrial Robots*. IFR Statistical Department, VDMA Services GmbH, Frankfurt am Main, Germany, 2023. ISBN 978-3-8163-0760-0.

Tushar Nagarajan, Christoph Feichtenhofer, and Kristen Grauman. Grounded human-object interaction hotspots from video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8688–8697, 2019.

Daniel Nyga. *Interpretation of Natural-language Robot Instructions: Probabilistic Knowledge Representation, Learning, and Reasoning*. PhD thesis, Universität Bremen, 2017.

Daniel Nyga, Subhro Roy, Rohan Paul, Daehyung Park, Mihai Pomarlan, Michael Beetz, and Nicholas Roy. Grounding robot plans from natural language instructions with incomplete world knowledge. In *Conference on robot learning*, pages 714–723. PMLR, 2018.

Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*, pages 3400–3407. IEEE, 2011.

Rafael Padilla, Sergio L Netto, and Eduardo AB Da Silva. A survey on performance metrics for object-detection algorithms. In *2020 international conference on systems, signals and image processing (IWSSIP)*, pages 237–242. IEEE, 2020.

Zengxi Pan, Joseph Polden, Nathan Larkin, Stephen Van Duin, and John Norrish. Recent progress on programming methods for industrial robots. *Robotics and Computer-Integrated Manufacturing*, 28(2):87–94, 2012.

Matteo Pantano, Thomas Eiband, and Dongheui Lee. Capability-based frameworks for industrial robot skills: a survey. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 2355–2362, 2022. doi: 10.1109/CASE49997.2022.9926648.

Matteo Pantano, Vladislav Klass, Qiaoyue Yang, Akhil Sathuluri, Daniel Regulin, Lucas Janisch, Markus Zimmermann, and Dongheui Lee. Simplifying robot grasping in manufacturing with a teaching approach based on a novel user grasp metric. *Procedia Computer Science*, 232:1961–1971, 2024.

Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *2009 IEEE International Conference on Robotics and Automation*, pages 763–768. IEEE, 2009.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Mikkel Rath Pedersen, Lazaros Nalpantidis, Rasmus Skovgaard Andersen, Casper Schou, Simon Bøgh, Volker Krüger, and Ole Madsen. Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing*, 37:282–291, 2016.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Affan Pervez and Dongheui Lee. Learning task-parameterized dynamic movement primitives using mixture of gmms. *Intelligent Service Robotics*, 11(1):61–78, 2018.

Jing Qi, Li Ma, Zhenchao Cui, and Yushu Yu. Computer vision-based hand gesture recognition for human-robot interaction: a review. *Complex & Intelligent Systems*, 10(1):1581–1606, 2024.

Junfei Qiu, Qihui Wu, Guoru Ding, Yuhua Xu, and Shuo Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016:1–16, 2016.

Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *Science China technological sciences*, 63(10):1872–1897, 2020a.

Zeju Qiu, Thomas Eiband, Shile Li, and Dongheui Lee. Hand pose-based task learning from visual observations with semantic skill extraction. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 596–603. IEEE, 2020b.

Mauro Queirós, João Lobato Pereira, Valdemar Leiras, José Meireles, Jaime Fonseca, and João Borges. Work cell for assembling small components in pcb. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, 2022. doi: 10.1109/ETFA52439.2022.9921654.

Oona Rainio, Jarmo Teuho, and Riku Klén. Evaluation metrics and statistical tests for machine learning. *Scientific Reports*, 14(1):6086, 2024.

RangeKing. Brief summary of yolov8 model structure - issue #189. https://github.com/ultralytics/ultralytics/issues/189, 2023.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015a.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015b.

Alexander Riedel, Nico Brehm, and Tobias Pfeifroth. Hand gesture recognition of methods-time measurement-1 motions in manual assembly tasks using graph convolutional networks. *Applied Artificial Intelligence*, 36(1):2014191, 2022.

Edoardo Romiti, Jörn Malzahn, Navvab Kashiri, Francesco Iacobelli, Marco Ruzzon, Arturo Laurenzi, Enrico Mingo Hoffman, Luca Muratore, Alessio Margan, Lorenzo Baccelliere, Stefano Cordasco, and Nikos Tsagarakis. Toward a plug-and-work reconfigurable cobot. *IEEE/ASME Transactions on Mechatronics*, 27(5):3219–3231, 2022. doi: 10.1109/TMECH.2021.3106043.

Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Ahmad Waleed Salehi, Shakir Khan, Gaurav Gupta, Bayan Ibrahimm Alabduallah, Abrar Almjally, Hadeel Alsolai, Tamanna Siddiqui, and Adel Mellit. A study of cnn and transfer learning in medical imaging: Advantages, challenges, future scope. *Sustainability*, 15(7):5930, 2023.

Emilio Sapena Masip, Lluís Padró, and Jorge Turmo Borras. Coreference resolution survey. 2008.

Sunita et al. Sarawagi. Information extraction. *Foundations and Trends® in Databases*, 1(3):261–377, 2008.

Matteo Saveriano. *Robotic Tasks Acquisition via Human Guidance: Representation, Learning and Execution.* PhD thesis, Technische Universität München, 2017.

Rosario Scalise, Shen Li, Henny Admoni, Stephanie Rosenthal, and Siddhartha S Srinivasa. Collaborative Manipulation Corpus. https://github.com/personalrobotics/collaborative_manipulation_corpus, 2018a.

Rosario Scalise, Shen Li, Henny Admoni, Stephanie Rosenthal, and Siddhartha S Srinivasa. Natural language instructions for human–robot collaborative manipulation. *The International Journal of Robotics Research*, 37(6):558–565, 2018b.

Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

Michaela Servi, Elisa Mussi, Andrea Profili, Rocco Furferi, Yary Volpe, Lapo Governi, and Francesco Buonamici. Metrological characterization and comparison of d415, d455, l515 realsense devices in the close range. *Sensors*, 21 (22):7770, 2021.

Dandan Shan, Jiaqi Geng, Michelle Shu, and David F Fouhey. Understanding human hands in contact at internet scale. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9869–9878, 2020.

Lin Shao, Toki Migimatsu, Qiang Zhang, Karen Yang, and Jeannette Bohg. Concept2robot: Learning manipulation concepts from instructions and human demonstrations. *The International Journal of Robotics Research*, 40(12-14): 1419–1434, 2021.

Lanbo She, Yu Cheng, Joyce Y Chai, Yunyi Jia, Shaohua Yang, and Ning Xi. Teaching robots new actions through natural language instructions. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 868–873. IEEE, 2014.

Bruno Siciliano and Oussama Khatib. Robotics and the handbook. In *Springer Handbook of Robotics*, pages 1–6. Springer, 2016.

Gunnar A. Sigurdsson, Abhinav Kumar Gupta, Cordelia Schmid, Ali Farhadi, and Alahari Karteek. Actor and observer: Joint modeling of first and third-person videos. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7396–7404, 2018. URL https://api.semanticscholar.org/CorpusID:4562167.

Vishwanath A Sindagi and Vishal M Patel. A survey of recent advances in cnn-based single image crowd counting and density estimation. *Pattern Recognition Letters*, 107:3–16, 2018.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models.

In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530, 2023. doi: 10.1109/ICRA48891.2023.10161317.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

Franz Steinmetz, Annika Wollschläger, and Roman Weitschat. RAZER—A HRI for Visual Task-Level Programming and Intuitive Skill Parameterization. *IEEE Robotics and Automation Letters*, 3(3):1362–1369, 2018. doi: 10.1109/LRA.2018.2798300.

Franz Steinmetz, Verena Nitsch, and Freek Stulp. Intuitive task-level programming by demonstration through semantic skill recognition. *IEEE Robotics and Automation Letters*, 4(4):3742–3749, 2019.

Maj Stenmark and Pierre Nugues. Natural language programming of industrial robots. In *IEEE ISR 2013*, pages 1–5, 2013. doi: 10.1109/ISR.2013.6695630.

R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998. doi: 10.1109/TNN.1998.712192.

Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

Moritz Tenorth, Daniel Nyga, and Michael Beetz. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *2010 ieee international conference on robotics and automation*, pages 1486–1491. IEEE, 2010.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Beatrice Van Eden and Benjamin Rosman. An overview of robot vision. In *2019 Southern African Universities Power Engineering Conference/Robotics and Mechatronics/Pattern Recognition Association of South Africa (SAUPEC/RobMech/PRASA)*, pages 98–104. IEEE, 2019.

Bram Vanherle, Steven Moonen, Frank Van Reeth, and Nick Michiels. Analysis of training object detection models with synthetic data. In *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*. BMVA Press, 2022. URL https://bmvc2022.mpi-inf.mpg.de/0833.pdf.

Andrea Vanzo, Danilo Croce, Emanuele Bastianelli, Roberto Basili, and Daniele Nardi. Grounded language interpretation of robotic commands through structured learning. *Artificial Intelligence*, 278, 2020a. doi: 10.1016/j.artint.2019.103181.

Andrea Vanzo, Danilo Croce, Emanuele Bastianelli, Roberto Basili, and Daniele Nardi. Human Robot Interaction Corpus. https://github.com/crux82/huric, 2020b.

Andrea Vanzo, Danilo Croce, Emanuele Bastianelli, Roberto Basili, and Daniele Nardi. Grounded language interpretation of robotic commands through structured learning. *Artificial Intelligence*, 278:103181, 2020c.

Sagar Gubbi Venkatesh, Raviteja Upadrashta, and Bharadwaj Amrutur. Translating natural language instructions to computer programs for robot manipulation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1919–1926. IEEE, 2021.

Pietro Vitiello, Kamil Dreczkowski, and Edward Johns. One-shot imitation learning: A pose estimation perspective. In *7th Annual Conference on Robot Learning*, 2023. URL https://openreview.net/forum?id=w5ONmpgnfG.

Weihua Wang, Xiaofei Li, Yanzhi Dong, Jun Xie, Di Guo, and Huaping Liu. Natural language instruction understanding for robotic manipulation: a multisensory perception approach. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9800–9806. IEEE, 2023.

Bowen Wen, Wenzhao Lian, Kostas Bekris, and Stefan Schaal. You only demonstrate once: Category-level manipulation from single visual demonstration. *Robotics: Science and Systems 2022*, 2022.

Gerhard Werling. *Produktorientierte automatische Planung von Prüfoperationen bei der robotergestützten Montage*. PhD thesis, 1993.

Kim Wölfel and Dominik Henrich. Grounding verbs for tool-dependent, sensor-based robot tasks. In *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 378–383, 2018. doi: 10.1109/ROMAN.2018.8525827.

Jianjia Xin, Lichun Wang, Kai Xu, Chao Yang, and Baocai Yin. Learning interaction regions and motion trajectories simultaneously from egocentric demonstration videos. *IEEE Robotics and Automation Letters*, 2023.

Gokul Yenduri, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, Rutvij H Jhaveri, Weizheng Wang, Athanasios V Vasilakos, Thippa Reddy Gadekallu, et al. Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *arXiv preprint arXiv:2305.10435*, 2023.

Francisco Yumbla, Juan Medrano, Edwin Valarezo, Hong-Ryul Jung, Tuan Luong, Sungwon Seo, Jinjae Shin, and Hyungpil Moon. Design of a conveyor belt manipulator for reposition of boxes in logistics centers. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 2225–2230, 2022. doi: 10.1109/CASE49997.2022.9926479.

Zhen Zeng, Zheming Zhou, Zhiqiang Sui, and Odest Chadwicke Jenkins. Semantic robot programming for goal-directed manipulation in cluttered scenes. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 7462–7469. IEEE, 2018.

Ze-Feng Zhan and Han-Pang Huang. Imitation system of humanoid robots and its applications. *IEEE Open Journal of Circuits and Systems*, 4:15–24, 2022.

Gaoyue Zhou, Liyiming Ke, Siddhartha Srinivasa, Abhinav Gupta, Aravind Rajeswaran, and Vikash Kumar. Real world offline reinforcement learning with realistic data source. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7176–7183, 2023. doi: 10.1109/ICRA48891.2023.10161474.

Yilun Zhou, Julie Shah, and Steven Schockaert. Learning household task knowledge from WikiHow descriptions. In Luis Espinosa-Anke, Thierry Declerck, Dagmar Gromann, Jose Camacho-Collados, and Mohammad Taher Pilehvar, editors, *Proceedings of the 5th Workshop on Semantic Deep Learning (SemDeep-5)*, pages 50–56, Macau, China, aug 2019. Association for Computational Linguistics. URL https://aclanthology.org/W19-5808.

Jordan Zlatev. Spatial Semantics. In *The Oxford Handbook of Cognitive Linguistics*, pages pp.318–350. Oxford University Press, 2007. ISBN 9780199738632. doi: 10.1093/oxfordhb/9780199738632.013.0013.

Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *Proceedings of the IEEE*, 2023.

Appendix A

Test instructions

1. *Begin by picking up the cuboid from the blue box and placing it in the gray box. Next, carefully lift the star and position it next to the cuboid in the gray box. Then, move the parallelogram, placing it beside the star in the gray box. Finally, pick up the octagon and place it adjacent to the parallelogram in the gray box.*
2. *Start by gently lifting the cuboid from the blue box and placing it precisely in the gray box. Then, grasp the star and carefully set it near the cuboid in the gray box. Afterwards, pick up the parallelogram and position it beside the star in the gray box. Conclude by placing the octagon next to the parallelogram in the gray box.*
3. *First, remove the cuboid from the blue box and place it in the gray box with precision. Following that, take the star and gently place it alongside the cuboid in the gray box. Continue by moving the parallelogram, placing it next to the star in the gray box. Finish by setting the octagon beside the parallelogram in the gray box.*
4. *Initially, pick up the cuboid from the blue box and carefully place it in the gray box. Next, take the star and delicately place it next to the cuboid in the gray box. Then, transfer the parallelogram, positioning it beside the star in the gray box. Finally, place the octagon adjacent to the parallelogram in the gray box.*
5. *Begin by lifting the cuboid from the blue box and placing it neatly in the gray box. Then, pick up the star and position it carefully next to the cuboid*

in the gray box. Next, move the parallelogram and place it beside the star in the gray box. Lastly, place the octagon next to the parallelogram, completing the sequence in the gray box.

Appendix B

Test instructions after coreference resolution

1. *Begin by picking up the cuboid from the blue box and placing cuboid in the gray box. Next, carefully lift the star and position the star next to the cuboid in the gray box. Then, move the parallelogram, placing the parallelogram beside the star in the gray box. Finally, pick up the octagon and place the octagon adjacent to the parallelogram in the gray box.*
2. *Start by gently lifting the cuboid from the blue box and placing the cuboid precisely in the gray box. Then, grasp the star and carefully set the star near the cuboid in the gray box. Afterwards, pick up the parallelogram and position the parallelogram beside the star in the gray box. Conclude by placing the octagon next to the parallelogram in the gray box.*
3. *First, remove the cuboid from the blue box and place the cuboid in the gray box with precision. Following that, take the star and gently place the star alongside the cuboid in the gray box. Continue by moving the parallelogram, placing the parallelogram next to the star in the gray box. Finish by setting the octagon beside the parallelogram in the gray box.*
4. *Initially, pick up the cuboid from the blue box and carefully place the cuboid in the gray box. Next, take the star and delicately place the star next to the cuboid in the gray box. Then, transfer the parallelogram, positioning the parallelogram beside the star in the gray box. Finally, place the octagon adjacent to the parallelogram in the gray box.*

5. *Begin by lifting the cuboid from the blue box and placing the cuboid neatly in the gray box. Then, pick up the star and position the star carefully next to the cuboid in the gray box. Next, move the parallelogram and place the parallelogram beside the star in the gray box. Lastly, place the octagon next to the parallelogram, completing the sequence in the gray box.*

Appendix C

Resulted database

Table C.1: Products

ObjectId	ObjectName	SourceLocation	TargetLocation	Color	TokenId	SentenceId	ClassId
0	cuboid	0			4	0	3
1	box			blue	8	0	1
2	cuboid		0		12	0	3
3	box			gray	17	0	0
4	star				5	1	5
5	star	2			9	1	5
6	box			gray	18	1	0
7	cuboid				14	1	3
8	parallelogram				4	2	2
9	parallelogram		1		8	2	2
10	star				11	2	5
11	octagon	4			4	3	4
12	box			gray	16	3	0
13	parallelogram				8	3	2

Table C.2: SourceLocations

LocationId	TrajectorObjectId	Descrpition
0	1	from
1	6	in
2	7	next to
3	12	in
4	13	next to

Table C.3: TargetLocations

LocationId	IndicatorObjectId	Descrpition
0	3	in
1	10	beside

Table C.4: TaskModelTargetObjects

TaskModelId	TargetObjectId
1	0
1	2
1	4
1	5
1	8
1	9
1	11

Table C.5: HandPositions

TimeId	TaskModelId	X	Y	Z
1	1	1.58	8.05	2.31
2	1	4.76	2.41	6.95
3	1	1.27	6.44	1.85
4	1	3.33	1.69	4.86
5	1	8.73	4.43	1.27
...	...	...	...	...

Table C.6: GroundedProducts

ClassId	ObjectName	SourceLocation	TargetLocation	Color
0	graybox			gray
1	bluebox			blue
2	parallelogram		1	
3	cuboid	0	0	
4	octagon	4		
5	star	2		

Table C.7: ProductPoseCuboid

TimeId	TaskModelId	X	Y	Z	Rz	Reach	Grasp	Move	Release
1	1	-0.092	-0.017	0.553	3				
2	1	-0.092	-0.017	0.554	3				
...	...	...	...	...	...	...	...	...	...
96	1	-0.396	-0.103	0.58	0.091	1			
97	1	-0.092	-0.017	0.55	3	1			
98	1	-0.092	-0.017	0.55	3	1			
...	...	...	...	...	...	...	...	...	...
122	1	-0.49	-0.128	0.72	0		1		
123	1	-0.49	-0.128	0.72	0			1	
124	1	-0.51	-0.134	0.75	0			1	
...	...	...	...	...	...	...	...	...	...
231	1	0.121	-0.008	0.55	1.9				1

Appendix D

Additional validation results

Table D.1: Validation results on the Semantic Evaluation (SemEval)-2014 Task 6

	Precision	Recall	F1-Score
`action`	99.31%	98.62%	98.96%
`trajector`	87.80%	94.74%	91.14%
`spatial indicator`	90.70%	92.86%	91.76%
`goal indicator`	94.25%	98.80%	96.47%
`color`	97.85%	99.13%	98.40%
`spatial relation`	82.22%	90.24%	86.05%
`move relation`	89.53%	91.67%	90.59%

Table D.2: Validation results on the Human Robot Interaction Corpus (HuRIC)

	Precision	Recall	F1-Score
`action`	98.08%	100%	99.03%
`trajector`	85.00%	80.95%	82.93%
`spatial indicator`	89.66%	100.00%	94.55%
`goal indicator`	100.00%	95.24%	97.56%
`spatial relation`	73.33%	84.62%	78.57%
`move relation`	95.00%	90.48%	92.68%

Table D.3: Validation results on the Collaborative Manipulation Corpus (HuRCM)

	Precision	Recall	F1-Score
`action`	99.14%	99.14%	99.14%
`trajector`	89.13%	95.35%	92.13%
`spatial indicator`	81.82%	93.75%	87.38%
`color`	93.68%	94.77%	94.22%
`spatial relation`	79.25%	87.50%	83.17%