FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Reasoning Capabilities for a Cognitive-Assistive Assembly System

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Ardian Koltraka
Matrikelnummer 1229395

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Inf. Dr.rer.nat. Jens Knoop
Mitwirkung: Dipl.-Ing.(FH) Dr.techn. Dietmar Schreiner

Wien, 28. Juni 2017

Ardian Koltraka	Jens Knoop

Reasoning Capabilities for a Cognitive-Assistive Assembly System

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computational Intelligence

by

Ardian Koltraka
Registration Number 1229395

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Inf. Dr.rer.nat. Jens Knoop
Assistance: Dipl.-Ing.(FH) Dr.techn. Dietmar Schreiner

Vienna, 28th June, 2017

Ardian Koltraka	Jens Knoop

Erklärung zur Verfassung der Arbeit

Ardian Koltraka
Mooslackengasse 21/406, 1190

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. Juni 2017

Ardian Koltraka

Acknowledgements

Hereby, I would like to express my very profound gratitude to my advisers Prof. Dr. Jens Knoop and Dr. Dietmar Schreiner, for their enthusiastic encouragement, fruitful critiques, and the patient guidance. They have gone beyond their duties to instil great confidence in both myself and my research work.

Moreover, I would like to thank the members of Profactor company for their support, valuable provided tools and materials. Here I have to give a special thank to Matthias Plasch who was my thesis-supervisor at Profactor GmbH, I am forever indebted to him.

I would express a deep sense of gratitude to my parents and my brother for being such a huge support all my life along. This accomplishment would not have been possible without them. Thank you.

I am also very thankful to my fiancée, for her continuous encouragement, support, and patients that she has given to me. I have never met anyone who believes in me more. Thank you for making me stronger than I am.

Finally, I would like to thank Almighty God for giving me the strength, knowledge, ability and opportunity to undertake this research work and to pursue and complete it satisfactorily. Without his blessings, this achievement would not have been imaginable.

Abstract

Modern production systems have to meet the challenges of changing market trends, high flexibility in product types and variants, and short innovation cycles in order to stay competitive. In this context, Human-Robot-Cooperation is considered as a key technology to improve efficiency and to reduce operating costs. In order to realize a cooperative and effective relationship between a human and a robot, e.g. when performing an assembly task together, concepts for communication and coordination among the actors, as well as for the representation of domain knowledge are required. Such a robotic assistant must do the right thing to the right object in the right way and reason about the task that it has to do in cooperation with the human. The ability of the robot to recognize and learn the current state of an assembly task is crucial, because it will be applied for further reasoning, e.g. establishing the next actions to be done by the robot or by the human and to derive assembly task variants. The overall aim of this master thesis is to develop a reasoning system which hypothesizes the current state of the assembly task by integrating and combining a knowledge base task model with the input from sensing and actuating components. A general task model which represents the task states has to be build, were each task state consists of a human state, robot state and a set of object relations. Having such a model whose construction conforms to the information received by sensing and actuating components enables the capability to predict the current state of the assembly task with the help of supervised machine learning algorithms.

Kurzfassung

Rasch veränderliche Marktsituationen, hohe Variantenvielfalt und Flexibilität sowie immer kürzer werdende Innovationszyklen, erfordern häufig eine schnelle Anpassung moderner Produktionssysteme an die neuen Gegebenheiten. Mensch-Roboter-Kollaboration (MRK) wird in diesem Zusammenhang als Schlüsseltechnologie zur Effizienzsteigerung und Kostenreduktion betrachtet. Zur Umsetzung effizienter MRK, z.B. in Montageprozessen, sind durchdachte Konzepte für die Kommunikation und Koordination unter den Akteuren, sowie die Abstraktion von domänenrelevantem Wissen notwendig. Die wesentliche Herausforderung an derartige Roboterassistenzsysteme ist das Setzen korrekter Handlungen zum richtigen Zeitpunkt und in nahtloser Kollaboration mit dem Menschen. Dabei sind das Lernen und Identifizieren von Zuständen in einem Montageprozess unverzichtbare Fähigkeiten, um auf nachfolgend auszuführende und zielführende Handlungen schließen zu können. Ziel dieser Masterarbeit ist die Entwicklung eines Reasoning-Systems um Hypothesen über den aktuellen Zustand des Montageprozesses aufstellen zu können. Als Informationsquellen für den Reasoning-Prozess sollen das vorhandene Domänenwissen und erfasste Daten von Sensor- und Aktuator-Systemen in kombinierter Weise herangezogen werden. Zur Wissensrepräsentation wird ein generisches Datenmodell zur Beschreibung des Montageprozesses aus Prozesszuständen entwickelt. Jeder Prozesszustand ist durch erfasste Zustände von Mensch und Roboter sowie der räumlichen Anordnung der relevanten Objekte (z.B. Bauteile) beschrieben. Dieses Datenmodell ist konform mit den Datenformaten der Sensor- und Aktuator-Systeme und ermöglicht qualitative Hypothesen über den aktuellen Prozesszustand unter Verwendung von Algorithmen zum überwachten, maschinellen Lernen.

Contents

CHAPTER 1

Introduction

Contemporary production systems have to be designed to deal with the growing challenges of market trends, high flexibility in product types and variants, and short innovation cycles in order to stay competitive. As in these production systems, robots are gradually getting more familiar with the human populated environments, they need to possess more complex reasoning abilities. Also cooperative robots are receiving greater acceptance in the human populated environments because the typical advantages provided by manipulators are combined with an intuitive usage. In this context, Human-Robot-Cooperation (HRC) is considered as a key technology to improve efficiency and to reduce operating costs. The goal of these robot systems is not only to operate efficiently and safely in natural, populated environments, but also be able to achieve higher levels of cooperation and communication with humans. The robotic assistant must do the right thing to the right object in the right way and reason about the task that it has to do in cooperation with the human. In such production systems, the focus is on a proximate cooperation where the human and the robot are co-located. In order to realise a cooperative and effective relationship between a human and a robotic system, e.g. the process of working together to accomplish an assembly task, concepts for communication and coordination among the actors, as well as for the representation of domain knowledge are required. The idea of combining an assembly robot behaviour with cognitive skills is highly challenging, and requires interdisciplinary cooperation between classical robotics, cognitive sciences, and psychology. Humans as nondeterministic factors make cognitive sciences and artificial intelligence important research fields in HRC (BWB08). The robot may learn from the task activity performed correctly by the human user. The ability of the robot to recognize and learn the current state of an assembly task is crucial, because it will be applied for further reasoning, e.g. deducing the next actions to be done by the robot or by the human and to derive assembly task variants. The overall aim of this master thesis is to develop a reasoning system which hypothesizes the current state of the assembly task. The main idea is to integrate and combine an assembly task knowledge with the input from sensing and actuating components and to consider the human and the robot capabilities in the assembly task execution. The reasoning system should be able to make a reliable analysis about the human states, the robot states and also to the object relations. A model which represents the states of the assembly task has to be build, were each task state consists of a human state, a robot state and a set of object relations. Such a meta model should express the information received by sensing and actuating components which also is used for the prediction of the current state of the assembly task with the help of supervised machine learning algorithms. A fast and a reliable classification of the task states is required, as well as strategies to conclude with the best assembly task state prediction.

1.1 Motivation

Although robot technology was primarily developed in the mid and late 20th century, it is important to note that the notion of robot-like behaviour and its implications for humans have been around for centuries in religion, mythology, philosophy, and fiction (GS07). "Robot" appears to have first been used in Karel Chapek's 1920's play Rossum's Universal Robots, though this was by no means the earliest example of a human-like machine. Indeed, Leonardo da Vinci sketched a mechanical man around 1495, which has been evaluated for feasibility in modern times (Ros06). The first assembly robot which was a die casting machine was invented by George Devol and was a prototype Unimate industrial robot, which worked on a General Motors assembly line at the Inland Fisher Guide Plant in Ewing Township, New Jersey, in 1961 (Tza00). It was a fairly simple robot, compared to the ones we have nowadays, because it was designed to perform only one task. In Europe the first industrial robots were installed in Sweden in 1967 at Svenska Metallverken in Upplands Väsby (Wal08). The robots did monotonous jobs like picking in and out. In 1969 Unimation installed its first robots for spot-welding, they were 26 robots for spot-welding car bodies. In 1972, Europe followed by setting up a spot-welding line with robots at Fiat (Wes00). The aim of the very first robots were to perform simple tasks such as pick and place, since they had not external sensing yet. Robots replaced humans in monotonous, repetitive, heavy and dangerous tasks. Industrial robots nowadays are divided in three different groups namely: material handling, process operations and assembly. In the end of the 1970s and the beginning of the 1980s when the robots started to manage both a more complex motion and had external sensor capacity, the robot development was mainly concentrated on assembly. On this time more complex robot applications followed, like welding, grinding, deburring (Wal08). Assembly robots were used to reduce costs, increase productivity, improve product quality and eliminate harmful tasks. As the robots are leaving factory environments and gradually moving to human populated environments, they need to have more complex cognitive abilities. They do not only have to operate efficiently and safely in natural, populated environments, but also be able to achieve higher levels of cooperation and communication with humans. Human-Robot-Collaboration is a research field with a wide range of applications, future scenarios, and potentially a high economic impact (BWB08).

To have an effective collaboration, recent research in the field of psychology has focused on cognitive processes of joint-action among humans (PvSB09). Psychological studies (GS03) show that collaborating in human teams, requires an effective coordination between participants that plan and execute their actions in relation to what they anticipate from the other team members, and not just react on the others current activities. A concept of a profitable working environment which is designed to support joint action of humans and industrial assembly robots is given where a safe collaboration is possible. The system anticipates human behaviour, based on knowledge databases and decision processes, ensuring an effective collaboration between the human and robot (LNR^{+}08). In this work we focus on a very specific HRC scenario that is realized at Profactor GmbH. The scenario is depicted in Figure 1.1, where the robot is in a cooperation with the human agent performing a steam cooker assembly task. The robotic system is equipped with a perception system, which recognizes human states (with the help of Kinect 2 sensor), robot states and spatial relations between objects (with the help of Asus Xtion sensor). The steam cooker assembly task consists of four objects, namely: Tray, Base, Ring, and Heater.

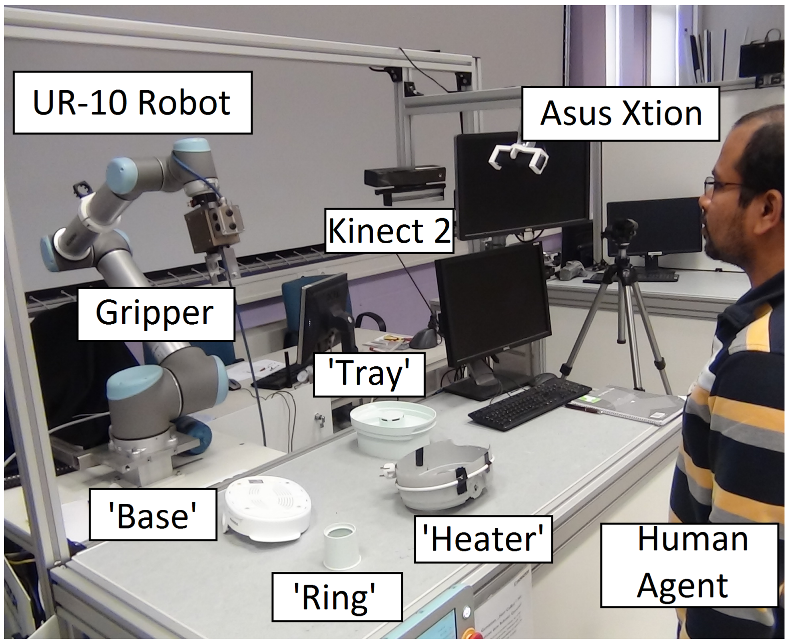


Figure 1.1: HRC in performing a steam cooker assembly task (APPR16).

The robotic system performs manipulations by using a gripper and a robotic arm combined with a manipulation planner software component. An important challenge of this robotic system is the recognition of the current state of the assembly task. In this work we address this particular challenge and propose a methodology based on machine learning techniques in order to solve it.

1.2 Problem Statement

Modern production systems have to meet the challenges of changing market trends, high flexibility in product types and variants, and short innovation cycles in order to stay competitive. In this context, Human-Robot-Cooperation is considered as a key technology to improve efficiency and to reduce operating costs. In order to realize a collaborative and effective relationship between a human and a robot, e.g. when performing an assembly task together, concepts for communication and coordination among the actors, as well as for the representation of domain knowledge are required. Such a robotic assistant must do the right thing to the right object in the right way and reason about the task that it has to do in cooperation with the human. The ability of the robot to recognize and learn the current state of an assembly task is important, in order to decide on the next actions to be done and to derive assembly task variants. The overall aim of this work is to develop a reasoning system which hypothesizes the current state of the assembly task by integrating and combining a knowledge base with the input from the perceived sensor and by considering its own capabilities in the assembly task execution.

1.3 Research Goal

The aim of this work is to develop a reasoning system, which reasons about the current state of an assembly task that is to be carried out by a robotic system collaborating with a human. The current state of the assembly task is composed of human state, robot state, and object relations.

The robotic system which is presumed for this thesis is equipped with sensing and actuating components, to perceive what is happening in the environment (Object Recognition and Human Action Recognition systems) and to perform manipulations using a Gripper and a Robotic arm combined with a manipulation planner software component. Figure 1.2 provides a high level overview of how an assembly task execution is carried out. Each state of an assembly task is characterized by a stable configuration of the involved objects and stable states of the Human Action Recognition, Manipulation planner and Gripper. There are some Pre/Post conditions for a task step to be satisfied that will lead to the next assembly task state (stable configuration). If the conditions are not met then there must be some deviation state. So it has to be a list of assembly task execution descriptions given to the robot and achieved by it. Since the robot actions are limited, the system needs an intelligent way to manage the data with the purpose of checking the robot capabilities for a given task. Based on our findings we eventually aim at answering the following two questions:

1. How does the reasoning system should be able to check the current state of an assembly task? By considering:

   – robot states,

   – object locations, and

- human states perceived by the robotic systems.

2. Which Machine Learning classifier is the most appropriate one for this problem?

The contribution in answering the above two questions regarding its usefulness for Profactor and more generally for the HRC problem is evaluated. Moreover, possible ideas and proposals for further research will be recommended concerning the perspectives of being able to extend the reasoning system.

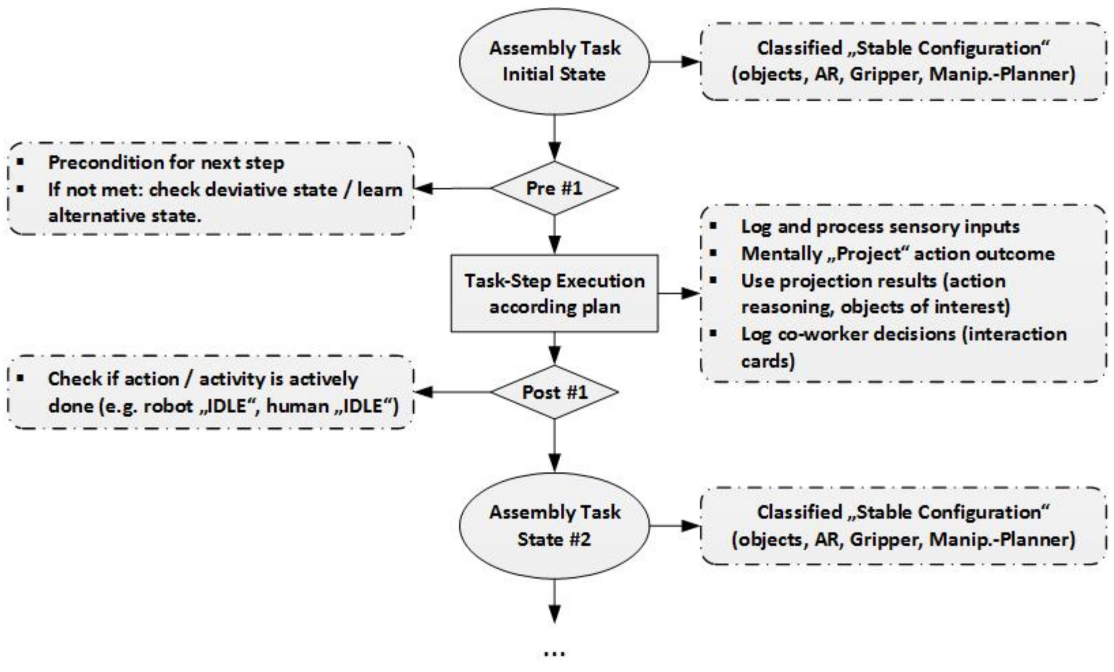

Assembly Task Initial State
Classified „Stable Configuration" (objects, AR, Gripper, Manip.-Planner)
Precondition for next step
If not met: check deviative state / learn alternative state.
Pre #1
Log and process sensory inputs
Mentally „Project" action outcome
Use projection results (action reasoning, objects of interest)
Log co-worker decisions (interaction cards)
Task-Step Execution according plan
Check if action / activity is actively done (e.g. robot „IDLE", human „IDLE")
Post #1
Assembly Task State #2
Classified „Stable Configuration" (objects, AR, Gripper, Manip.-Planner)
...

Figure 1.2: Assembly task execution sequence.

1.4 Contributions

The main contribution in this master thesis is the design, implementation, and evaluation of a reasoning system that is used in a proximate cooperation. The reasoning system is able to predict the current state of an assembly task, and consists of two main developed modules namely:

1. a module that provides the use of machine learning (ML) classifiers,

2. and another module in which:

   a) an ontology is constructed to model the assembly task description,

b) reasoning about the constructed ontology with the help of Prolog queries is made,

c) the service provided by the ML module is called in order to predict the unknown class (task state) of the fetched data from the perception system.

The constructed ontology model that represents the task description is crucial in our work. The ontology model consists of human states, robot states, object relations and task states of the assembly task. Object relations are given by a binary predicate. The developed reasoning system is initially used for a steam cooker assembly task at Profactor, which fully meets our expectations. In considering more general HRC at Profactor our contribution is suited to be applied on those assembly tasks where their ontology model evolves as an extension of the base ontology model proposed by us. The extended ontology model must conform to the base ontology model. Our contribution can also be applied in a more general HRC problem as long as the ontology model of the assembly task can be constructed by extending the base one developed by us, and again the constructed ontology model must conform to the base one. Thus, to use our contribution as a "model for the world" the essential constraint is that the constructed ontology model of the intended assembly task must conform to the ontology model developed in our contribution.

After the reasoning system is designed and implemented, our focus is put on ML classifiers evaluation, in order to have a feeling, which classifier should be used to predict the task state of an assembly task.

1.5 Structure of thesis

Figure 1.3 shows the structure of the thesis and the motivation behind it is elaborated. Before the various sections and their content are presented, a general remark concerning the overall structuring of this specific work is necessary.

As specified in the previous sections and detailed thereafter, the goal of this thesis is to design, implement and evaluate a reasoning system which hypothesizes the current state of an assembly task. In Chapter 1 we illustrate the general motivation behind the conducted research, outline the research problem statement, the research goal and our contributions. In Chapter 2 and 3 we introduce the basic concepts and systems that are needed to accomplish the goal of this work and also existing methods and approaches that are dealing with similar problems. In Section 2.3 we introduce existing ML methods which are used in manufacturing that also can be used to predict the current state of the assembly task. In Chapter 4 the ontology model is developed, which is spread in three sections, namely modeling classes (Section 4.1), individuals (Section 4.2) and properties (Section 4.3). In Chapter 5 we describe the design and implementation of the reasoning system that will take the task model concept introduced in Chapter 4 and use it for further processing and reasoning. Chapter 6 gives an overview on performance evaluation between the classifiers that will be used to make the task state prediction.

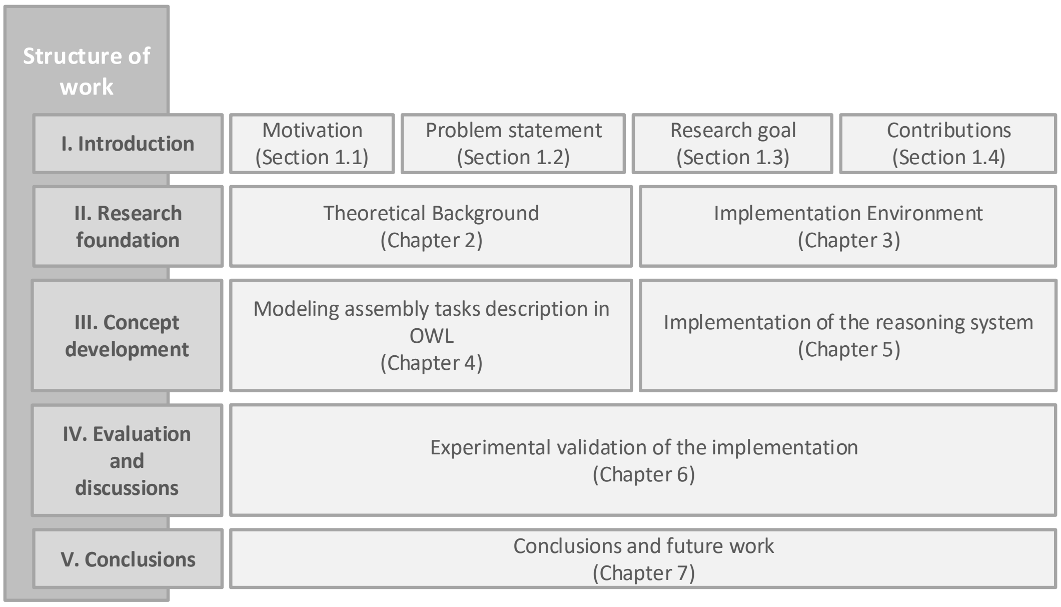

Structure of work
I. Introduction
Motivation (Section 1.1)
Problem statement (Section 1.2)
Research goal (Section 1.3)
Contributions (Section 1.4)
II. Research foundation
Theoretical Background (Chapter 2)
Implementation Environment (Chapter 3)
III. Concept development
Modeling assembly tasks description in OWL (Chapter 4)
Implementation of the reasoning system (Chapter 5)
IV. Evaluation and discussions
Experimental validation of the implementation (Chapter 6)
V. Conclusions
Conclusions and future work (Chapter 7)

Figure 1.3: Structure of this thesis.

Chapter 7, structured in two parts summarizes the research work, concludes the findings, and gives an outlook into potential future directions in this research domain. It reviews the achieved results and knowledge gained by this work and puts it in the greater context, and it presents an outlook that identifies further research areas related to the findings.

CHAPTER 2

Theoretical Background

In the previous chapter we showed the general motivation in this research, outlined the research problem, the research goal and our contribution. Therefore the basic concepts and systems that are needed in the accomplishment of our goal should be introduced. In this chapter, we summarise the basic concepts of Description Logics (DLs), Web ontology Language (OWL), and some applications of ML techniques that will be applied to identify the current state of an assembly task. DLs and OWL are important for this work because as we noted earlier the task model of the assembly task will be represented by an ontology. The simplified examples that are used to illustrate the concepts in this chapter are taken from our constructed ontology model, which is described in Chapter 4.

2.1 Description Logics

This section is based on (BCM$^+$03). It recapitulates the basic notions of DL as a formal language for representing knowledge and reasoning about it. DL is the most modern name used in knowledge representation (KR) formalisms which represents the knowledge of an application domain by first defining the relevant concepts of the domain, and then specify properties of objects and individuals which occur in the domain based on the defined relevant concepts. Most DLs are fragments of classical first order logic (FOL) and are very closely related to modal logic. But in contrast to FOL, DLs are decidable, their syntax is well-suited for representing structured knowledge etc. The design space of DLs is seen as an interval which ranges from lightweight DLs (not expressive enough) to very expressive DLs. The language of a DL is based on:

1. A vocabulary consisting of concept names, role names, and individuals.
2. A set of concept constructors and role constructors, to build more complex concepts and roles from the basic names.

3. Rules for writing Knowledge Bases (KBs) which comprises two components, the TBoxes (terminological boxes) and the ABoxes (assertional boxes).

Concepts names intention is to denote atomic classes, unary predicates (Student, Robot etc). Individuals are constants. Role names, denote binary relationships between individuals. More complex concepts and roles can be build by the help of available constructors. There are two types of constructors namely concept constructors and role constructors. An example which represents the construction of more complex concepts and roles is given as follows by the use of (see Table 2.3 for the Syntax and Semantics of DL):

1. Concept constructors:

RobotStates $\sqcap$ HumanStates	TaskObjects $\sqcup$ RobotStates
TaskObjects $\sqcap \neg$ HumanStates	≥ 2 toTheLeftOf.TaskObjects
RobotStates $\sqcap \forall$ hasHumanState.Idle	$\exists$toTheRightOf.$\{$Tray$\}$

2. Role constructors:

inCenterOf $\cup$ toThesideOf	hasHumanState $\cap \neg$ hasRobotState

Elementary descriptions are atomic concepts and atomic roles. Complex descriptions can be built from them inductively with concept constructors. A KR system based on Description Logics provides facilities to set up knowledge bases, to reason about their content, and to manipulate them. Figure 2.1 sketches the architecture of such a system.

The TBox is a set of terminological axioms that state how concepts or roles are related to each other, i.e., the vocabulary of an application domain, while the ABox contains concept membership assertions and role membership assertions about named individuals in terms of this vocabulary. Two main kinds of terminological axioms are:

1. General concept inclusions (GCIs): $C \sqsubseteq D$.
2. Definitions: $A \equiv D$, where A is a concept name.

The semantics of the KBs is given in terms of interpretations, similar to the ones used in FOL. The interpretation consists of:

1. A non-empty domain.
2. An interpretation function.
   - It gives meaning to the basic symbols in the vocabulary.

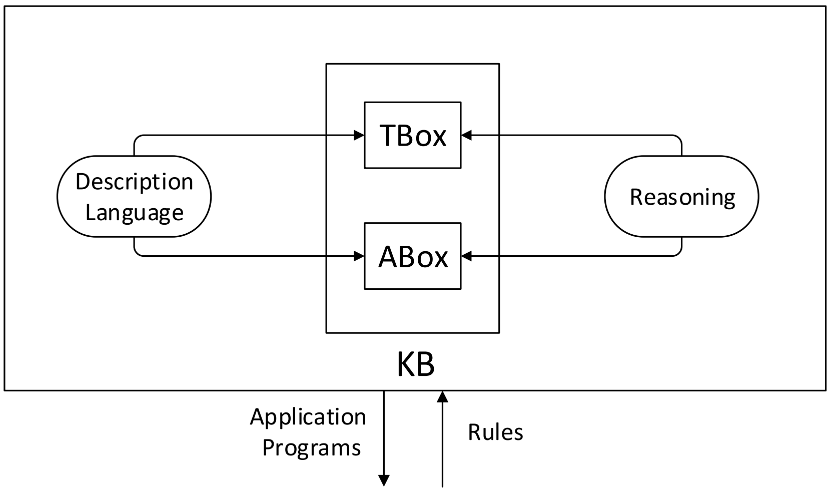


Figure 2.1: Architecture of a knowledge representation system based on Description Logics.

- It is extended to complex concepts and roles, following the rules that define the different constructors.
- It is used to determine satisfaction of axioms in the TBox and assertions in the ABox.

An interpretation satisfies a knowledge base (is a model of the knowledge base), if it satisfies both the ABox and the TBox. We will give formal definitions of the syntax and semantics of the basic DL called $\mathcal{ALC}$ which is the most widely used DL reasoning service. The name $\mathcal{ALC}$ stands for "Attributive concept Language with Complements" (BCM$^+$03). The meaning of the $\mathcal{ALC}$ is formed by starting from a basic DL $\mathcal{AL}$ (Attributive Language), the addition of a constructor is indicated by appending a corresponding letter, $\mathcal{ALC}$ is obtained from $\mathcal{AL}$ by adding the complement operator ($\neg$). The DL that consists only of the following set of constructors (i.e., conjunction, disjunction, negation, existential restriction and value restriction) is called $\mathcal{ALC}$. The $\mathcal{ALC}$ syntax consists of a countable set N_C of concept names, a countable set N_R of role names and a countable set N_I of individual names. The sets of $\mathcal{ALC}$-concept descriptions is the smallest sets such that:

1. Every concept name $A \in N_C$ is an $\mathcal{ALC}$-concept.
2. $\top$ and $\bot$ are $\mathcal{ALC}$-concepts.
3. If C is an $\mathcal{ALC}$-concept, then $\neg C$ is a $\mathcal{ALC}$-concept.
4. If C_1 and C_2 are $\mathcal{ALC}$-concepts, then $C_1 \sqcap C_2$ and $C_1 \sqcup C_2$ are $\mathcal{ALC}$-concepts.

5. If $R \in N_R$ is a role and C is an $\mathcal{ALC}$-concept, then $\forall R.C$ and $\exists R.C$ are $\mathcal{ALC}$-concepts.

In $\mathcal{ALC}$ we only have concept constructors. An $\mathcal{ALC}$ Knowledge Base $\mathcal{K}$ is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. The TBox $\mathcal{T}$ is a finite set of GCIs ($C_1 \sqsubseteq C_2$), and the ABox $\mathcal{A}$ is a finite set of concept and role membership assertions (C(a), R(a,b)). TBox can be used for assigning names to complex descriptions. Terminological axioms in TBox have the form:

$$C \sqsubseteq D \ (R \sqsubseteq S) \text{ or } C \equiv D \ (R \equiv S)$$

where C, D are concepts (and R, S are roles). The first type of axioms are called inclusions, and the second type of axioms are called equalities. An equality in which the left-hand side is an atomic concept is a definition. Definitions are used to introduce symbolic names for complex descriptions. For instance, the axiom

$$\mathit{RobotStates} \equiv HumanState \sqcap \neg \exists hasHumanState.\{\mathit{Rotate}\}$$

associates on the left hand side of the description we associate the name *RobotStates*, the only difference between the robot and the human is that the robot is not able to rotate objects. For example, define $HumanState$ analogously to *RobotStates* as following:

$$HumanState \equiv \mathit{RobotStates} \sqcup \exists hasHumanState.\{\mathit{Rotate}\}$$

The $TaskStates$ is defined as:

$$TaskStates \equiv HumanStates \sqcup \mathit{RobotStates} \sqcup \mathit{ObjectRelations} \sqcup TimeStamp$$

We call a finite set of definitions $\mathcal{T}$ a terminology or TBox if no symbolic name is defined more than once, that is, if for every atomic concept A there is at most one axiom in $\mathcal{T}$ whose left-hand side is A. The inclusion whose left-hand side is atomic is called a specialization. For example to define the concept "RobotStates" in detail, it means that every "RobotStates" concept is a "TaskStates" concept with specialisation

$$\mathit{RobotStates} \sqsubseteq TaskStates.$$

A concept D subsumes a concept C if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every interpretation $\mathcal{I}$, we write it as $\models C \sqsubseteq D$. In Table 2.1 a TBox with assembly task state relation is given. Some of the concept and role atoms in the ABox may be defined names of the TBox. In the ABox, individuals are introduced, by giving them names, and then properties of these individuals are asserted. Using concepts C and roles R, assertions of the following two kinds can be made in an ABox:

C(a), R(b,c).

The first kind which is called concept assertions, the meaning is that "a" belongs to (the interpretation of) C, the second kind is called role assertions, the meaning is that "c" is a filler of the role R for "b".

$RobotStates \sqsubseteq TaskStates$	$RobotStates \equiv HumanState \sqcap \neg\exists hasHumanState.\{Rotate\}$
$RobotStates \sqsubseteq TaskStates$	$HumanState \equiv RobotStates \sqcup \exists hasHumanState.\{Rotate\}$
$ObjectRelations \sqsubseteq TaskStates$	$TaskStates \equiv HumanStates \sqcup RobotStates \sqcup$ $ObjectRelations \sqcup TimeStamp$
$TimeStamp \sqsubseteq TaskStates$	

Table 2.1: A terminology (TBox) with concepts about assembly task states relationships.

Table 2.2 shows an example of an ABox, in which $Tray$, $Base$, $Heater$ and $Ring$ are individual names, then $TaskObjects(Heater)$ means that $Heater$ is a an assembly task object, and $toTheLeftOf(Heater, Ring)$ means that $Ring$ is at the left of $Heater$. An ABox, denoted as $\mathcal{A}$, is a finite set of such assertions and can be seen as an instance of a relational database with only unary or binary relations.

toTheLeftOf(Heater, Ring)	TaskObjects(Heater)
inFrontOf-Generally(Ring, Base)	TaskObjects(Tray)
inFrontOf-Generally(Heater, Tray)	TaskObjects(Ring)
toTheRightOf(Base, Tray)	TaskObjects(Base)

Table 2.2: A world description (ABox).

The semantics of $\mathcal{ALC}$ is given by an interpretation $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\triangle^{\mathcal{I}}$ is a non empty set called domain and $\cdot^{\mathcal{I}}$ is the interpretation function. The interpretation function $\cdot^{\mathcal{I}}$ maps

1. every concept name A to a subset of $\triangle^{\mathcal{I}}$, $A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$
2. every role name R to a set of pairs from $\triangle^{\mathcal{I}}$, $R^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$
3. every individual a to an element of $\triangle^{\mathcal{I}}$, $a^{\mathcal{I}} \sqsubseteq \triangle^{\mathcal{I}}$

To illustrate the meaning of the interpretation for atomic symbols we consider the following:

1. Consider the domain: $\{b, h, t, r\}$
2. Each individual is interpreted as one element
   $Heater^{\mathcal{I}} = h$, $Tray^{\mathcal{I}} = t$, $Ring^{\mathcal{I}} = r$, $Base^{\mathcal{I}} = b$ ...

3. Concepts are interpreted as a set of elements
   $TaskObjects^{\mathcal{I}} = \{b, h, t, r\}$, ...

4. Roles are interpreted as sets of pairs
   $toTheLeftOf^{\mathcal{I}} = \{(h, r)\}$, $toTheRighttOf^{\mathcal{I}} = \{(b, t)\}$,
   $inFrontOf - Generally^{\mathcal{I}} = \{(r, b), (h, t)\}$ ...

In Table 2.3 it is shown how the interpretation function is extended to all concepts.

Constructor	Syntax	Semantics
top/verum	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
bottom/falsum	$\bot$	$\bot^{\mathcal{I}} = \emptyset$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
universal rest	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}} .(R^{\mathcal{I}}(d_1, d_2) \to d_2 \in C^{\mathcal{I}})\}$
existential rest	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}} .(R^{\mathcal{I}}(d_1, d_2) \wedge d_2 \in C^{\mathcal{I}})\}$

Table 2.3: Interpretation function extended to all concepts.

With the help of Table 2.3 and the example for atomic symbols, we make an example for complex concepts in the following way:

1. Consider the domain: $\{b, h, t, r\}$

2. Each individual is interpreted as one element
   $Heater^{\mathcal{I}} = h$, $Tray^{\mathcal{I}} = t$, $Ring^{\mathcal{I}} = r$, $Base^{\mathcal{I}} = b$ ...

3. Concepts are interpreted as a set of elements
   $TaskObjects^{\mathcal{I}} = \{b, h, t, r\}$, ...

4. Roles are interpreted as sets of pairs
   $toTheLeftOf^{\mathcal{I}} = \{(h, r)\}$, $toTheRighttOf^{\mathcal{I}} = \{(b, t)\}$,
   $inFrontOf - Generally^{\mathcal{I}} = \{(r, b), (h, t)\}$ ...

5. The atomic expressions fix the meaning of all the complex ones, e.g.,

$$\begin{array}{lcl}
(\exists toTheLeftOf.TaskObjects)^{\mathcal{I}} & = & \{h\} \\
(\exists toTheRighttOf.TaskObjects)^{\mathcal{I}} & = & \{b\} \\
(\exists inFrontOf - Generally.TaskObjects)^{\mathcal{I}} & = & \{r\} \\
(\forall toTheLeftOf.TaskObjects)^{\mathcal{I}} & = & \{b, h, t, r\} \\
(\forall toTheRighttOf.TaskObjects)^{\mathcal{I}} & = & \{b, h, t, r\} \\
(\forall inFrontOf - Generally.TaskObjects)^{\mathcal{I}} & = & \{b, h, t, r\}
\end{array}$$

OWL 2 that will be used by the ontology model construction is based on $\mathcal{SROIQ}$. Therefore we will present the features of $\mathcal{SROIQ}$, which is formed by taking $\mathcal{ALC}$ and extend it as following:

1. $\mathcal{ALC}$ extended with transitivity axioms that are expressions of the form $trans(R)$ for a role R, asserting that R is transitive, forms DL $\mathcal{S}$,

2. $\mathcal{S}$ extended with limited complex role inclusion axioms; reflexivity (Ref(R)), irreflexivity (Irr(R)), Symmetry (Sym(R)) and role disjointness (Disj(R; S)) $\mathcal{R}$ forms $\mathcal{SR}$,

3. $\mathcal{SR}$ extended with nominals (allows to build concepts from a set of individuals $\{a_1, ..., a_n\}$ whose semantics is given by $\{a_1^{\mathcal{I}}, ..., a_n^{\mathcal{I}}\}$) $\mathcal{O}$ forms $\mathcal{SRO}$,

4. $\mathcal{SRO}$ extended with inverse properties (R^-, the semantics of R^- is given by $\{(d_2, d_1)|(d_1, d_2) \in R^{\mathcal{I}}\}$) $\mathcal{I}$ forms $\mathcal{SROI}$,

5. $\mathcal{SROI}$ extended with qualified cardinality restrictions ($\geq nR.C$, the semantics of $\geq nR.C$ is $\{d_1|\#(\{d_2|(d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}) \geq n\}$ and $\leq nR.C$, the semantics of $\leq nR.C$ is $\{d_1|\#(\{d_2|(d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}) \leq n\}$) $\mathcal{Q}$ forms $\mathcal{SROIQ}$.

2.2 OWL Ontologies

In this section the relevant concepts for this work regarding OWL ontology are introduced, which are taken from (HKR$^+$09). An ontology is a conceptualisation of the domain, which separates the conceptual level from the actual data. It provides a common view of possibly heterogeneous data sources, and can be shared by different task-specific applications. DLs are used as prominent languages for writing ontologies. Complex concepts can therefore be built up in terms of simpler concepts. The most recent development in standard ontology languages is OWL from the World Wide Web Consortium (LÖ09). An OWL ontology consists of Individuals, Properties, and Classes. A short description for them is given next:

1. *Individuals*, represent objects in the domain in which we are interested. OWL does not use the Unique Name Assumption (UNA), the meaning of UNA is that two different names could refer to the same individual. Individuals are also known as instances. Individuals can be referred to as being 'instances of classes'.

2. *Properties*, are binary relations which link two individuals together. For example the property `toTheLeftOf` links the individual `Heater` and individual `Ring` and property `toTheRightOf` links individual `Base` to individual `Tray` as are shown at Table 2.2. Properties can have inverses, can be limited by having a single value (being functional), they can also be either transitive or symmetric. Properties may also have domains and ranges. In OWL there are two main types of properties namely object property and datatype property. A datatype property gives a relation

between instances of classes and RDF literals or XML schema datatypes. An object property gives the relations between individuals of two classes. Properties are known as roles in description logics.

3. *OWL classes*, are interpreted as sets that contain individuals. Classes are described using formal descriptions that state precisely the requirements for membership of the class. For example the class `TaskObjects` in Table 2.2 contains all individuals that are assembly task objects in our domain of interest. Classes may be organised into a superclass-subclass hierarchy, which is also known as a taxonomy. Subclasses are subsumed by their superclasses. For example consider the classes `TaskStates` and `HumanStates`, `HumanStates` is a subclass of `TaskStates`, which means that all human states are task states (all members of class `HumanStates` are members of class `TaskStates`). Being a `HumanStates` implies that it is a `TaskStates`. In OWL classes are constructed from descriptions that specify the conditions which must be satisfied by an individual to be a member of the class. Classes are also known as concepts in description logic.

The languages of the OWL family use the open world assumption, which means the following: if a statement cannot be proven to be true with current knowledge, it is concluded that the statement is false.

2.3 Application of Machine Learning to identify the task state

ML has been successfully used multiple times in various process optimization, monitoring and control applications in manufacturing industries (Alp14), since manufacturing industry nowadays is experiencing with a never seen increase in available data (WWIT16). It is argued that supervised learning is a good fit for most manufacturing applications due to the fact that the majority of manufacturing applications can provide labelled data (WWIT16).

ML is not only a problem which is related to databases, it is also a topic related to artificial intelligence (AI). ML helps to find solutions to many problems in robotics, vision and speech recognition. It programs machines to optimize a performance criterion using synthetic data or experience data of the past. We have a defined model, and learning is related to a program that optimizes the parameters of the model by usage of training data or past experience. The model is used to make predictions in the future, or it can be descriptive to gain knowledge, or both. ML uses the statistics theory in building mathematical models, since the task is to train a model from the given data. Additionally to predictive accuracy, the efficiency of learning or inferring information (knowledge) is of great interest, due to space and time complexity which are very related to manufacturing problems (Alp14).

Supervised ML may benefit from the generated data in manufacturing tasks related to statistical process control purposes and the fact that this data is labelled (HSSK06), this also holds for the problem which is considered in this thesis. The general process of Supervised ML is shown in Figure 2.2. The very first step is collecting the dataset, where it is very helpful to have a domain expert available which is aware of the most informative attributes. If such a support is not available then the data are taken by a brute force method which considers all the provided features.

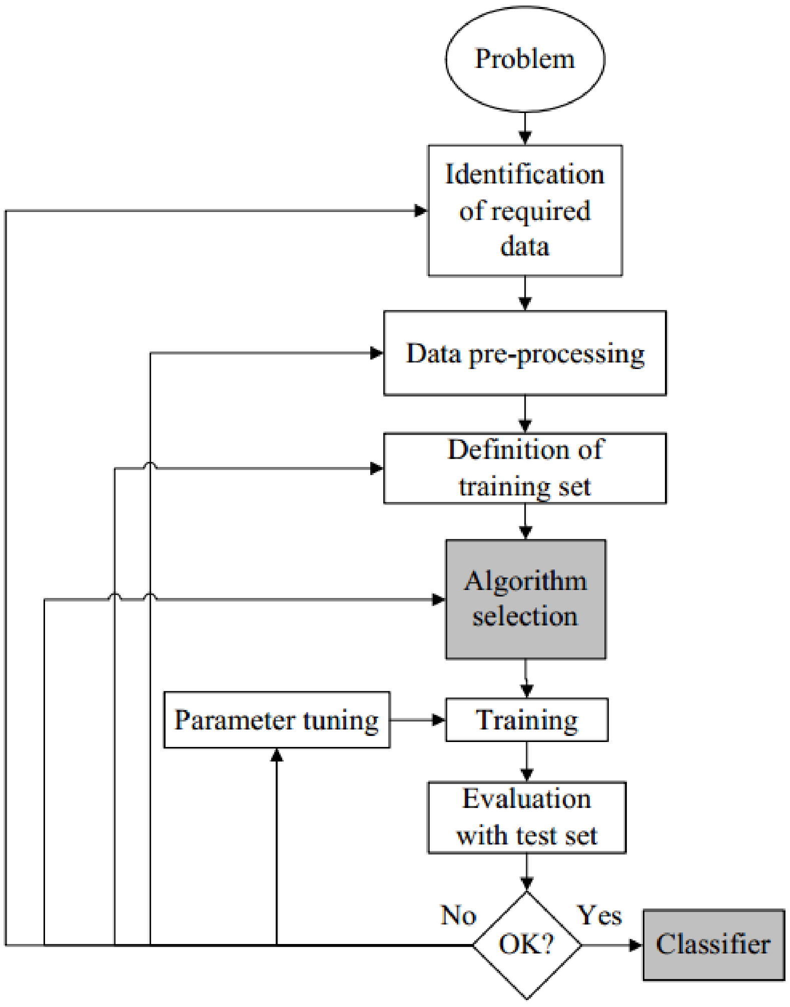


Figure 2.2: The process of supervised ML(Kot07).

But brute force methods require significant preprocessing (ZZY03). The second step is about the data preparation and preprocessing, where depending on the circumstances of application a lot of methods needs to be considered (deal with missing values, instance selection, feature selection). After the data preparation and preprocessing step is finished then a training set is defined on which the learning algorithms should be trained.

The choice of a learning algorithm is a very critical step. The classifier evaluation is usually based on the prediction accuracy. For calculating the classifier accuracy different techniques are available (Kot07).

When the ML model is learned, then the error rate is calculated and if its value is not satisfiable, we have to go to a previous step of supervised ML process. Multiple factors have to be checked. It can happen that most related features are not used, a training set is required, the selected classifier does not fit with the dataset etc.

Since the constructed task model in this work is presented by nominal values, we will recapitulate the definition of categorical data taken from (Agr06). A categorical variable has a measurement scale consisting of a set of categories. Categorical variables have two main types of measurement scales, namely: nominal and ordinal. Categorical variables having ordered scales are called ordinal variables.

Categorical variables having unordered scales are called nominal variables. There are different supervised ML algorithms which can be used, each of these algorithms has its advantages and disadvantages to a particular application as it is sketched in Table 2.4. Based on Table 2.4 and on the further properties of classifiers that can fit to our problem better, we choose four supervised ML algorithms which can handle nominal data, namely: Decision Trees, Naïve Bayes, k-Nearest Neighbors (kNN), and Support Vector Machine (SVM). For each of the classifiers a short description will be provided in the following subsections.

Naive Bayes and kNN classifiers can be easily used as incremental learners (the model's knowledge that is defined in this thesis may be extended time to time). Naive Bayes requires little storage space during both the training and classification stages: the strict minimum is the memory needed to store the prior and conditional probabilities. Decision trees classifier has a highly flexible hypothesis space, as the number of nodes (or depth) of the tree increase, decision tree can represent increasingly complex decision boundaries. Decision trees are capable of handling datasets that may have errors. With the recent growth in the amount of data model collected by assembly task experts, decision trees classifier provides methods that can deal with this problem (RM14). SVMs provide sparseness of solution when dealing with large data sets.

As the Chapter 6 gives an overview on performance evaluation between the chosen classifiers, the need to describe the concepts that are considered in the performance evaluation arises. `Precision` and `Recall` are the measures that measure how well an information retrieval system fetches the relevant requested documents. `Precision` (SW11) is defined as a ratio of true positives (TP) over the total number of positives predicted by a model, therefore it can be defined in terms of true positives and false positives (FP) as follows:

$$Precision = \frac{TP}{(TP + FP)}. \tag{2.1}$$

`Recall` (SW11) is a measure of information retrieval performance. Recall is the total number of documents retrieved that are relevant/Total number of relevant documents in the database, and it is defined as:

$$Recall = \frac{TP}{(TP + FN)}, \tag{2.2}$$

were FN is the abbreviation for false negative. `F-measure` (SW11) is a measure of information retrieval performance, and it is given as:

$$F - measure = \frac{2 * Recall * Precision}{(Recall + Precision)}. \tag{2.3}$$

2.3.1 Decision trees

A decision tree (SW11) is a tree-structured classification model that can be efficiently induced from training data. Each node in the decision tree represents a feature in an instance to be classified, and each branch represents a value that the node can assume. Leaf nodes represent events/outcomes, other nodes represent decisions. Figure 2.3 shows a decision tree sample for an assembly task states dataset, where an example of assembly task state conditions (ObjRelations, RobotState, HumanState) is considered, and the class is to define whether the conditions are appropriate to the InitialTaskState or to the FinalTaskState. To classify a new instance the work starts at the top node which is the root, and the value which is corresponding to this node is considered (in this example is $ObjRelations$). Then it moves to the descendants that correspond to a particular value of the attribute, arriving at a new node with a new attribute. This process is repeated until the classification reaches the leaf node, which is labeled with the class value, that stands for being in the initial task state or at the final task state. For all instances for which the classification reaches a particular leaf, the leaf value will be considered as the predicted class value of that instance. In Figure 2.3 leaf nodes are shown as rectangular boxes, and the inner nodes are shown as ellipses. It can happen that not all training data are used for training the classifier, and this leads to not include particular attributes in the construction of the decision tree. Attributes which are close to the root of the tree are more important. And important attributes have a stronger influence when the classification work is moving across the tree until it reaches the class values (e.g., ObjRelations will always be tested, whereas RobotStates and HumanStates will only be tested under particular conditions). In decision trees there is a concept called `pruning`, and it is used to remove nodes from the decision tree after the training has finished. Pruning is especially used when there is noisy or useless data. There are some stopping criteria on the pruning method that is called pre-pruning. The use of pruning helps to remove redundant nodes and sometimes leads to a remodeled tree.

Learning Algorithm

The decision tree is learned in a top-down fashion, the Algorithm 2.1 is known as *Top-Down Induction of Decision Trees (TDIDT), recursive partitioning, or divide and-conquer* learning. The algorithm selects the attribute from which the best split is gained as the

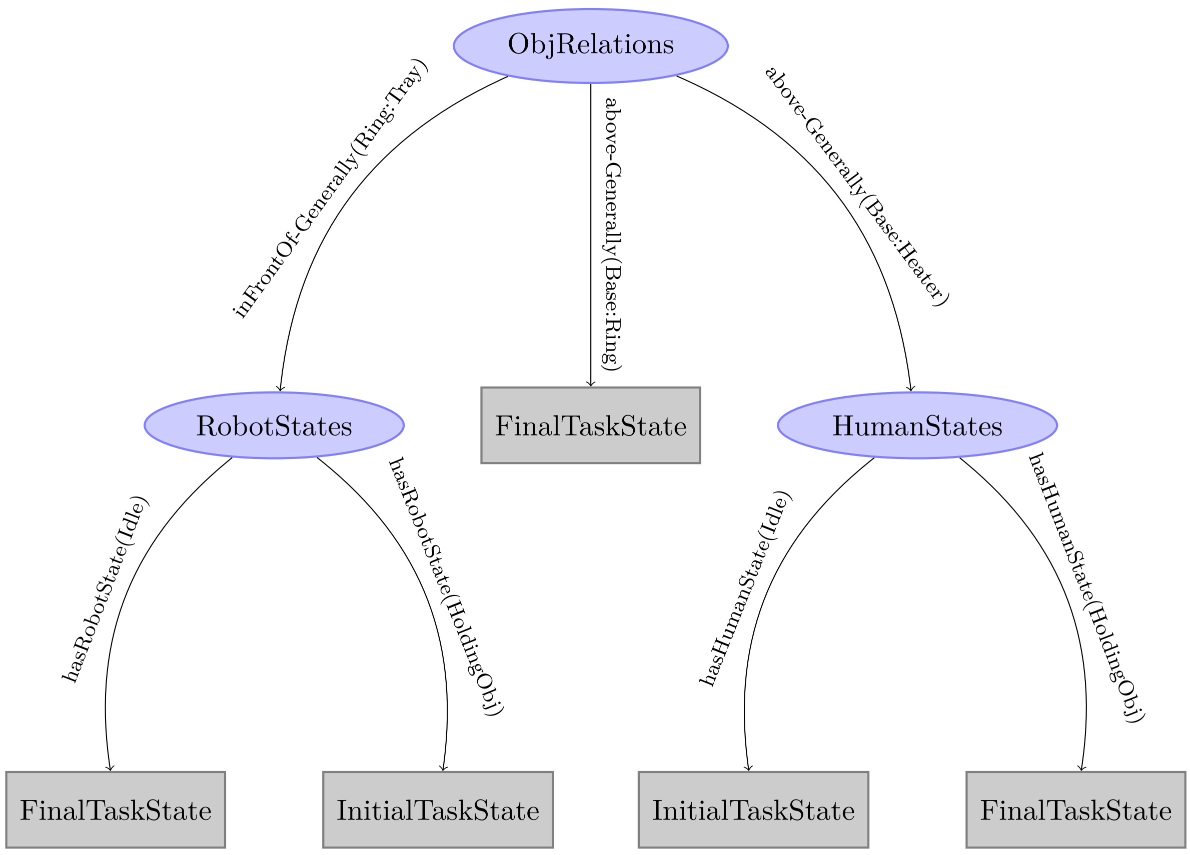


Figure 2.3: A decision tree describing the assembly task states dataset.

root node of the decision tree, splits the set of instances into disjoint sets, and adds the corresponding branches and nodes to the tree. The most simplest splitting criterion is for discrete attributes, in which each test is formed as follows:

$$t \leftarrow (A = \nu),$$

where ν is one value of the attribute A. The corresponding set S_t contains all training examples for which the attribute A has the value ν.

After the first split of the dataset is accomplished and it is partitioned according to the value of the attribute which is split on, then the procedure is recursively applied to each of the resulting datasets. A particular node is not split if all matching records have the same output value, if not then add an interior node and associate to it the best splitting attribute for the particular set where that attribute is included in, as it is described previously. Therefore, the dataset is separated into non overlapping smaller datasets up to the point were each particular sub-dataset contains samples of the same class which is called pure node. Once the decision tree is constructed it does not have to store all the

Algorithm 2.1: TDIDT(S)

```
   Input: $S$, a set of labeled examples.
   Output: $Tree$
1  $Tree$ = new empty node
2  if all examples have the same class c or no further splitting is possible then
3      // new leaf
4      $Label(Tree) = c$
5  else
6      // new decision node
7      $(A, T) = FindBestSplit(S)$
8      for each test $t \in T$ do
9          $S_t$ = all examples that satisfy $t$
10         $Node_t = TDIDT(S_t)$
11         $AddEdge(Tree \xrightarrow{t} Node_t)$
12     end
13 end
14 return $Tree$
```

training set but only use the structure of the tree, the parameters of the decision nodes and the leaves. Decision trees use the tree structure, were the leaf is found in a fast way with a small number of comparisons.

Attribute Selection

An important step in decision tree algorithm is the choice of the right attribute to perform the split. Most of the leafs contain a single training example, therefore the termination criterion in many cases is trivial to be satisfied. The tree in Figure 2.3 is a simple one and it classifies the training data correctly. Sometimes it can be observed that simple trees are more accurate than more complex trees.

A typical criterion takes use of a function that measures the impurity of a node. Two impurity measures are:

1. Entropy(Qui86),
$$Entropy(S) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \cdot log_2 \left(\frac{|S_i|}{|S|} \right) \tag{2.4}$$

2. Gini index (BFOS84),
$$Gini(S) = 1 - \sum_{i=1}^{c} \left(\frac{|S_i|}{|S|} \right)^2 \tag{2.5}$$

S is the set of training examples, S_i is a subset of training examples that corresponds to class c_i. Both equations reach the maximum when the classes are equally distributed, and they reach the minimum when a particular S_i contains all examples s.t. $|S_i|/|S| = 1$. A good split divides the data set into subsets that are as pure as possible, the ideal case is when all the subsets contain examples of only one class. High entropy and high Gini implies that their value has to be reduced in order to achieve a better prediction. To achieve a better prediction the so-called *gain* is used,

$$Gain(S, A) = Impurity(S) - \sum_t \frac{|S_t|}{|S|} \cdot Impurity(S_t) \tag{2.6}$$

which is the amount by which the original impurity can be reduced by splitting into subsets. The value of *gain* ranges from at least zero up to the impurity value. A stands for an attribute, and t is a test on A that partitions the set S into non overlapping disjoint subsets S_t. The term *Impurity(S)* is a constant that is considered for all attributes.

2.3.2 Naïve Bayes

Naïve Bayes classifier treats the classification problem in terms of probabilities and the definitions are taken mainly from (SW11). Naïve Bayes is a simple learning algorithm that utilizes the Bayes rule together with a strong assumption that the attributes are conditionally independent, given the class. Naïve Bayes classifier has three main concepts, namely: conditional probability, Bayes Theorem, and the Bayes decision rule.

1. The conditional probability $P(E|H)$ is used to define independent events (MM07), is given by:

$$P(E|H) = \frac{P(E \cap H)}{P(H)}, \tag{2.7}$$

   where $P(E|H)$ is the probability of event E given evidence H. In the same way we define $P(H|E)$:

$$P(H|E) = \frac{P(H \cap E)}{P(E)}, \tag{2.8}$$

   where $P(H|E)$ is the probability of event H given evidence E. It follows that:

$$P(H \cap E) = P(E)P(H|E). \tag{2.9}$$

2. The Bayes Theorem starts with an initial degree of belief that an event will occur, and then with new information this degree will be "updated" (MM07). These degrees are represented by the prior probability event H, $P(H)$ (probability of event before evidence is seen) and posterior probability of H, $P(H|E)$ (probability of event after evidence is seen), whose relation is given by :

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}. \tag{2.10}$$

3. The Bayes decision rule states that based on the posterior probabilities, it is possible to assign an element x to a class that has the largest probability.

Then Naïve Bayes for classification is used to answer the question "what is the probability of the class given an instance?" where instance is Evidence E and the class value for instance is the event H. Naïve Bayes for classification uses the naïve assumption which states that evidence is split into parts (i.e. attributes) that are independent. And Naïve Bayes for classification is defined as follows:

$$P(H|E) = \frac{P(E_1|H)P(E_2|H)...P(E_n|H)P(H)}{P(E)}. \tag{2.11}$$

Naïve Bayes's desirable properties:

- Computational effciency: Training time is linear with respect to both the number of training examples and the number of attributes, and classification time is linear with respect to the number of attributes and unaffected by the number of training examples.
- Low variance: It does not utilize search but at the cost of high bias.
- Incremental learning: Naïve Bayes operates on estimates of low order probabilities that are derived from the training data. These can readily be updated as new training data are acquired.
- Direct prediction of posterior probabilities.
- Robustness in the face of noise: Naïve Bayes always uses all attributes for all predictions and hence is relatively insensitive to noise in the examples to be classified, also it is insensitive to noise in training examples.
- Robustness in the face of missing values: Because naïve Bayes always uses all attributes for all predictions, if one attribute value is missing, information from other attributes is still used, resulting in graceful degradation in performance.

2.3.3 k-Nearest Neighbors

This section is based on the definitions in (DHS00). The k-Nearest Neighbors (kNN) classifier classifies a given instance x by assigning to it the most frequently class of the k nearest samples, in other words by examining the classes of the k nearest neighbors and by majority vote a decision is made to assign the class to x. In Figure 2.4 it is illustrated a two-class problem with an odd value of k (to avoid ties), as can be seen in the case were $k = 1$ and $k = 5$ the class for x is *blue*, and when $k = 3$ the assigned class for x is black.

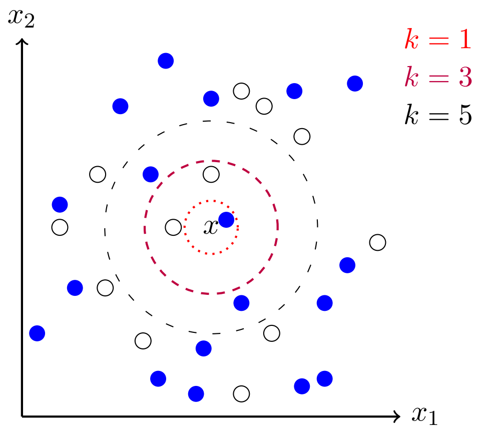


Figure 2.4: Polynomial projection.

The kNN classifier is very sensitive to noise and provides no generalisation. The effect of noise is reduced by taking large values of k, because it makes boundaries between classes less distinct. If $k = 1$, kNN is the same as the nearest neighbour algorithm. The value of k has to be specified by the user and the best choice depends on the data. The value of k can be arbitrary increased when the training data set is large in size. It can handle non linear separation and no training time is needed which means that computation is made on the classification step. Memory consumption depends on the training data. kNN is computationally expensive when it is faced with many items to classify.

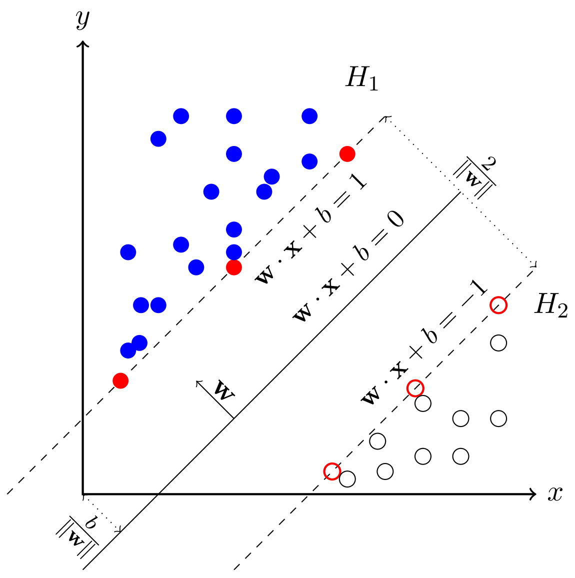
2.3.4 Support Vector Machine

The content of this section is based on (SW11). Support Vector Machine (SVMs) are part of linear algorithms that are used for classification, regression, and other applications. When a two class classification problem is considered, the work of SVMs is to find a hyperplane that separates the data in two classes with as wide as possible margin. Margin is the width that the boundary between the classes could be increased to, before hitting any data-point as shown in Figure 2.5. This method offers a good generalization accuracy on unknown data. The funded hyperplane is based on the data points lying in the margin that are called support vectors. In most cases of the real data linear separation is not possible, but SVMs can be extended to handle this nonlinearity by the use of kernels. SVMs can be trained by quadratic programming that:

1. makes theoretical analysis easier, and
2. provides much convenience in designing efficient solvers that scale for large datasets.

In real-world data, SVMs often have performance in accuracy, flexibility, robustness, and efficiency. To find the optimal hyperplane for separating the data points in two classes, an example is considered where the training set is $\{(x_i, y_i)\}_{i=1}^n$, x_i is the input feature vector for the $i-$th example that belongs to a binary class $y_i \in \{1, -1\}$, the example is positive ($y_i = +1$) or negative ($y_i = -1$). We assume that the set of positive and negative examples are linearly separable, therefore there exists a function $f(x) = \langle w, x\rangle + b$ (w is the weight vector, and b determines offset from origin (intercept/bias)) such that:

$$\langle w, x_i\rangle + b > 0 \text{ for } y_i = +1$$
$$\langle w, x_i\rangle + b < 0 \text{ for } y_i = -1.$$



y
x
H_1
H_2
$\mathbf{w}\cdot\mathbf{x}+b=1$
$\mathbf{w}\cdot\mathbf{x}+b=0$
$\mathbf{w}\cdot\mathbf{x}+b=-1$
$\mathbf{w}$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$

Figure 2.5: Largest Margin.

The $\langle w, x_i\rangle + b = 0$ is called decision plane and there can be multiple such planes that separate class 1 and class -1, see Figure 2.5. The aim of SVM is to find a particular hyperplane which maximizes the margin. Mathematically, it is easy to check that the distance from a point x_i to a hyperplane $\langle w, x_i\rangle + b = 0$ is $||w||^{-1}|\langle w, x_i\rangle + b|$. SVM is looking for the optimal w, b of the following optimization problem:

$$\underset{w\in\mathbb{R}^p, b\in\mathbb{R}}{maximize} \ \underset{1\leq i\leq n}{min} \frac{|\langle w, x_i\rangle + b|}{||w||}, \ s.t. \begin{cases} \langle w, x_i\rangle + b > 0 & \text{if } y_i = +1 \\ \langle w, x_i\rangle + b < 0 & \text{if } y_i = -1 \end{cases} \ \forall i. \tag{2.12}$$

Therefore, to fix the scale, the numerator of the objective $\underset{1 \leq i \leq n}{min} \; |\langle w, x_i \rangle + b|$ is equivalently set to 1, and minimize the denominator $||w||$:

$$\underset{w \in \mathbb{R}^p, b \in \mathbb{R}}{minimize} \; ||w||^2, \;\; s.t. \begin{cases} \langle w, x_i \rangle + b \geq 1 & \text{if } y_i = +1 \\ \langle w, x_i \rangle + b \leq -1 & \text{if } y_i = -1 \end{cases} \quad \forall i. \tag{2.13}$$

This constrained quadratic program can be solved efficiently. Hence it becomes the most commonly used form of SMV for linear separable case. Some properties of SVM classifier are listed below:

- A SVM may be viewed as a binary classifier. It abstracts a linear decision boundary from the data and uses it to classify unknown data belonging to the two classes.
- SVMs learn from data, by considering the maximum constructed margin.
- If two classes are linearly separable, then it does not have to implement any additional feature, it just computes the corresponding maximum margin by given data and uses it.
- To have a better generalization model, SVM uses soft margins which will be introduced in 2.3.4.
- If the data are not linearly separable, the SVM classifier uses Kernel function to deal with this problem as introduced in 2.3.4.
- If the data contain more than two classes, then the SVM classifier reduces them to multi binary problems as will be shown in 2.3.4.

Soft Margins

Sometimes linear separation is not possible, or it can happen that linear separation would lead to a badly generalising model. This is also applied to the application scenario on this master thesis, since big assembly tasks can have a very large number of assembly task states, therefore the linear separation might not be possible. To avoid such cases the usage of soft margins can be considered whose main goals are that to construct a hyperplane that splits "as cleanly as possible/desirable" and to maximise the margin value. As it is shown in Figure 2.6 the cyan circle (○) is not considered as -1 class because the idea is to have a better generalization model. The constraints in 2.13 can be equivalently written as $y_i(\langle w, x_i \rangle + b) \geq 1$. An introduction to the slack variables is made, their goal is to penalise misclassification and adapts constraints $y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i$, and incorporate a penalty into the original objective to derive the soft margin SVM:

$$\underset{w, b, \xi_i}{minimize} \; \lambda \, ||w||^2 \; + \frac{1}{n} \sum_{i=1}^{n} \xi_i \;\; s.t. \; y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i, \; and \;\; \xi_i > 0 \;\; \forall i. \tag{2.14}$$

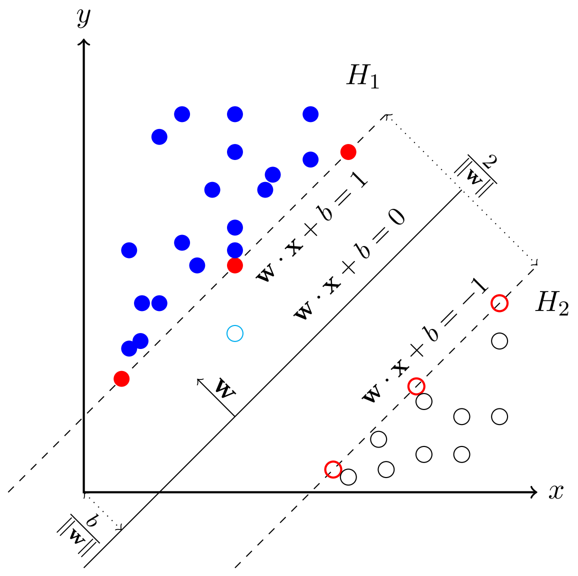

y
H_1
$\frac{2}{\|\mathbf{w}\|}$
$\mathbf{w} \cdot \mathbf{x} + b = 1$
$\mathbf{w} \cdot \mathbf{x} + b = 0$
$\mathbf{w} \cdot \mathbf{x} + b = -1$
H_2
$\mathbf{w}$
x
$\frac{b}{\|\mathbf{w}\|}$

Figure 2.6: Soft Margin.

$\lambda > 0$ is a trade-off factor, and ξ_i can be written as $\xi_i = max\{0,\ 1 - y_i(\langle w, x_i \rangle + b)\}$, which is called hinge loss and is depicted in Figure 2.7. Using soft margins does not mean to have necessarily 100% classification accuracy on the training set but it is used to achieve a better generalization model.

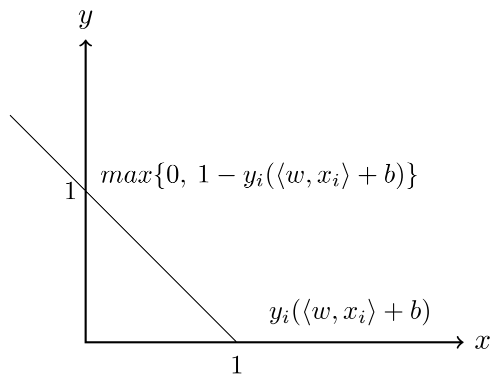
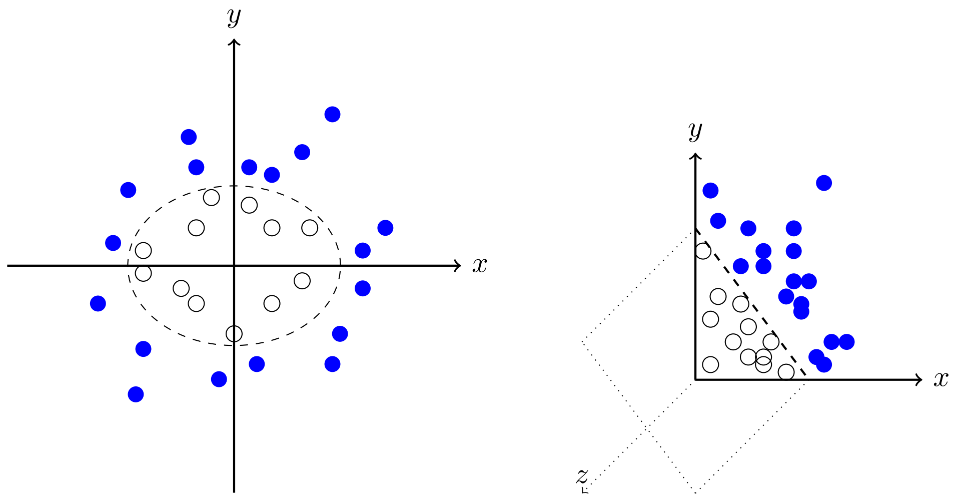


Figure 2.7: Graph of hinge loss (Qui86).

Non linear separation

SVMs so far have been presented as a classifier for linearly separable data with the addition of slack variables. However, some data can not be linearly separated, then the idea of SVMs is to project the data into a higher dimensional space with the assumption that projection of data may be linear separable in this space.



(a) Original data. (b) Polynomial projection.

Figure 2.8: Projection from two dimensional input space with non linear separable classes into a linear separable feature space.

Projection in the higher space is achieved as the multiplication result of vectors by the kernel matrix, were the kernel matrix gives the shape of possible separators. Figure 2.8 depicts a projection that maps a two dimensional input space with non linear separable classes (Figure 2.8a) into a linear separable feature space (Figure 2.8b). Some common kernels are: quadratic, polynomial, radial basis function sigmoid etc. To select the best kernel function, it requires a check of all functions for a particular data that might faced with. So the choice of the kernel function is made by taking the model and train it with different kernels and pick the best performing one. Working with high dimensional data is computationally expensive, but a SVM depends only on the dot product between vectors. It uses the kernel trick to replace the dot product by the kernel function, this is computationally inexpensive.

Multi class SVM

The standard SVM classifier is used for binary problems (i.e. two classes). Multi class problems deal with three or more classes. SVMs can be used for multi class problems by reducing them to multi binary problems, like described below:

1. One of the labels to the rest (one versus all), where a binary classifier is used that distinguishes between a particular class i and the rest ($i = 1, ..., \#numClasses$). The classifier with highest output function is the winner that is done by using a winner-takes-all strategy.
2. Between every pair of classes (one versus one), again a binary classifier for each pair of classes is made. Classification is done by a max-wins voting strategy, in

which every classifier assigns the instance to one of the two classes. Then the vote for the assigned class is increased by one vote, and finally the class with most votes determines the instance classification.

	Decision Trees	Neural Networks	Naïve Bayes	kNN	SVM	Rule-learners
Accuracy in general	**	***	*	**	****	**
Speed of learning with respect to number of attributes and number of instances	***	*	****	****	*	**
Speed of classification	****	****	****	*	****	****
Tolerance to missing values	***	*	****	*	**	**
Tolerance to irrelevant attributes	***	*	**	**	****	**
Tolerance to redundant attributes	**	**	*	**	***	**
Tolerance to highly interdependent attributes (e.g.parity problems)	**	***	*	*	***	**
Dealing with discrete/binary/ continuous attributes	****	***(not discrete)	***(not continuous)	***(not directly discrete)	**(not discrete)	***(not directly continuous)
Tolerance to noise	**	**	***	*	**	*
Dealing with danger of overfitting	**	*	***	***	**	**
Attempts for incremental learning	**	***	****	****	**	*
Explanation ability/ transparency of knowledge/ classifications	****	*	****	**	*	****
Model parameter handling	***	*	****	***	*	***

Table 2.4: Comparing learning algorithms (**** stars represent the best and * star the worst performance) (Kot07).

CHAPTER 3

Implementation Environment

In the last chapter we have summarised the basic concepts that are needed in the accomplishment of our goal. This chapter provides a short overview about the topic of this thesis and analyzes existing notions and systems concerning knowledge processing and presentation. These knowledge processing and presentation systems are based on the concepts introduced in the previous chapter. A Knowledge Processing Infrastructure for Cognition-enabled Robots (KnowRob) is used which employs DLs as a formalism to represent encyclopedic and common sense knowledge, in particular the OWL. Approaches and systems will be introduced, that are targeting similar goals in the domain of cognitive, assistive, robotic systems and human robot collaboration.

3.1 KnowRob knowledge processing system

The robotic system at Profactor uses KnowRob as a processing tool, thus our reasoning system is designed and implemented on top of KnowRob. Therefore an overview about the KnowRob system is needed, the overview is based on (TB13). KnowRob is a knowledge processing system and it is implemented in SWI-Prolog (WSTL12). SWI-Prolog is also used as central knowledge store. In KnowRob, all knowledge is represented in the Web Ontology Language (OWL) (LÖ09). OWL is a XML-based format that allows to formally describe relational knowledge in a Description Logics (DLs) dialect. To formally model the knowledge, it is very useful to distinguish between general relations and environment-specific information. In OWL, this is reflected by the distinction between classes and instances. Class knowledge is described in the so-called TBox, knowledge about instances of these classes is contained in the ABOX. The relation between classes and instances is similar to object-oriented programming. In the KnowRob system, Prolog is used for loading, storing and reasoning on the knowledge base which is represented in OWL. Content and functionality of KnowRob can be extended to different modules. The system structure is given at Figure 3.1. The central component of KnowRob is the knowledge base that provides the mechanisms to store and retrieve information about actions, objects, processes, temporal events, their properties, and relations.

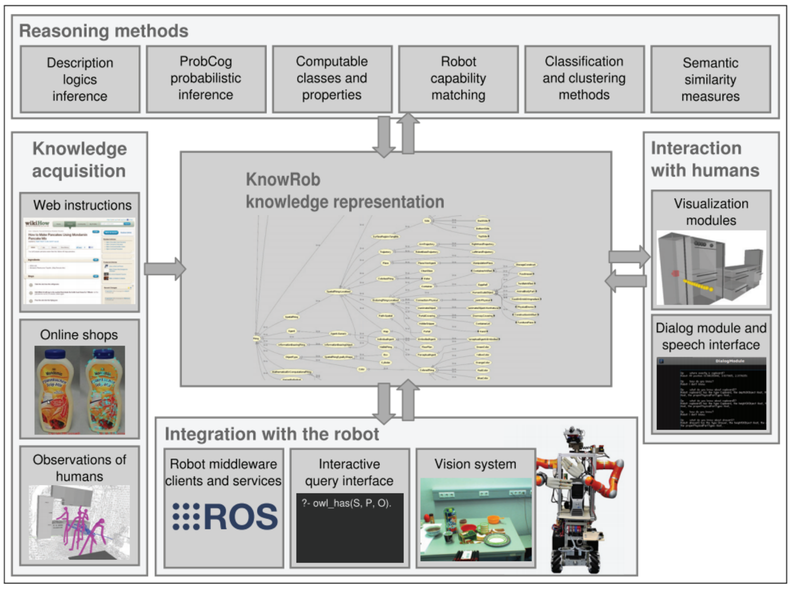

Reasoning methods
Description logics inference
ProbCog probabilistic inference
Computable classes and properties
Robot capability matching
Classification and clustering methods
Semantic similarity measures
Knowledge acquisition
Web instructions
Online shops
Observations of humans
KnowRob knowledge representation
Interaction with humans
Visualization modules
Dialog module and speech interface
Integration with the robot
Robot middleware clients and services
ROS
Interactive query interface
?- owl_has(S, P, O).
Vision system

Figure 3.1: The KnowRob system provides several components for knowledge acquisition and representation, for reasoning about this knowledge, and for grounding it in the robot's perception and action system (TB13).

The KnowRob ontology is used in this system and presents the general vocabulary and representation into which other representations and inference methods can be incorporated. The core system is able to be extended by using extensions for a particular functionality.

3.2 Knowledge Processing for Autonomous Personal Robots

In order to obtain an ontology model that represents task states in our work, an approach which is focused on knowledge representation and processing for autonomous personal robots is considered.

In this approach which is proposed by Tenorth and Beetz (MM09) the knowledge is modeled in description logics using OWL, they use both modeling levels of DLs, namely: concepts and instances. Classes contain terminological knowledge like: events, types of objects and actions that are organized in a taxonomic way. Instances contain concrete and physical objects or actions that are performed actually. The relations are represented in triples (Subject, Predicate, Object).

This approach is used by the Autonomous Personal Robots to achieve a more flexible and general behaviour and better performance.

This system integrates encyclopedic knowledge, environment model, human observations and action based models. In this system queries of continuous data observed in a real-world environment can be applied. The concept of computable classes and properties that are used to create instances from observed data and action models is introduced. Components and mechanisms introduced in this work are applied in a practical knowledge processing system that is designed for autonomous robots. The main contributions of this work are:

1. It introduces a complete knowledge processing system that combines encyclopedic knowledge, an environment model, action-based reasoning, and human observations. This information is accessed in a symbolic and uniform way.
2. The system offers symbolic queries in the continuous data received by the sensor in real-world environment.
3. It introduces computable classes and properties that are used to create instances from observed data and action models as a powerful means for discovering a class structure among action-related concepts.

3.3 Unified Representation for Reasoning

In the modeled ontology, we developed a class that represents the object relations and a class that represents robot states of an assembly task execution. The development of these two classes is inspired by Tenorth and Beetz who give a unified representation for reasoning about robot actions, processes and their effects on objects (TB12). They provide a way how the robot can be equipped with sufficient knowledge when there might be some knowledge gaps in the action descriptions. They give a system that integrates several sources of knowledge and combines them for filling knowledge gaps in instructions for manipulation tasks. A representation called "object transformation graph" is given that semantically describes how objects are transformed during a task and allows to reason about these transformations. They present methods to project the results of processes and actions, for including processes into the action planning procedure. They illustrate their approach by showing how a task description that can be generated by natural-language instructions (which leads to incomplete task descriptions) can be completed by the proposed methods. The approach gives from where the additional information that makes task completion possible is obtained from.

3.4 RoboEarth Action Recipe Execution

Since in our contribution the need for a base ontology model that can be further extended to model a particular assembly task is crucial, the idea of a globaly base model comes from Marco, Tenorth, Häussermann, Zweigle and Levi (dMTH$^+$12) who present an approach which proposes a way how to reuse task execution plans by introducing the concept of a globally accessible database. In this approach knowledge reasoning and processing are essential for planning and decision making. The globally accessible database stores: action recipes, object detection models, navigation maps, etc. Generated data can be taken from humans, robots or processor programs which are running in the database. To interpret and execute an abstract task description they have developed an execution engine which serializes and triggers all command executions, supervises them, manages incoming events and handles failure states. Every file of this database is annotated with an OWL description for the sake of allowance of semantic queries. Recipes are stored in description logic. This approach takes into account the robot capabilities for recipes stored in the globally accessible database and checks if the robot is appropriate for a specific recipe. In this approach KnowRob is used as a knowledge processing framework which is able to access the global database and the world model as well that is used for answering object positions queries. The execution engine makes requests to the KnowRob framework for retrieving information like plans and object positions. In order for KnowRob framework to have a generated plan, it queries the database and gives a matching task plan to the execution engine. KnowRob is used as an interlingua framework to exchange knowledge between different components in the RoboEarth system. When the execution engine sends a query request to KnowRob for a particular recipe, then KnowRob queries the global shared database and downloads the recipes, checks if there are missing components and then triggers their downloads. Our developed ontology model can serve as a global model in a HRC with respect to assembly tasks. Their approach can be further extended by taking our contribution into account, and have an ontology model for each assembly task description.

3.5 Human activity analysis

The developed ontology model in our work must represent the human activities, therefore a class that expresses the human states is constructed. The motivation in contructing this class comes from an approach that represents human activity analysis (AR11). In this approach Aggarwal and Ryoo provide a complete overview of the state-of-the-art of human activity recognition methodologies. Here various types of methodologies for recognition of different levels of activities are discussed. They are focused on simple human actions and on high-level activity recognition methodologies designed for the analysis of human actions, interactions, group activities, and discuss recent research trends in activity recognition.

3.6 Mental Model and Shared Grounds in Human-Robot Interaction

Mental models and shared grounds in HRC must be taken in consideration when constructing the ontology model. To do so we have treated an investigation by Hwang, Lee, and Kwon at (HLK05). They have investigated the role of mental model and shared grounds in Human-Robot interaction in order to achieve an efficient collaboration between human and robot. They conceptualize a three way relationship through human, robot and world, and connect them in terms of construction of shared grounds. They develop a mediate interface which provides a communication and a coordination between the human and the robot by using multimodal icon symbols. By using this mediate interface they carry out a pilot experiment that is used to evaluate the role of a mental model in constructing the shared grounds. They have shown that a mental model helps users to understand what a robot is able to do, which commands need to be used to operate the robot, how the robot works, and determine how users interact with the robot. They have shown that these mental models have a big effect in constructing shared grounds between a human and a robot.

CHAPTER 4

Modeling the assembly task description in OWL

In the preceding chapter we have analysed existing notions and systems concerning knowledge processing and presentation, this analysis is mainly considered on the development of the ontology learning model. In this chapter, we introduce the learning model of the assembly task. The main goal is to provide a core ontology assembly model that can be imported and extended for different assembly tasks. This step is crucial for interpreting the task model in an appropriate manner in order to identify the state of a task during the task execution by applying ML classifiers. The concept of this constructed model incorporates object relations, human states, robot states and the time stamp on which a specific task state is identified. Since KnowRob will be used as a knowledge processing system where the knowledge is represented in OWL, the learning task model has to be represented in OWL as well. The ontology model was constructed using Protégé (Mus15), which is a feature rich ontology editing environment with full support for the OWL 2, and direct in-memory connections to description logic reasoners. An assembly task execution sequence is formed by a set of states as shown in Figure 4.1. On the constructed ontology model each task state must have information about a robot state, a human state, object relations and a time stamp that corresponds to the time on which the task state is asserted in the model.

The natural way of the assembly task execution sequence will be used in order to give the constructed ontology task model. As mentioned in (HKR^{+}09) the proposed ontology assembly task model will also have individuals, properties and classes, which are going to be presented in the following sections of this chapter. To have a better understanding of the constructed ontology model, we describe the developed ontology model for the Steam Cooker assembly scenario that is realized at Profactor GmbH which is depicted in Figure 1.1.

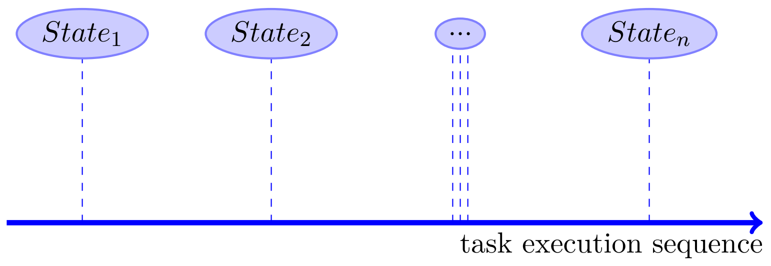


Figure 4.1: Assembly task execution sequence formed by states.

4.1 Modeled classes

Now we introduce the constructed classes of one ontology task model. OWL classes are interpreted as sets that contain individuals. The proposed ontology assembly task model defines five different classes that will contain, robot states, human states, time stamps, task objects and task states. Each defined class will be explained in the following subsections. The classes will be represented by circles by help of Venn diagrams (ven). The base class of all ontology classes is the class `Thing`.

4.1.1 Robot States

The given robot states are represented as individuals and they are grouped together in a class called *RobotStates*.

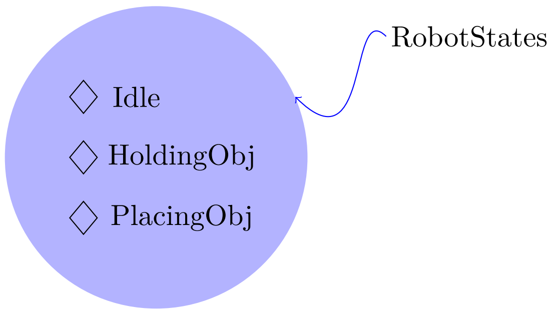


Figure 4.2: Representation of RobotStates class.

Figure 4.2 gives an abstraction of the *RobotStates* class formed by three individuals namely *Idle*, *HoldingObj*, *PlacingObj*.

4.1.2 Human states

Human states are represented as individuals and all of them are part of a class that is called *HumanStates*.

Figure 4.3 gives an abstraction of the *HumanStates* class formed by three individuals namely *Idle*, *HoldingObj*, *PlacingObj*. In Figure 4.3 the defined individuals are the same

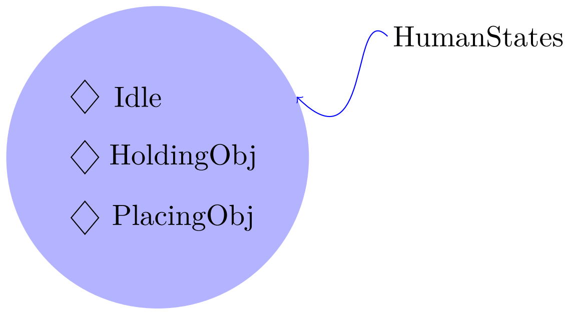


Figure 4.3: Representation of HumanStates class.

as the robot states individuals, but this does not mean that the robot states have to be the same as the human states.

4.1.3 Time Stamp

The *TimeStamp* class covers all time stamp individuals of an assembly task. Each task state is assigned exactly one time stamp individual. A time stamp individual gives the number of seconds between a particular date and the Unix Epoch (`Thursday, January 1, 1970, 00:00 UTC`). For example individual `1484736563.1557156` is equivalent to `Wednesday, 18-Jan-17 10:49:23 UTC`.

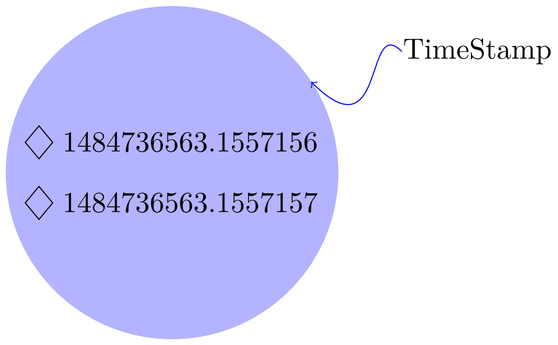


Figure 4.4: Representation of TimeStamp class.

An abstraction of the *TimeStamp* class is given in Figure 4.4.

4.1.4 Objects of the task

To collect all the objects that are participating in a specific assembly task, a class is constructed for this reason and it is named *TaskObjects*. Figure 4.5 illustrates an example of *TaskObjects* class used in a steam cooker assembly task.

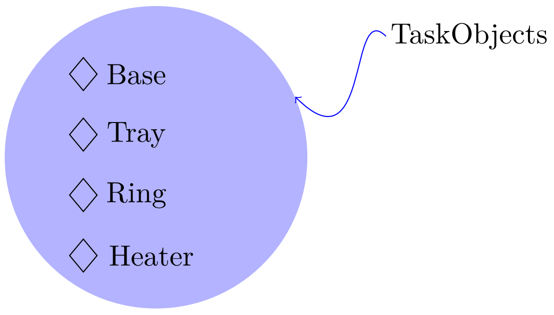


Figure 4.5: Representation of TaskObjects class.

4.1.5 Task States

In this subsection the task state concept and its development is illustrated by using a class-instance relation as introduced in Section 3.1. The class *TaskStates* in the ontology task model is the class that will include all the states of an assembly task. Task states that represent an assembly task description are defined as subclasses of *TaskStates* class. Based on the location of objects and relations between them, the same task state could have various numbers of configurations that represent object relations which have to be defined as different instances of the task description model. Each configuration of objects together with the human state, robot state and time stamp will be defined as an instance of a particular state class that represents them. Figure 4.6 sketches an abstraction of the *TaskStates* class that has a sub class which is called *InitialTaskState*. *InitialTaskState* class is one of the task states that is used to construct the learning model. As described above, a task state can have more than one configuration between objects (e.g. in one configuration of initial state Obj_1 is in front of Obj_2 and in another configuration of initial state Obj_1 is to the left of Obj_2), to handle this situation it should be constructed a sub class of that task state for each such configuration. For example Figure 4.6 illustrates this idea by *FCOfInitialTaskState* class that stands for first configuration of the initial task state. Since Prolog is used in KnowRob, and a program in Prolog consists of one or more predicates, the object relations are defined as predicates. In first-order theories, predicates are often associated with sets. For each positive integer n, a denumerable list of symbols called $n-ary$ predicates, or predicates of degree n (Smu95). In Prolog, the arity of a predicate is the number of arguments that the head of the clause has.

FCOfInitialTaskState class which is a subclass of *InitialTaskState* class is created by:

1. Instances (i.e. $FCOfInitialTaskStateObjRel_1$ in Figure 4.6 that stands for the first object relation of the first configuration of the initial task state) representing the object relations. For each task state there must be $n(n-1)/2$ such relations as explained in Section 4.3.4 therefore $n(n-1)/2$ instances, n is the number of objects of the task. To define such relations (role assertions in DLs dialect), properties as listed below, are used:

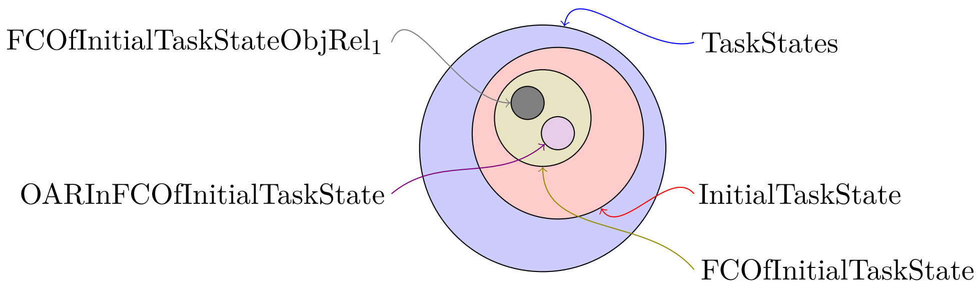


Figure 4.6: Representation of TaskStates class.

a) one object property (i.e. inFrontOf), which will be used as a predicate of arity two to represent the object relation, i.e. *inFrontOf(Obj$_1$, Obj$_2$)* which means that Obj$_2$ is in front of Obj$_1$.
b) and two sub-properties of the object property 1a, namely:
    i. lhsObject, which is an object property and will be used to take the value of the first argument of its super-property, i.e. *lhsObject(Obj$_1$)*,
    ii. rhsObject, which also is an object property and will be used to take the value of the second argument of its super-property, i.e. *rhsObject(Obj$_2$)*.

Such an instance in the Protégé environment is defined as follows:

a) in the equivalent class section the object relations are defined as:

```
(lhsObject value Heater)
and (rhsObject value Ring)
and (toTheLeftOf value Heater)
and (toTheLeftOf value Ring)
```

   This defined instance is formulated on a steam cooker assembly ontology model.
b) In the subclass section the corresponding name of its super-class should be placed, an example is:

```
FCOfInitialTaskState
```

2. And another instance (i.e.*OARInFCOfInitialTaskState* in Figure 4.6 that stands for the one arity relation (a set) in the first configuration of the initial task state) which will be described only by unary predicates (concept assertions). This instance will be used to represent human state, robot state and timestamp for this particular state configuration, an example of such an instance is given as follow:

```
(hasHumanState value Idle)
 and (hasRobotState value Idle)
 and (hasTimeStamp value 1478854453.4594500)
```

4.2 Modeled individuals

OWL allows the definition of individuals and to assign properties about them. Assume that human states of a task have to be specified, first all the human states have to be asserted to the constructed ontology. For example, let us say that human states are: 'Idle', 'HoldingObj', 'PlacingObj' and all of them have to be asserted into the ontology model. To do so, the previously created *HumanStates* class is populated with the given individuals. The individual insertion is in the same manner for the *RobotStates*, *TaskObjects* and *TimeStamp* classes.

4.3 Modeled properties

Based on (HKR$^+$09) it is known that there are two main types of properties, namely: Object properties and Datatype properties. The properties in the proposed ontology model will be mainly object properties since object properties link individuals of different classes. There is also another property which is called Annotation property, and it will be used to provide metadata about classes, individuals and object properties. In the developed ontology model we have used OWL object properties in order to represent relationships by considering human states, robot states, time-stamp and object relations, these relations are explained in the following subsections.

4.3.1 Human state property

An object property is defined which is called *hasHumanState*. Its goal is to provide information about a human state to the task state when the assembly task state is defined by the assembly task expert. The range of *hasHumanState* object property will be the class *HumanStates*.

4.3.2 Robot state property

Also another object property is asserted in the ontology which is called *hasRobotState*. Its goal is to provide information about the robot state to the task state when it is defined by the assembly task expert. The range of *hasRobotState* object property will be the class *RobotStates*.

4.3.3 TimeStamp property

Learning the assembly task model can be extended from time to time and the provided results of classifiers at a latter point in time will change for sure as the task model will be updated. To keep track on classifier results and also on the updated versions of the ontology model, an object property is constructed that is called *hasTimeStamp*, its goal is to timestamp each state of the assembly task model and its value will be the time when the task state is constructed (the task state's timestamp has to be defined as an individual at *TimeStamp* class). The range of *hasTimeStamp* will be the class *TimeStamp*.

4.3.4 Object Relations

We have given the OWL properties that represent human states, robot states and timestamp, to complete our job in defining the OWL object properties we are left with the property that will represent object relations. This is accomplished in this subsection. To represent a relation between two objects the naming conventions as they are given in KnowRob are used (i.e. aboveOf, belowOf, inCenterOf etc). An assembly task can have a various number of objects that have to be dealt with. And the way how the object relations concept is constructed comes from the definition of complete graphs.

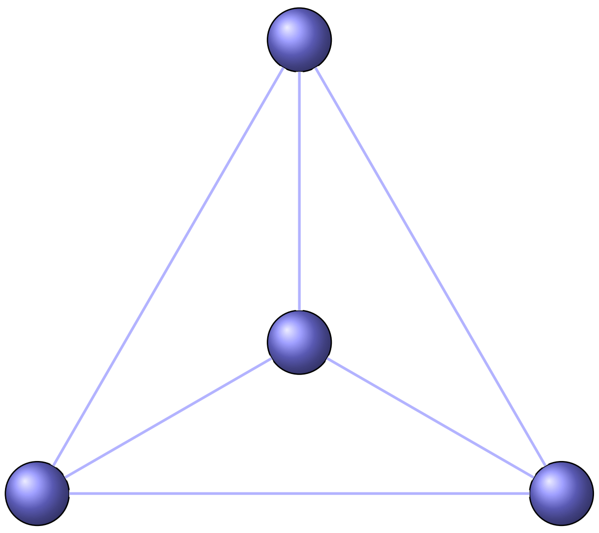


Figure 4.7: A complete graph example.

A complete graph is a simple graph in which any two vertices are adjacent (BM07). A complete graph formed by n vertices is denoted by K_n, and K_n has $n(n-1)/2$ edges. Figure 4.7 shows an example of a complete graph formed on 4 vertices (K_4). Based on the definition of a complete graph it is made a mapping from objects that are used in the assembly task to the vertices in the complete graph and a mapping that maps the relation between objects in the assembly task to edges on the complete graph. Therefore this formulation implies that there have to be $n(n-1)/2$ object relations in the assembly task, where n is the number of objects of the assembly task. To illustrate the idea of mapping it is constructed a complete graph that represents the relation of objects in the steam cooker assembly task as sketched in Figure 4.8. In a steam cooker assembly task there are four objects namely: Base, Tray, Ring and Heater. Hence there must be six relations between these four objects as shown in Figure 4.8. The relation between the assembly task objects are defined as object properties in Protégé, which represents the relationships between two individuals (in the assembly task case it represents the relationships between two objects of the assembly task).

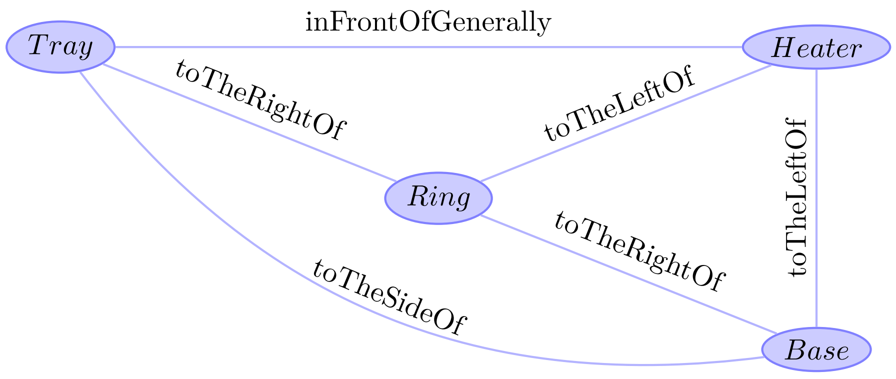


Figure 4.8: A complete graph representing object relations in the steam cooker assembly task.

Figure 4.9 depicts the hierarchy of object properties that will represent the relations between objects in an assembly task. The naming conventions for such relations are taken from the KnowRob *comp_spatial* (TB13) package, in order to have the same names with the data perceived from the KnowRob system. As it was shown in Section 4.1.5, two sub-properties (lhsObject, rhsObject) of object relation properties are presented in the hierarchy of the object relations. The range of all these object properties is the *TaskObjects* class. Object property *hasObjectRelation* is defined as a super-property of all object properties that will be needed to describe the object relations in the assembly task model.

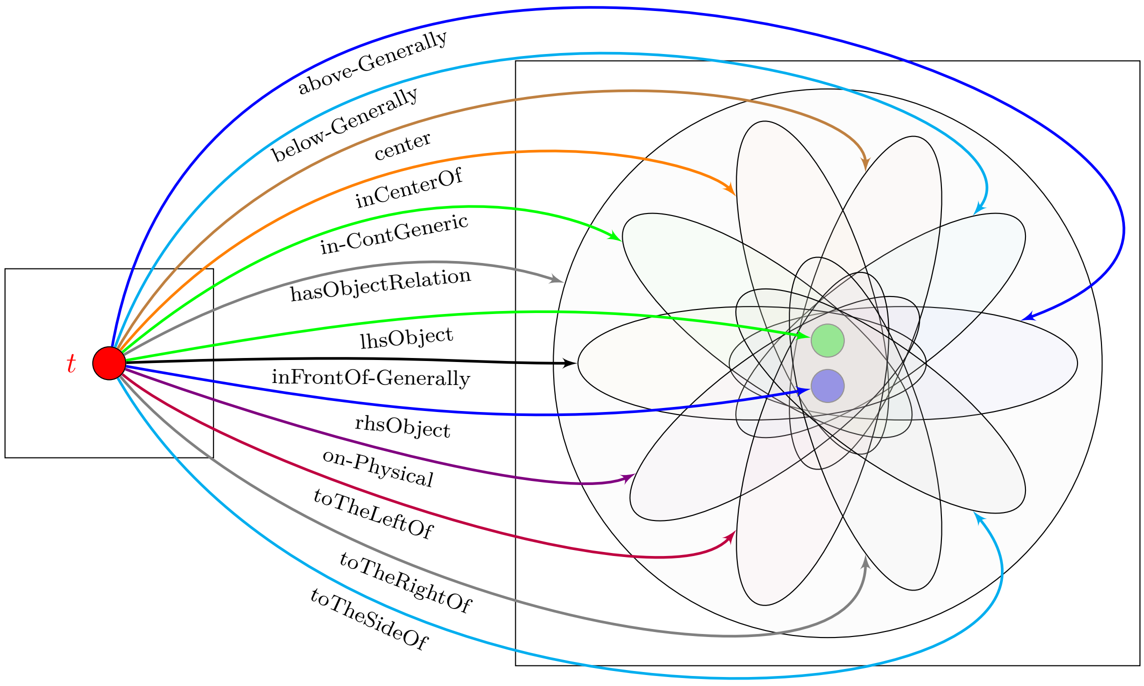


Figure 4.9: Object relation properties, $\forall\ t \in Thing$.

CHAPTER 5

Implementation of the reasoning system

While the last chapter dealt with the ontology modeling of the assembly task, the focus of this chapter is to describe the design and implementation of our reasoning system that will take the ontology model and use it for further processing and reasoning. This chapter is structured into three main parts, the first one gives how the system architecture is designed, the second one presents the development of a ML package and the third one provides the development of a reasoning package that will operate with the services provided by the ML package.

5.1 Architecture of the reasoning system

The reasoning system architecture is build up according to three modules, namely: The Perception system module, the KnowRob (`komoprod_task_reasoning`) module and the RosJava (`komoprod_reasoning`) module as shown in Figure 5.1. The Perception system module is not within the scope of this master thesis, it is simply used here to give the idea from where are the perceived data coming from. The `RosJava` module provides the services of ML techniques, namely: train different classifiers on the given training data model, use the trained model to make prediction, classifier evaluation, and data preprocessing. The ontology assembly task model is placed in the `KnowRob` module, also the Prolog ontology processing predicates in order to construct the data in a right format (list of lists). The `KnowRob` module provides the knowledge processing for the fetched data as well. A detailed explanation of both modules will be given in the following two sections. Perceived data that are provided by the perception module should be in the same format as the extracted data from the ontology assembly task model. Perceived data are fed into the `KnowRob` module and together with the training data are sent into the `RosJava` module. In `RosJava` module they are preprocessed and converted to nominal data. And the last step is that the given classifier which is defined in a Prolog predicate in the `KnowRob` module is used to train the ML model, and from the ML trained model the prediction of the unknown task state on perceived data is achieved.

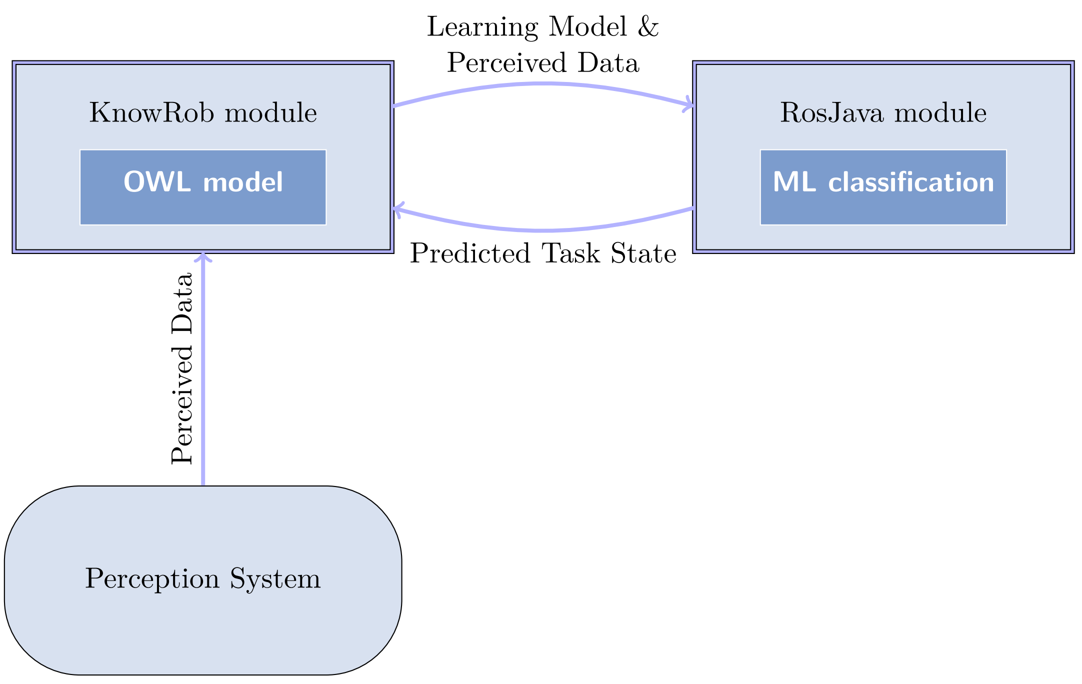


Figure 5.1: Architecture of the reasoning system.

The predicted class (task state) is returned to the `KnowRob` module as a string value.

5.2 RosJava package implementation

In Section 2.3 it has been established that ML classifiers may be generally suitable for the prediction of the current state of an assembly task. The success of the predicted task state by considering human state, robot state, and object relations is essential in this work. To reach the goal of the master thesis, a ML classifier package will be mainly used with the help of those classifiers that are introduced in Section 2.3. In this section the application of ML techniques will be presented by building a Rosjava (Con) package that will operate with Weka (WFH11) (the weka version is 3-9-0) Application Programming Interface (API) in order to be able to access the modeled assembly task ontology and to use it for further processing. RosJava represents the Java implementation of the Robot Operating System (ROS) (QCG$^+$09) framework. ROS is an open-source, meta operating system for robots. Weka workbench is a collection of ML algorithms and data preprocessing tools. Since the work in this master thesis is part of the KnowRob system, and KnowRob is implemented in SWI-Prolog and SWI-Prolog also is used as the central knowledge store, a connection between Weka and SWI-Prolog is required. Therefore it is decided to adjoin Java Interface to Prolog (JPL)(DW) into the developed package. JPL is a set of Java classes and C functions providing an interface between Java and Prolog. The constructed package is placed in the KnowRob work space in order to be able to extend its functionality. Prolog predicates are developed in this package in order to make use of the constructed Java classes that operate with Weka API. The developed predicates will then be called by other packages that will need the ML services obtained by this package. The work of Weka classifiers in this package is constrained to only those classifiers that can deal with nominal values, since the processing of ontology knowledge will be focused on nominal data. As explained in Section 2.3 in the operation with Weka we will consider only four of its classifiers.

We developed the following main prolog predicates in RosJava package:

```
trained_classifier(Classifier, TrainingData),
classify_data(TrainingData, TestData, Pred),
directly_classify(Classifier, TrainingData, TestData, Pred),
evaluate(Classifier, TrainingData),
save_data(Data).
```

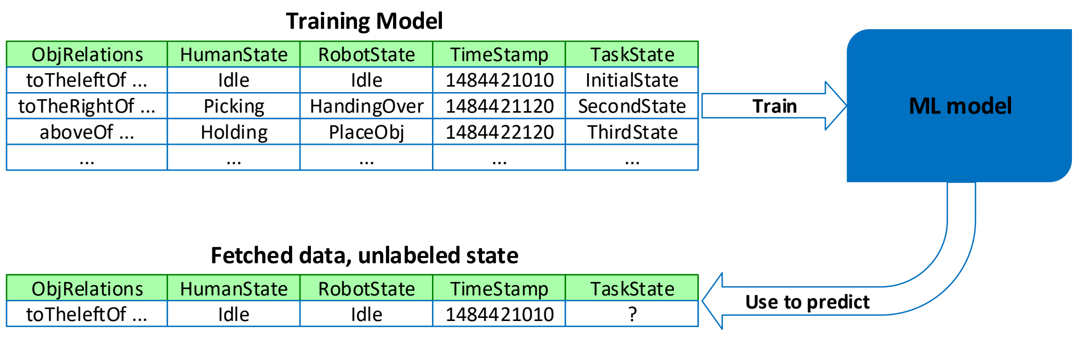
The very first step of classification is to provide the classifier with the training data, to handle this step the `trained_classifier` predicate is developed. Predicate `trained_classifier` has arity two. Its first argument is the classifier that will be used and the second argument represents the training data. This predicate will be called for the training step of the classifiers. In order to use the trained model for classification a `classify_data` predicate is developed and it has arity three. The attributes of the `classify_data` predicate are: training data, test data, and the returned predicted class value (predicted assembly task state). Both training data and testing data are given in the `classify_data` predicate because testing data is represented as a two dimensional string array and it has to have the same attribute values as the training data. The *TestData* corresponds to the perception data which is provided by the robotic system. The predicted value for the unknown class is returned at `Pred` variable. Another predicate that combines training and testing together to provide a quicker way of using the classification is the `directly_classify` predicate, like the previous one it returns the predicted value that is mapped to `Pred` variable by feeding with respective information the other three arguments namely: `Classifier`, `TrainingData`, and `TestData`. The intention of the `evaluate` predicate is to evaluate the classifier by showing the evaluation results, it has to be fed with the classifier (`Classifier`) and training data (`TrainingData`). The `save_data` predicate is made to save the data in Attribute-Relation File Format `ARFF` (arf), that allows to use the model outside the KnowRob tool.

5.3 KnowRob package implementation

This section introduces the development of a KnowRob (TB13) package in which the main developed predicates of the previous section will be called to predict the current state of an assembly task. Some of the so called `computables` (MM09) are developed, which serve for computing relations that would otherwise be manually asserted or inferred using logical inference in the constructed ontology learning model. The constructed ontology model will be queried from this package from which a list with model instances will be generated. The ontology model is placed on this package, which is used as a core model and it can be imported to other specific assembly task models. The most important Prolog predicate used for this package is the `training_data(TrainingData)` predicate, which returns all training data (as a list of lists) that are specified in the modeled ontology assembly task.

The idea of applying ML classifiers to predict the current state of an assembly task will be illustrated by making some Prolog queries. Figure 5.2 represents the abstraction of applying the ML classifiers to the task state prediction problem.

The first step is to start by training the classifiers on training data using a Prolog query. Therefore a ML model will be produced, and then another Prolog query will be applied to predict the task state of fetched data.



ObjRelations	HumanState	RobotState	TimeStamp	TaskState
toTheleftOf ...	Idle	Idle	1484421010	InitialState
toTheRightOf ...	Picking	HandingOver	1484421120	SecondState
aboveOf ...	Holding	PlaceObj	1484422120	ThirdState
...	...	...	...	...

ObjRelations	HumanState	RobotState	TimeStamp	TaskState
toTheleftOf ...	Idle	Idle	1484421010	?

Figure 5.2: Abstraction of the reasoning system.

So to realise a training and a prediction there are two steps that have to be performed, as following:

1. `train` the classifier by using the query:

```
training_data_array(FArrayArray),
jpl_new('weka.classifiers.trees.J48', [], Classifier),
trained_classifier(Classifier, FArrayArray).
```

The purpose of the above query is to train the J48 (Decision Trees) classifier on the given data model FArrayArray by the help of the trained_classifier predicate. A new instance (Classifier) of a classifier is created by the jpl_new predicate and given as the first attribute in the trained_classifier predicate. After the execution of this query a learned model is saved in the RosJava package and is ready to be used for classification.

2. predict the unknown state on the fetched data:

```
training_data_array(FArrayArray),
TestFeatures = [list of fetched data],
fetched_data_array(FTestArray, TestFeatures),
classify_data(FArrayArray, FTestArray, Pred).
```

The predicate fetched_data_array, converts the list of the fetched data (TestFeatures) into a two dimensional string array and binds it to FTestArray variable. Therefore the classify_data(FArrayArray, FTestArray, Pred) predicate will return the predicted state at the Pred variable.

This two steps approach gives only one example how to train and then use the trained model for prediction, but once the training model is learned it does not have to be updated (re-execute the first step Prolog query) until some changes to the ontology model are made. In case of a big data model, the training step may require a lot of time and therefore it is not practical to make it frequently. But the second step Prolog query can be executed as many times as it is needed for any fetched data that has the same format as the data model.

For situations where the data model is small and the use of train and predict by one predicate is practical, as shown in the previous section a predicate is constructed that is able to do this job. An example of such a Prolog query that does training and predicting by one predicate is given as follows:

```
training_data_array(FArrayArray),
jpl_new('weka.classifiers.trees.J48', [], Classifier),
TestFeatures = [list of fetched data],
fetched_data_array(FTestArray, TestFeatures),
directly_classify(Classifier, FArrayArray, FTestArray, Pred).
```

As it can be observed from the query, first training data are loaded, then a new instance of the J48 (Decision Trees) class is created by the jpl_new predicate. The fetched data list has to be assigned to the TestFeatures variable which is converted to an array by calling the fetched_data_array(FTestArray, TestFeatures) predicate. At the end the `directly_classify` predicate is called that is fed with the classifier, training data and fetched data, and returns the predicted state to the `Pred` variable.

Another example will be shown that is used to evaluate a ML classifier, to do so a Prolog query as follows is needed:

```
jpl_new('weka.classifiers.trees.J48', [], Classifier),
eval_classifier_10FCVLD(Classifier).
```

The above Prolog query creates an instance of the decision trees classifier and feeds it to the `eval_classifier_10FCVLD` predicate. The `eval_classifier_10FCVLD` predicate makes the evaluation based on 10 fold cross validation.

CHAPTER 6

Experimental validation of the implementation

It is known that there are many different ML methods, tools and techniques available, each with distinct advantages and disadvantages. The domain of ML has grown to an independent research domain. After the reasoning system is designed and implemented the focus is put on classifier evaluation. In this chapter we discuss our experimental performance evaluation of learning algorithms and summarize our findings, performance evaluation is made between suitable classifiers introduced in Section 2.3. The performance evaluation is based on experiments with the chosen classifiers on three data sets, which are constructed by an assembly task human expert. The first data set consists of 24 instances, and all the attributes have nominal values (as explained in Section 5.2). The second data set consists of 48 instances, and all attributes have nominal values. The third data set consists of 80 instances, and again all attributes have nominal values. Since our work at Profactor GmbH dealt with a very specific assembly task scenario, the constructed datasets are based on a steam cooker assembly task description model. The knowledge represented by the assembly task model is given by the number of samples per dataset, which varies from too weak (3 configurations for each state) to too strong (10 configurations for each state, approximately all combinations of object relations). The initial data set has 8 states and 3 configurations for each state (24 instances), the second data set has 8 states and 6 configurations for each state (48 instances), and the third data set is based on 8 states and 10 configurations for each state (80 instances). Except taken time to build the model and correctly classified instances the interest is on `precision`, `recall` and `F-measure` to evaluate the classifiers performance. To perform the experiments, we use the following machine/software:

```
Intel Core i7-3687U CPU @ 2.10GHz x 4
8GB RAM
```

```
Ubuntu 14.04 LTS (64 bit)
OpenJDK RTE (build 1.8.0_111-8u111-b14-3~14.04.1-b14)
WEKA 3.9.0
ROS indigo
```

6.1 Experiments based on the first data set

In the first experiment we evaluate the performance of the chosen classifiers in 10 fold cross validation test mode. These results (see Table 6.1) are obtained by running the algorithms on the default input parameters. As it can be seen the used time to build the model is negligible for all classifiers. The best performance with respect to the correctly classified instance is achieved by Decision Trees followed by kNN and Naïve Bayes. What is surprising in this first experiment is that the run of SVM classifier implies that the number of correctly classified instances on 10 fold cross validation test mode is 0 (as it will be seen later this is not the case when kernel type estimator is changed to linear). Its best performance on the first data set by using the function is on 3 fold cross validation test mode where the number of Correctly Classified Instances is 22 (91.6667 %).

Classifier	Time taken to build the model (seconds)	Correctly classified instances	Precision (avg)	Recall (avg)	F-measure (avg)
Decision Trees	0.01	19 (79.1667 %)	0.844	0.792	0.788
Naïve Bayes	0	14 (58.3333 %)	0.698	0.583	0.615
kNN	0	18 (75 %)	0.845	0.750	0.754
SVM	0	0 (0 %)	0.000	0.000	0.000

Table 6.1: Evaluation of classifiers on first dataset with 10 fold cross validation.

6.1.1 Optimizing parameters of classifiers on the first dataset

Now, the achieved results reported in Table 6.1 are considered, to improve the classifiers performance by changing their input parameters.

Decision Trees

In experimenting with the Decision Trees classifier to check if the performance of the classifier can be improved, the minimum number of objects and the confidence factor are changed. The minimum number of objects is the minimum number of instances in the leaf nodes, the confidence factor gives a relation between the pruning and the error on each decision node. Default parameters are: minNumObj = 2, confidenceFactor = 0.25. By changing minNumObj to 1 and changing the confidenceFactor to 0.5 and 0.1 the same results as in the default parameter values are produced. When minNumObj is increased to 3 and 4 while the confidenceFactor was changed again to 0.5, and 0.1 the performance is getting worse than in the default parameter configuration.

kNN

In the kNN case the parameter k is changed in order to see if are achieved better results. To avoid ties an odd distance is used. The experiment presented here tries to find the best distance. As sketched in Table 6.2 the best results for the kNN classifier are achieved when the distance is 1 (default configuration). One can observe, that an increased distance leads to worse results. The time taken to build the model is 0 seconds on all three cases.

Distance	Correctly Classified Instances	Precision (avg)	Recall (avg)	F-Measure (avg)
1	18 (75 %)	0.845	0.750	0.754
3	14 (58.3333 %)	0.573	0.583	0.515
5	8 (33.3333 %)	0.254	0.333	0.240

Table 6.2: kNN results on different distance parameters.

SVM

To improve the results of the SVM classifier different kernel estimators are used. The results are summarised in Table 6.3.

kernelType estimator	Correctly Classified Instances	Precision (avg)	Recall (avg)	F-Measure (avg)
Linear	23 (95.8333 %)	0.969	0.958	0.957
Polynomial	0 (0 %)	0.000	0.000	0.000
Radial	0 (0 %)	0.000	0.000	0.000
Sigmoid	0 (0 %)	0.000	0.000	0.000

Table 6.3: SVM results on different kernel functions.

By the use of the linear kernel estimator the best result achieved so far in the first dataset is gained. The time used to build the model with the linear kernel estimator is 0.02 seconds and with the other ones is 0 seconds.

6.2 Experiments based on the second data set

The classifier performance is again evaluated in 10 fold cross validation test mode. These results (see Table 6.4) are achieved by running the algorithms on the default input parameters. As shown in Table 6.4 the time taken to build the model is negligible for all classifiers. The best performance with respect to the correctly classified instance is achieved by Naïve Bayes followed by kNN and Decision Trees. Again, as in the previous section in the case of the SVM classifier the number of correctly classified instances is 0. Later it is shown that this is not the case when the kernel type estimator is changed to linear. But again changing the test mode to 3 fold cross validation the best performance of SVM is gained in which the number of Correctly Classified Instances is 46 (95.8333 %).

Classifier	Time taken to build the model (seconds)	Correctly classified instances	Precision (avg)	Recall (avg)	F-measure (avg)
Decision Trees	0.01	43 (89.5833 %)	0.915	0.896	0.898
Naïve Bayes	0	46 (95.8333 %)	0.964	0.958	0.958
kNN	0	45 (93.75 %)	0.951	0.938	0.936
SVM	0.01	0 (0 %)	0.000	0.000	0.000

Table 6.4: Evaluation of classifiers on second dataset with 10 fold cross validation.

6.2.1 Optimizing parameters of classifiers on the second dataset

Similar to Section 6.1.1, this section provides a description on how to improve the performance of the classifiers based on the results shown in Table 6.4 by changing their input parameters.

Decision Trees

For the experiments we change the minimum number of objects and the confidence factor. The results are shown in Table 6.5. Note that there are some minor improvements when the minNumObj value is set to 1. The experiments are performed for three values of confidenceFactor starting from 0.1, 0.25 (which is the default), and 0.5 but the results do not change. On this dataset the best performance of Decision Trees is made for minNumObj = 1, and confidenceFactor = 0.25.

minNumObj\ confidenceFactor	0.5			0.25		
	Precision (avg)	Recall (avg)	F-measure (avg)	Precision (avg)	Recall (avg)	F-measure (avg)
1	0.958	0.938	0.939	0.958	0.938	0.939
2	0.915	0.896	0.898	0.915	0.896	0.898
5	0.915	0.896	0.898	0.915	0.896	0.898
7	0.464	0.417	0.387	0.468	0.438	0.394

Table 6.5: Decision tree results for different parameter settings.

kNN

To achieve better performance of the kNN classifier, experiments where parameter k is changed are performed. Odd values of k are considered in order to avoid ties. In the applied experiments the attempt is to find the best distance. As sketched in Table 6.6 the best results for the kNN classifier are achieved when the distance is 1 (default configuration). It can be noted that increasing the distance leads to worse results. The time taken to build model is 0 seconds on all three cases.

Distance	Correctly Classified Instances	Precision (avg)	Recall (avg)	F-Measure (avg)
1	45 (93.75 %)	0.951	0.938	0.936
3	44 (91.6667 %)	0.940	0.917	0.918
5	41 (85.4167 %)	0.920	0.854	0.854

Table 6.6: kNN results on different distance parameters for the second data set.

SVM

The experiments using the SVM classifier are carried out with different kernel estimators. The results are shown in Table 6.7.

kernelType estimator	Correctly Classified Instances	Precision (avg)	Recall (avg)	F-Measure (avg)
Linear	48 (100 %)	1.000	1.000	1.000
Polynomial	0 (0 %)	0.000	0.000	0.000
Radial	0 (0 %)	0.000	0.000	0.000
Sigmoid	0 (0 %)	0.000	0.000	0.000

Table 6.7: SVM results for different kernel functions.

Again, the best results are achieved by using the linear kernel estimator. The time used to build the model on linear kernel estimator is 0.04 seconds and for the other ones is 0.01 seconds.

6.3 Experiments based on the third data set

Similar to Section 6.1 and 6.2, the classifier performance is evaluated in 10 fold cross validation test mode. The results (in Table 6.8) are achieved by running the algorithms on the default input parameters. The time taken to build the model is again negligible for all classifiers. The best performance with respect to the correctly classified instance is achieved by Naïve Bayes and SVM followed by kNN and Decision Trees. In contrast to the previous uses of SVM with the default kernel function configuration, on this data set the SVM classifier has achieved one of the best results.

Classifier	Time taken to build the model (seconds)	Correctly classified instances	Precision (avg)	Recall (avg)	F-measure (avg)
Decision Trees	0.01	75 (93.75 %)	0.945	0.938	0.938
Naïve Bayes	0	79 (98.75 %)	0.989	0.988	0.987
kNN	0	77 (96.25 %)	0.968	0.963	0.962
SVM	0.01	79 (98.75 %)	0.989	0.988	0.987

Table 6.8: Evaluation of classifiers based on third data set with 10 fold cross validation.

6.3.1 Optimizing parameters of classifiers for the third dataset

This section gives an overview on the performed experiments whose purpose is to improve the classifiers performance based on the results shown in Table 6.8 by changing their input parameters.

Decision Trees

For the experiment based on Decision Trees, the minimum number of objects and the confidence value are changed to try to improve performance. The tests are performed for three values of confidenceFactor starting from 0.1, 0.25 (which is the default), and 0.5 but the results are the same. On this dataset the best performance of decision tree is made for minNumObj = 1, and confidenceFactor = 0.25 where the number of correctly classified instances is 77 (96.25 %).

kNN

In the case of the kNN classifier experiments start again by changing parameter k in order to see if better results are achieved. Odd distance is used in order to avoid ties. The goal of the experiments is to find the optimal distance. As shown in Table 6.9 the best results for the kNN classifier are achieved if the distance is 1 (default configuration). By increasing the distance to 3 and 5, similarly worse results are obtained. The time taken to build the model is 0 seconds on all three cases.

SVM

Experiments using the SVM classifier are performed with different kernel estimators. The results are shown in Table 6.10. The best results for the third dataset from the SVM classifier, are achieved by using the linear kernel estimator (correctly classified instances are 100 %). The time taken to build the model for the linear kernel estimator is 0.05 seconds and for the other ones is 0.03 seconds. Compared to applying SVM on

Distance	Correctly Classified Instances	Precision (avg)	Recall (avg)	F-Measure (avg)
1	77 (96.25 %)	0.968	0.963	0.962
3	76 (95 %)	0.960	0.950	0.951
5	76 (95 %)	0.960	0.950	0.951

Table 6.9: kNN results for different distance parameters for the third data set.

the previous data sets, the SVM performance in this data set is generally better for each kernel function.

kernelType estimator	Correctly Classified Instances	Precision (avg)	Recall (avg)	F-Measure (avg)
Linear	80 (100 %)	1.000	1.000	1.000
Polynomial	76 (95 %)	0.953	0.950	0.950
Radial	79 (98.75 %)	0.989	0.988	0.987
Sigmoid	79 (98.75 %)	0.989	0.988	0.987

Table 6.10: SVM results for different kernel functions.

6.4 Concluding remarks on the evaluation results

SVM classifier usually requires a large data model in order to achieve maximum prediction accuracy. This behaviour is observed in our experiments as well. In the very first two experiments the correctly classified instances on SVM run with the default kernel function configuration are 0 (0 %), this happens because the training sets are too small. Since the projection of all three training sets to higher dimensional space leads to linearly separable classes, therefore in the run of SVM with linear kernel function, the best percentage of correctly classified instances is achieved in all three training sets. And the best performance of SVM is achieved for the second and third training sets with linear kernel function. In the second training set the correctly classified instances are 48 (100 %), and in the third training set the number of correctly classified instances is 80 (100 %). The worst performance of SVM is made on the first and the second training sets with Polynomial, Radial and Sigmoid kernel functions where the number of correctly classified instances is 0 (0 %), such a failure can happen because the training sets are too small.

Whereas Naïve Bayes performs better in case of rather small models, and as it can be seen along our experiments the performance of Naïve Bayes is one of the best ones. The Naïve Bayes classifier is one of the classifiers that is considered to have a high bias value, because it summarises the dataset by a single probability distribution and implies that this model is sufficient to discriminate between classes. The training time that is needed by the Naïve Bayes classifier is very short, since it needs only one pass through the model to count the frequency of the discrete values. Concerning the memory space for the Naïve Bayes classifier, it needs a little memory during the training and classification stage, the minimum required is to store prior and conditional probabilities. The best performance of the Naïve Bayes classifier is achieved for the third training set where the number of correctly classified instances is 79 (98.75 %). The worst performance of Naïve Bayes is for the first training set in which the number of correctly classified instances is 14 (58.3333 %).

On classifiers with a high variance it is usually the case that complex models are generated that fit data variations more strictly. Based on the performed experiments the classifiers which have high variance are decision trees and SVM. The best performance of decision trees is reached for the last training set where the minNumObj = 1, the confidenceFactor = 0.25 and the number of correctly classified instances is 77 (96.25 %). Decision trees in the first training set performs significantly better compared to the other classifiers. The decision trees performance on the first training set is the worst one compared to the decision trees performances on the second and third training sets.

There is general agreement that kNN is very sensitive to irrelevant features. In the constructed ontology model introduced in Chapter 4 the most irrelevant attribute is time-stamp which can be removed from the model as a preprocessing step. The kNN classifier requires zero training time because the training instances are simply stored. The kNN memory consumption depends on training data and becomes computationally expensive with many items to classify. The best performance of kNN is achieved for the third training set where the distance parameter (k) is 1 and the correctly classified instances are 77 (96.25 %). The basic kNN has usually only a single parameter (k) which is relatively easy to tune, but SVMs have more parameters.

Based on (Kot07) and on our experimental results of Sections 6.1, 6.2 and 6.3, Table 6.11 is constructed in order to give a compact view of comparison between the chosen classifiers.

Our experimental results indicate that for ontology models that have less than 24 instances the best classifier is decision trees, for ontology models that have more than 24 instances and less than 48 instances the best classifier is Naïve Bayes, and for models that have more than 48 instances the best classifier is SVM.

	Decision Trees	Naïve Bayes	kNN	SVM
Accuracy in small data sets	***	****	***	***
Accuracy in general	**	**	**	****
Speed of learning with respect to number of attributes and the number of instances	***	****	****	*
Speed of classification	****	****	*	****
Dealing with danger of overfitting	**	***	***	**
Attempts for incremental learning	**	****	****	**
Model parameter handling	***	****	***	*

Table 6.11: Comparison of algorithms (**** stars represent the best and * star the worst performance)

CHAPTER 7

Conclusions and Future Work

This chapter, structured in two parts summarizes the research work and concludes the findings before giving a short outlook into potential future directions in this research domain.

Conclusions

As initially stated, modern production systems have to be designed to deal with the growing challenges of market trends, high flexibility in product types and variants, and short innovation cycles in order to stay competitive. In this context, Human-Robot-Cooperation is considered as a key technology to improve efficiency and to reduce operating costs. The research problem of this thesis is focused on improving the quality of the Human-Robot-Cooperation, by designing, implementing and evaluating a reasoning system that makes a robot able to hypothesize the current state of an assembly task. The design and implementation of the reasoning system is based on two modules, namely: one module that does the Machine Learning reasoning task, and the other module which provides the ontology model of the assembly task and also provides a mechanism to access and process the ontology model.

The first developed module is a RosJava package, on which a Machine Learning RosJava project is build. Since KnowRob is used as a knowledge processing system and it is developed in SWI-Prolog, the need of a Java Prolog communication rises. For this purpose it is used the JPL interface, and by using JPL a Prolog module is developed in the constructed package. The aim of this Prolog module is to provide some constructed predicates that can interact with the developed ML Java code, therefore the learning algorithms can be called and evaluated from these constructed predicates.

The second developed module is a KnowRob package. This package provides a mechanism for storing an ontology that expresses the assembly task model, and also provides a mechanism for accessing and processing the constructed ontology model. The stored ontology represents an assembly task model and will be written by the assembly task expert. The provided ontology model describes the information about the human states, the robot states, and object relations as initially declared as the research question, and also for the task states. Each task state consists of a specific human state, a robot state, a time stamp and a number of relations between objects.

The second developed package together with the first developed package reasons about the constructed ontology model and uses the derived Machine Learning model for making class (task state) prediction on the fetched data taken from the perception system.

Since the prediction of the current state of an assembly task is achieved, the successful implementation of the reasoning system is indicated. The developed reasoning system will serve as a valuable extension to the task execution tool of a robotic assembly system, and it can be used as a demonstrator in research projects dealing with human robot collaboration.

Finally, chosen ML classifiers are evaluated and compared to the established requirements in order to suggest the most appropriate classifier.

Concluding, the presented ontology model allows us to identify assembly task states of an assembly task in robotic assembly systems. The ontology model is able to be utilized by different assembly tasks in assembly robot applications. This fits nicely with the robot systems that uses the KnowRob processing tool. It can be safely said that the proximate cooperation between the human and the robot in an assembly task execution will be improved by applying our contribution in this master thesis. This work can be used by manufacturing companies and manufacturing research.

This Human-Robot-Cooperation improvement is not constraint to a specific assembly task, but it can be applied to any assembly task to which an ontology can be modeled by extending our developed ontology model. This way, knowing the current assembly task state indirectly contributes to a sustainable growth of knowledge in human robot cooperation.

To the best of our knowledge there is no similar work that is targeting the same problem. Therefore, we do not discuss the related work explicitly, but instead we provided Chapter 3 an implementation environment which gives a short overview about the topic of this thesis and analyzes existing notions and systems concerning knowledge processing and presentation. The most relevant work found so far is the one which presents the design of a joint-action assembly demonstrator that is used in cooperation between humans and robots in a shared workspace ((LNR+08)). The main idea given by this work is to achieve an efficient peer-to-peer collaboration, and the only similar work mentioned on their research is that the information which is perceived by the perception system is used in the decision making process to decide the next action step on the assembly task. It is given no clear explanation on how this decision making process is implemented and it has no similarities with our work in reasoning about the current state of an assembly task.

Future Work

The research presented in this thesis raises many more questions of both general and technical nature.

One of the bigger aspects of future research is the possibility to extend the reasoning system to a version that is capable to give a prediction on what the next state should be in the assembly task execution sequence. This is expected to strengthen the focus on a learning model that represents the assembly task states on a sequential mode.

The remaining lines of research follows from Chapter 4 where the development of a complex assembly task model is performed manually by a human assembly expert.

How to implement a user friendly interface in which the development of the ontology model will be easier? Since the Protégé environment might be too complex for the assembly task expert, it is definitely valuable to have a less complex user interface for constructing the ontology model.

Lastly, how to extend the reasoning system to a version that is able to validate the correctness of the modeled ontology? Because missing validation can lead to wrong asserted data in constructing the ontology model, e.g. Obj_1 is to the right of Obj_1 ($\texttt{toTheRightOf(Obj}_1\texttt{, Obj}_1\texttt{)}$).

List of Figures

List of Tables

Bibliography

[Agr06] Alan Agresti. *An Introduction to Categorical Data Analysis*, pages 1–20. John Wiley, Sons, Inc., 2006.

[Alp14] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2014.

[APPR16] Sharath Chandra Akkaladevi, Matthias Plasch, Andreas Pichler, and Bernhard Rinner. Human robot collaboration to reach a common goal in an assembly process. pages 1–12, September 2016.

[AR11] J. K. Aggarwal and Michael Ryoo. Human activity analysis: A review. *ACM Comput. Surv.*, 43(3):16:1–16:43, 2011.

[arf] Attribute-Relation File Format (ARFF). Available at `http://www.cs.waikato.ac.nz/ml/weka/arff.html` (Accessed: 2017/01/15).

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[BFOS84] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

[BM07] Adrian Bondy and U. S. R. Murty. *Graph theory.* Graduate texts in mathematics. Springer, New York, London, 2007. OHX.

[BWB08] Andrea Maria Bauer, Dirk Wollherr, and Martin Buss. Human-robot collaboration: a survey. *I. J. Humanoid Robotics*, 5(1):47–66, 2008.

[Con] Kohler Conley. Rosjava. Available at `http://wiki.ros.org/rosjava` (Accessed: 2017/01/14).

[DHS00] Richard Duda, Peter Hart, and David Stork. *Pattern Classification (2Nd Edition).* Wiley-Interscience, 2000.

[dMTH+12] Daniel di Marco, Moritz Tenorth, Kai Häussermann, Oliver Zweigle, and Paul Levi. Roboearth action recipe execution. In *12th International Conference on Intelligent Autonomous Systems*, 2012.

[DW] Paul Singleton Fred Dushin and Jan Wielemaker. Java prolog interface (jpl). Available at `http://www.swi-prolog.org/packages/jpl/` (Accessed: 2017/01/14).

[GS03] Knoblich Günther and Jordan Scott. Action coordination in groups and individuals: learning anticipatory control. *J Exp Psychol Learn Mem Cogn*, 29(5):1006–1016, September 2003.

[GS07] Michael Goodrich and Alan Schultz. Human-robot interaction: A survey. *Foundations and Trends in Human-Computer Interaction*, 1(3):203–275, 2007.

[HKR+09] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools, 2009.

[HLK05] Jung-Hoon Hwang, KangWoo Lee, and Dong-Soo Kwon. The role of mental model and shared grounds in human-robot interaction. In *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, pages 623–628, Aug 2005.

[HSSK06] Jenny A. Harding, Muhammad Shahbaz, Srinivas, and Andrew Kusiak. Data Mining in Manufacturing: A Review. *Journal of Manufacturing Science and Engineering*, 128(4):969–976, 2006.

[Kot07] Sotiris Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[LNR+08] Claus Lenz, Suraj Nair, Markus Rickert, Alois Knoll, Wolfgang Rösel, Jürgen Gast, Alexander Bannat, and Frank Wallhoff. Joint-action for humans and industrial robots for assembly tasks. In *The 17th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2008, Munich, Germany, August 1-3, 2008*, pages 130–135, 2008.

[LÖ09] Ling Liu and Tamer Özsu, editors. *World Wide Web Consortium*, pages 3559–3559. Springer US, Boston, MA, 2009.

[MM07] Wendy Martinez and Angel Martinez. *Computational Statistics Handbook with MATLAB, Second Edition (Chapman & Hall/Crc Computer Science & Data Analysis)*. Chapman & Hall/CRC, 2007.

[MM09] Tenorth Moritz and Beetz Michael. KNOWROB - knowledge processing for autonomous personal robots. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 11-15, 2009, St. Louis, MO, USA*, pages 4261–4266, 2009.

[Mus15] Mark Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015.

[PvSB09] Edita Poljac, Hein van Schie, and Harold Bekkering. Understanding the flexibility of action–perception coupling. *Psychological Research PRPF*, 73(4):578–586, 2009.

[QCG$^+$09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[Qui86] Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[RM14] Lior Rokach and Oded Maimon. *Data Mining with Decision Trees - Theory and Applications. 2nd Edition*, volume 81 of *Series in Machine Perception and Artificial Intelligence.* WorldScientific, 2014.

[Ros06] Mark Elling Rosheim. *Leonardo's lost robots.* Springer, 2006.

[Smu95] Raymond M. Smullyan. *First-order Logic.* Dover books on advanced mathematics. Dover, 1995.

[SW11] Claude Sammut and Geoffrey Webb. *Encyclopedia of Machine Learning.* Springer Publishing Company, Incorporated, 1st edition, 2011.

[TB12] Moritz Tenorth and Michael Beetz. A unified representation for reasoning about robot actions, processes, and their effects on objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, October, 7–12 2012.

[TB13] Moritz Tenorth and Michael Beetz. Knowrob: A knowledge processing infrastructure for cognition-enabled robots. *I. J. Robotics Res.*, 32(5):566–590, 2013.

[Tza00] Spyros Tzafestas. *Handbook of Industrial Robotics, Shimon Y. Nof (ed.)*, volume 28. 2000.

[ven] Venn diagram. Encyclopedia of Mathematics. Available at `https://www.encyclopediaofmath.org//index.php?title=Venn_diagram&oldid=32575` (Accessed: 2017/01/28).

[Wal08] Johanna Wallén. The history of the industrial robot. *Linköping: Linköping University Electronic Press*, 2008.

[Wes00] Lars Westerlund. *The Extended Arm of Man: A History of Industrial Robot.* Informationsförlaget, 2000.

[WFH11] Ian Witten, Eibe Frank, and Mark Hall. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[WSTL12] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. Swi-prolog. *TPLP*, 12(1-2):67–96, 2012.

[WWIT16] Thorsten Wuest, Daniel Weimer, Christopher Irgens, and Klaus-Dieter Thoben. Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45, 2016.

[ZZY03] Shichao Zhang, Chengqi Zhang, and Qiang Yang. Data preparation for data mining. *Applied Artificial Intelligence*, 17(5-6):375–381, 2003.