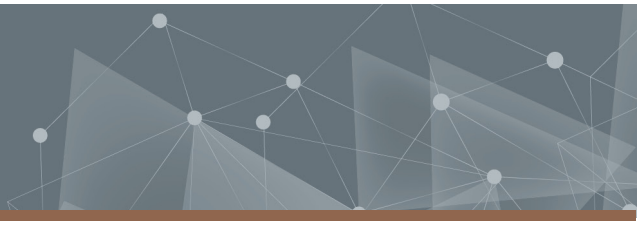
CHALMERS
UNIVERSITY OF TECHNOLOGY



Learning human actions on-demand based on graph theory

Master's thesis in Systems, control and mechatronics

JING ZHANG

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

Learning human actions on-demand based on graph theory

JING ZHANG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Systems and Control Division
Mechatronics
Chalmers University of Technology
Gothenburg, Sweden 2024

Learning human actions on-demand based on graph theory
JING ZHANG

Supervisor: Karinne Ramirez Amaro, Department of Electrical Engineering
Examiner: Karinne Ramirez Amaro, Department of Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Systems and Control Division
Mechatronics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

Learning human activities on-demand based on graph theory
JING ZHANG
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Collaborative robots (Cobots) are designed to work side-by-side with humans, sharing space and skills to achieve common goals. However, as human tasks become increasingly complex, Cobots must adapt to unfamiliar tasks. Traditional machine learning methods, while offering potential solutions, tend to focus on learning low-level physical activities. This lack of interpretability makes it difficult for humans and robots to understand and predict each other's behavior, hindering effective collaboration. In addition, machine learning methods rely heavily on human demonstrations, limiting the robot's ability to generalize to new scenarios.

In this work, each task (e.g., putting a spoon in the drawer) can be segmented into an interpretable activity sequence (e.g., Open, Grasp, Drop, etc.) based on human activities in real-time. We propose a method that can automatically construct different sequences for different tasks using a single human demonstration. Given that human demonstrations can vary and may include mistakes, this method reconstructs the most representative activity sequence from multiple demonstrations, thus robot could understand and predict human activities, and this method could extend to unseen scenarios.

We use a semantic reasoning method to transform low-level data into high-level concepts understandable by humans. Decision trees are trained to capture specific activity characteristics defined by predicates, e.g., when a human grasps a spoon, data about velocity and spatial relations are translated into predicates like `inHand`(spoon) as the input of decision tree, then this movement is inferred by out method as "Grasp", allowing real-time prediction and segmentation of human activities into activity sequences even for new experiments. Activities are parameterized using ontology knowledge, enabling robots to adapt to various objects and tasks. If an object, e.g., a bottle, is not inside the ontology, then we use Large Language Models(LLMs) to categorize the object according to the predefined ontology. For instance, both "spoon" and "bottle" belong to the category "Objects," making the activity "Grasp" identical in a high-level context. Since humans may perform the same task in different ways, de Bruijn graph and sequence assembly algorithms streamline these sequences by eliminating redundant activities and representing repetitive patterns, then reconstructing the most representative activity sequence by finding a path traversing each edge of the graph. This approach enhances the ability of Cobots to understand and predict human activities, thereby improving their collaboration with humans in dynamic environments.

Keywords: Human-Robot Collaboration, Ontology, Online Semantic Segmentation, De Bruijn Graph, LLM, Robot Learning

Acknowledgements

The completion of this thesis indicates that I will soon complete my master's degree and embark on a new journey related to robotics. Here, I would like to express my special thanks to Prof. Karinne. Almost a year ago, but earlier, she provided me with an opportunity and was willing to let me try to participate in robotics research even though I had no systematic robotics research experience. I am very grateful for her guidance and encouragement during the research process. Over the past year or so, she has provided me with a lot of valuable advice and help, not only in academics, but also in personal ability growth and life. It is also a pleasure to continue working with Prof. Karinne in my upcoming doctoral studies. Her trust and support in me make me full of confidence in my future research.

Jing Zhang, Gothenburg, May 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

HRC	Human-Robot Collaboration
Cobots	Collaborative Robots
ML	Machine Learning
LSTM	Long Short Term Memory
LLMs	Large Language Models
ROS	Robot Operating System
KG	Knowledge Graph
RDF	Resource Description Framework
RDFS	RDF Schema
OWL	Web Ontology Language
CCP	Cost-complexity Pruning
REP	Reduced-error Pruning
VR	Virtual Reality
DNA	Deoxyribonucleic Acid
SMOTE	Synthetic Minority Over-sampling Technique
SMOTETomek	SMOTE and Tomek Links

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Parameters

x	Predicate parameter
y	Predicate parameter
E	Entropy
D	Dataset
IG	Information gain
C	Cost in Cost-complexity pruning
k	Length of the k-mer in DNA sequencing
S	Sequence
P	Substrings
l	A value by which the length of k-mer is reduced to form $(k-l)$-mer
G	De Bruijn graph
G'	Aggregated Maximal Weighted Subgraph

Variables

o_t	Visual observation
o_g	Visual goal
i	Class in decision tree
p_i	The probability of class i
a	Attribute in decision tree
V	Attribute number
v	v-th attribute

n	Total class number
α	Parameter that describe complexity
T	Number of leaf nodes

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

Human-Robot Collaboration (HRC) is defined by the interaction between humans and robots within a shared workspace [2] as Figure 1.1 shows, collaborative robots (Cobots) are supposed to work with humans while sharing information and skills to achieve common goals. This concept of collaborative work requires that Cobots not only be able to coexist harmoniously with humans in physical space but also that humans and Cobots need to understand each other. The first requirement means Cobots need to recognize human actions, although traditional Machine Learning based methods offer potential solutions, they typically focus on learning low-level physical actions, as Figure 1.2 (b) depicts, robots are good at dealing with sensor data (e.g., here the data represents the position and orientation of a plate) while humans prefer high-level abstract concepts in communication, such as understanding the intention behind knowing a plate's position, like reaching to fetch it. The lack of interpretability [3] in these sensor data interpretations poses significant barriers when Cobots and humans attempt to achieve effective alignment and seamless collaboration, potentially leading to untrustworthy and even dangerous situations. For example, imagine a robot wants to help wash a plate, then goes directly to a human without any human-understandable explanation, it can cause confusion or even fear. However, suppose robots could interpret data in a manner that is comprehensible to humans, in that case, it translates data related to the plate to the activity it intends to achieve, such as reaching the plate, and notifying humans of the activity in an understandable way, as depicted in (c) of Figure 1.2, this would enable humans to easily understand the robot and decide the next movement (e.g., approaching the robot or standing and waiting), thereby enhancing collaboration.

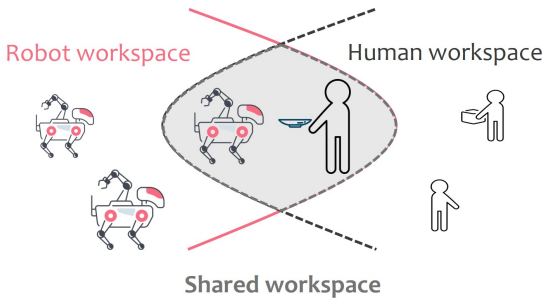


Figure 1.1: Human-robot collaboration in a shared workspace.

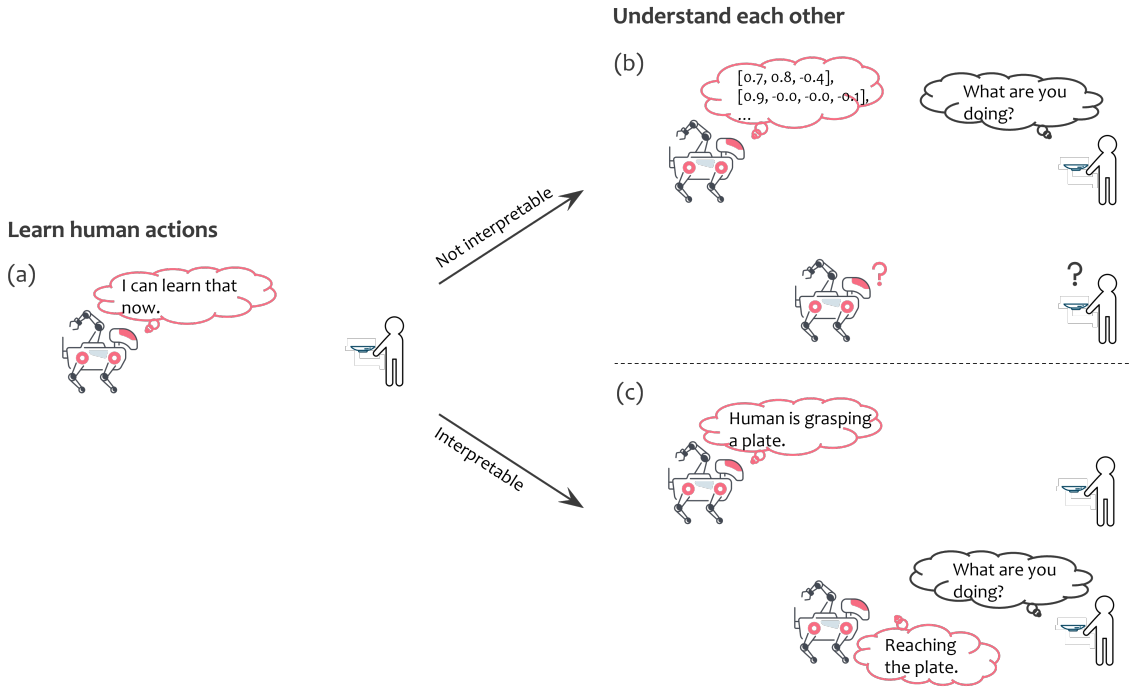


Figure 1.2: Different situations when collaboration, the data in (b) represents the position and orientation of the plate.

1.2 Problem statement

Imagine a situation when two people finished breakfast. Then, both persons need to collaborate to clean the table. In this work we would like to have a robot helping with this task. For instance, one person grasps a plate, robot then recognizes this movement and predicts the next action should be putting it in the dishwasher. This collaborative effort allows humans to focus on other tasks. Such task planning can be learned from humans. Task graph [4] is a commonly used method to model semantic activity and activity transition as Figure 1.3 showed, as long as robot knows the holistic plan to perform such task, in this example, the activity sequence is `Idle` → `Grasp` → `Put` → `Idle`. At the beginning of this example, online segmentation allows the robot to recognize that a human is grasping a plate. Following this recognition, the robot can predict subsequent activities based on the learned activity sequence, here the next action is to put it in the dishwasher. This proactive behavior helps achieve the cleaning goal more efficiently. Since these activities are semantically defined, robot can communicate with humans about its intention in an interpretable way, otherwise it would be hard to understand robot behavior as discussed in Section 1.1. Task graphs not only provide a clear representation of the activities but also highlight potential interaction points, identifying moments where HRC is necessary and beneficial, thereby enabling better collaboration.

Human demonstrations are commonly used for teaching robots to execute tasks. However, these demonstrations are not always the most efficient or accurate repre-

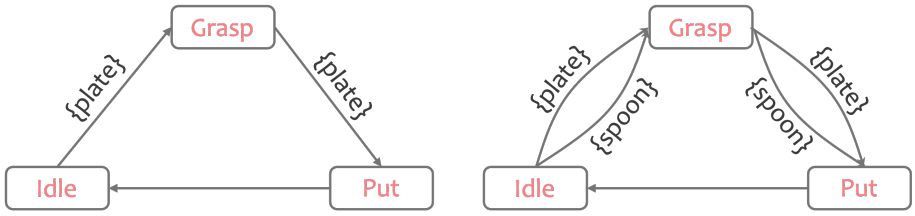


Figure 1.3: Task graphs for cleaning kitchen table. The right fig. shows while the specific object can be different, the overall activity is the same at a high-level concept.

sentations of a task. Errors can occur during demonstrations; for example, a human might accidentally drop a plate, which is an action that the robot should not learn. Additionally, there is inherent variability in how different individuals perform specific tasks. Complex tasks may contain repetitive patterns, which further complicate task graph structure like Figure 1.4. Cycles bring many challenges, including 1) ambiguity in task sequencing: a cycle between these actions might make it unclear whether the robot should repeat the sequence, how many times to repeat it, or under what conditions to move on to the next action. 2) increased complexity in task representation: representing tasks with cycles requires more sophisticated algorithms for parsing and executing actions because the robot must be able to identify and manage the repetition. 3) A cycle in task graph may indicate a repetitive task or a need to retry due to a failure.

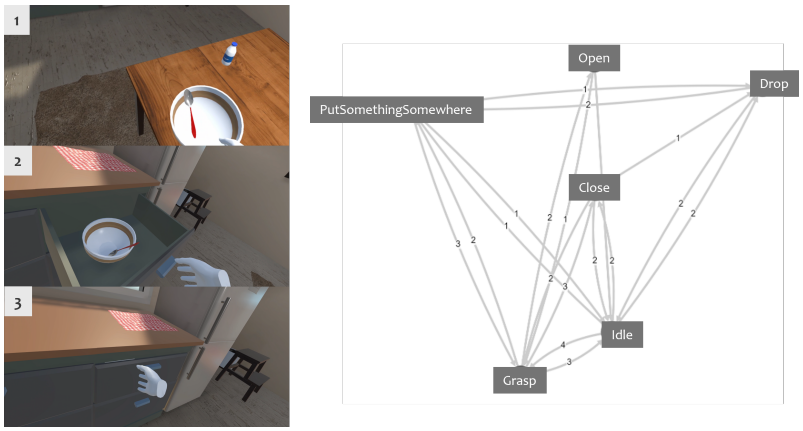


Figure 1.4: Human activities and corresponding task graph for storing two objects in a drawer, weight in the edge represents the transition frequency from one activity to another.

This thesis aims to bridge the gap between low-level sensor data processing by robots and high-level abstract communication by humans. It seeks to enable robots to segment human actions online, translate these actions into high-level concepts easily understood by humans, and use these concepts to build a graph structure to learn to schedule tasks effectively. The main problems can be summarized as:

1. Online human actions recognition and segmentation from demonstrations.
2. Reconstructing the most representative activity sequence for performing a task from the task graph and varied human demonstrations.

1.3 Related works

1.3.1 Segmentation and Recognition of Human Activities

Activity recognition in HRC aims to enable robots to understand and infer human actions. Some work has investigated recognition based on machine learning: Wang et al. [5] proposed a teaching-learning-prediction (TLP) model, this model enables the robot to learn and predict human hand-over intentions from natural multimodal human demonstrations, but it relies on wearable sensors and is limited to hand-over tasks; Yan et al. [6] proposed a multi-layer stacked deep LSTM model, which improves the recognition accuracy of human intentions by combining the advantages of single-layer LSTM and deep network structure, but it cannot achieve real-time. Chen et al. [7] introduce a stiffness estimation and intention detection method using surface electromyography signals and convolutional neural networks, validated on four wrist motion operators (up, down, left, right). However, the approach still needs to incorporate more diverse and complex operators not only restrict to the wrist information to enhance flexibility and performance. Orsag et al. [8] used visual sensing patterns decomposed into multiple levels, joint coordinates are captured to form a human skeleton, and human activities are recognized through an LSTM network. However, some operations will be misclassified due to similar motion characteristics, and the LSTM network limits the real-time performance of the system.

Other studies are mainly based on semantic representation. Ramirez-Amaro et al. [9] proposed a method to extract semantic rules of daily human activities. They extracted low-level information from sensor data and then inferred high-level information by reasoning about human expected behavior. However, their semantic rules could not describe the spatial relationship between objects, thus limiting their generalization ability. Vigné et al. [10] use dynamically created state machines based on semantic facts derived from environmental observations, activities are represented as sequences of geometric changes in the environment. The proposed system can recognize activities performed by robots and humans in real-time scenarios. However, it can only recognize a limited set of actions, specifically picking and placing, and is unable to recognize more complex behaviors. [11] proposed an approach that uses a sub-symbolic data recorder to record the environment and human gaze frame by frame, and uses predefined key activity frames (e.g., transport, slide) to detect and segment tasks. However, these predefined key frames limit the flexibility of the system, making it difficult to handle activities that do not fully conform to the predefined grammatical structure, and also it mainly focuses on simple tasks like pick-and-place.

1.3.2 Task Graph Representations

Task graph is traditionally used to model states, activities, and state transitions for procedural tasks [4]. In this context, the procedure planning problem [12] is defined as follows: given a current visual observation o_t and a visual goal o_g that indicates the desired final configuration, the model should plan a semantic high-level action sequence to bring the state from o_t to o_g. Mao et al. [4] focuses on actions and their temporal dependencies, providing a more compact and efficient representation and improving action prediction accuracy, but the performance is affected by initial localization errors in task tracking, which makes it hard to predict the next action and generate the task plan. These are problems which will be explored in this project. Existing literature directly encodes video information into state nodes and manually labels semantic activities [4, 12, 13]. However, these methods face challenges in transferring learned activities to unseen scenarios. Zhang et al. [14] used the RL model to generate the best task sequence in the HRC system and infer the correct assembly operation, but were unable to handle human repeated operations and the task allocation was not interpretable. These are problems which will be explored in this project.

1.4 Goals of the thesis

This thesis aims to enable robots to segment human actions online, translate these actions into high-level concepts easily understood by humans, and learn to schedule tasks effectively, these can be summarized as the following parts:

1. Use predicates and semantic reasoning to interpret low-level data into high-level concepts for human understanding.
2. Use supervised learning to get the core segmentation tool - decision tree.
3. Perform online segmentation for human actions, such segmentation could be generalized for unseen scenarios.
4. Collect segmented activity sequences, eliminate activity variations and noise as discussed in Section 1.2.
5. Reconstruct activity sequence that could represent specific task execution order most.

1.5 Boundaries of the thesis

1. All experiments were conducted by using Virtual Reality together with Unity. The choice of VR allows an ideal situation that easily captures all necessary information without having to deal with challenges related to perception, mapping and etc.
2. To simplify the problem, experiments are performed only with one hand.
3. The possibility exists, although very low (nearly 5%), that an LLM may provide an unexpected classification response.

4. Current method only contains the high-level task planning part, without any low-level execution specificity. For example, while the concept of "Grasp" can be applied to both a spoon and a cup, the actual grasping strategies for these objects are different as a cup can be broken.

1.6 Thesis structure

Chapter 1 introduces the research background, problems and an overview of the proposed method.

Chapter 2 covers the theory and concepts needed to understand this work, including knowledge representation, task graph, decision tree, sequence assembly and relevant graph theory.

Chapter 3 Introduces the proposed methods and gives more details about the whole implementation process of this work.

Chapter 4 shows the experiment setting, experiments about this system's generalization ability and reconstruction results.

Chapter 5 includes a summary and brief discussion of this work.

2

Theory

This chapter covers essential concepts and theories in knowledge representation, decision tree, and sequence assembly. Section 2.1 discusses knowledge representation and ontology. Section 2.2 explains the structure of decision trees, classification criteria, and pruning techniques. Section 2.3, drawing inspiration from DNA sequencing, addresses sequence assembly and covers topics such as the de Bruijn graph.

2.1 Knowledge Representation

Knowledge representation [15] is a method of substituting real-world objects with symbols or models, allowing us to reason about the world without taking physical action. It involves a set of ontological commitments, which define the categories of things that exist and the relationships that can hold among them. There exist various methodologies for knowledge representation, among which the knowledge graph (KG) is the most prevalent for depicting real-world entities and relationships using a graph structure, where nodes symbolize real-world entities or atomic values (attributes), edges denote relations [16] and statement set. Generally, ontology is used as KG schemas.

2.1.1 Ontology

In computer science, ontology is a taxonomy used to formally model the structure of a system through entities and relationships between them [17], and mostly developed for question-answering and problem-solving applications [18]. Ontology is often developed using specific languages designed for semantic representation. The most commonly used languages include:

- **RDF** (Resource Description Framework) [19]: A standard model for data interchange on the web, RDF is used to represent information about resources.
- **RDFS** (RDF Schema) [19]: An extension of RDF, RDFS provides the relationships of taxonomical structures.
- **OWL** (Web Ontology Language) [20]: Designed for creating complex ontologies, OWL adds semantics and allows for detailed descriptions of classes, properties, and relationships between entities. OWL is also based on RDF.

Ontologies provide a structured framework for organizing information and enable systems to interpret and infer new knowledge, setting the foundation for semantic reasoning.

2.2 Decision Tree

Decision trees are a popular method in supervised learning for decision-making. They are used for both classification and regression tasks, and even multioutput tasks [21]. Decision trees can be visualized as a series of nested if-else statements, making them highly interpretable.

2.2.1 Structure

A decision tree is a commonly used non-parametric supervised machine learning algorithm used to build prediction models and classification models. It is based on a tree structure and divides the data set into different subsets by analyzing the relationship between data features and target variables to gradually build decision rules. Decision tree models are intuitively interpretable.

The structure of a decision tree is similar to an inverted tree, consisting of nodes and branches. The top of the tree is called the root node, and each node represents a feature attribute or judgment condition. An example is shown in Figure 2.1. The key components of a decision tree are:

- **Root Node**: The initial split.
- **Internal Nodes**: Internal Nodes perform tests on one or more attributes and split the dataset based on the outcome of the test.
- **Leaf Nodes** (Terminal Nodes): Nodes that provide the predicted value (for regression) or class label (for classification).
- **Branches** (Edges): Each branch corresponds to a possible outcome of the decision rule applied at the parent node, and connects the next node.

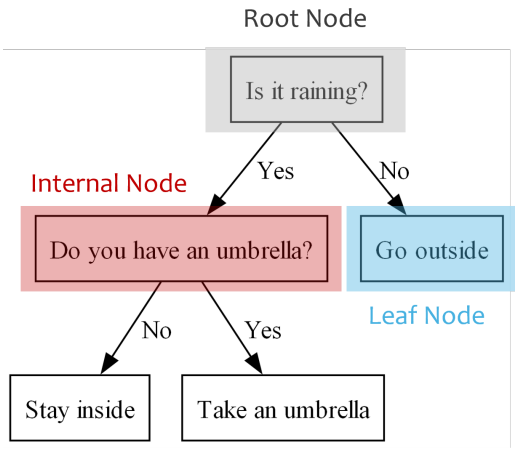


Figure 2.1: Decision tree example.

2.2.2 Classification Criteria

Starting from the root node, branches are carried out according to different feature attributes until the leaf nodes, which represent the final prediction results. This selection aims to achieve the most informative partitions based on specific criteria. The most common criteria for classification tasks are:

- **Entropy** [21]: The concept of entropy used to describe about uncertainty and information content. A set's entropy is zero when it contains instances of only one class. It is measured by the formula:

$$E(D) = -\sum_{i=1}^{n} p_i \log_2 (p_i) \tag{2.1}$$

  where D is the dataset, p_i is the probability of class i.
- **Information Gain**: A reduction of entropy is often called an information gain [21]. Give a dataset D, assume where an attribute a has V possible values $a^1, a^2, \ldots, a^V$, and if a is selected as the dividing attribute, then V branching nodes are generated, where the v-th branching node contains all the samples in D that take the value a^v on attribute a, denoted as D^v ,It is calculated as:

$$IG(D, a) = E(D) - E(D|a) = E(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} E(D^v) \tag{2.2}$$

  and $E(D)$ is the entropy defined in Eq. 2.1, $E(D|a)$ is also called as conditional entropy of D given attribute a.

  Information gain selects the attribute that maximizes the expected reduction in entropy from the root node to the leaves. Higher information gain indicates a more significant split.
- **Gini Impurity** [22]: Indicates the probability that a randomly selected sample in the set of samples will be missegregated. It is calculated as:

$$Gini(p_i) = 1 - \sum_{i=1}^{n} p_i^2 \tag{2.3}$$

  where p_i is the probability of class i. A lower Gini impurity indicates a purer node, the smaller the Gini index is, the smaller the probability that a randomly selected sample in the set will be misclassified. When all samples in the set are in one class, the Gini impurity is 0.

2.2.3 Pruning

Pruning reduces the size of a decision tree by removing parts of it that contribute less to classifying instances. Pruning helps prevent the model from capturing the noise in the data instead of the underlying distribution, i.e. overfitting. There are two main ways of pruning:

2.2.3.1 Pre-Pruning

Pre-pruning is to consider whether it can improve the performance of the decision tree during each division during the decision tree generation process. If the performance of the decision tree can be improved, it will be divided, otherwise, it will stop growing, i.e., stopping criteria. This method mainly uses heuristics to deal with noise during learning [23].

2.2.3.2 Post-Pruning

Post-pruning is to construct a complete decision tree first, and then check a group of nodes with the same parent node from the bottom up. Based on certain criteria, judge whether this group of nodes can be merged into one node. Post-pruning is to delete some subtrees and replace them with their leaf nodes. The biggest difference from pre-pruning is whether the decision tree grows completely. The generation of decision trees is to learn a local model, while post-pruning is to learn an overall model. The most common post-pruning methods are:

- **Cost-complexity Pruning** (CCP) [22]: The cost complexity criterion is defined as:

  $$C_\alpha = C(T) + \alpha|T|$$

  where $C(T)$ is the misclassification cost of the tree T, T is the number of leaf nodes, and α is used to adjust the balance between prediction error and complexity, that is, to adjust the fitting ability and generalization ability of the model. Subtrees are pruned if they do not reduce the overall cost.
- **Reduced-error Pruning** (REP) [24]: The available data is split into two sets of examples: a training set to form the learned decision tree and a separate validation set to evaluate the accuracy of the decision tree on subsequent data. It contains the following steps:
  1. Nodes are examined from the bottom up and replaced with leaf nodes.
  2. The validation set is used to evaluate the performance of the pruned decision tree.
  3. Pruning is performed if the error of the replaced tree on the validation set decreases or remains the same.
  4. This process is continued until the validation error cannot be reduced further.

Pruning improves the generalization of the decision tree, making it more effective on unseen data by reducing its complexity and preventing it from capturing noise in the training data.

2.3 Sequence assembly

In genomics and bioinformatics, sequence assembly refers to the process of reassembling short DNA sequence fragments (reads) into longer continuous sequences (contigs) [25]. There are similarities with our task which aims to construct an activity sequence from activity primitives, as described in Section 3.1.2. The introduction of de Bruijn graphs [26] provides a method for dealing with the problem of noise and

cycles in graph structures caused by human demonstrations, this has been discussed in Section 1.2.

2.3.1 De Bruijn graph

De Bruijn graph is a graphical representation method for efficiently processing sequence data and is often used to assemble genome sequences. It identifies overlaps between different short DNA sequences (reads), which can be used to reconstruct longer DNA sequences.

In Figure 2.2, (a) represents a genome sequence S as TAGCCAATTGT...TAGCC AATTGT. (b) In the de Bruijn graph derived from the sequence in (a), two branches are illustrated in distinct colors, red and blue. (c) The sequence showcases two unique streams of strings, with their differences underlined. (d) For the sequence in (c), two divergent de Bruijn graphs are depicted, showcasing minor variances between segments, illustrated by a bulge with dual branches (highlighted in red and blue). (e) The sequence features a consistently recurring $k-mer$, $k-mers$ are substrings of length k contained within a larger biological sequence, where the $k-mers$ consist of nucleotides [27], here is TAG.

Small differences between these genome sequences can form so-called bulge structures in De Bruijn graphs, i.e., small cycles or redundant paths in the graph. In Figure 2.2, (f) The De Bruijn graph for the sequence in (e) displays the recurring k-mer (TAG), forming a cycle within the graph.

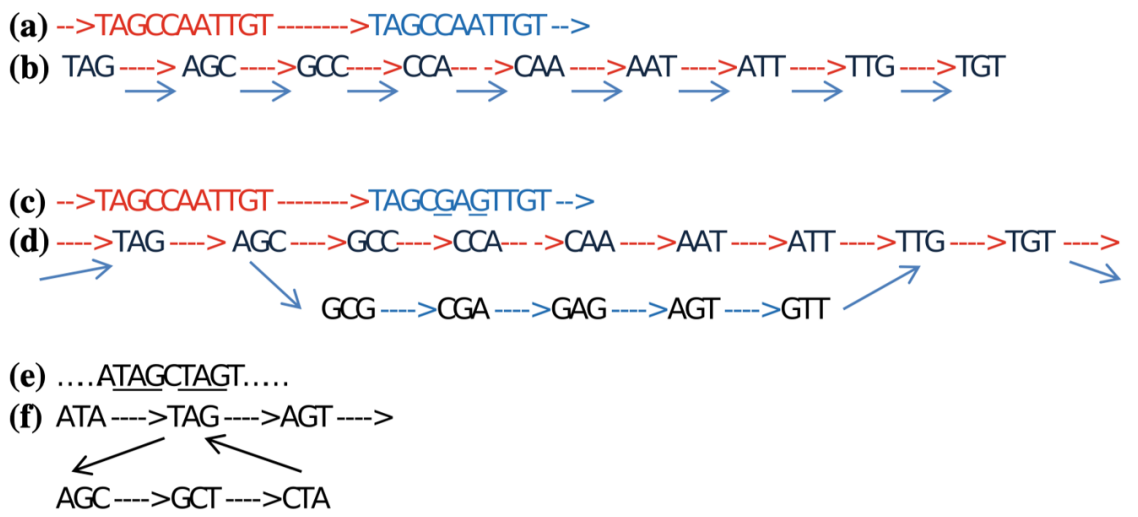


Figure 2.2: Figure containing De Bruijn graph generated from the sequence, bulge, and cycle structure represented in the De Bruijn graph.

3
Methods

In this chapter, detailed implementation steps are described. Section 3.1 contains the predefined ontology, completion method and semantic reasoning. Section 3.3 introduces the task graph. Section 3.3 detailed explains how to get the decision tree and relevant performance evaluation, pruning and tree selection. Section 3.3.3 includes online segmentation and sequences collection, and Section 3.4 proves the theory used to reconstruct activity sequence. Section 3.5 shows the whole process of the proposed system.

3.1 Knowledge Representation

3.1.1 Ontology Definition

Our system tracks three categories of knowledge within the framework of knowledge representation: object classifications and relationships, activity classifications, and a graph of possible transitions between activities known as the task graph.

As depicted in Fig. 3.1, a pre-defined ontology has been established. By querying the current entity's category and its siblings within this ontology, activities can be parameterized with various entities, enabling a more flexible task representation.

However, traditional ontologies rely on preset rules or limited semantic associations and have difficulty in expanding their scope [28]. In this regard, LLMs such as ChatGPT provide a novel and powerful solution. By pre-training on extensive text data, deep language structures and rich world knowledge are learned. These models cannot only generate natural language, but also perform complex reasoning and knowledge abstraction. Especially when dealing with entities beyond pre-defined ontology, LLMs can automatically analyze and assign unseen objects to a category based on context, thus supporting the automatic completeness of the ontology.

3.1.2 Semantic Reasoning

Building on the structured knowledge provided by ontology, semantic reasoning involves interpreting and deriving logical conclusions from this information. In this thesis, predicates are used to interpret human demonstrations into understandable human activities. Based on data collected from the environment—such as velocity,

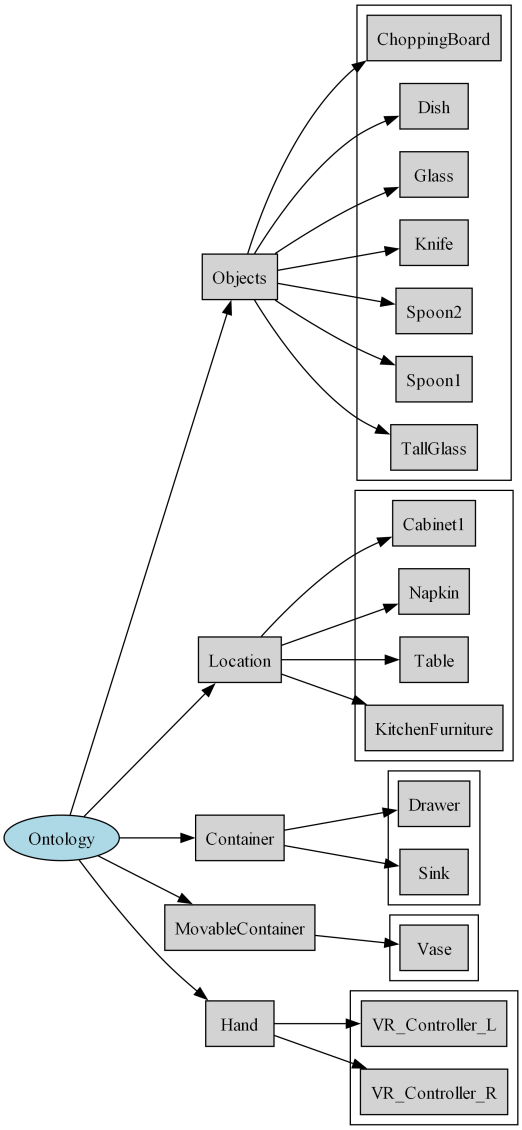


Figure 3.1: Pre-defined ontology representing the relationships between different categories and entities.

position, and the spatial relationships between the hand and objects, as well as between objects themselves—demonstrations can exhibit many specific features. These features are divided into two primary groups: Hand Motions and Object Properties, as detailed in Table 3.1. For instance, `inHand` indicates that an object is in hand, `onTop` signifies that an object is on top of another object, and `motionOpen` denotes that an object has been opened.

Activities can be defined and segmented by these features. For example, given the following predicate, x and y are predicate parameters:

- `onTop`(x, y): x is on top of y
- `inHand`(x): x is in hand
- `motionMoving`(x): x is moving

Table 3.1: Features used to represent environment state

Hand Motions	`inHand`
Object Properties	`onTop` `motionMoving` `motionOpen` `inside`

- `motionOpen`(x): x is opened
- `inside`(x, y): x is inside y

If in the current timestep, the set of features is:

$$\neg\exists y\, \texttt{onTop}(\text{spoon}, y) \wedge \texttt{inHand}(\text{spoon}) \wedge \texttt{motionMoving}(\text{spoon}) \tag{3.1}$$

the corresponding activity primitive could be labeled as: `PutSomethingSomewhere`. Based on such rules, we can define human actions in a semantic and interpretable way.

3.2 Task Graph

Task graph is traditionally used to model states, actions, and state transitions for procedural tasks [4]. A task graph is represented as a directed graph, where each node represents a specific activity, and edges represent the transition between activities [29]. One example is shown in Figure 1.3, where edge represents the transition between two activities, and weight indicates the transition frequency.

3.3 Decision Tree

3.3.1 Training a Decision Tree

The decision tree, shown in Figure 3.2, was trained using data collected from a single human demonstration by the Classification and Regression Tree (CART) [22] algorithm. CART uses the Gini impurity to decide the optimal split point. The process of selecting the optimal split point includes the following steps: first, traverse each feature and test all possible values of the feature, and calculate the Gini impurity of each value as the split point; second, for each candidate split point, calculate the impurity of the child node after the split; finally, select the feature that can minimize the impurity and its value as the optimal split point. The smaller the Gini impurity, the higher the purity of the dataset, indicating that the split point can better classify the dataset. According to these rules, starting from the root node, the dataset is divided into two subsets according to the optimal split point. This process is recursively performed for each subset until the stopping condition is met (such as reaching the maximum depth, the minimum number of node samples, or the gain after splitting being less than a certain threshold). In this work, the dataset is a single human demonstration collected for the task "put a spoon and vase together in the drawer", features are derived from predicates as discussed in Section 3.1.2, each possible value of a feature is encoded to a specific number. For example, if the

feature `inHand` has the value "Drawer," then the encoded number is 0, more details are shown in Table 3.2. The predictions (i.e., labels) of the decision tree are manually labeled activities based on a set of predicates, one example is shown in Eq. 3.1. Different paths may eventually point to the same leaf node due to local optimality of gini impurity and result in the same prediction results. For example, in our decision tree, in our decision tree, regardless of the path taken through the feature `inHand` or `motionOpen`, it is possible to ultimately reach the leaf node labeled "Grasp".

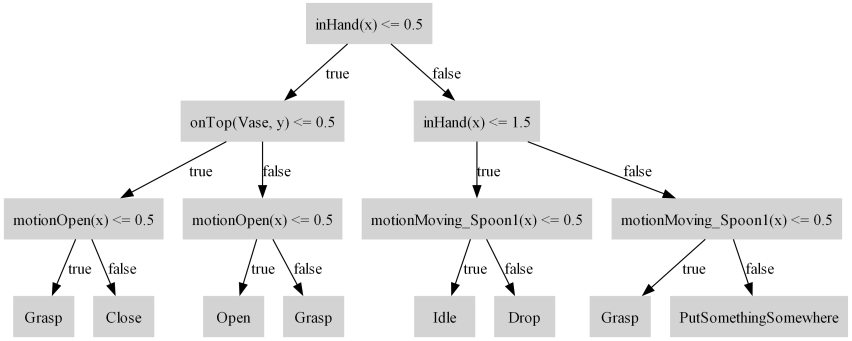


Figure 3.2: Decision tree classifier. x and y are potential values during prediction. Details are shown in Table 3.2.

Table 3.2: Encoded features during the training process. Predicates are defined in Section 3.1.2. For example, if the current feature is `inHand`(x) and the value of x is "Drawer," then the encoded value of the feature `inHand`(x) is 0.

Feature	Feature Encoding
`inHand`(x)	{"Drawer", 0}, {"Nothing", 1}, {"Spoon1", 2}, {"Vase", 3}
`onTop`$(Spoon1, y)$	{"Nothing", 0}, {"Table", 1}
`onTop`$(Vase, y)$	{"Nothing", 0}, {"Table", 1}
`motionOpen`(x)	{"Drawer", 0}, {"Nothing", 1}
`inside`$(Vase, y)$	{"Drawer", 0}, {"Nothing", 1}
`inside`$(Spoon1, y)$	{"Drawer", 0}, {"Nothing", 1}, {"Vase", 2}
`motionMoving_Vase`(x)	{"Nothing", 0}, {"Vase", 1}
`motionMoving_Spoon1`(x)	{"Nothing", 0}, {"Spoon1", 1}

3.3.2 Model Evaluation and Pruning

The classifier achieved an overall accuracy of 99%, as illustrated by the confusion matrix and the classification report (Figure 3.3). These results demonstrate the strong performance of our decision tree model. It is important to note that the activities primarily describe human activities, particularly hand movements. Activities such as `Drop`, `Open`, and `Close` are considered "instant activities" and consequently have fewer samples in the dataset compared to more frequent activities like `Idle` and `PutSomethingSomewhere`.

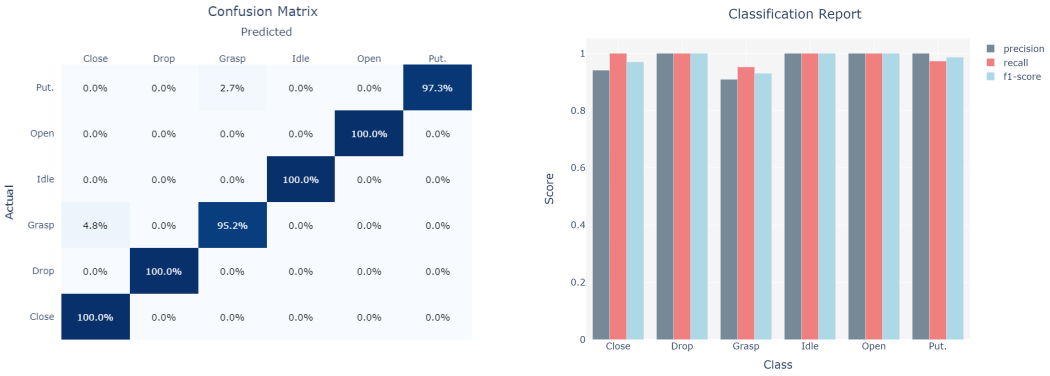


Figure 3.3: Confusion matrix and classification report for the decision tree which is shown in Figure 3.2. "Put." is the abbr. of `PutSomethingSomewhere`.

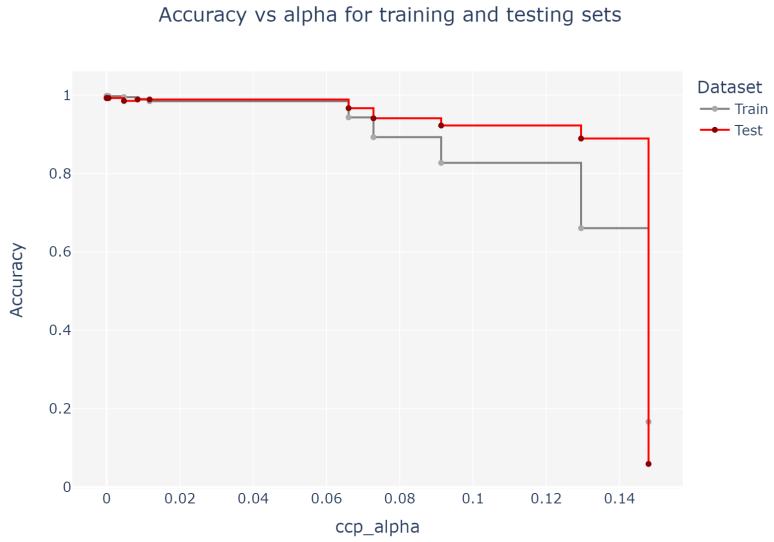
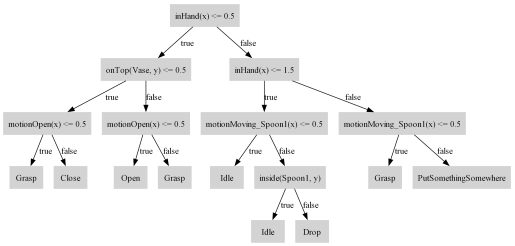


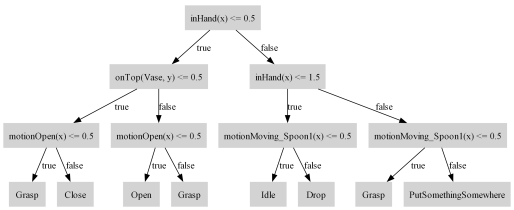
Figure 3.4: Model accuracy vs. complexity α after pruning.

To address the class imbalance, we employed the SMOTETomek technique [30], which combines both over-sampling and under-sampling methods to resample the dataset. Additionally, to prevent overfitting, cost-complexity pruning [24] is applied during training. Figure 3.4 illustrates the relationship between the accuracy and the complexity parameter α of cost-complexity pruning for both the training and testing sets. α acts as a penalty term and prevents model overfitting by balancing model complexity and its performance on the training set. A lower α value results in a more complex tree with more nodes and branches, as the penalty for complexity is smaller. Conversely, a higher α value simplifies the tree by increasing the penalty for additional nodes, resulting in fewer branches. Initially, as α increases, the decision tree becomes simpler, but the accuracy on the test dataset does not drop significantly, and this simplification helps reduce overfitting. However, once $\alpha \geq 0.7$, the accuracy on the test set drops because the model becomes too simple to capture the underlying patterns in the data, resulting in a drop in accuracy on

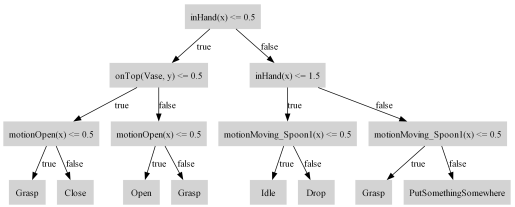
both the training and test sets. Figure 3.5 shows different trees after pruning by using different α. Comprehensively considering the complexity and classification performance of decision trees, any value of α between $0.02 \sim 0.06$ can be used as the final pruning parameter.



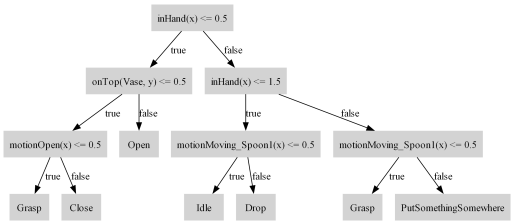
(a) $\alpha = 0.01$



(b) $\alpha = 0.03$



(c) $\alpha = 0.05$



(d) $\alpha = 0.07$

Figure 3.5: Different decision tree of different complexity α after pruning.

3.3.3 Implementing the decision tree to Segment and Recognize activities

The purpose of online segmentation is to dynamically segment human actions into interpretable and intuitive activity sequences in real-time as they are performed in the VR environment. Participants are asked to perform a task in the VR environment, such as putting a vase and a spoon together in a sink. Online segmentation involves continuously monitoring the participant's actions, segmenting them into distinct activities and building a task graph that represents activity transition simultaneously. One example is shown in Figure 3.6. Edge weights mean the transition frequency between two nodes in the task graph. Grey notation is the newly appeared transition. "Put." is the abbr. of "PutSomethingSomewhere". The loop between "Grasp" and "Put." is caused by the movement is not fast during transport, e.g., turning around in another direction, stopping for a while.

Since it's impossible to define every task by humans, parameterizing activities that allow for the prediction and generalization of different tasks is vital. An ontology is pre-defined and categorizes objects, locations, containers, and other relevant entities essential for describing the environment and activities shown in Figure 3.1. When a new entity that is not listed in the encoded value table (Table **??**) is encountered by the decision tree, and special handling is required. For instance, if the

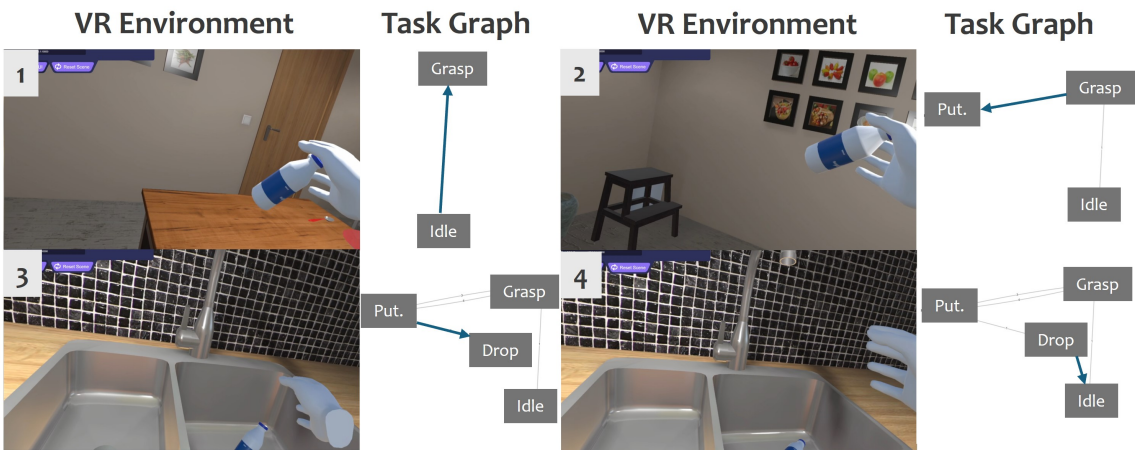


Figure 3.6: Online segmentation example for the task "put the bottle in the sink". The blue arrow highlights the newly appeared edge that is essential for the task execution.

current feature `inHand` has a value "Spoon2," which is not present in the feature encoding, by acquiring the category and siblings for the new entity from ontology, here according to Figure 3.1, "Spoon2" is under the category "Objects", and it has a sibling "Spoon1", which is in the feature encoding table, which means they are identical in high-level concept. Through this method, an adaptive representation of activities can transfer to unseen entities. However, it is also impractical to define every possible entity within an ontology. Thus, we complement our ontology with LLM, which serves as a resource for entities not included in the existing knowledge base. This hybrid approach ensures our method can generalize to unseen scenarios and new objects.

3.4 De Bruijn Graph and Sequence Reconstruction

3.4.1 Building a De Bruijn Graph

As discussed in Section 2.3.1, those small changes in DNA sequence are consistent with the challenges we face when automatically generating task graphs from VR demonstrations. The sequence modification algorithm can be summarized into the following two ways:

- **Initial condition**: Given a threshold C, if the number of links of a bulge is less than C, then it can be considered to be simplified. Here, bulge is formed by two substrings P_1 and P_2 in sequence S.
- **Sequence modification**: To remove the bulge, the algorithm modifies the sequence by replacing all occurrences of P_1 with P_2 in S. This means selecting one variant in the sequence and ignoring the other.

In this project, DNA sequencing is useful to eliminate small changes in human

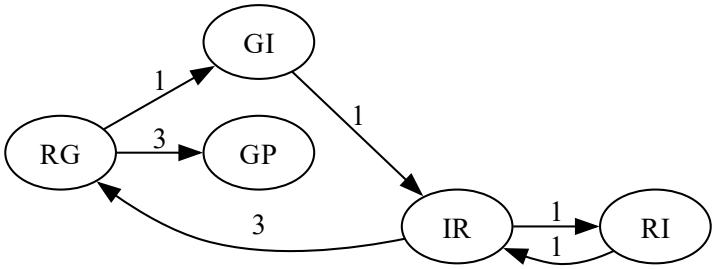


Figure 3.7: De Bruijn graph for the given activity sequence

demonstration sequences for one specific task. For example, if we want to grasp one object and put it to another place, three possible sequences extracted from human demonstration could be:

1. `Idle` → `Reach` → `Idle` → `Reach` → `Grasp` → `PutSomethingSomewhere`.
2. `Idle` → `Reach` → `Grasp` → `Idle` → `Reach` → `Grasp` → `PutSomethingSomewhere`.
3. `Idle` → `Reach` → `Grasp` → `PutSomethingSomewhere`.

Use the following abbreviation for each activity: `I` for `Idle`, `R` for `Reach`, `G` for `Grasp` and `P` for `PutSomethingSomewhere`. Use $3-mer$ to divide them into reads:

1. `IRI`, `RIR`, `IRG`, `RGP`.
2. `IRG`, `RGI`, `GIR`, `IRG`, `RGP`.
3. `IRG`, `RGP`.

Take each length-3 input string and split it into two overlapping substrings of length 2. Call these the left and right $2-mers$.

1. `IR`, `RI`, `IR`, `RG`, `GP`.
2. `IR`, `RG`, `GI`, `IR`, `RG`, `GP`.
3. `IR`, `RG`, `GP`.

In the de Bruijn graph, each node represents a $2-mer$, and each directed edge represents an overlap between two $2-mer$, from each left $2-mer$ to corresponding right $2-mer$ as shown in Figure 3.7. In this simple example, several conclusions can be easily derived:

1. Edge represents activity transition during execution.
2. Weight represents the frequency of transition from one activity to another.
3. A higher weight indicates a more necessary step for executing a task.

Thus the hypothesis is: If a path that traverses all necessary edges can be found, then this path is the most representative sequence to execute the task.

3.4.2 Sequence Reconstruction

As demonstrated in Section 2.3.1, the sequences are divided into reads to build de Bruijn graphs, where nodes represent $k-1$-mers and edges represent k-mers, indicating transitions between activities. De Bruijn graphs offer a more compact representation in the representation of repetitive patterns and cycles within activity

sequences. The goal of sequence reconstruction from the de Bruijn graph requires finding a sequence that covers the most representative edges, i.e., activity transitions. This problem is similar to the Seven Bridges of Königsberg Problem.

3.4.2.1 Seven Bridges of Königsberg Problem

The Seven Bridges of Königsberg problem is based on the layout of the city of Königsberg, which consists of two large islands connected to each other and to the mainland by seven bridges, the illustration is shown in Figure 3.8. The problem is how to walk across the seven bridges in the city in a single walk, crossing each bridge exactly once. By representing the land as nodes, the bridges as edges and formulating the problem as a question about whether there is an Eulerian circuit in the graph, Euler solved the Seven Bridges of Königsberg Problem [31].

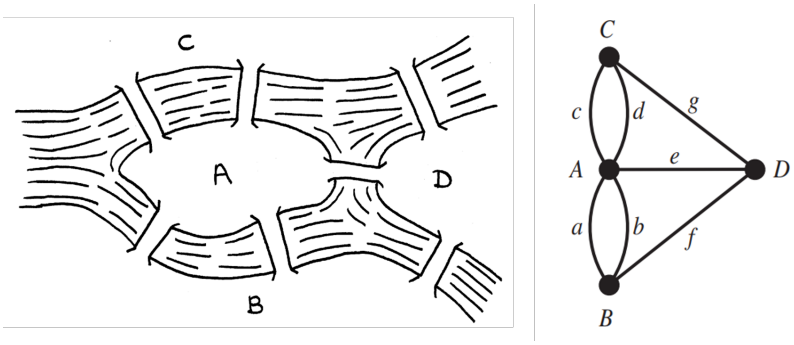


Figure 3.8: Seven Bridges of Königsberg Königsberg and its topology [1]

3.4.2.2 Eulerian Path

Eulerian path is also called the Eulerian trail:
Theorem 1. *for undirected graph [32]:*

- *A non-trivial connected graph has an Euler circuit iff. each vertex has even degree.*
- *A connected graph has an Euler trail from a vertex x to a vertex $y \neq x$ iff. x and y are the only vertices of odd degree.*

Since the de Bruijn graph is a directed graph, then given Theorem 2:
Theorem 2. *For the existence of an Eulerian path in a directed graph [33]:*

- *the underlying graph is connected, and*
- *for each vertex except two, the in-degree equals the out-degree; of the remaining two vertices, either both have in-degree equal to the out-degree, or else one has in-degree = out-degree +1*

and also:
Theorem 3. *De Bruijn graph has an Eulerian path [34].*

Thus an Eulerian path can be found from a de Bruijn graph, which means the reconstruction of activity sequence is possible. Moreover, a corollary especially relevant to our task is:

Corollary 4. *Let G' be a de Bruijn graph derived from the combination of multiple de Bruijn graphs, where the edge weights correspond to or greater than the number of sequences. If all sequences from the same task share the same start vertex and end vertex, then G' contains an Eulerian path.*

To prove this,

Proof. Consider a combined de Bruijn graph G'. The restriction on the edge weights implies that some internal edges may be removed. However, since all sequences from the same task share the same start vertex s and end vertex t, these vertices will not be removed. This ensures that the degree condition for the vertices remains intact. Specifically, we analyze the degrees of the vertices as follows:

- The start vertex s of the sequence has an out-degree that is one more than its in-degree.
- The end vertex t of the sequence has an in-degree that is one more than its out-degree.
- All other vertices have equal in-degrees and out-degrees, as they are constructed continuously from the sequences.

thus we proved that G' contains an Eulerian path according to Theorem 2 and 3. □

In the following part, such graph G' is named as *Aggregated Maximal Weighted Subgraph*.

Based on above analysis, we could conclude that an *Aggregated Maximal Weighted Subgraph* has such features:

- each edge represents an activity transition that appears in all of the sequences
- it covers every edge in the graph

These features ensure the reconstruction results in the most representative activity transition and sequence for a specific task. The most standard format of de Bruijn graph is built from de Bruijn sequence [35]: a de Bruijn sequence for a given alphabet is a cyclic sequence that contains every possible substring of a specified length exactly occurs once. However, in experiments, the sequence collected from human demonstration is not always a de Bruijn sequence. More details will be illustrated in the next section.

3.4.3 Example for Reconstructing Sequence

For example, given a task "put spoon and vase into drawer", the corresponding sequences are (`I`←`Idle`, `O`←`Open`, `G`←`Grasp`, `P`←`PutSomethingSomewhere`, `D`←`Drop`, `C`←`Close`):

1. IGOIGPGDIGPDIGC
2. IGOIGPGDIGPGPDIGCGC

the corresponding *Aggregated Maximal Weighted Subgraph* which built by 3-mer is showed in Figure 3.9.

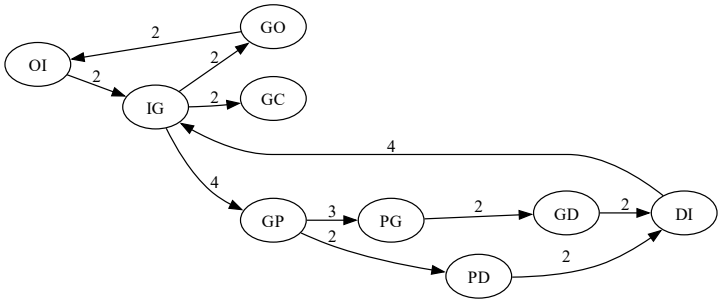


Figure 3.9: Aggregated Maximal Weighted Subgraph (3-mer) for the given task.

Node `GP` and `DI` violate the degree balance, showcasing they don't occur just once. Consequently, there is no Eulerian path, and further processing methods are required. Path 1: `GP` → `PG` → `GD` → `DI` and path 2: `GP` → `PD` → `DI` achieve the same transition essentially, such degeneracy can be eliminated by sequence modification (described in Section 2.3.1): replace one sequence with another. After replacing path 1 by 2, the new aggregated subgraph is shown in Figure 3.10:

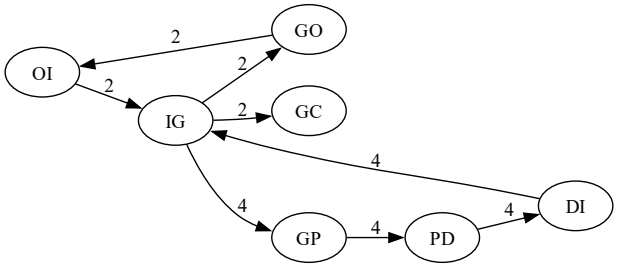


Figure 3.10: Aggregated Maximal Weighted Subgraph (3-mer) after sequence modification.

This change is reflected in the original sequence, indicating that some unnecessary activities have been removed:

1. IGOIGP~~G~~DIGPDIGC
2. IGOIGP~~G~~DIGP~~GP~~DIGCGC

Additionally, some weights are greater than the number of sequences means a repetitive pattern during activity transition, 3-mer is not enough length to capture such a pattern, increasing length will capture more transition context to solve the problem.

3.5 Integration

Figure 3.11 illustrates the entire process from online segmentation to sequence assembly. Using the example of putting a bottle into a sink, predicates describe hand motions and object properties, which serve as the input for the decision tree. In this scenario, the entity "bottle" is not within the scope of the encoded value. To address this, a LLM is employed to complete the ontology. Given a predefined ontology, the LLM is queried to determine the most suitable category for the "bottle." For instance, if the LLM identifies the category as "objects," then query ontology directly for its siblings. This process reveals that "spoon1" is a sibling of "bottle" and is also within the scope of the decision tree. This finding indicates that task "put spoon1 into sink" is the equivalent substitution of original task, other activities can be derived similarly.

The task initially exists in the idle interval for human thinking and movement. Assuming that LLMs cannot give bottle a category in the first query, since predicates are sent as long as they are triggered, the general physical calculation step is less than 20 milliseconds. Continuous and very short query intervals will ensure that LLMs give an answer, and then insert the classification result of the bottle into the corresponding category of the ontology.

Once we collected sequences with the same start vertex and end vertex, they could be combined and filtered to get the *Aggregated Maximal Weighted Subgraph*, by judging weighs and degree balance to determine if an Eulerian path can be found directly (see Corollary 4 and corresponding proof) or this subgraph still need modification and re-build to satisfy the Eulerian path requirements, the process is shown in Figure 3.11 "Sequence Assembly" part.

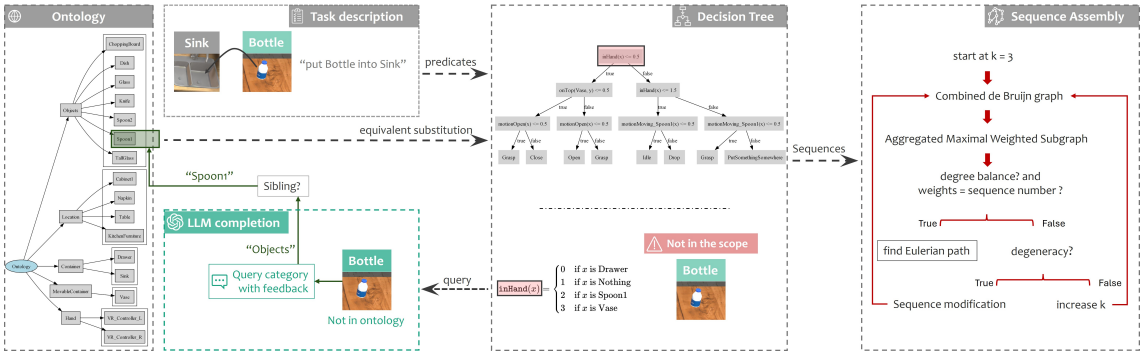


Figure 3.11: Online segmentation and sequence assembly integration.

4

Results

4.1 Experiment Setting

Our experiments are conducted in a VR kitchen environment developed using Unity, as illustrated in Fig. 4.1. This environment simulates a realistic kitchen setting, allowing for controlled and repeatable experiments. To ensure a comprehensive dataset that covers a wide range of tasks for this study, the initial dataset focuses on the tasks of storing a vase (referred to as "`Vase`" in the ontology shown in Figure. 3.1) and a spoon (referred to as "`Spoon`") into a drawer (referred to as "`Drawer`").

Figure 4.1: The VR kitchen environment in Unity.

Participants interact with the VR environment using VR controllers, and a set of features are used to describe hand motions and object properties as shown in Table 3.1. Based on these features, semantic activities are labeled to represent various activities. In this study, we utilize the following task primitives: `Idle`, `Open`, `Grasp`, `PutSomethingSomewhere`, `Drop`, and `Close`. The dataset is gathered as Section 2.1 demonstrated.

4.2 Online Segmentation in Unseen Scenarios

We conducted several tasks to test the generalization ability of our method. Since the original dataset was trained with {`Spoon`, `Vase`, `Drawer`}, we further defined three unseen tasks:

1. **Task 1**: Put "`Spoon2`" (in ontology, but not in the decision tree value scope) and "`Vase`" into "`Sink`" (new container).
2. **Task 2**: Put "`Bottle`" (not in the current ontology) into "`Sink`".
3. **Task 3**: Put "`Spoon1`" and "`Vase`" on "`Napkin`" (new location), put them into "`Drawer`", and put "`Bottle`" into "`Drawer`" (change the execution order and introduce repeat tasks with more complexity).

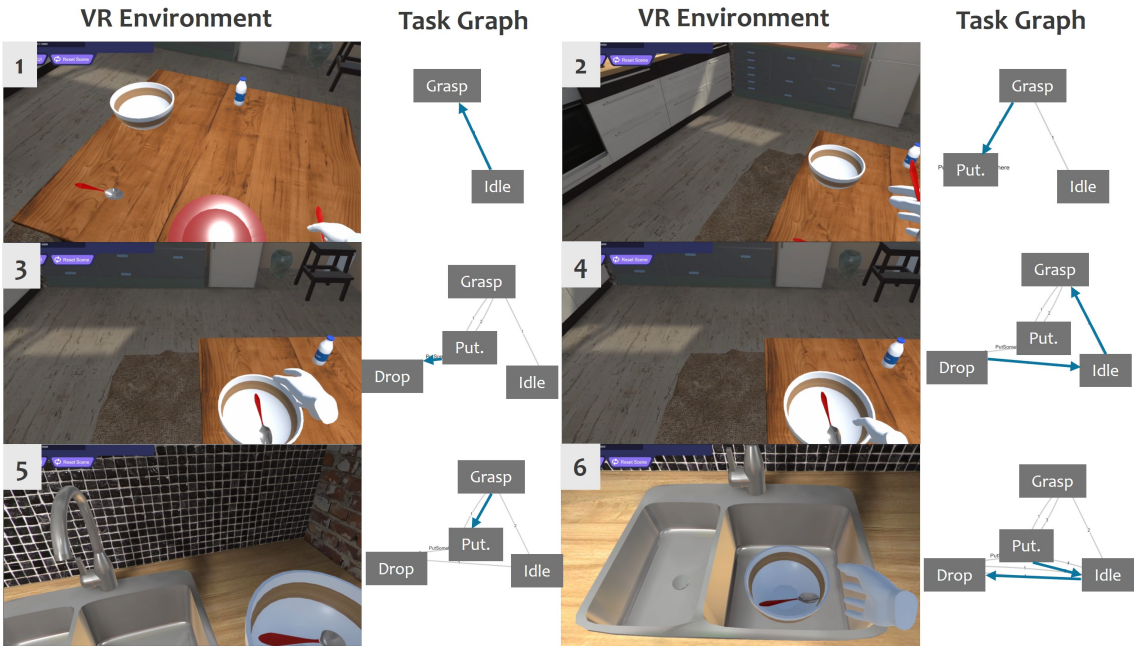


Figure 4.2: Online segmentation for unseen task 1. The blue arrow highlights the newly appeared edge that is essential for the task execution.

As figs. 4.2 to 4.4 depict, our method generalizes well to unseen scenarios, effectively recognizing the necessary activities. These high-level representations of entities' positions, velocities, and spatial relationships are intuitive and easily understood by humans. Moreover, they are also interpretable for robots as numeric data.

However, Figures 4.2 steps 2 to 3 and 4.4 reveal an inevitable situation: not only variations in human demonstrations introduce discrepancies in the task graph (as shown in Figure 1.4), but also hardware limitations and noise from the simulation engine further contribute to this issue, often resulting in cycles that exceed the expected activity sequence. These cycles, depicted as redundant loops in the task graph, can complicate the interpretation and execution of tasks by robots.

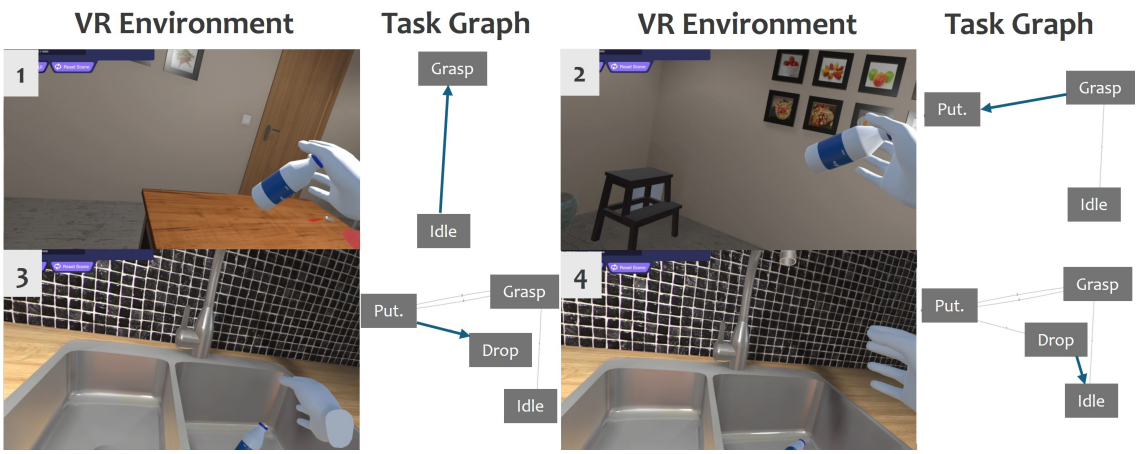


Figure 4.3: Online segmentation for unseen task 2.

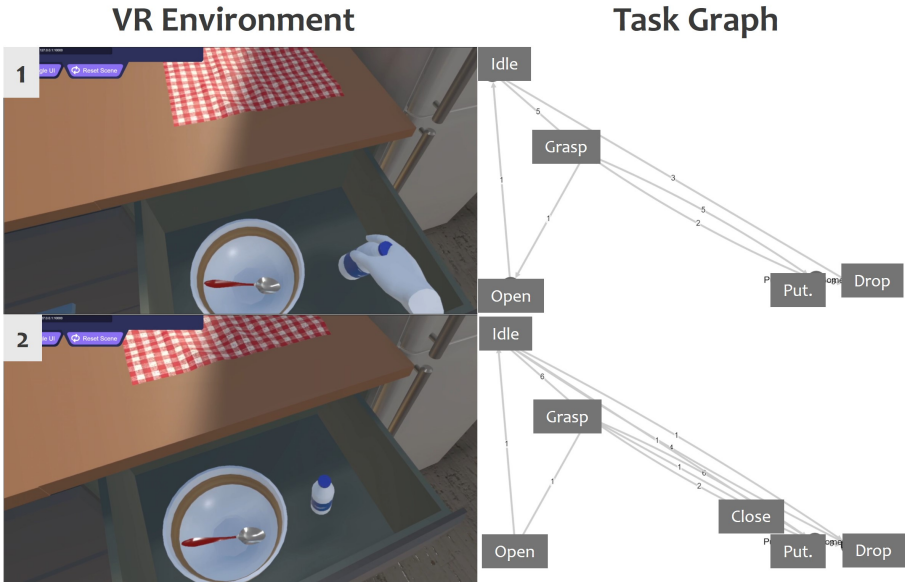


Figure 4.4: Online segmentation for unseen task 3 with two keyframes.

4.3 Activity Sequence Reconstruction

Based on Corollary 4 which is explained in Section 3.4, we define a simplified combined De Bruijn graph as the Aggregated Maximal Weighted Subgraph, where each edge weight corresponds to the number of sequences. This subgraph has an Eulerian path, provided that the original graphs have the same start and end vertices. For each task corresponding in Section 4.2, two demonstrations were collected, and used the activity's first letter to refer to the activity (`I`←`Idle`, `O`←`Open`, `G`←`Grasp`, `P`←`PutSomethingSomewhere`, `D`←`Drop`, `C`←`Close`).

4.3.1 Task 1: put spoon (a new object) and vase into sink (a new container)

As for task 1 described in Section 4.2: Put "Spoon2" (in ontology, but not in the decision tree value scope) and "Vase" into "Sink" (new container). The collected 2 demonstrations are:

1. `IGOIGPGDIGPDIGC`
2. `IGOIGPGDIGPGPDIGCGC`

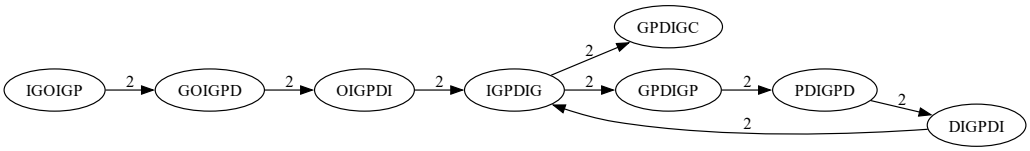


Figure 4.5: Aggregated maximal weighted subgraph for task 1.

In sequence 1, the repeated string `GPGP` and `GCGC` are mainly caused by intermittent motion, which means that when transporting the object, if there is a short pause, activity "PutSomethingSomewhere" will change to "Grasp", a common situation in reality, but also introduces unexpected cycles in task graph. These are the additional noises that need to be removed.

For task 1, the detailed analysis is discussed in Section 3.4, when $k = 7$, the *Aggregated maximal weighted subgraph* is shown in Figure 4.5 and the reconstructed sequence is `IGOIGPDIGPDIGC`, precisely described the most compact representation to complete the given task from the demonstrations. By comparing the reconstructed sequence with the original sequences, our method demonstrates its ability to handle variations not only caused by humans but also by machines.

4.3.2 Task 2: put a bottle (an unseen object) into the sink (a new container)

For task 2, the De Bruijn graphs are shown in Figures 4.6 and 4.7, and the corresponding reconstructed activity sequence is: `IGPDI`. In Figure 4.6, it's obvious that only two nodes satisfy the condition that $|in - out|$ degree $= 1$, thus don't need to specify the start node manually. The collected demonstration 2 showcases a situation where the object dropped unexpectedly when transported. The experiment results showed that our method has the capability to handle unexpected situations and reconstruct the essential steps for the execution of a task.

1. `IGPDI`
2. `IGPGPDIGPDI`

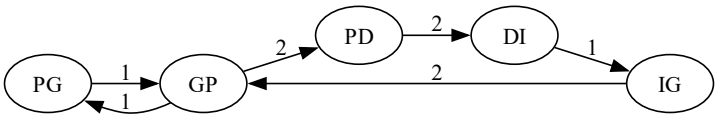


Figure 4.6: Combined de Bruijn graph for task 2.

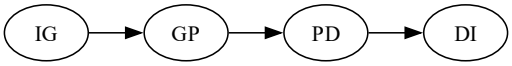


Figure 4.7: Aggregated maximal weighted subgraph for task 2.

4.3.3 Task 3: clean the table

This task contains a new location and unseen object with different task execution order. For task 3, the collected two demonstrations and the corresponding de Bruijn graphs are shown in Figures 4.8 and 4.9, where the reconstructed sequence is `IGPDIGOIGIPGDGC`.

1. `IGPDIGPDIGOIGPGPDIGIPGDGC`
2. `IGPDIGPDIGOIGPDIGIPGDGC`

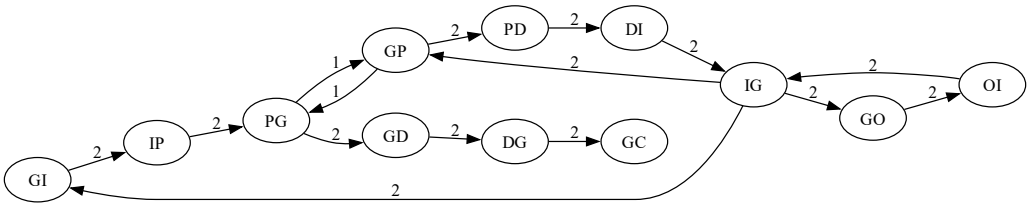


Figure 4.8: Combined de Bruijn graph for task 3.

4.3.4 Discussion

There are some challenges and problems encountered during experiments. The first is to train the decision tree, the manually labeled action "Close" occurred after the "Vase" was placed in the "Drawer". The trained decision tree captured this feature and as a result, if we want to correctly detect the action of "Close" in experiments, "Vase" needs to be ensured is in the "Drawer".

Software and hardware limitations affect experimental performance and introduce additional noise. When more objects are involved in collision detection, especially when containers are involved, resulting in slower physics calculations, and the timestep of the physics calculation engine is uneven, frames are dropped, and sometimes even the transmitted predicates are missing, thus creating noisy activity

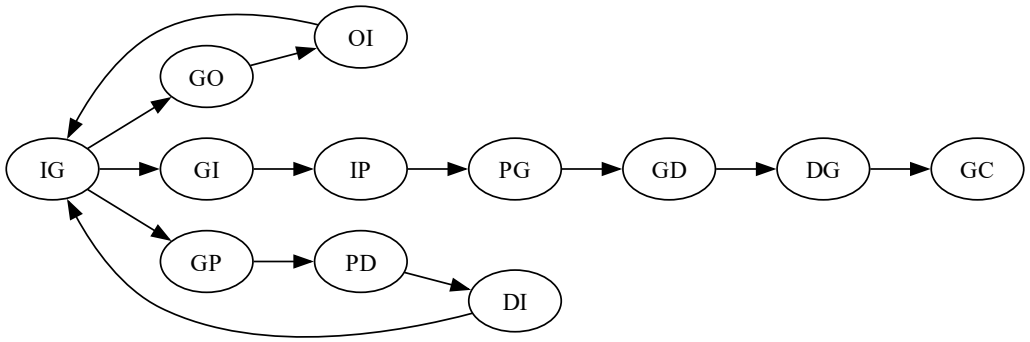


Figure 4.9: Aggregated maximal weighted subgraph for task 3.

sequences that make reconstruction, especially for complex tasks, difficult.

Due to the above problems, additional obstacles exist to collect sequences that meet the conditions. For example, when putting other objects in the drawer, the ideal activity transition as the end is "Close" (the drawer) to "Idle", but the action of closing the drawer will cause the object to roll over in the drawer. Since "Drop" is recognized based on the downward velocity vector, sometimes, the segmentation system will recognize a "Drop" after "Close", which will affect the collection. Moreover, the frame drop will also introduce a velocity peak and affect the recognition results related to the calculation of velocity. Switch debounce algorithm might help to eliminate the velocity peak caused by frame drop, and prevent wrong recognition.

Another potential solution is adding another predicate, like `objectActedOn`, to describe an intention about the hand motion, once such intention is determined, then we don't need to consider the states of other objects. In the previous example, before "Close", if such an intention points at "Drawer", then we can exclude the states of other objects. Thus, even if the rollover situation still happens, the system won't mis-predicate the activity. This approach can help isolate the primary action and reduce the impact of secondary movements on the recognition system.

5
Conclusion

This thesis presents a method to bridge the gap between low-level sensor data processing by robots and high-level concepts for human understanding. By developing an interpretable online segmentation and recognition system using supervised learning and decision trees, robots can now segment human actions in real-time and translate them into high-level concepts understandable to humans. The method also demonstrates effective generalization to unseen scenes. The use of de Bruijn graph and sequence processing algorithms can effectively remove variation and noise in human demonstrations, so that the most representative task sequences can be reconstructed, further improving task scheduling. Despite the limitations of specific manual labeling requirements and hardware constraints, this work enhances bidirectional understandable communication in HRC.

There are several potential improvements for future work. First, in experiments, the activity `Idle` occupies the majority of the time (as indicated in Figure 3.3). However, this period often includes moments when the human is thinking and performing decision-making, not truly idle. Eye tracking could be useful to let the robot know where the human is gazing, assisting in predicting and preempting human actions more proactively.

To eliminate noise or potential segmentation errors, a ground truth task graph can be defined alongside the manual labeling of human demonstrations. Then, LLMs can act as supervisors, performing basic reasoning tasks to correct these errors.

The ontology structure can be further optimized to adapt to different low-level strategies. For example, although both a spoon and a cup have a common activity `Grasp` in high-level concept, the exact methods for grasping them are different in practice. Introducing affordances as entity properties in the ontology can ensure that entities with the same affordance (e.g., spoon and knife are similar, but not with the cup as cup is easily broken) share the same low-level strategy.

Bibliography

[1] B. Hopkins and R. J. Wilson, "The truth about königsberg," The College Mathematics Journal, vol. 35, no. 3, pp. 198–207, 2004.

[2] E. Matheson, R. Minto, E. G. Zampieri, M. Faccio, and G. Rosati, "Human–robot collaboration in manufacturing applications: A review," Robotics, vol. 8, no. 4, p. 100, 2019.

[3] M. Mayr, "Learning with skill-based robot systems: Combining planning & knowledge representation with reinforcement learning," 2024.

[4] W. Mao, R. Desai, M. L. Iuzzolino, and N. Kamra, "Action dynamics task graphs for learning plannable representations of procedural tasks," arXiv preprint arXiv:2302.05330, 2023.

[5] W. Wang, R. Li, Y. Chen, and Y. Jia, "Human intention prediction in human-robot collaborative tasks," in Companion of the 2018 ACM/IEEE International Conference on Human-Robot Interaction, HRI '18, (New York, NY, USA), p. 279–280, Association for Computing Machinery, 2018.

[6] L. Yan, X. Gao, X. Zhang, and S. Chang, "Human-robot collaboration by intention recognition using deep lstm neural network," in 2019 IEEE 8th International Conference on Fluid Power and Mechatronics (FPM), pp. 1390–1396, 2019.

[7] X. Chen, Y. Jiang, and C. Yang, "Stiffness estimation and intention detection for human-robot collaboration," in 2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA), pp. 1802–1807, 2020.

[8] L. Orsag, T. Stipancic, and L. Koren, "Towards a safe human–robot collaboration using information on human worker activity," Sensors, vol. 23, no. 3, 2023.

[9] K. Ramirez-Amaro, M. Beetz, and G. Cheng, "Understanding the intention of human activities through semantic perception: observation, understanding and execution on a humanoid robot," Advanced Robotics, vol. 29, no. 5, pp. 345–362, 2015.

[10] A. Vigné, G. Sarthou, and A. Clodic, "Primitive action recognition based on semantic facts," in Social Robotics (A. A. Ali, J.-J. Cabibihan, N. Meskin, S. Rossi, W. Jiang, H. He, and S. S. Ge, eds.), (Singapore), pp. 350–362, Springer Nature Singapore, 2024.

[11] A. Haidu and M. Beetz, "Automated acquisition of structured, semantic models of manipulation activities from human vr demonstration," in 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 9460–9466, IEEE, 2021.

[12] C.-Y. Chang, D.-A. Huang, D. Xu, E. Adeli, L. Fei-Fei, and J. C. Niebles, "Procedure planning in instructional videos," in European Conference on Computer Vision, pp. 334–350, Springer, 2020.

[13] J. Bi, J. Luo, and C. Xu, "Procedure planning in instructional videos via contextual modeling and model-based policy learning," in Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 15611–15620, 2021.

[14] R. Zhang, Q. Lv, J. Li, J. Bao, T. Liu, and S. Liu, "A reinforcement learning method for human-robot collaboration in assembly tasks," Robotics and Computer-Integrated Manufacturing, vol. 73, p. 102227, 2022.

[15] H. J. Levesque, "Knowledge representation and reasoning," Annual review of computer science, vol. 1, no. 1, pp. 255–287, 1986.

[16] Z. Chen, Y. Zhang, Y. Fang, Y. Geng, L. Guo, X. Chen, Q. Li, W. Zhang, J. Chen, Y. Zhu, et al., "Knowledge graphs meet multi-modal learning: A comprehensive survey," arXiv preprint arXiv:2402.05391, 2024.

[17] S. Staab and R. Studer, Handbook on ontologies. Springer Science & Business Media, 2013.

[18] A. Cangelosi and M. Asada, Cognitive robotics. MIT Press, 2022.

[19] B. McBride, The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS, pp. 51–65. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[20] D. L. McGuinness, F. Van Harmelen, et al., "Owl web ontology language overview," W3C recommendation, vol. 10, no. 10, p. 2004, 2004.

[21] A. Géron, Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. " O'Reilly Media, Inc.", 2022.

[22] L. Breiman, Classification and regression trees. Routledge, 2017.

[23] J. Fürnkranz, "Pruning algorithms for rule learning," Machine Learning, vol. 27, pp. 139–172, 1997.

[24] J. R. Quinlan, "Simplifying decision trees," International journal of man-machine studies, vol. 27, no. 3, pp. 221–234, 1987.

[25] N. Nagarajan and M. Pop, "Sequence assembly demystified," Nature Reviews Genetics, vol. 14, pp. 157–167, 2013.

[26] P. E. Compeau, P. A. Pevzner, and G. Tesler, "Why are de bruijn graphs useful for genome assembly?," Nature biotechnology, vol. 29, no. 11, p. 987, 2011.

[27] Ž. Fišer Pečnikar and E. V. Buzan, "20 years since the introduction of dna barcoding: from theory to application," Journal of applied genetics, vol. 55, pp. 43–52, 2014.

[28] L. N. DeLong, R. F. Mir, M. Whyte, Z. Ji, and J. D. Fleuriot, "Neurosymbolic ai for reasoning on graph structures: A survey," arXiv preprint arXiv:2302.07200, 2023.

[29] T. Bates, K. Ramirez-Amaro, T. Inamura, and G. Cheng, "On-line simultaneous learning and recognition of everyday activities from virtual reality performances," in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3510–3515, IEEE, 2017.

[30] G. E. Batista, A. L. Bazzan, M. C. Monard, et al., "Balancing training data for automated annotation of keywords: a case study.," Wob, vol. 3, pp. 10–8, 2003.
[31] L. Euler, "Solutio problematis ad geometriam situs pertinentis," Commentarii academiae scientiarum Petropolitanae, pp. 128–140, 1741.
[32] B. Bollobás, Modern graph theory, vol. 184. Springer Science & Business Media, 2013.
[33] G. Pólya, R. E. Tarjan, and D. R. Woods, Hamiltonian and Eulerian Paths, pp. 157–168. Boston: Birkhäuser Boston, 2010.
[34] P. E. Compeau, P. A. Pevzner, and G. Tesler, "How to apply de bruijn graphs to genome assembly," Nature biotechnology, vol. 29, no. 11, pp. 987–991, 2011.
[35] N. G. De Bruijn, "A combinatorial problem," Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam, vol. 49, no. 7, pp. 758–764, 1946.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY