Studies on Interactive Knowledge Acquisition and Reuse for Teaching Industrial Robots

2010

Lei Wang

Design-Systems Laboratory
Department of Mechanical Engineering and Science
Graduate School of Engineering
Kyoto University
JAPAN

Acknowledgments

First and foremost, I would like to gratefully thank my supervisor, Professor Tetsuo Sawaragi, for his support, guidance, and encouragement during my stay at Kyoto University. He gave me a lot of freedom in my study, at the meantime, his rigorous research attitude influenced me greatly.

I must give special thanks to Ph.D. Yajie Tian. She has given me excellent guidance and support both in my research and my living in Kyoto.

I would particularly like to thank Mr. Akio Noda and Dr. Haruhisa Okuda, Mitsubishi Electric Corporation, Japan, for their help and support of the experimental facilities and technical suggestions. I have greatly benefited from the internship in Mitsubishi Electric Corporation, Japan, and have learned a lot about Japanese business culture.

I would also like to thank Ph.D. Yukio Horiguchi, Ph. D. Hiroaki Nakanishi, Ms. Shinobu Minato, and all the members in Sawaragi lab. They have been always willing to help me.

Last but not the least, I thank my parents for their love, understanding, and encouragement.

Abstract

The traditional automated line production is hard to deal with the diversified and rapid changing requirements of consumers anymore. This is because the automated line production systems are inflexible and are not easy to alter a design or production process after a production line is implemented. As a result, cell production is turned to be the manufacturing style that suits to the modern agile manufacturing philosophy, which focuses on meeting the demands of customers by adopting flexible and reconfigurable manufacturing practices. Cell production has the mass production line split into a number of self-contained units. Each cell is responsible for a significant part of the finished article and, rather than each person only carrying out one very specific task, team members are skilled at a number of roles. Therefore, in cell production, it is easy to reconfigure workplace and is flexible to assign jobs for new manufacturing tasks. So far, cell production is still mainly implemented by human workers.

On the other hand, because of the increasing human labor cost and long training time for novice workers, along with the fast advances of robotics technology and factory automation technology, robot cell production or autonomous cell production robotic systems will stand for the future development direction of the manufacturing industry. Currently, robot cell production is still in its initial developing stage. A critical problem of robot cell production is that the task of teaching robots is excessively time-consuming, which leads to increase in time cost and becomes the bottleneck of increasing the flexibility and popularity of robot cell production.

The essence of teaching robots is to transfer knowledge from human workers to robots. Thus, to solve the robot teaching problem, it is significant to acquire human knowledge for robots and make robots be able to automatically and properly reuse the acquired knowledge for similar new manufacturing tasks. Herein, the human knowledge refers to how human workers adjust the robot program for a certain manufacturing task according to the given environment state by tuning the positions of seizing/placing points of the target workpiece, by inserting intermediate points and time delays in the motion trajectory, or by changing the robot motion speed and so on. Such human knowledge plays a crucial role in ensuring and improving the stability and efficiency of the performance of the industrial robots, especially when they are set into the automatic running mode.

Thus, by acquiring and reusing the human knowledge, the time cost of employing robots can be dramatically reduced. Moreover, by accumulating human knowledge for robots, even after experienced workers are retired, it is still possible to keep the productivity and manufacturing efficiency of a company. In addition, this also gives a chance that novice workers are able to learn from robots during their implementation of robot teaching tasks. Therefore, the originality of this research is on developing a system to realize interactive knowledge acquisition and reuse for teaching industrial robots. Herein, ‘interactive’ means that not only robots are able to learn from human workers, but also it is possible for human workers to learn from robots through their implementing the robot-teaching tasks. This research appears especially important in the recent years, during which more and more experienced workers were going to retire from their jobs and it takes a long time to train new workers.

In this research, a knowledge-based system has been developed for acquiring and reusing human knowledge in the robotic assembly domain. For an assembly task (i.e. the assembling process of a certain workpiece from picking it up from its initial position to placing it at/in the destination) is not

trivial but complicated in which many factors such as workpiece features (i.e., shape, size, weight and so on), tool features (i.e., number of fingers, open width, length, load of weight, and seizing type such as gripping by two fingers, absorbing by a vacuum and so on), environments (i.e., the number of obstacles nearby the destination, and the relative positions and distances between the obstacles and the destination) and so on should be considered, a novel concept of hierarchical knowledge modularization has been proposed. Hierarchical knowledge modularization means that a knowledge-based system acquires and represents the knowledge implemented in a robotic manufacturing task in terms of a hierarchical set (i.e., different levels) of knowledge modules, organizes these knowledge modules as a plan for the observed task, and saves the plan and the task description as a case in the case base of the system. In this way, the system can flexibly retrieve a case or modules of several cases and reuse it or them for a new task. In other words, the system can re-organize different levels of knowledge modules in different cases to generate the solution for a new task.

Explanation-based learning (EBL) is a deductive learning method that can learn from a single training example with the help of a pre-encoded domain theory. It is used to acquire human knowledge by observing robot programs of experienced workers and corresponding task descriptions. In the developed knowledge-based system, the original EBL has been modified with regards to the operationality criterion. The new proposed operationality criterion is a set of rules that demand the learned knowledge should be expressed in two ways: 1. easily understandable by human workers, i.e., explanations of the observed robot program are saved in the learned result; and 2. reusable by robots to generate program automatically, i.e., the observed robot program is generalized as a program schema. This new proposed operationality lays the foundation for interactive knowledge acquisition and reuse for teaching industrial robots. On the one hand, by reusing the learned robot program schemata, robots can automatically generate programs for tasks that are similar as the ones they have learned from human workers. On the other hand, the explanations of the programs generated by the robots can be provided for novice human workers, which makes the novice human workers be able to learn from robots during they perform robot-teaching tasks.

The modified EBL has been further repeatedly and hierarchically used to abstract different levels of knowledge modules from the observed assembly task. The learning result, i.e., the generalized assembly plan is composed of different levels of knowledge modules. Together with its corresponding task description, they are represented and saved as a case in the case base of the knowledge-based system. In addition, the modified EBL has further been improved to have two learning modes. The first learning mode is learning from an example directly given by human workers, which is the modified EBL method. The second learning mode is learning expertise on error recovery by observing revisions made by human workers in handling execution errors that occur in teaching the robots or in reusing previously acquired knowledge. A criterion for saving the learned knowledge has also been proposed to control the number of the cases in the knowledge base.

Case-based reasoning (CBR) is the process of solving new problems based on the solutions of similar past problems. In this research, a modified CBR method has been developed to reuse the acquired cases to automatically generate robot programs for new assembly tasks. Because the saved cases are composed of hierarchical knowledge modules in this proposed system, a past case not only can be reused as a whole, but also can be reused partly by synthesizing different parts of several cases to generate a program for a new complex task in a variant environment.

In the knowledge-based system, EBL and CBR are integrated to reduce the utility cost of the acquired knowledge. This is because the cost of reusing the cases of knowledge modules is

increasing along with more and more knowledge is acquired and saved in the knowledge base. By abstracting the training examples into generalized knowledge modules, the improved EBL helps reduce the number of the saved cases. By providing a flexible retrieving method that is able to partly reuse the past cases for new tasks in variant environments, the modified CBR method improves the reusing efficiency of the saved cases. In addition, the knowledge-based system is possible to be applied as a media in integrating product design and manufacturing for reducing the lead time for companies. This is because the proposed system has saved errors occurred in the manufacturing process, and some errors that are difficult to be settled during the robot-teaching process can be solved easily by making changes in the design of a product. By using the proposed system, a product designer can simulate the robotic manufacturing process and be aware of the errors that may occur during the robotic manufacturing process. In this way, the designer is able to consider such errors from the product designing perspective and may further solve some of these errors by revising the product design before the robot-teaching process. In addition, the proposed system can automatically generate the preliminary robot program for assembling a product as soon as the data of its design has been input. Thus, the lead time can be reduced by using the proposed system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The traditional automated line production system, or also can be called mass production system, is dedicated to meeting the enormous customer needs of a single or small group of standardized products, in which each product passes through the same sequence of operations, and the machines and other equipments are laid-out in the order they are used.

The profits of line production come from several sources. The primary cause is a reduction of nonproductive effort of all types. In craft production, the craftsman must bustle about a shop, getting parts and assembling them. He must locate and use many tools many times for varying tasks. In line production, each worker repeats one or a few related tasks that use the same tool to perform identical or near-identical operations on a stream of products. The exact tool and parts are always at hand, having been moved down the manufacturing/assembly line consecutively. The worker spends little or no time retrieving and/or preparing materials and tools, and so the time taken to manufacture a product using line production is shorter than when using traditional methods. As factory automation (FA) has been introduced into line production, more and more tasks are predominantly carried out by machines and robots. Therefore, the probability of human error and variation is also reduced. In addition, a reduction in labor costs, as well as an increased rate of production, enables a company to produce a larger quantity of one product at a lower cost than using traditional, non-automated-line-production methods.

However, automated line production system is inflexible because it is not easy to alter a design or production process after a production line is implemented. Also, all products produced on one production line will be identical or very similar, and introducing variety to satisfy individual tastes is very difficult.

In recent years, as the requirements of consumers for products has become manifold and has been altering rapidly, the product types are becoming more diversified and the period of product demand is becoming shorter. Therefore, coupled with the fierce global competition of production, it is the trend that production of products is shifting from mass production of few product types to multi-product production in varying volume. As a result, cell production, or cellular manufacturing, is turned to be the manufacturing style that suits to the modern agile manufacturing philosophy [1], which focuses on meeting the demands of customers by adopting flexible and reconfigurable manufacturing practices.

Cell production has the mass production line split into a number of self-contained units. Each team or ‘cell’ is responsible for a significant part of the finished article and, rather than each person

only carrying out only one very specific task, team members are skilled at a number of roles, so it provides a means for job enrichment and job rotation. Cell production is a form of team working and helps ensure worker commitment, as each cell is responsible for a complete unit of work. Cells would usually have responsibility for organizing work rosters within the cell, for covering holiday and sickness absences and for identifying recruitment and training needs. Cells deal with other cells as if they were customers, and take responsibility for quality in their area.

So far, cell production is still mainly implemented by human workers. In other words, the current cell production is still human cell production. Although human cell production is more adaptable to the high-mix low-volume market demand fluctuations and has lower production cost and higher quality compared with mass production, along with the fast advances of robotics technology and factory automation technology, robot cell production or autonomous cell production robotic systems will stand for the future development direction of the manufacturing industry [2-4].

1.1 Next Generation Manufacturing: Robot Cell Production

In human cell production, the product quality and task performance are greatly influenced by the skills, experiences, physical and mental conditions, and other personal factors of human workers. Thus, the disadvantage of human cell production is that it is difficult to maintain the production quality and efficiency at a reliable level due to human factors such as experienced worker retirement and novice worker recruitment. An ideal solution for this problem is to employ robots in cell production to perform the physical manufacturing tasks instead of human workers.

The advantages of robot cell production are as the following: 1. Product quality and production efficiency can be steadily maintained. 2. 24-hour continuous production is possible with the robot operation. This is beneficial to flexibly scheduling the manufacturing jobs and reducing the stock cost of parts and finished products. 3. The cost of employing robot is much lower than that of hiring human labors. This is especially true for developed countries. 4. A robot work cell can be configured in a relatively small limited space. Thus, it helps save the cost of plant construction area.

It is true that robot cell production is superior in many respects to human cell production. Nevertheless, this does not indicate that humans are completely being replaced by robots in robot cell production. The term ‘robot cell production’ just signifies that robots are utilized to the fullest extent in the whole manufacturing process including preliminary work such as part feeding and post process such as quality inspection of finished products in a work cell. The objective is to integrate robotics technology and various factory automation technologies to achieve the optimal design and control of the work cells in cell production.

Generally, a robot work cell is composed of a working platform, several robots, sensors, and other automated equipments such as part feeders and fixtures. All of the robots and equipments are confined in the limited space on the working platform. With the support of sensors and other

automated equipments, robots fetch out parts or workpieces from part feeders or bins, assemble the parts to make the products, inspect the quality of the finished products, and at the end put out the qualified products onto conveyors.

The ultimate purpose of the development of a robot cell production system is to the four goals described below [2]:

1) A system suitable for multi-product production in varying volumes;
2) A flexible system that can be adjusted easily to the product types and the size of production;
3) A system which is versatile and which can be standardized, in order to reduce the introduction cost of the system;
4) A system compliant with international safety standards, which is developed with the latest control safety technologies and can be used worldwide.

1.2 Problem of Excessive Robot Teaching Time in Robot Cell Production

Currently, robot cell production is still in its initial developing stage. There are many problems to be researched and solved such as the dexterity of robot tools and the accuracy of sensors. Among many problems in robot cell production, a critical one is that the task of teaching robots is excessively time-consuming [5-7]. This has become a bottleneck to improving the flexibility of robot work cells, which keeps small and medium size enterprises (SMEs) away from robotic automation. This problem is especially severe in the robotic assembly domain.

The robot system used by nowadays manufacturers consists of four parts, including the robot arm, the robot controller, a teaching pendent, and a personal computer (PC). Robot teaching is the most important and difficult work in robotic manufacturing. Workers teach a robot in the following 4 steps: 1. making a program for the task of assembling a workpiece; 2. setting the robot in teaching mode, enabling the teaching pendant, and teaching the robot the coordinates of points in the program with the teaching pendant; 3. checking the effectiveness of the teaching by letting the robot execute the commands one by one with the teaching pendant; 4. disabling the teaching pendant, setting the robot in playback mode to check the automatic execution of the robot program.

Robot teaching is awfully time-consuming for the following two main reasons. First, workers have to teach robots repeatedly even for similar assembly tasks. For example, to screw four bolts into four holes in a box, workers have to teach the robot four times, although the mechanics of screwing are common each time. Second, the robot program usually has to be revised repeatedly for new tasks in order to become robust in the real-life manufacturing environment. This is because workers often have to repeat their efforts many times to find the appropriate positions for the seizing point, placing point, and approach points of a given workpiece. In addition, workers often have to revise the robot program in response to the occurrence of errors in the playback mode. This is

because, in the teaching mode, the actions of the robot are discrete, and the speed of motion is very slow. However, in the playback mode, the robot's actions are executed continually at high speed. Thus, unexpected errors that did not happen in the teaching mode may occur in the playback mode. In this case, the workers have to revise the corresponding part of the robot program to correct such errors by tuning the robot motion speed, time delays, signal communications, or key point positions and so on.

By far, various methods have been proposed to facilitate the task of teaching robots [8]. Many researchers have applied virtual reality technologies [9-11], while others have used intuitive teaching methods such as leading robots directly by human hands [12] or instructing robots by voice [13]. Although these methods indeed simplify the task of teaching robots, they overlooked the essential fact the task of teaching robots is to transfer knowledge from human workers to robots.

1.3 Contributions of the Dissertation

The purpose of this research is to integrate the expertise and experiences of skilled workers that are accrued from long years performing of manufacturing and assembly tasks into the robot cell production systems. This has great significance to robot cell production, because it is only human having the unique advantages of excellent adaptability to new environments, unrivaled flexibility to uncertainties, remarkable learning ability, and creativity of novel knowledge. This is also why robots can not completely replace humans and should be taught by humans for carrying out inexperienced tasks in robot cell production.

If we take a robot cell production system as a human, then the robots and other automated equipments in the system is the body of the human, human workers can be regarded as the soul of the human, and the work of this research is to create the brain of the human, which is a knowledge-based system. The objective of developing this knowledge-based system is to acquire human knowledge of teaching robots assembly operations, accumulate the knowledge in the knowledge base, and reuse the knowledge for similar assembly tasks. In this research, the novel concept of hierarchical knowledge modularization has been proposed.

Toward implementing the hierarchical knowledge modularization idea in real-world robot cell production systems, this dissertation contributes in the following senses:

- Presenting what is the knowledge in robotic assembly, from what the knowledge can be acquired, and the method used in acquiring the knowledge (Chapter 3);
- Providing a new systematic approach for hierarchically generalizing all the know-hows implemented by human workers in the whole process of teaching robots assembly operations, systematizing the generalized knowledge into different levels of reusable knowledge modules, learning new knowledge from error recovery examples, and evaluating whether the acquired knowledge modules are worth saving into the knowledge base

(Chapter 4);

- Proposing the method how to flexibly reuse the acquired knowledge modules for new assembly tasks and how the automatically generate robot programs for robots in the robot cell production systems (Chapter 5).

1.4 Overview of the Dissertation

The followings are the general description s of the contents of each chapter.

Chapter 2 introduces the basic approaches, i.e. explanation-based learning (EBL) and case-based reasoning (CBR), used for knowledge acquisition and reuse, and how EBL and CBR are integrated to construct the proposed knowledge-based system.

Chapter 3 presents the overall structure of the proposed knowledge-based system for acquiring, sharing, and reusing human knowledge in the robotic assembly domain. The original EBL is modified to acquire tacit knowledge from the robotic programs of skilled workers. A new operationality criterion within the EBL framework is proposed, which demands that the learned tacit knowledge should be understandable by workers and reusable by robots. By accessing the knowledge accumulated in the knowledge base, workers can learn the tacit knowledge implemented in robotic programs made by other workers. In this way, robotic programs become a vehicle for transferring tacit knowledge of skilled workers. This enables skill-succession even after the original skilled workers have retired. However, the modified EBL is only developed in acquiring knowledge from lower level tasks such as ‘pickup’ and ‘place’ but not a whole assembly task.

Chapter 4 improves the modified EBL method developed in Chapter 3. The improved EBL method has two learning modes. The first learning mode is a hierarchical use of the modified EBL presented in Chapter 3 for higher level tasks. The second learning mode can learn from error recovery training examples, which is the novel proposal of the improved EBL method. Evaluation criteria of saving the acquired knowledge are also provided in the improved EBL method. The most significant distinction of the improved EBL method is that the acquired knowledge is represented and saved as different levels of knowledge modules in assembly plans. The knowledge modules in an assembly task not only can be reused in whole, but also can be flexibly reused in part.

Chapter 5 proposes an application of CBR in offline automatically programming for robots in robot cell assembly production by reusing knowledge acquired from human workers. The features and rules used for retrieving different levels of the saved knowledge modules are presented. The method of automatically generating robot programs by reusing the retrieved knowledge modules is provided. The knowledge modules acquired from the same assembly task is saved as a case in the knowledge base. The past learned cases can be reused flexibly in three ways: 1. one case is reused wholly for a new task; 2. one case is partly reused for a new task; 3. several cases are partly reused to synthesize the robot program for a new task in a variant environment.

Chapter 6 discusses the merits of integrating EBL and CBR in constructing the proposed knowledge-based system. The proposed system also can help share knowledge between product designers and manufacturing workers. The method for integrating the product design process and the product manufacturing process by using the proposed system is addressed. This further contributes to reduce the lead time in robot cell production.

Finally, Chapter 7 summarizes this dissertation and points out the directions for future works.

Chapter 2

Methodology of Knowledge Acquisition and Reuse

This chapter introduces the basic concepts and algorithms of the methods used in this study.

Section 2.1 introduces the concept and the algorithm of the knowledge acquisition method: explanation-based learning (EBL).

Section 2.2 presents the framework and the working mechanism of the knowledge reusing method: case-based reasoning (CBR).

Section 2.3 explains how EBL and CBR are applied and integrated for developing the knowledge-based system proposed in this dissertation.

2.1 Explanation-based Learning (EBL)

2.1.1 The Concept of EBL

EBL is a deductive learning method that learns from untutored observation, in which the observed human expert needs no knowledge of the internal workings of the system and is never asked to articulate the information or methods he/she uses [14]. With a pre-encoded knowledge base, called domain theory in EBL, EBL can perform a knowledge-intensive analysis of a single training example [15]. The analysis involves first explaining why the training instance is an example of the concept to be learned, called goal concept, and then generalizing the explanation structure in a principled manner so it can be used to recognize a larger class of instances of the goal concept than the original training instance. A seeming paradox of EBL is that in order to produce its final description of the goal concept, the learning system must possess an initial description of that same concept [16]. In fact, without an initial description of the goal concept, it would be impossible for the system to explain why the given training instance is an example of the goal concept. The way to untangle the paradox is to acknowledge that learning can involve knowledge transformation, as well as knowledge acquisition [17]. EBL methods do not acquire truly “new” knowledge, but rather transform existing knowledge that is unusable or impracticable into a form that is usable [18, 19]. Hirsh described the implementation of EBL within a logic-programming environment [20].

Figure 2.1 shows the structure of EBL. Traditionally, an EBL problem is defined as the following:

Given:

· Goal Concept: A concept definition describing the concept to be learned. (It is assumed that this concept definition fails to satisfy the Operationality Criterion.)

· Training Example: An example of the goal concept.

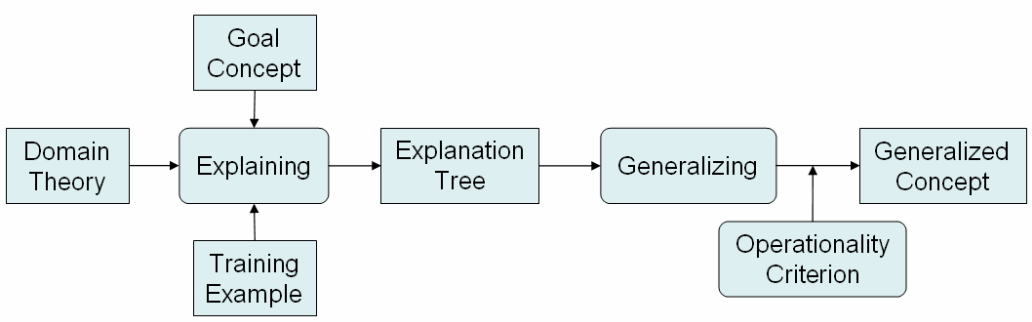

Figure 2.1 The structure of EBL.

· Domain Theory: A set of rules and facts to be used in explaining how the training example is an example of the goal concept.

·Operationality Criterion: A set of predicates or rules specifies the form in which the learn concept definition must be expressed.

Determine:

· A generalization of the training example that is a sufficient concept definition for the goal concept and that satisfies the operationality criterion.

The EBL method has the following two steps [15]:

1. Explain: Construct an explanation in terms of the domain theory that proves how the training example satisfies the goal concept definition.

· This explanation must be constructed so that each branch of the explanation structure terminates in an expression that satisfies the operationality criterion.

2. Generalize: Determine a set of sufficient conditions under which the explanation structure holds, stated in terms that satisfy the operationality criterion.

· This is accomplished by regressing the goal concept through the explanation structure. The conjunction of the resulting regressed expressions constitutes the desired concept definition.

2.1.2 Operationality Criterion

Operationality criterion is a key concept in EBL, which determines the utility of the new learned knowledge. Operationality criterion is one of the inputs in the Mitchell's system [15], and is defined as: operationality criterion specifies the form in which the learned concept definition must be expressed; the concept definition must be expressed in terms of the predicates used to describe the training example or other selected, easily evaluated, predicates from the domain theory. DeJong and Mooney [14] argued that there were two problems with Mitchell's definition of operationality criterion: 1. The process of specifying the operationality criterion for a particular domain is not itself operational; there is no mention of how these predicates are actually selected or how easy their evaluation must be to meet the operationality criterion. 2. The specification of predicates alone will not insure ease of evaluation.

DeJong and Mooney gave their solution as the following [14]. It should be acknowledged that EBL system has two logically distinct components: a learning element and a performance element. The purpose of learning is to improve the processing ability of the performance element. The notion of operationality must be judged with respect to the abilities of a particular performance element. While both the learning and the performance element operate on the same knowledge base, they reflect the world knowledge in subtly different ways. In DeJong and Mooney's EBL system which learns schemata, the operationality criterion is a set of goals that the system could easily achieve using normal means by simply instantiating an existing schema. There are two important implications of this definition: 1. the operationality criterion is derivable from the system's performance and, therefore, should not be independently input to the system, 2. The operationality criterion is dynamic, not static; as the system learns new schemata, additional goals become operational since the system can use the new schemata as building blocks to construct future explanation.

Segre built the ARMS system [21-23] on DeJong and Mooney's EBL model which learns schemata [14], and examined the operationality/generality trade-off and how it affected performance of EBL systems [24]: If the newly acquired knowledge structure is available to the system, it is said to be operational. If the new structure does not improve the performance of the system in some fashion, it is not worth learning. However, not all operational knowledge is created equal: the cost of using the new knowledge structure is called operationality. The more operational the structure, the easier (e.g., less expensive) it is for the system to apply it. The diversity of examples covered by the new structure is directly related to its generality: likely to be useful. The more general the structure, the more expensive its application tends to be.

Keller redefined operationality criterion in terms of performance system that uses the learned concept description, and in terms of the criteria for evaluating that system's performance [17]. The revised operationality definition is: given a concept description, a performance system that makes use of the description to improve performance, and performance objectives specifying the type and extent of the system improvement desired; then the concept description is considered operational if it satisfies the following two requirements: 1. Usability - the description must be usable by the performance system, 2. Utility - when the description is used by the performance system, the system's performance must improve in accordance with the specified objectives.

2.1.3 An Example of EBL Application

Figure 2.2 shows the work flow of the EBL algorithm, mainly on how the explanation structure is constructed.

Safe-To-Stack is an example used by Mitchell [12] to illustrate the EBL algorithm. The task is to learn to recognize pairs of objects <OBJ1, OBJ2> such that it is safe to stack OBJ1 on top of OBJ2

(In this example, the uppercase items such as OBJ1 and numbers are constants; the lowercase items such as x, y are variables).

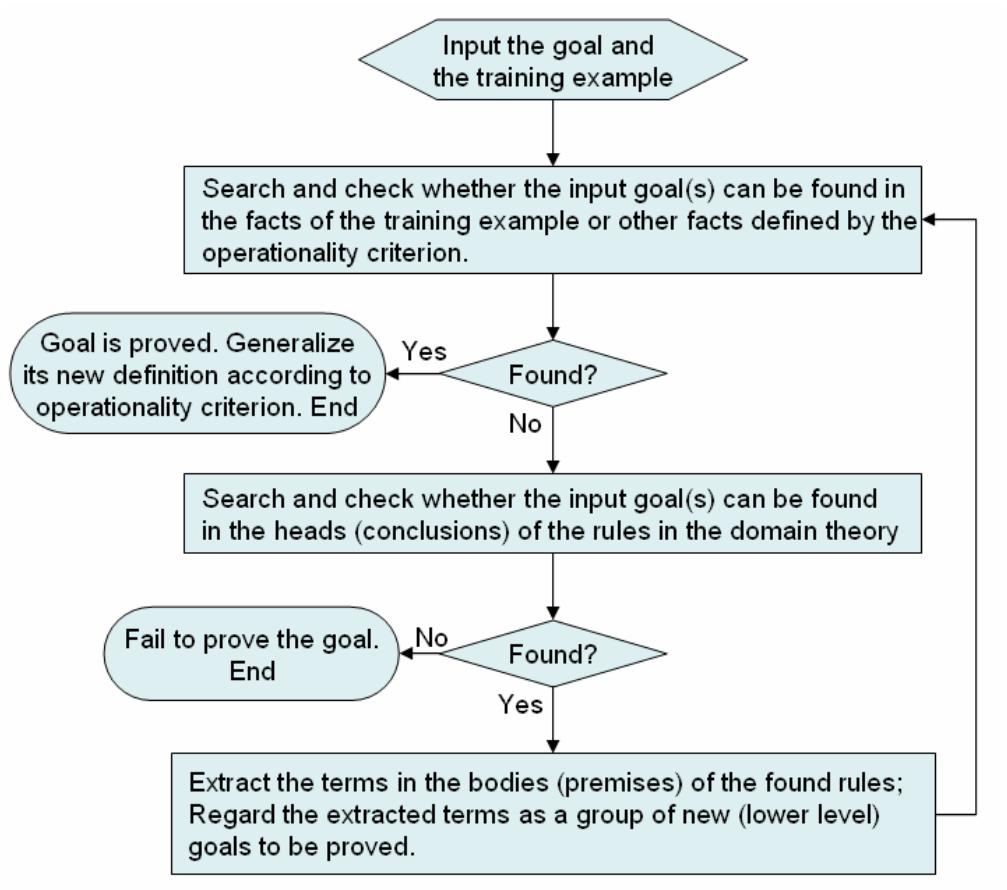

Figure 2.2 Flowchart of the EBL algorithm.

The problem is given as the following:

Given:

•Goal Concept: Pairs of objects <x, y> such that Safe-To-Stack (x, y).

• Training Example:

On (OBJ 1, OBJ2)

Type (OBJ 1, BOX)

Type (OBJ2, ENDTABLE)

Color (OBJ1, RED)

Color (OBJ2, BLUE)

Volume (OBJ 1, 2)

Density (OBJ 1, 0.3)

• Domain Theory:

Lighter (x, y) => Safe-To-Stack (x, y)

Volume (x, vx) & Density (x,dx) & Equal (wx,vx*dx) => Weight (wx, vx*dx)

Weight (x, wx) & Weight (y, wy) & Less-Than (wx, wy) => Lighter (x, y)

Type (y, ENDTABLE) => Weight (y, 5) (default)

Less-Than(0.6, 5)

• Operationality Criterion: The concept definition must be expressed in terms of the predicates used to describe examples (e.g., Volume, Color, Density) or other selected, easily evaluated, predicates from the domain theory (e.g., Less-Than).

Determine:

• A generalization of training example that is a sufficient concept definition for the goal concept and that satisfies the operationality criterion.

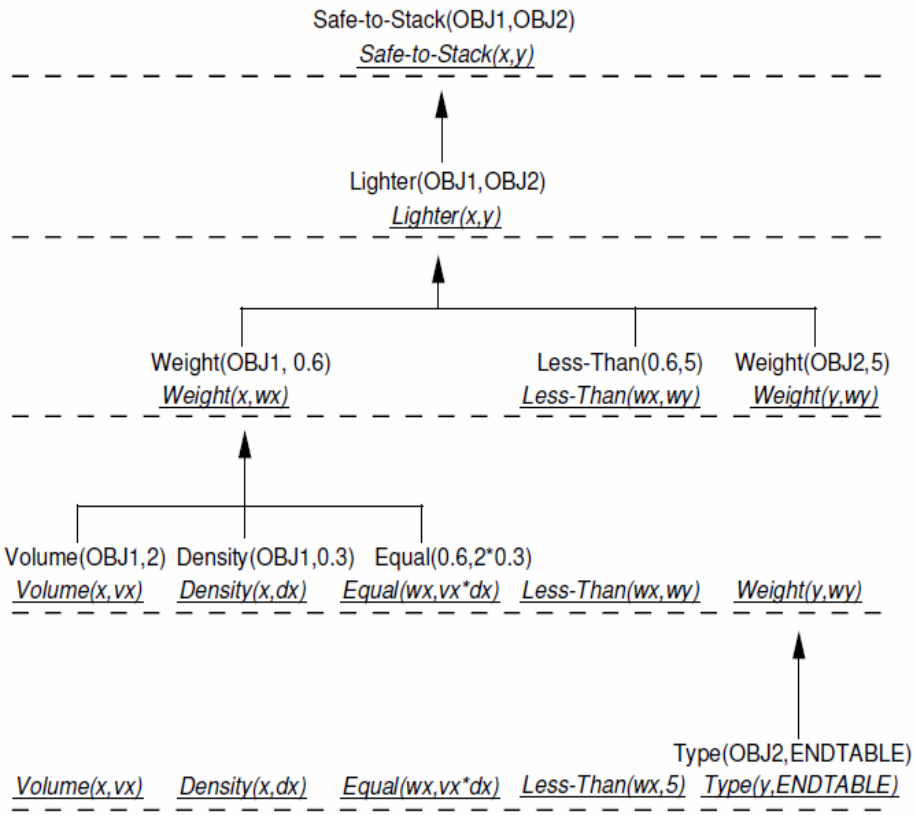

Figure 2.3 Learning procedure of 'Safe-to-Stack'.

Figure 2.3 shows how the operational concept definition for the goal concept Sate-To-Stack(x, y) is generalized from a specific example that OBJ1 is safe to stack on OBJ2.

In Figure 2.3, there are two layers of predicates above each dashed line. The first layer that is not underlined is the specific explanation tree for the goal concept. The second layer that is underlined is the generalized explanation tree for the goal concept. The first layer and the second layer of the explanation structure correspond to the first step and the second step in the EBL method.

The first step is to construct an explanation of how the training example satisfies the goal concept.

As shown in the first layer in figure 3, the pair of objects <OBJ1, OBJ2> satisfies the goal concept Safe-To-Stack because OBJ1 is Lighter than OBJ2. Furthermore, this is known because the Weights of OBJ1 and OBJ2 can be inferred. For OBJ1, the Weight is inferred from its Density and Volume, whereas for OBJ2 the Weight is inferred based on a rule that specifies the default weight of ENDTABLEs in general. Through this chain of inferences, the first layer of the explanation structure demonstrates how OBJ1 and OBJ2 satisfy the goal concept definition. Note that the specific explanation tree has been constructed so that each of its branches terminates in an expression that satisfies the operationality criterion (e.g., Volume (OBJ1, 2), Less-Than (0.6, 5)).

In order to determine general sufficient conditions under which the explanation holds, the second step involves regressing (back propagating) the goal concept step by step back through the explanation structure. The second layer in figure 3 illustrates the second step of Mitchell's EBL method in the context of the Safe-To-Stack example. In the topmost step of this layer, the goal concept expression Safe-To-Stack (x, y) is regressed through the rule Lighter (x, y) => Safe-To-Stack (x, y) to determine that Lighter (x, y) is a sufficient condition for inferring Safe-To-Stack (x, y). Similarly, regressing Lighter (x, y) through the next step in the explanation structure yields the expression Weight (x, wx) & Weight (y, wy) & Less-Than (wx, wy). This expression is in turn regressed through the final steps of the explanation structure to yield the operational definition for Safe-To-Stack (x, y). To illustrate the goal regression process in greater detail, consider the final steps of figure 3 in which the expression Weight (x, wx) & Weight (y, wy) & Less-Than (wx, wy) is regressed through the final steps of the explanation structure. Each conjunct of the expression is regressed separately through the appropriate rule, in the following way. The conjunct is unified (matched) with the consequent (right-hand side) of the rule to yield some set of substitutions (particular variable bindings). The substitution consistent with the example is then applied to the antecedent (left-hand side) of the rule to yield the resulting regressed expression. Any conjuncts of the original expression which cannot be unified with the consequent of any rule are simply added to the resulting regressed expression (with the substitutions applied to them). As illustrated in the figure, regressing the conjunct Weight (x, wx) through the rule Volume (x, vx) & Density (x, dx) & Equal (wx,vx*dx) => Weight (x, vx*dx) therefore yields Volume (x, vx) & Density (x, dx) & Equal (wx, vx*dx). Regressing the conjunct Weight (y, wy) through the rule Type (y, ENDTABLE) => Weight (y, wy) yields Type (y, ENDTABLE). Finally, since no rule consequent can be unified with the conjunct Less-Than (wx, wy), this conjunct is simply added to the resulting regressed expression after applying the substitutions produced by regressing the default rule Type (y, ENDTABLE) => Weight (y, 5), which yield the conjunct Less-Than (wx, 5). The final, operational definition for Safe-To-Stack (x, y) is therefore:

Volume (x, vx) & Density (x, dx) & Equal (wx, vx*dx) & Less-Than (wx, 5) & TYPE (y, ENDTABLE) => Safe-To-Stack (x, y).

This expression represents a justified generalization of the training example, for which the explanation structure serves as a justification. It is the generalized operational concept definition for the goal concept Safe-To-Stack (x, y), and can be used by other similar examples.

2.2 Case-based Reasoning (CBR)

2.2.1 The Concept of CBR

CBR is the process of solving new problems based on the solutions of similar past problems [52-58]. Schank and Jona [59] described the nature of CBR as follows: “Most people prefer not to have to think hard if they can help it. They will try to get by with whatever worked before, even if it is less than optimal. We believe that, roughly speaking, people's everyday cognition consists of about 90% retrieving of past solutions and only about 10% or less of actual novel problem solving.”

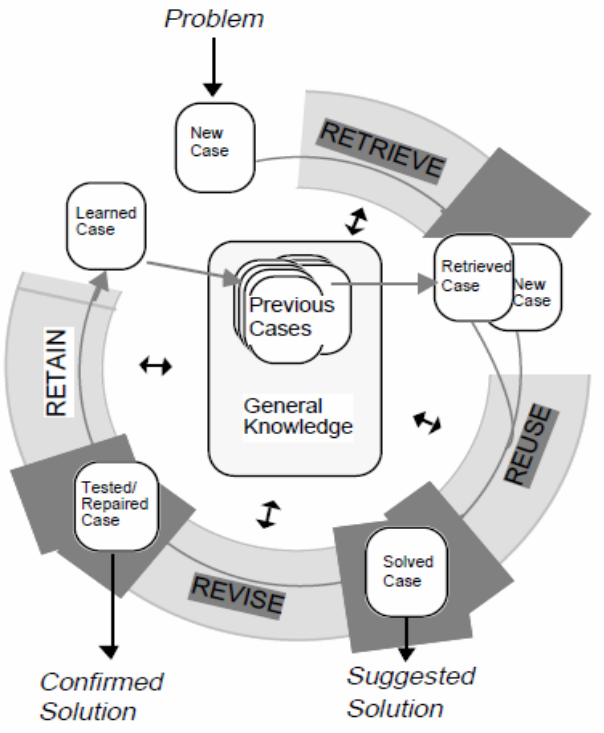

Figure 2.4 The working process of CBR.

Figure 2.4 illustrates the work process of CBR. The core of any CBR system is a knowledge base. It is composed of a case base and a general knowledge base. The case base contains previous problem-solving cases. It saves the past problem-solving experiences and knowledge in the form of cases. The general knowledge base holds techniques in the form of rules or algorithms that are used in the 4 Rs in a CBR cycle. 4 Rs (i.e. retrieve, reuse, revise, retain) are the four key steps and functions of a CBR system:

1. Retrieve: Given a target problem, retrieve cases from case base that are relevant to solving it. A

case consists of a problem, its solution, and, typically, annotations about how the solution was derived.

2. Reuse: Map the solution from the previous case to the target problem. This may involve adapting the solution as needed to fit the new situation.

3. Revise: Having mapped the previous solution to the target situation, test the new solution in the real world (or a simulation) and, if necessary (e.g., an unexpected effect or error occurs), revise.

4. Retain: After the solution has been successfully adapted to the target problem, store the whole resulting experience as a new case in case base.

2.2.2 An Example of CBR Application

Case-based Menu Planner (CAMP) developed by Marling, et al. is a pure case-based planner [25]. CAMP employs the case-based reasoning (CBR) technique to suggest menus to users. The heart of a CBR system is its case base. CAMP's case base holds dozens of daily menus that were compiled from reputable sources and modified as needed to ensure that they satisfy the RDIs (Reference Daily Intakes) and the Dietary Guidelines of Americans and Aesthetic standards. In a CBR system, a case contains a past solution and the features that indicate when the solution is likely to be useful again. A solution in CAMP is a daily menu. Features that indicate a menus' usefulness are: its nutrient vector, the types of meals and number of snacks included, and included foods.

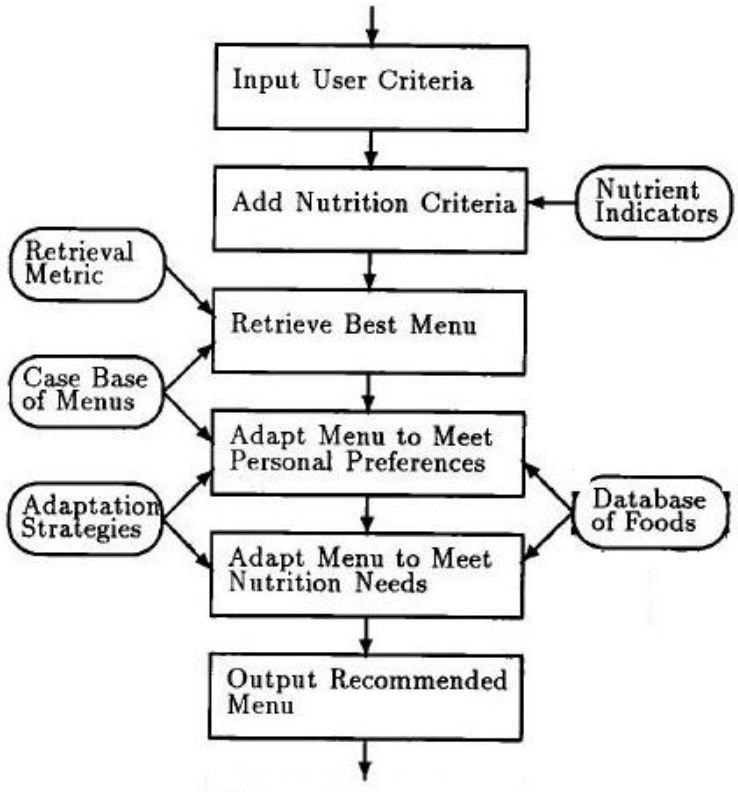

Figure 2.5 A flow chart for CAMP [25].

CAMP operates by retrieving and adapting daily menus from its case base. A flow chart for CAMP is show in Figure 2.5. An individual’s calorie level and any optional nutrition and personal preference criteria are input first. Nutrition criteria are added to ensure that the RDIs are met. The menu best suiting the criteria is retrieved from the case base.

A reusability metric is used to select and retrieve a case based on the ease of adapting it to meet current goals. Before a case can be reused in CAMP, it must be adapted until it meets all user-specified constraints, plus additional constraints imposed as minimum RDIs. To find the best case, CAMP checks each case against all constraints. Any case meeting all constraints constitutes an exact match and is retrieved. When a case does not comply with a constraint, a penalty score is assigned based on how difficult it would be to bring the case into compliance. The penalty scores were determined through consultation with the expert and refined through trial and error. CAMP finds the case that is easiest to adapt, striking a balance between the number and severity of constraint violations.

CAMP's retrieval algorithm is shown as the below:

For each case in the case base do: Apply the reusability metric to obtain a penalty score;

Sort cases by penalty score;

Retrieve the case with the lowest penalty score.

A detailed account of case retrieval in CAMP is available in [26].

If the best case, as determined by the reusability metric, is not an exact match, it is adapted until it complies with any unmet constraints. Adaptation is generally considered to be the most difficult part of CBR. The primary case adaptation methods are substitution, transformation, and derivational analogy [27]. In substitution, replacements are found for parts of an old solution which do not suit current needs. In transformation, individual components of an old solution are modified to suit the current problem. In derivational analogy, as exemplified by PRODIGY/ANALOGY [28], an old problem-solving method, rather than an old problem solution, is reused. CAMP employs substitution and transformation during adaptation.

CAMP's adaptation framework, based on the expert's manual approach to adapting menus, is as follows:

1. Check the number of snacks. Adjust, if necessary.
2. Check meal types. Swap meals to accommodate preferences, if necessary.
3. Eliminate any forbidden food items.
4. Check calorie level. Adjust serving sizes, if necessary.
5. Fix any nutrient specific deficiencies.

CAMP's database plays two roles in case adaptation. First, it maintains the small, medium, and large serving sizes for each food, used to adjust serving sizes. Second, it maintains the nutrient vector for each food, used to calculate the effects of making changes to the menu.

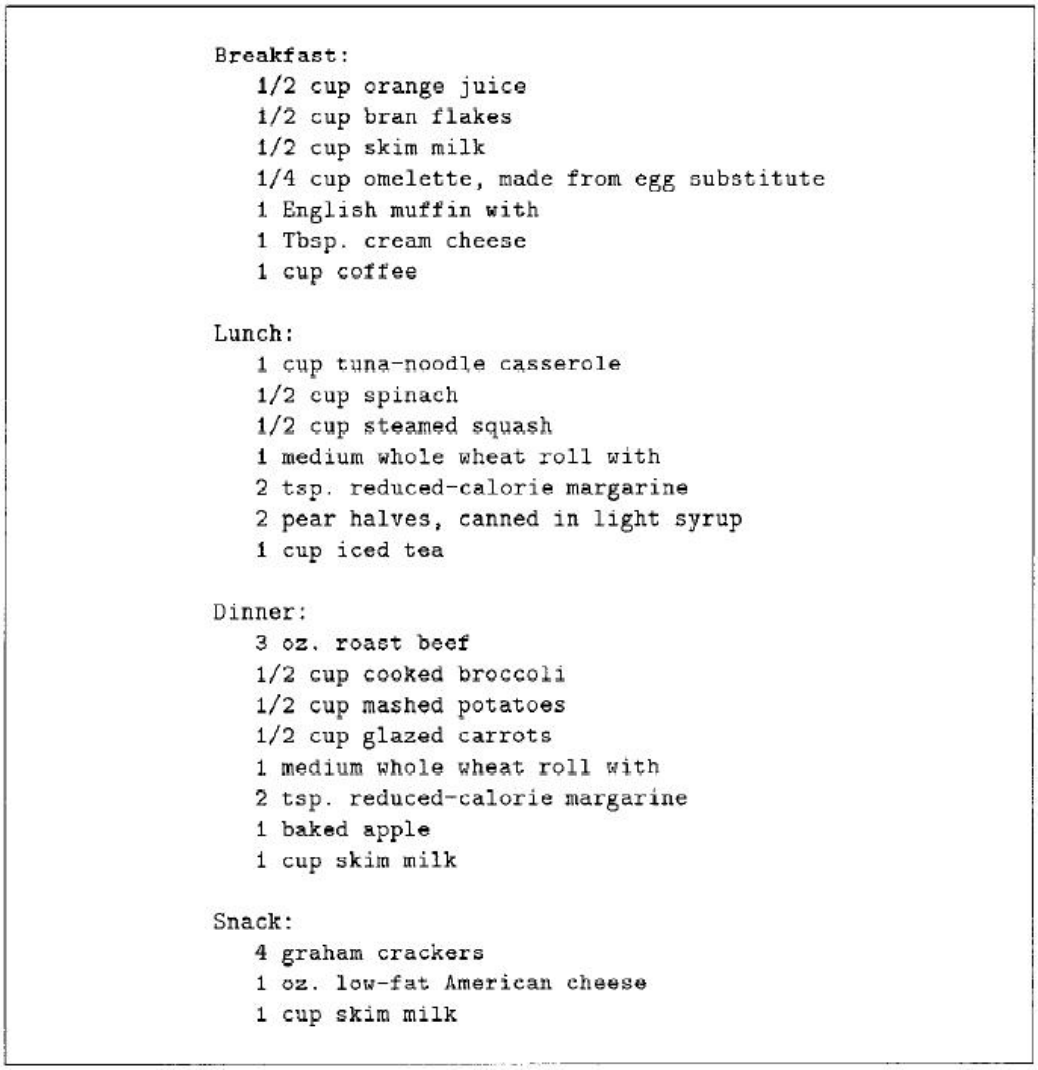

Figure 2.6 A menu planned by CAMP [25].

The adapted menu becomes the system output. Figure 2.6 shows a representative menu planned by CAMP. The menu shown was constrained by the user to include one snack, a total of 1800 to 2200 calories, at least 800 mg of calcium, and no more than 30% of calories from fat. It provides one snack, 2109 calories, 1557 mg of calcium, and 26% of calories from fat, meeting all constraints.

2.3 The Knowledge Acquisition and Reuse Method in This Dissertation

In this dissertation, the EBL method is used to acquire knowledge and the CBR method is employed to reuse the acquired knowledge. The originality of this dissertation lies in its novel development of a method for hierarchical knowledge modularization in the robotic manufacturing domain by integrating EBL and CBR, as shown in Figure 2.7. Hierarchical knowledge modularization means that a knowledge-based system acquires and represents the knowledge implemented in a robotic manufacturing task in terms of a hierarchical set (i.e., different levels) of

knowledge modules, organizes these knowledge modules as a plan for the observed task, and saves the plan and the task description as a case in the case base of the system. In this way, the system can flexibly retrieve a case or modules of several cases and reuse it or them for a new task. In other words, the system can re-organize different levels of knowledge modules in different cases to generate the solution for a new task.

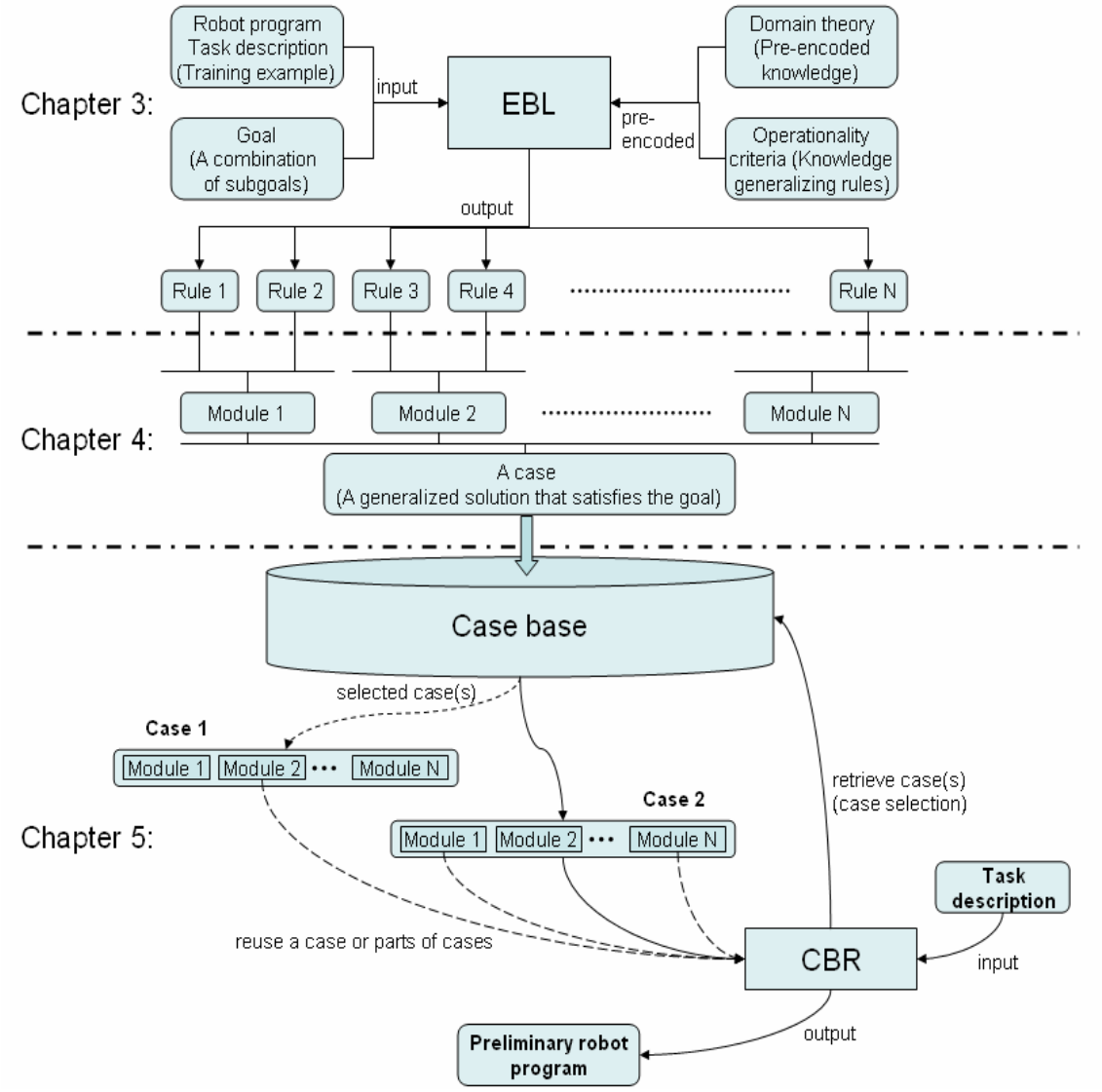

Figure 2.7 The knowledge acquisition and reuse method by integrating EBL and CBR.

In this dissertation, the traditional EBL is used to generalize the rule for a lowest level subgoal from the training example (Chapter 3). By the hierarchically multiple use of the traditional EBL method for different levels of goal and subgoal, the knowledge in the training example can be learned and generalized as rules and a hierarchical set of knowledge modules (Chapter 4). The reason for this novel hierarchical application of EBL is that the tasks in robotic manufacturing should be and can be hierarchically analyzed and divided [10]. In addition, in this dissertation, a new

definition of operationality criteria for the EBL method is proposed, which requires the acquired knowledge should be represented in a way that satisfies two requirements. The first is operability in that it needs to be easy to transform into robot programs (i.e., commands and parameters). The second is understandability in that it needs to be easy for human workers to understand the knowledge implemented in the robot program.

CBR is employed for knowledge reuse (Chapter 5). Its main task is to generate the preliminary robot program for a new input task by reusing a cases or modules of several cases in the case base (i.e., a part of the knowledge base). In this dissertation, two modifications are made based on the traditional CBR method. The first is that EBL is used to acquire new cases. The second is that a past case not only can be reused in whole, but also it can be reused in part by synthesizing different modules of several cases to generate the program for a new complex task in a varying environment.

In this dissertation, the novel concept of hierarchical knowledge modularization has been proposed. The fundamental method for knowledge acquisition and reuse to realize this concept has been developed. The future study will focus on the following issues. 1. How to integrate other machine learning method with EBL to acquire knowledge. This is because EBL is good at learning strategic knowledge for high level abstract tasks, while other methods such as reinforcement learning is adept in discovering basic knowledge for low level specific tasks. 2. How to apply hierarchical knowledge modularization in fault analysis and error recovery. Hierarchical knowledge modularization eases identifying the location of the error by specifying and organizing learned knowledge in different levels of knowledge modules and is able to provide solution support from the perspective of knowledge reuse. 3. How to develop a better human-system interaction interface to facilitate human to add or revise knowledge modules in the knowledge base.

Chapter 3

The Construction of the Knowledge-based System

3.1 Introduction

This chapter presents the overall structure of the knowledge-based system (KBS) and the modified EBL method proposed in this dissertation. The modified EBL is used in learning tacit knowledge from robot programs of lower level tasks such as ‘pickup’ and ‘place’ in the robotic assembly domain. With the modified EBL, a new operationality criterion is proposed for the generalized tacit knowledge, which demands that it should be expressed in two ways: 1. easily understandable by human workers, and 2. reusable by robots to generate program automatically.

Section 3.2 explains what the tacit knowledge is in the robotic assembly domain.

Section 3.3 explains why EBL is used in this research.

Section 3.4 describes the overall structure of the proposed KBS and the construction of the domain theory.

Section 3.5 presents the leaning mechanism of the modified EBL method with a concrete example.

Section 3.6 gives the experiments and analysis to explain the limitation of the proposed method.

Section 3.7 compares the proposed method with other methods and presents the importance of learning tacit knowledge.

Section 3.8 gives the conclusion

3.2 Tacit Knowledge in Robotic Assembly

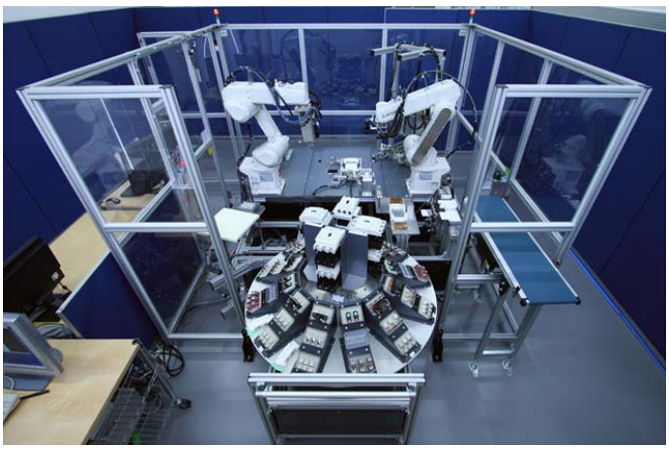

Figure 3.1 Robotic assembly work cell [30].

As shown in Figure 3.1, robotic assembly refers to using two or more robots in a work cell to complete the assembly of a wide variety of products that are made of various workpieces.

In the robotic assembly domain, whether workers are moving a robotic arm to pick up or place a workpiece, the task is non-trivial, and a number of contingencies could arise, each affecting the applicability or outcome of the robotic program. Thus, the nuances of the real world necessitate that workers have to be familiar with the physical properties of workpieces in a particular situation as well as their kinematic characteristics when they are moved by robots in high speed. In addition, workers have to deal with various environment situations. For example, suppose that the task is to palletize a group of blocks in a plate, as shown in Figure 3.2. When placing the first block into the plate, to make a stable insertion, the approach position of the block should be close to the target hole. In placing the second block, the robotic operation seems to be the same as the first one. However, to avoid collision with the first palletized block, an intermediate approach point is added, which makes the second block first approach the target hole at a position above the palletized block before approaching at a closer position to its destination.

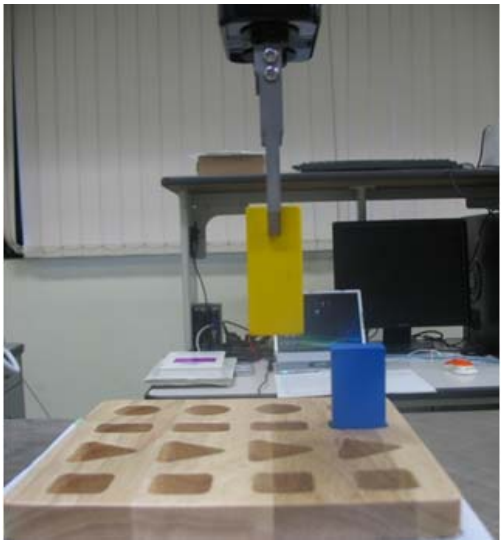

Figure 3.2 Example: adding an intermediate point.

It is difficult to observe these types of knowledge explicitly in robotic operations. However, such knowledge is indeed implemented in the form of robotic programs by inserting intermediate points, by adjusting the specific positions and velocities of the robots, by inserting extra commands to adjust the timing of the operations, and so on. Experienced workers could interpret such programs and selectively recognize which parts of the program are critical for its successful performance. However, for less-experienced workers, it would be quite difficult to understand robotic programs made by skilled workers, which gives rise to the problem of skill-succession.

Such assembly skills and experiences that are implemented within robotic programs are the tacit knowledge owned by skilled workers. The efficiency of a robotic assembly work cell could be dramatically improved if the tacit knowledge of skilled workers could be shared with other workers and transferred to robots. In this paper, we propose a knowledge-based system based on EBL to share and reuse the tacit knowledge of skilled workers.

3.3 The Reason for Using EBL in Knowledge Acquisition

The basic idea of EBL is that with sufficient background knowledge, humans appear to learn quite a lot from one example and use the learned results to guide their problem solving efforts next time around. Two features of EBL make it appropriate for learning tacit knowledge in the robotic assembly domain: 1. EBL can learn from a single example. This feature is very important, because it is difficult to obtain many training examples in the robotic assembly domain due to the long teaching time. Moreover, each robot teaching example usually contains distinguished knowledge, specific to its unique assembly situation. 2. EBL can generalize from an example without human tutoring. This feature makes EBL appropriate for learning tacit knowledge. However, in the proposed KBS, to adapt EBL to acquiring knowledge from robotic programs, two modifications should be made to the construction of the domain theory, and the definition of the operationality criterion.

Segre has first applied EBL in the robotic manufacturing domain in his ARMS system [21-23]. The ARMS was developed on a simulated environment, and was aimed to construct an autonomous system. Different from the ARMS system, the KBS proposed in this study is built on real robotic manufacturing systems. Furthermore, the proposed system emphasizes knowledge sharing between human workers and robots, besides the function of automatically reusing the learned knowledge in similar tasks. Thus, a new operationality criterion is proposed within the EBL framework, which demands that the learned knowledge should be expressed in two ways: 1. easily understandable by human workers, and 2. reusable by robots to generate program automatically.

In addition to EBL, many other methods have also been applied in a variety of robotic manufacturing problems for acquiring and reusing expert knowledge. According to the way how the expert knowledge is acquired and reused, we classify these methods into three types: self-adapting (SA) method, learning from demonstration (LfD) method, and knowledge based (KB) method.

In the recent 2 decades, the SA method has prevailed in the AI area and in the robotic manufacturing domain. Nuttin [31] and Prabhu [32] used reinforcement learning (RL) and artificial neural networks (ANN) in designing learning controllers respectively for peg-into-hole assembly task and deburring task. Lopez-Juarez [33] and Chen [34] applied ANN and online-learning method in peg-into-hole assembly. Noda [35, 36] developed an active search algorithm based on active learning for optimizing robot motion trajectory. In the SA method, the researchers use their prior knowledge to model the objective system/problem. Then the parameters in their models are refined

or optimized by responding to feedback of the online robot operations.

Within LfD method, a state-action mapping policy is learned from examples or demonstrations provided by a teacher. Then the learned policy is reused to guide the robot operation in future tasks. Argall [4] made a survey on LfD and categorized LfD research in terms of demonstrator, problem space, policy derivation and performance. Hovland [37] employed hidden Markov model (HMM) to transfer human peg-into-hole assembly skill to robot based on a set of training data gathered from human. Friedrich [38] developed an extended programming by demonstration method (PbD) to create a plan for a given task not only on the basis of the given demonstration data. Additionally the user is asked for the intention he followed with the demonstration. Therewith a generalized plan from a single demonstration can be reused for a whole set of tasks. The idea is the same as EBL. Dillmann [39] improved the PbD method, in which the goal is to modify information gained by the demonstration in that way that different target systems are supported.

In the KB method, human knowledge is encoded as facts and rules in a knowledge base. And task plans can be generated based on the knowledge base. Hwang [40] developed a knowledge based framework to support task-level programming and operational control of robots. Fujita [41] employed the KB method in the assembly shop process design support system of his company.

Generally, the SA method requires a large number of training data or experiments; the LfD method concentrates on making a robot mimic human operations without explaining the know-how in the operations; the KB method focuses on planning for a task by existing rules in a knowledge base. In this study, the aim is to acquire the tacit know-how from a single robot teaching example of human workers. This is because each robot teaching example contains the unique know-how of the human teacher. Therefore, in the proposed method, a modified EBL is used to explain the operation of the human worker and to generalize it as a rule, and the KB method is used to generate program for new similar tasks.

3.4 The Structure of the KBS

As Figure 3.3 shows, the proposed KBS works as a medium between workers and the robotic system in a work cell. Workers interact with the KBS through a human-system interface. The KBS is mainly composed of the learning part, the planning part, and the domain theory.

3.4.1 The Learning Part

The learning part consists of the explainer and the generalizer. The function of the learning part is to acquire knowledge from robotic teaching demonstrations of skilled workers. The inputs of the learning part are a task and its corresponding training example, i.e., the robot program for the task. Its output is the acquired knowledge for the task.

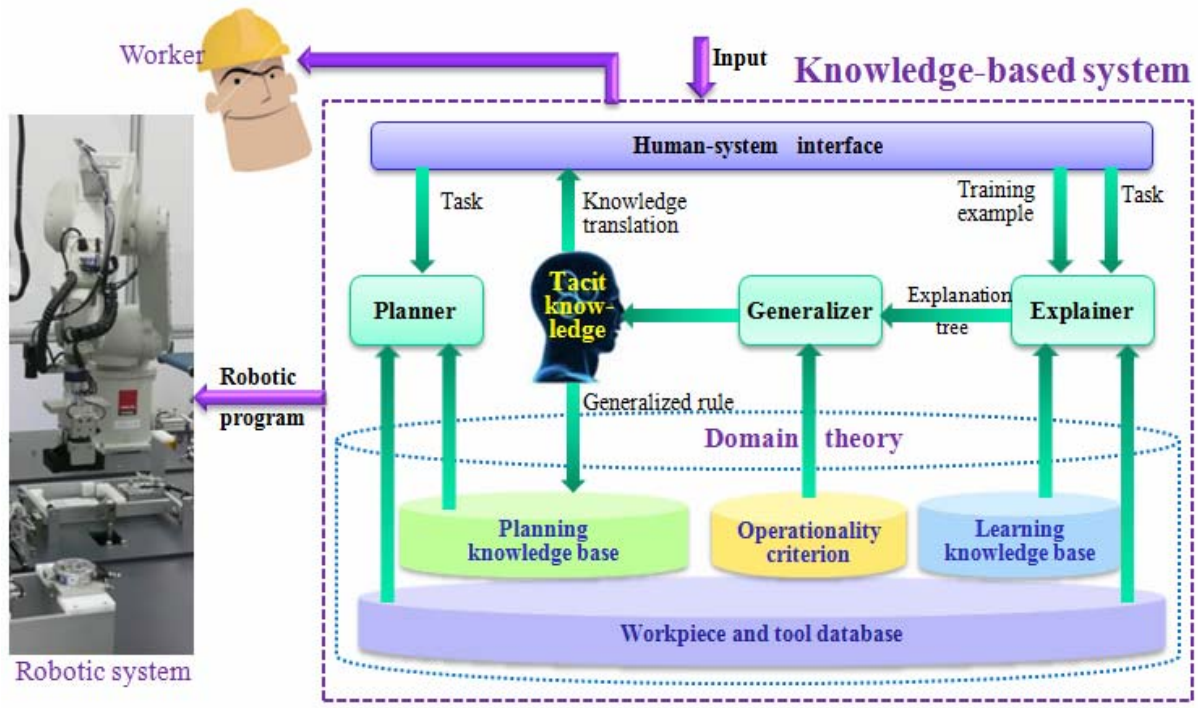

Figure 3.3 The structure of the knowledge-based system.

In the KBS, the tasks are classified into 2 types: 1. Pick-up, and 2. Placing. This classification is based on the following considerations. The assembly process of any workpiece can be segmented into the pick-up phase and the placing phase. In factories, workers also program and teach robots for the pick-up and the placing operations of a workpiece separately. In fact, this classification of the tasks is the classification of the knowledge implemented by workers. Thus, classifying the tasks in this way can make the acquired knowledge more flexible and reusable for workers in actual factory environments.

As Figure 3.3 shows, a task and its corresponding training example are input into the explainer of the learning part. The explainer explains how the training example accomplishes the task by using the rules in the learning knowledge base and the data retrieved from the workpiece and tool database. Its result is an explanation tree. Then the generalizer generalizes knowledge from the explanation tree according to the operationality criterion. The generalizer outputs the generalized knowledge in two ways. One represents the knowledge in terms that is easily understood by workers, i.e., the knowledge translation shown to workers through the human-system interface. The other is a generalized rule that can be used by the planning part, and is saved into the planning knowledge base.

3.4.2 The Planning Part

The planning part is the planner in Figure 3.3. Its function is to generate robot programs for input tasks. The planner generates robot programs by using rules in the planning knowledge base and data

in the workpiece and tool database. The output robot programs will be sent to the robot system.

3.4.3 The Domain Theory

The domain theory, i.e., the knowledge base, is the most important part of the KBS. It is composed of the learning knowledge base (LKB), the operationality criterion (OC), the planning knowledge base (PKB), and the workpiece and tool database (WTD).

The WTD provides data on workpieces and robot tools that can be used by both the learning part and the planning part. For example, the block data are given in the form:

wp(ID,Type-Subtype,[Height,Length,Width])[1].

The following is a specific example:

wp(5,block-1,[49.5,24.5,14.5]).

The OC is a set of language processing rules, which is pre-encoded in the domain theory. It defines the terms in which the knowledge learned from the learning part should be expressed. It requires the learned knowledge to be expressed in two ways: 1. a representation in natural language that can be easily understood by workers; and 2. a rule formulated with predicates from the planning part, i.e. a rule that can be reused by the planning part.

The LKB and the PKB are the core of the KBS, as they the parts that contain knowledge.

3.4.4 The Learning Knowledge Base (LKB)

The LKB is composed of a base of basic learning rules and a base of analysis rules.

The training example is composed of robotic commands and parameters. To analyze the training example, the learning part must understand the meanings of the robotic commands. The basic learning rules are the rules that explain each robotic command or each commonly used combination of robotic commands. For example, consider the rule

```
grip(T1,HN,T2):-
    cmd(N1,dly(T1)),
    cmd(N2,hclose(HN)),
    cmd(N3,dly(T2)),
    T1>0.2, T2>0.2,
    N2=:=N1+1,
    N3=:=N2+1.
```

It means that the action of gripping is realized by a combination of three commands ‘dly(T1)’, ‘hclose(HN)’, and ‘dly(T2)’. ‘T1>0.2’, ‘T2>0.2’ mean that the delay time should be longer than 0.2

[1] In the following descriptions of predicates, characters initiated with capital letters or upper case characters represent variables, while lower case characters are constants.

second. ‘N2=:=N1+1’, ‘N3=:=N2+1’ mean that the sequence of the three commands is ‘dly(T1)’, ‘hclose(HN)’, and ‘dly(T2)’. In other words, the gripping action is achieved by closing a robot tool, and there should be delays both before and after closing the robot tool. This combination of commands is a rule/operation that is commonly used in writing robotic programs for seizing workpieces. Thus it is pre-encoded in the domain theory.

The analysis rules are used to analyze the work cell state of a task and the corresponding tacit knowledge of skilled workers implemented in the training example. There are three main kinds of rules in the base of analysis rules:

1. Rules for analyzing work cell state data, which are mainly used to analyze the robot movement speed in automatic mode, the relationship between robot tool and target workpiece, and the environment state nearby the assembly destination.
2. Rules for analyzing parameters, which are used to explain how human workers chose the coordinates of the seizing, placing, and approach points for a workpiece.
3. Rules for analyzing robot commands, which are used to analyze which operation strategy (i.e., robotic program schema) is used in the robot program in a training example.

There are four types of operation strategies:

1. Essential strategies – There are two essential strategies for pickup and placing tasks respectively. They are almost the same, and the only one difference is whether the command to the robot tool is ‘close gripper’ or ‘open gripper’: “move to safe point, approach the target point, get at the target point, close/open gripper, retreat to approach point, move back to safe point.”
2. Time-delay strategies – To make an assembly operation stable, or to coordinate cooperation between a robot and other facilities, sometimes it is necessary to add time delays ‘dly N’ into an essential strategy. These are the time-delay strategies. For example, if a pickup task requires high precision, to make the robot tool stable in high-speed motion, it needs to delay 1 second before the tool gets to the target point.
3. Speed-changing strategies – These strategies are used for adjusting the speed of a robot tool by adding a speed-changing command ‘ovrd’ into an essential strategy. For example, in a placing task ‘screw a pin’, to make the pin align precisely with the target hole, the robot tool must move the pin at a very low speed from the approach point to the hole. However, in other parts of the trajectory, the robot tool does not need to move at such a low speed. In other words, to accomplish a task both stably and efficiently, speed-changing strategies play an important role.
4. Intermediate-point-adding strategies – Usually, the environment state in a work cell is complicated. In any a pick-up or placing task, to avoid collisions, it is often necessary to add intermediate points into an essential strategy.

All of the rules and strategies in the LKB are explicit knowledge that we can generalize from the operation manual of robots or common sense. Thus, we summarize them and encode them to construct the LKB, which is the most important part of the KBS.

On the other hand, the tacit knowledge of skilled workers is contained in their robotic programs, i.e., the training examples. It is how skilled workers selectively combine the above rules and strategies to construct a complicated operation strategy for a pickup/placing task. In most cases, the pickup/placing task is given in a particular environment or with particular requirements. In other words, the tacit knowledge of skilled workers can be generalized as a rule: Given the features/ requirements of a particular pickup/placing task, the rule determines which type of complicated operation strategy should be selected.

In brief, the LKB provides general knowledge of robotic assembly in the form of discrete rules and strategies. It is used by the learning part to analyze the input tasks and training examples. The output of the learning part is the acquired tacit knowledge, which is generalized as a rule and saved in the PKB.

3.4.5 The Planning Knowledge Base (PKB)

The PKB is composed of a basic planning base and a planning rule base.

The basic planning base is composed of 2 parts:

1. Robotic program schemata – Most robotic programs are composed of several staple commands (e.g., ‘mov’, ‘mvs’, ‘dly’, ‘ovrd’, ‘hclose’, ‘hopen’, etc.). In addition, robots are often operated within routine schemata. Thus, it is possible to summarize all the potential complicated operation strategies for the pickup/placing tasks in the robotic assembly domain. Hence, the robotic program schemata of the potential complicated operation strategies are generalized and encoded in the basic planning base.
2. Parameter calculation rules – These rules are used to calculate the exact coordinates of point parameters in the robotic program.

The planning rule base contains two main kinds of rules:

1. Rules for analyzing tasks – When a task is input into the planning part, the planning part first analyzes the features or requirements of the task with these rules. Then, it uses the rules learned from the learning part to select a robotic program schema from the planning basic base. Subsequently, given the state data in the input task, it uses the parameter calculation rules to calculate the point coordinates, and generates the robotic program that can be used by the robotic system. The features and requirements of a task refer to the type of the target workpiece, the type of the robot tool, the relationship between the workpiece and the tool, the speed with which the robot is required to run in automatic mode, etc.

2. Rules learned from the learning part – In fact, most of the complicated operation strategies (i.e., robotic program schemata) in the robotic assembly domain can be generalized and encoded in the domain theory of the KBS. They are explicit knowledge. The tacit knowledge of skilled workers is in knowing how to select an appropriate operation strategy and use it flexibly for a particular task. Thus, the tacit knowledge must be learned from training examples. To make the tacit knowledge reusable, it is generalized as indexing rules for robotic program schemata in the planning rule base.

3.5 The Modified Explanation-based Learning (EBL) Method

In this section, a specific example is given to illustrate how the tacit knowledge is generalized as a rule and shared with workers by the modified EBL method.

Figure 3.4 Example: palletizing blocks.

As shown in Figure 3.4, there are two queues of blocks with the same frictional properties and density but different weights. The same color blocks have the same heights, but have different size cross-sections. The blocks in the right queue are thinner than the blocks in the left queue. The green blocks are the highest, while the blue ones are the lowest. The red blocks are higher than the yellow ones. Note that the width of the blocks, but not the color, is the key feature that affects the operation strategies.

In the example, four tasks and their corresponding training examples are given in turn to be learned by the KBS. They are picking up and placing the blue and the yellow blocks in the right queue in Figure 3.4. Because the learning mechanism is the same for the four tasks, only the learning process for picking up the blue block is presented in details. The other three are only given to compare the learning results.

In learning from picking up the blue block, the task input to the learning part of the KBS is as follows:

Task goal: pickup.

Robot motion speed in automatic mode: auto_ovrd(80).

Initial position of the target block:

ini_pos(5, block-1, [691.27, -219.85, 49.5, -179.89, 0.38, 0.02]).

The fact ‘auto_ovrd(80).’ indicates that the robot’s motion speed in automatic mode is 80% of its maximum motion speed. In the fact ‘ini_pos(5, block-1, [691.27, -219.85, 49.5, -179.89, 0.38, 0.02]).’, ‘5’ is the ID number of the blue block. ‘block-1’ is its type. ‘[691.27, -219.85, 49.5, -179.89, 0.38, 0.02]’ is the coordinate of the point that indicates its initial position, at the center of the upper side of the target block.

The other input, the training example is as follows:

Robot commands and parameters:

```
cmd(1,mov(phome)).          %Mov Phome
cmd(2,mov(p5,50)).          %Mov P5,-50
cmd(3,mvs(p5)).             %Mvs P5
cmd(4,dly(0.3)).            %Dly 0.3
cmd(5,hclose(1)).           %HClose 1
cmd(6,dly(0.3)).            %Dly 0.3
cmd(7,mvs(p5,50)).          %Mvs P5,-50
cmd(8,mov(phome)).          %Mov Phome
point(phome, [612.390, 0.180, 463.520, -179.890, 0.380, 0.020]).
point(p5, [691.270, -219.850, 210.000, -179.890, 0.380, 0.02]).
```

In the above, the elements in the right column (Mov Phome, etc.) are the primitive robotic commands in the robotic program. The left column (cmd(1,mov(phome))., etc.) is the transformation of the primitive robotic commands. The primitive robotic commands are transformed into facts of the training example, which can be understood by the KBS. Each robotic command is tagged with a number to indicate its order in the robotic program. In this way, the repeated commands can be distinguished by their sequence numbers.

After the above two inputs are input into the learning part, the “explainer” starts to explain how the training example accomplishes the pickup task. The explaining process is a search through the rules in the LKB, as shown in Figure 3.5.

The search tries rules one by one to construct an explanation tree, and terminates when all the leaf nodes of the explanation tree have been proved by the facts of the training example and the task, or by facts in the LKB. Such facts are the underlined terms in Figure 3.5. Because of limitations of space, Fig. 5 cannot show the whole explanation tree. Thus, the sub-trees of seize_point and pickup_act are not shown completely. After all the leaf nodes are proved, the arguments in the searched rules (i.e., the terms above the dashed lines in Figure 3.5) are instantiated with the values in the facts. In this way, as the terms under the dashed lines in Figure 3.5 show, an instantiated explanation tree is obtained.

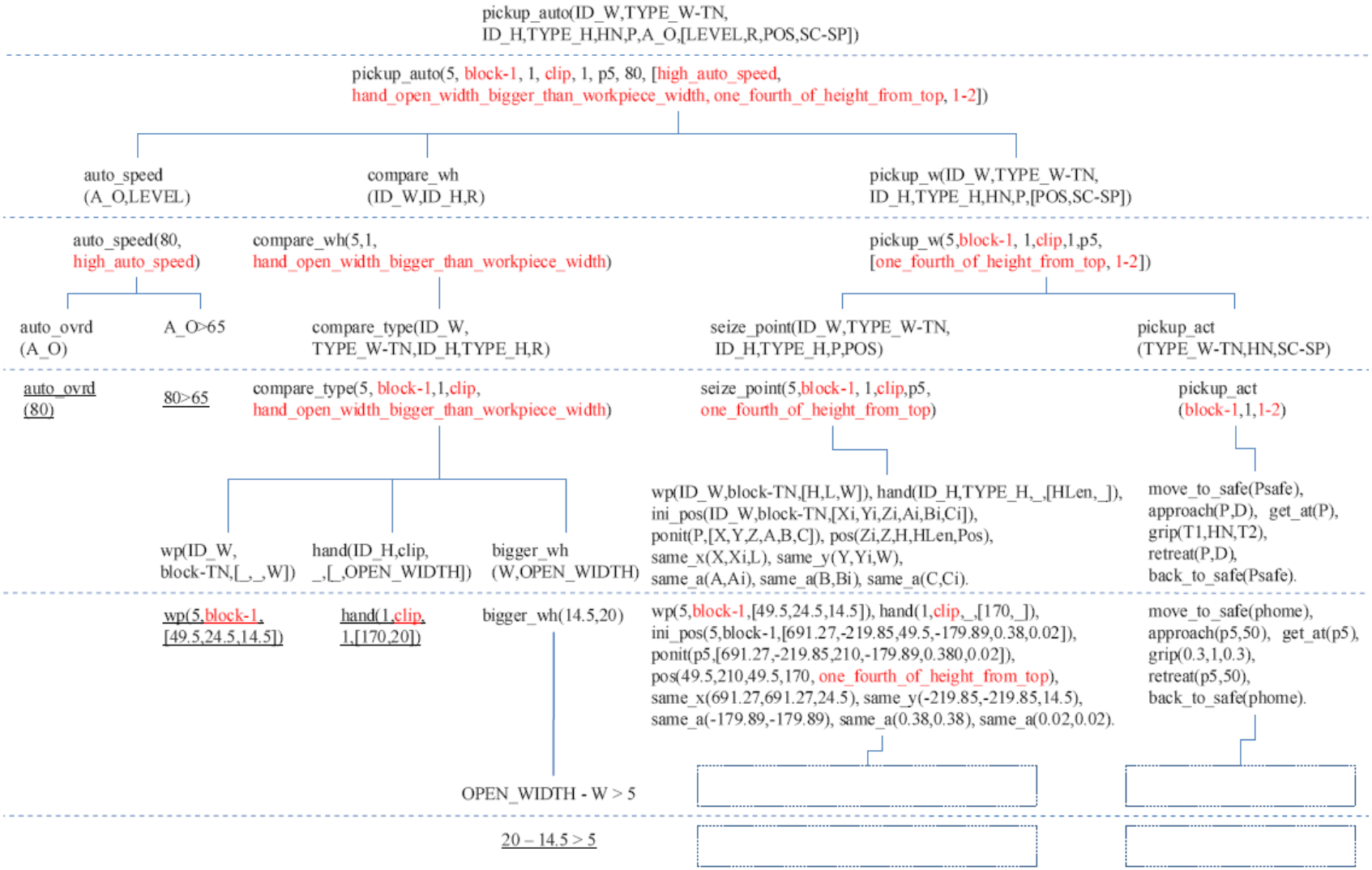

Figure 3.5 Explanation tree of the example: picking up the blue block in the right queue.

Then, according to the operationality criterion, the “generalizer” generalizes tacit knowledge from the instantiated explanation tree by selecting key arguments and transforming them into an understandable knowledge translation and a reusable generalized rule. In Figure 3.5, the red arguments are the selected key arguments, which present the features/requirements of the pickup task, the seizing position of the blue block, the pickup operation strategy, etc.

The learning result is made up of two parts. The first is the knowledge translation for workers:

IF:

The automatic running speed is: high_auto_speed;

The workpiece type is: block-1;

The robot tool type is: clip;

The robot hand open width is: hand_open_width_bigger_than_workpiece_width;

THEN:

The seizing position on the workpiece is: one_fourth_of_height_from_top;

The pickup operation is:

move to a safe point, approach the seizing point at a distant position (to avoid collision with nearby workpieces), move to the seizing point, seize the target workpiece, retreat to the approach point, move back to the safe point.

The second is a generalized rule for the planning part:

```
pickup_plan(ID_W,block-1,
ID_H,clip,P_name,hand_open_width_bigger_than_workpiece_width,high_auto_speed,
[P_Coordinate,Command_Sequence ]):-
seizeponit(ID_W,block-1,ID_H,one_fourth_of_height_from_top,P_name,P_Coordinate),
pickupact(ID_W,block-1,clip,1-2,P_name,Command_Sequence).
```

The above generalized rule does not include the detailed pickup operation actions. Instead, it contains the code for the pickup strategy used in the training example, which is ‘1-2’. In the planning part, this is a rule to transform ‘1-2’ into robotic commands by using the robotic program schema whose code is ‘1-2’ in the basic planning base. In ‘1-2’, ‘1’ indicates the type of the operation strategy, and ‘2’ indicates its sub-type.

The pickup strategy used here is an essential one, since picking up the blue block is a simple task. The tacit knowledge in this example is that the robot tool should approach the blue block at a distant position. Because there are other blocks near the blue one, approaching it from a distance can avoid collisions in moving it after picking it up.

In placing the blue block, the placing strategy used is also an essential one, whose code is ‘1-1’. The tacit knowledge in this task is that the robot should approach at a position near the destination to make the blue block align with the target hole. In addition, the releasing point is at half the depth of the hole to make the blue block slide smoothly into the hole. (In the palletizing example in Fig. 4, the

cross-sections of the target holes are bigger than the cross-sections of their corresponding blocks. Thus the placing tasks are loose insertions.)

For picking up the yellow block in the right queue, the tacit knowledge learned is the same as that of picking up the blue block. Thus, the KBS does not need to save the generalized rule learned from this task in the PKB, while it simply presents the knowledge translation for the workers.

However, for placing the yellow block, the tacit knowledge learned is different from that of placing the blue block. For this task, the placing strategy used is '2-1':

> move to a safe point, approach the releasing point at a distant position (to avoid collision with obstacles nearby destination), approach at a position near the releasing point, move to the releasing point, release the target workpiece, retreat to the distant approach point, move back to the safe point.

This is a complicated placing strategy, which is formulated by adding a distant approach point (i.e., an intermediate point) into the essential placing strategy '1-1'. In placing the yellow block, the previously placed blue block is taken as an obstacle near the destination as shown in Fig. 2. To avoid collision with the blue block, a distant approach point should be added. The generalized rule learned from this task is different from that of placing the blue block, thus the KBS will save it in the PKB.

The four tasks given in this section illustrated the Modified EBL method. In the next section, we show how the learned knowledge is reused by the planning part of the KBS.

3.6 Experiments and Analysis

As mentioned above, the KBS learned the knowledge of palletizing the blue and the yellow blocks in the right queue in Figure 3.4, and the learned knowledge is generalized as reusable rules to palletize the remaining blocks without being taught by the human. Through this learning procedure, the PKB of the KBS can be updated, since new generalized rules are added into its planning rule base.

In this section, experiments are conducted by using the planning part of the KBS to generate robot programs for palletizing the other six blocks. The purpose is as follows: 1. to show how the planning part works, 2. to test the integrity and consistency of the domain theory, and 3. to show the boundaries of reuse implied by the generalized rules.

Three aspects of the experimental analysis are presented. Aspect 1 shows how the planning part works and that the generalized rule is valid for similar tasks. Aspects 2 and 3 show cases in which the generalized rule is not valid, that is, problems that can occur in the PKB.

3.6.1 How the Planning Part Works

To generate the robot program of picking up the red block in the right queue, the task input into the "planner" is:

Task goal: pickup.

Robot motion speed in automatic mode: auto_ovrd(80).

Initial position of the target block:

ini_pos(7, block-1, [691.27, -219.85, 70, -179.89, 0.38, 0.02]).

The “planner” first uses rules in the PKB to analyze the features/requirements of the task. Then it matches these feature/requirements with those in the premises of the generalized rules. In other words, the planning process is also a search of the rules. Unlike learning, it is a search through the PKB. Figure 3.6 shows the searched rules.

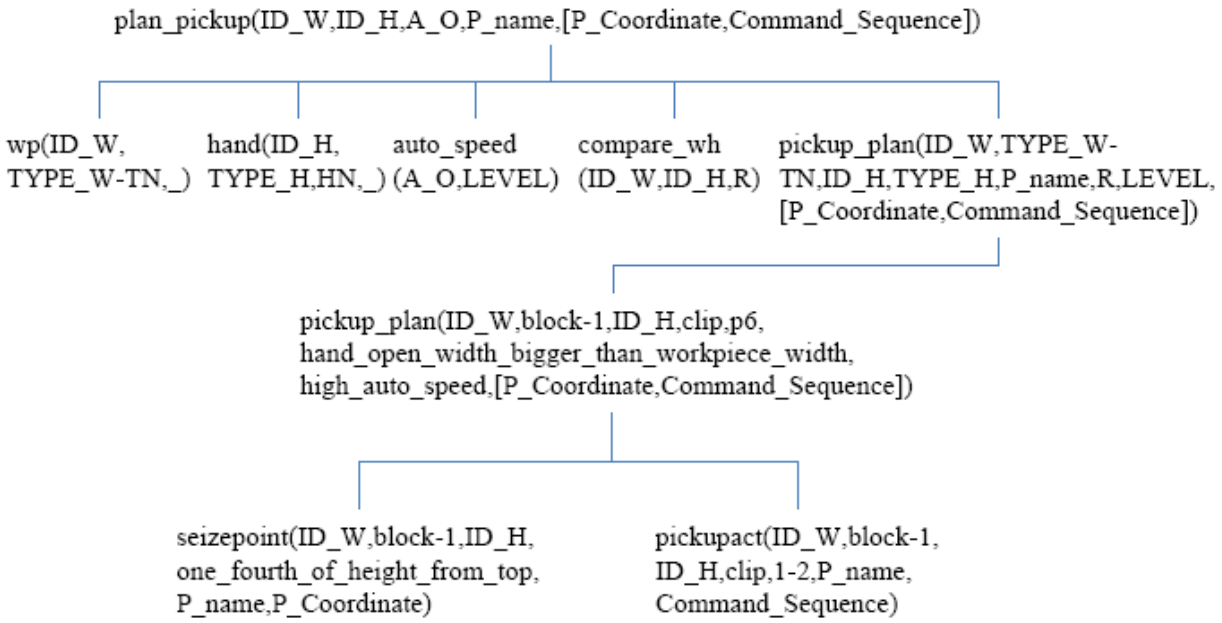

Figure 3.6 Tree of searched rules for picking up the red block in the right queue.

Finally, the generalized rule learned from picking up the blue block matches with the input task. This rule retrieves the pickup robotic program schema ‘1-2’ and the corresponding parameter calculation rule from the basic planning base. In this way, the “planner” outputs the robot program for picking up the red block as follows:

The seize point coordinate is

[691.27, -219.85, 204.5, -179.89, 0.38, 0.02]

The pickup command sequence is:

mov(psafe) , mov(p6, 50) , mvs(p6) , dly(0.3) ,

hclose(1) , dly(0.3) , mvs(p6, 50) , mov(psafe).

This robot program can be executed successfully to pick up the red block in the right queue. In the same way, the “planner” generates a valid program for placing the red block. By reusing the learned knowledge in this way, the robot-teaching time can be reduced.

3.6.2 Inconsistent Theory Problem

In Figure 3.4, the blocks in the right queue belong to the same type ‘block-1’. This is because their widths are almost the same. However, the “planner” fails to generate a program for picking up the green block in the right queue. This is caused by a confliction between the rules in the domain theory. In WTD, the “planner” finds that the width of the green block is 15, its type is ‘block-1’, and the open width of the robot tool is 20. In the planning rule base, there is a rule ‘compare_wh’ saying that:

If:

robot _tool_open_width – workpiece _width > 5,

Then: hand_open_width_bigger_than_workpiece_width;

If:

robot _tool_open_width – workpiece _width =< 5,

Then: similar_hand_open_width_and_workpiece_width.

This rule is used for analyzing the relationship between a robot tool and the target workpiece. Therefore, for the green block, the task features/requirements analysis result is as follows:

The workpiece type is: block-1;

The robot hand open width is: similar_hand_open_width_and_workpiece_ width.

However, in the planning rule base, the rule learned from picking up the blue block is:

IF:

The automatic running speed is:

high_auto_speed;

The workpiece type is: block-1;

The robot tool type is: clip;

The robot hand open width is: hand_open_width_bigger_than_workpiece_ width;

THEN:

The seize position on the workpiece is: one_fourth_of_height_from_top;

The pickup operation is:

move to a safe point, approach the seizing point at a distant position (to avoid collision with nearby workpieces), move to the seizing point, seize the target workpiece, retreat to the approach point, move back to the safe point.

This means that there is no rule in the PKB that can be used for picking up the green block. This failure is caused by the conflict between the rule ‘compare_wh’ and the learned rule for picking up the blue block.

In this example, the learned rule determines the pickup strategy by considering both the type of the block and the comparison between its width and the open width of the tool. The comparison is decided by the pre-encoded rule ‘compare_wh’, which does a calculation with the data (i.e. the

width) of the block and the tool. In this case, a minor variance of the block data will lead to completely different comparison result. If the width of the green block was not 15, but was 14.99, then there would be no such ‘planner’ failure.

Or on the other hand, assume that we change the ‘compare_wh’ rule to be

If:

robot _tool_open_width – workpiece _width > =5,

Then: hand_open_width_bigger_than_workpiece_width;

If:

robot _tool_open_width – workpiece _width < 5,

Then: similar_hand_open_width_and_workpiece_width.

In this case, there also would be no conflict between the rule ‘compare_wh’ and the learned rule. Then, the “planner” will be able to reuse the learned rule successfully to generate a robotic program for picking up the green block.

This experiment reveals that due to real-time uncertainty factors (e.g. the minor variance of the workpiece data), it is difficult to make the pre-encoded rules to be completely consistent with the new learned rules. This is the inconsistent theory problem that may exist in any knowledge-based system. In the proposed KBS, the method of solving this problem is that an expert manually revises the rules to resolve their contradiction. Another method is to introduce a feature weighing mechanism to compare the importance/influences of the conflicting features in the rules. Then the KBS uses the rules by respecting the feature of the greatest importance/influence. The feature weighing mechanism will be the subject of our future research.

3.6.3 Incomplete Theory Problem

In Figure 3.4, the blocks in the left queue belong to type ‘block-2’, because their widths are almost the same. After the task of picking up the blue block in the left queue is input into the KBS, the “planner” finds that there is no rule in the PKB available for picking it up. This is the incomplete theory problem of the PKB. In the proposed KBS, the method of solving this problem is to give a training example of this task. Then the proposed KBS can learn from the training example and update the PBK.

In this experiment, a training example of picking up the blue block in the left queue is given to the KBS. The KBS learns the following rule:

IF:

The automatic running speed is:

high_auto_speed;

The workpiece type is: block-2;

The robot tool type is: clip;

The robot hand open width is: similar_hand_open_width_and_workpiece_ width;

THEN:

The seize position on the workpiece is: one_fourth_of_height_from_top;

The pickup operation is:

move to a safe point, approach the seizing point at a distant position (to avoid collision with nearby workpieces), slow down the speed (to avoid collision between the robot tool and the target workpiece), approach the seizing point, move to the seizing point, seize the target workpiece, retreat to the approach point at the distant position, resume normal speed, move back to the safe point.

In the basic planning base, the code for the pickup strategy used in this training example is '3-2'. This pick-up strategy is a complicated one, which is formulating by adding an intermediate approach point and a speed change into the essential pickup strategy. Because the width of the blue block is similar to the open width of the robot tool, in the high speed automatic mode, the robot tool is prone to collide with the block. Therefore, the speed of the robot tool should slow down when it moves to the seizing point. In addition, an approach point that is close to the block should be added to align the robot tool with the block. In the following experiments, the rules learned from picking up and placing the blue block are reused successfully in picking up and placing the rest blocks in the left queue.

3.7 Discussion

3.7.1 Comparison with Other Method

In this section, we compare the proposed method with other methods, i.e., the SA method, the LfD method and the KB method introduced in Section 3.3 from the perspective of the ability of sharing and reusing expert knowledge.

In the SA method, the expert knowledge is used to model the objective system or to define the feedback/reward function. Then the system is optimized through online learning from its exploration. Thus, the SA method is an unsupervised learning method. Within the SA method, the knowledge is inductively learned from multiple/repetitive explorations by the system itself, but is not learned from human experts. Therefore, the limitation of the SA method is that it sacrifices the communication with human so as to lack expressiveness and inferential richness of the learned knowledge. It makes the SA method lose the chance that expert knowledge may help it converge within less exploration times. On the other hand, the proposed method emphasizes knowledge sharing. The knowledge acquired by the proposed method can not only be reused by the system, but also can be understood and reused by human.

The LfD method acquires human knowledge from demonstrations given by human. However, it just learns the state-action mapping policy, but ignores the analysis of the demonstrations. Thus, the quality of the learned knowledge is limited by the quality of the given demonstrations. Moreover, the LfD method often requires a set of training demonstrations for learning. The proposed method can learn from a single training example by analyzing it with the LKB and generalizing the explanation with the OC. The analyzing ability makes the proposed method can critically learn from an example instead of learning a state-action mapping policy in a passive way. In addition, the ability of learning from a single example is important in that each example has the particular knowledge of the expert who gave the example. Especially in real robotic manufacturing, it is difficult to provide many training examples.

The KB method is the method used in the planer of the proposed KBS. Compared to other KB system, the proposed KBS uses a modified EBL to update its PKB, which makes the KB method enhance the extendibility and practicability.

3.7.2 Importance of Sharing Tacit Knowledge

Within factory automation, many kinds of automations like robotic technologies are introduced and did contribute to improving the productivity of manufacturing. However, these trends are causing another novel problem, the so-called de-skilling hypothesis of human workers induced by the automation [42]. In other wards, the more automation is incorporated into the manufacturing process, the less the worker understands the process; the more sophisticated the machine becomes, the less control and comprehension of the machine the worker has.

In the current factories, skills on how to operate and maintain the automation, e.g., robots, to keep them work continuously without failures are tacit knowledge. The tacit knowledge is of importance because the technical hitches caused by the wrong instructions to the robot do make the production lead time longer, but there is no way to share and reuse the tacit knowledge needed to avoid those kinds of technical hitches that are currently grasped in mind by only few experienced workers. Thus, construction of a novel knowledge infrastructure is needed that can support the maintenance and the transfer of the tacit knowledge of manufacturing among the human workers.

As for knowledge management and knowledge creation, the SECI model proposed by Nonaka is well known [43]. He mentioned that the creation of knowledge is a continuous process of dynamic interactions between tacit and explicit knowledge. We completely agree with this idea, but so far few discussions have been made on how to realize that knowledge creation cycle technically with the support of machine learning methodologies and how to design an entire system from a perspective of human-machine collaborative systems. Therefore, in this paper, based on the considerations above, we propose the KBS based on a modified EBL method to facilitate tacit knowledge sharing and reusing between human workers and robots.

3.8 Conclusion

In this chapter, the overall structure of a KBS to acquire tacit knowledge for sharing and reusing in the robotic assembly domain is proposed. The original EBL has been modified to acquire tacit knowledge from the robotic programs of skilled workers. A new operationality criterion is proposed, which demands that the learned tacit knowledge should be understandable by workers and reusable by robots. With the KBS, workers can learn the tacit knowledge implemented in robotic programs made by other workers. In this way, robotic programs become a vehicle for transferring tacit knowledge of skilled workers. This enables skill-succession even after the original skilled workers have retired. In addition, the proposed KBS can generate robotic programs for assembly tasks in similar cases. Thus, it enables human workers to avoid repeatedly teaching robots similar assembly tasks. This reduces the robot teaching time dramatically. The proposed KBS thus helps share tacit knowledge in the robotic assembly domain, and improves the efficiency and flexibility of robotic manufacturing.

However, in this chapter, the modified EBL is only used in acquiring knowledge from lower level tasks such as ‘pickup’ and ‘place’ but not a whole assembly task. The knowledge reusing method is also confined to lower tasks. In the following chapters, more complicated to acquire and reuse knowledge for higher level tasks are developed and presented.

Chapter 4

The Hierarchical Knowledge Modularization Method

4.1 Introduction

In this chapter, the modified EBL method proposed in the Chapter 3 is improved to make it to be able to learn from error recovery training examples. The improved EBL is also hierarchically used to acquire knowledge from higher level tasks, which are whole assembly tasks.

Section 4.2 briefly introduces the improved EBL method.

Section 4.3 presents the hierarchical knowledge acquisition process of the improved EBL method with a concrete real-world assembly example.

Section 4.4 describes how the learned knowledge is reused and how the improved EBL is used to learn from an error recovery example.

Section 4.5 gives the saving evaluation and reusing selection methods of the acquired knowledge.

Section 4.6 gives the conclusion.

4.2 Improvement of the Modified EBL

The most distinguished feature of the improved EBL method is that it has two modes of learning: 1. learning from an example directly given by human workers, which has been presented in Chapter 2; 2. learning expertise on error recovery by observing revisions made by human workers in handling execution errors that occur in reusing previously acquired knowledge. The second learning mode of the improved EBL method is completely new and different from the original EBL. As acquired human expertise is reused for similar assembly tasks, errors may occur in the execution of robot programs generated by reusing the generalized plans. In this case, human workers will revise the program to resolve the errors. Then, the second learning mode can be used to learn from these error recovery examples.

In this study, an assembly task is defined as a process to assemble a single workpiece including picking it up from its initial position and placing it at its destination. Human expertise (i.e., knowledge) means how workers plan the robot's actions (i.e., commands) for a given assembly task and how they design the trajectory for robot motion (i.e., point parameters) according to a certain environment.

Figure 4.1 outlines the flowchart of the improved EBL method, which is divided into two: the learning part and the planning part (i.e., the grey part in Figure 4.1). In the learning part, the inputs

are an assembly task (i.e., the goal) and its training example (i.e., the robot program). The output is the acquired human expertise, which will be saved in the planning knowledge base. The input in the planning part is a new given assembly task that has to be accomplished. The output is the executable robot program for the given task and hints for human workers. The hints represent human expertise that should be paid attention (i.e., mainly how to decide the seizing/placing and approach point in the robot motion trajectory) in the execution of the generated robot program. Then, human workers upload the generated robot program to the robot controller and test it in the playback mode. The planning part will ask workers whether error occurs in the test. If the answer is no (i.e., no error occurs), the planning part will make saving evaluation of the new task to decide whether it is worth being learned and saved. If the answer is yes (i.e., error occurs), workers must revise the robot program. And the planning part will ask workers to input the revised program. This is how the workers interact with the planning part of the system (i.e., dashed-line framed parts 'Test', 'Error?', 'Revising' in Figure 4.1). Then, the learning part will learn from the error recovery. In this case, both the generated program and the revised program will be taken in as the training example. And the assembly task includes the new given assembly task description, and the original task description of the generated program that is saved in the domain theory.

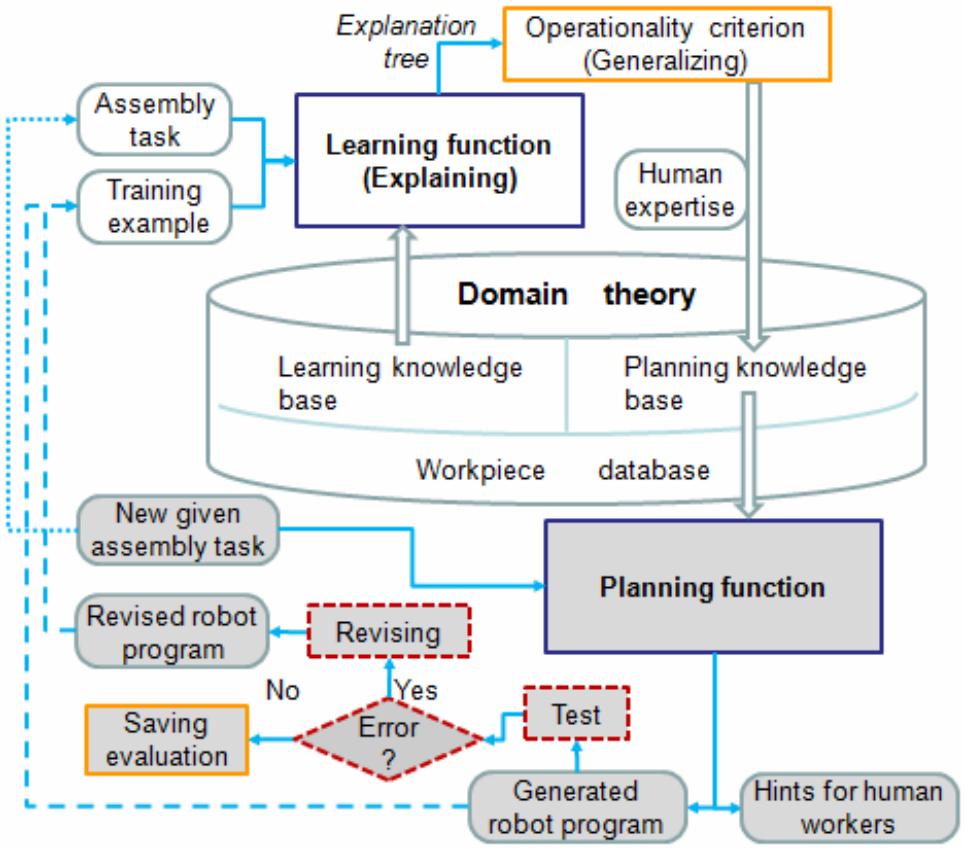

Figure 4.1 The structure of the improved EBL method.

4.3 Acquisition of Human Expertise

4.3.1 Description of an Assembly Task and Its Training Example

The left photo of Figure 4.2 shows the workpiece to be assembled, i.e., a fin-block (a fin-block is a device for extinguishing arcs in circuit breakers). The right photo shows its destination and the destination environment for the assembly task, in which the assembled fin-block is shown in its final assembled state. The improved EBL method requires two inputs to learn from the robot-teaching demonstration of assembling this fin-block: the first is its assembly task description and the second is the robot program for attaining the task.

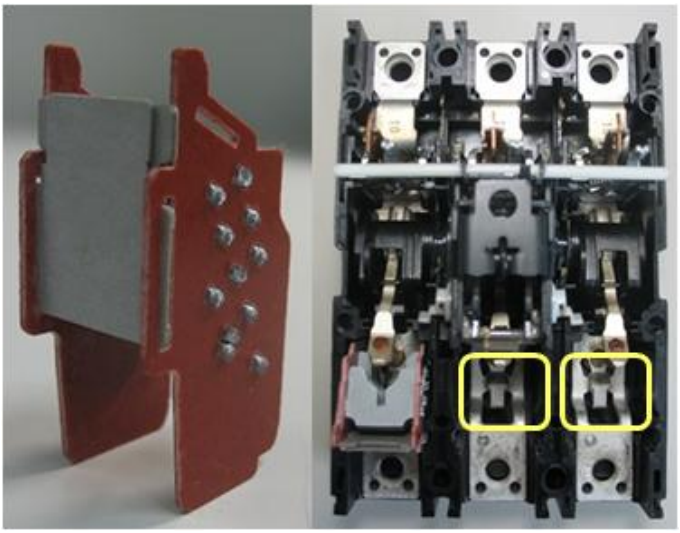

Figure 4.2 Example: assembling fin-block in circuit breaker.

The assembly task description is:

Task ID: 28.
Task name: assemble fin-block.
Workpiece: id(10), name(fin-block).
Tool: id(6), name(gripper2).
Assembly operation: simple_peg_hole_insertion.
Initial position:
p_ini(209.260,-615.490,228.720,-180.000,0.000,72.810).
Destination position:
p_des(585.050,-308.610,272.310,179.980,-0.200,157.490).
Destination environment:
hole_width(25),hole_depth(33),obstacle(front,5,69).

Here, ‘Task ID’ indicates the order for the assembly task that distinguishes it from other assembly tasks in the assembly series, which is a sequence of assembly tasks for assembling a product (e.g., the product to be assembled in Figure 4.2 is a circuit breaker). The ‘Task name’ is the name of the

task. As there are repetitive tasks in an assembly series, several tasks can have the same task name (e.g., there are three fin-blocks to be assembled, which have the same task name ‘assemble fin-block’ and different task IDs ‘28’, ‘29’, and ‘30’ are assigned to each). The IDs and names of the workpiece and tool can help extract data on the fin-block and data on the robot tool used to assemble it from the workpiece database. The ‘simple_peg_hole_insertion’ describes the assembly operation to insert the fin-block into a hole (i.e., a square hole) in the body of the circuit breaker. The ‘Initial position’ of the fin-block, i.e., ‘p_ini’, is indicated by the center of the underside of the fin-block. The ‘Destination position’, i.e., ‘p_des’, is indicated by a point that is on the inner surface of the body of the circuit breaker, which corresponds to the center of the fin-block at the bottom when it is assembled. The destination environment is described as a hole that is 25 mm in width with a depth of 33 mm. There is also an obstacle nearby, which is 5mm in front of the destination and has a height of 69 mm.

The experienced workers’ robot program for this assembly task is:

%pickup
Commands:
Ovrd M_NOvrd
HOpen 1
Mov p1
Mov p2,-50
Ovrd 10
Dly 0.1
Mvs p2
Dly 0.3
HClose 1
Dly 1
Mvs p2,-50
Ovrd M_NOvrd
Dly 0.1
Mov p3
Parameters:
p1(344.060,430.570,423.240,106.130,-20.520,154.730).
p2(209.260,-615.490,260.810,-180.000,0.000,72.810).
p3(368.950,31.110,505.450,-179.800,-0.030,-109.290).
%place
Commands:
Mov p3

Mov p4,-150

Mvs p4,-35

Ovrd 10

Dly 0.1

Mvs p4

Dly 0.3

HOpen 1

Dly 0.3

Ovrd M_NOvrd

Mov p4,-150

Mov p3

Parameters:

p4(585.050,-308.610,310.330,179.980,-0.200,157.490).

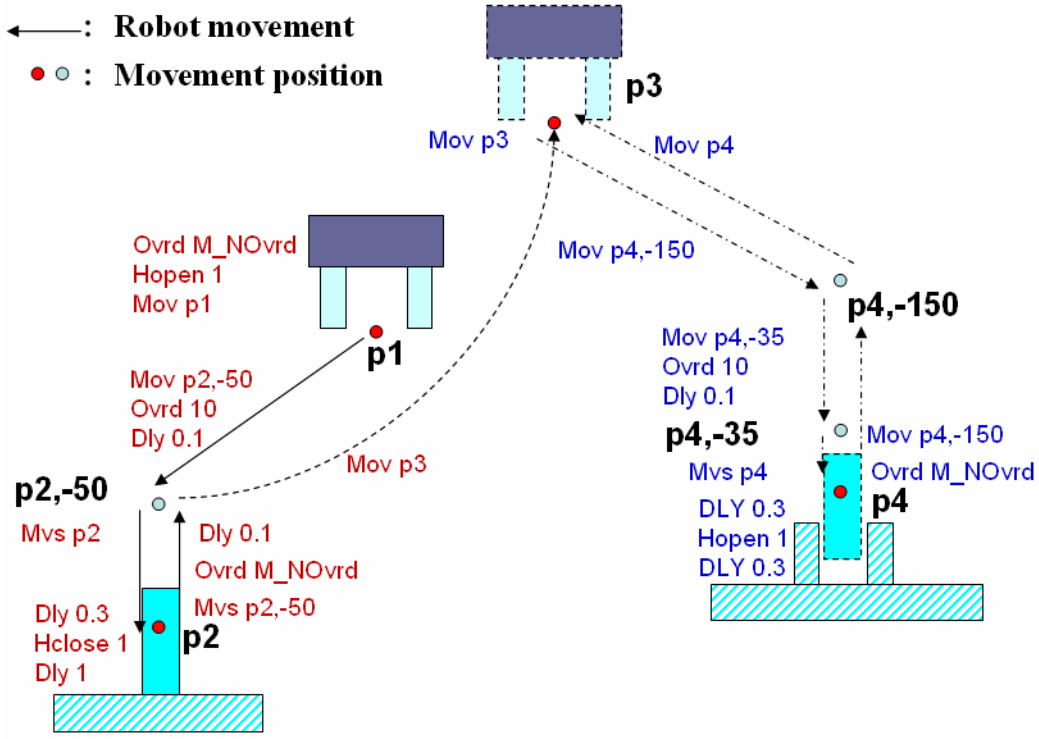

Figure 4.3 Robot's motion trajectory in ‘assemble fin-block’.

The parameters in the program are the point coordinates for the center of the robot’s tool tip in its motion trajectory. As shown in Figure 4.3, 4 points (i.e., p1, p2, p3, and p4) are indispensable in an assembly task. ‘p1’ and ‘p3’ respectively are safe points in the ‘pickup’ part and ‘place’ part. The role of safe points is to make the robot tool move to a safe position when it picks up or places the target workpiece. In this way, the robot tool can avoid collision with obvious obstacles. It is easy for human workers to teach safe points. ‘p2’ is the seizing point where the robot tool seizes the

fin-block. ‘p4’ is the placing point where the robot tool places the fin-block while assembling it. Workers need expertise to teach the seizing point and the placing point. There are always approach points near the seizing point (i.e., ‘p2,-50’ in this example) and the placing point (i.e., ‘p4,-150’, ‘p4-35’ in this training example). The approach points are useful to make accurate ‘pickup’ and ‘place’ operations. This is human expertise, such as how the point parameters are determined, and how the delays are set in the robot program.

4.3.2 The Hierarchical Knowledge Modularization Learning Process

The process of acquiring human expertise by using the improved EBL also consists of two steps: explaining and generalizing. The improved EBL is hierarchically used to explain the goal (i.e., ‘assemble’ in Figure 4.4) and its different levels of subgoals of the input assembly task (i.e., ‘pickup’, ‘place’, ‘command’ and ‘parameter’ in Figure 4.4)

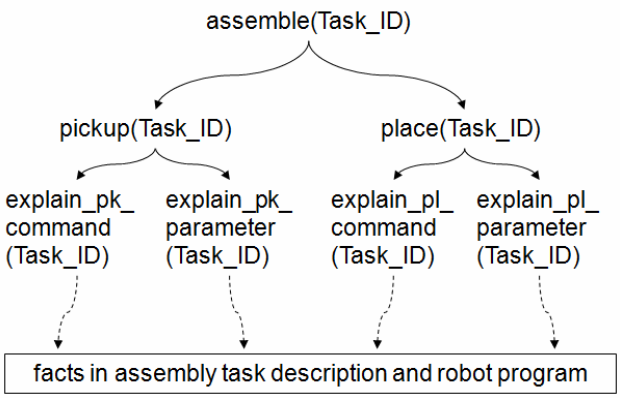

Figure 4.4 The hierarchy of the explaining process.

The explaining process involves a search through the learning knowledge base. At the start, all the arguments in the goal and the rules that are searched are variables. As Fig. 5 shows, the goal is to accomplish a given assembly task whose ID is ‘Task_ID’. The rules explain the picking-up part and the placing part of the assembly task separately. In both parts, the commands and parameters in the robot program are also explained separately. After all the leaf nodes have been proved by the facts in the assembly task description and the robot program, the arguments in the rules and goal are instantiated with the values in the facts. Then, an instantiated explanation tree that is specific to the given assembly task is obtained. The instantiated explanation tree explains how this particular robot program accomplishes the assembly task.

The explaining process for the robot commands involves translating them into a language that can be easily understood by humans. For example, with the following three rules in the learning knowledge base:

Rule 1: seize_the_target_workpiece(W_Name):-

wp(W_ID,W_Name,_),

grip(T1,HN,T2).

Rule 2: delays_before_and_after_close_tool(T1,T2):-

grip(T1,HN,T2).

Rule 3: grip(T1,HN,T2):-

cmd(N1,dly(T1)),

cmd(N2,hclose(HN)),

cmd(N3,dly(T2)),

T1>0.2, T2>0.2,

N2=:=N1+1,

N3=:=N2+1.

and the fact ‘Workpiece: id(10), name(fin-block)’ in the assembly task description and the facts ‘Dly 0.3’, ‘HClose 1’, and ‘Dly 1’ in the training example, the improved EBL method can translate the commands ‘Dly 0.3, HClose 1, Dly 1’ in the training example in into ‘seize the target fin-block’, ‘delay T1 before and T2 after closing the robot tool’, and ‘T1 and T2 should be longer than 0.2 sec’, which are easily understood by human.

Similarly, explaining the parameters involves analyzing the point parameters in the robot program. The learning function analyzes the point parameters by taking into consideration the related workpiece data, tool data and facts on the environment. The explanation gives reasons for how and why experienced workers determined the seizing point ‘ps’, the placing point ‘pp’, and the approach points as those in the robot program.

The workpiece data and tool data can be retrieved from the workpiece database by using their respective IDs given in the assembly task description. In the ‘assemble fin-block’ example, the data of the fin-block is:

ID: 10.

Name: fin-block.

Type:block.

Length: 21 mm.

Width: 20 mm.

Height: 42 mm.

Max_length: 22 mm.

Max_width: 22 mm.

Max_height: 50 mm.

Weight: 25 g.

Material: plastic, metal.

Texture: hard.

Color: brown, silver.

The data of the robot tool used to perform the ‘assemble fin-block’ task is:

ID: 6.

Name: gipper2.

Type: two-finger.

Inside_open_width: 25 mm.

Outer_open_width: 55 mm.

Finger_length: 120 mm.

Max_load: 5000 g.

Figure 4.5 is part of the explanation tree of a subgoal that explains the placing point ‘pp’ of the fin-block, i.e., the point ‘p4’ in the ‘place’ part of the robot program. This partial explanation tree shows how the rules in the learning knowledge base are searched to explain the subgoal ‘placing_position(H,R)’. The search terminates until the leaf nodes are proved by the facts in the assembly task description and the robot program, or the facts in the workpiece data base, which are the underlined items in Figure 4.5.

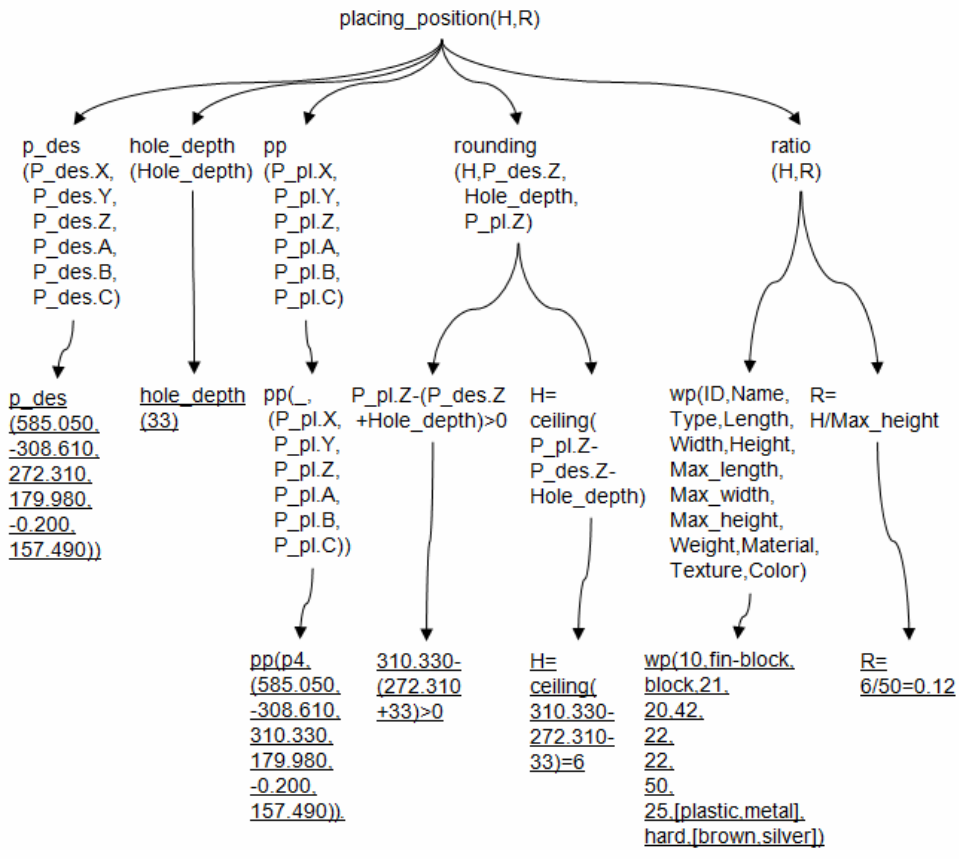

Figure 4.5 Partial explanation tree of the placing point subgoal.

In Figure 4.5, the placing point ‘pp(P_pl.X, P_pl.Y, P_pl.Z, P_pl.A, P_pl.B, P_pl.C)’ of the fin-block is instantiated by the point ‘p4(585.050, -308.610, 310.330, 179.980, -0.200, 157.490)’ in the ‘place’ part of the robot program. The destination position ‘p_des(P_des.X, P_des.Y, P_des.Z, P_des.A, P_des.B, P_des.C)’ of the fin-block is instantiated by the destination position ‘p_des(585.050, -308.610, 272.310, 179.980, -0.200, 157.490)’ (i.e., a point on the bottom surface of the left hole in the body of the circuit breaker) in the assembly task description. The coordinates of the destination position ‘p_des’ are obtained from the sensor. The coordinates of the placing point ‘pp’ (i.e., the point ‘p4’ in the robot program) is taught by human workers based on the coordinates of the destination position ‘p_des’. Here, the ‘X, Y’ coordinates and the ‘A, B, C’ angle coordinates of the placing point ‘pp’ of the fin-block are the same as the corresponding ‘X, Y’ and ‘A, B, C’ coordinates of the destination position, only their ‘Z’ coordinates (i.e, P_pl.Z and P_des.Z) are different. In other words, the human workers place the fin-block above the target hole. Therefore, the subgoal ‘placing_position(H, R)’ involves figuring out the height (i.e., ‘H’) of the placing point ‘pp’ above the hole’s top orifice and the ratio (i.e., ‘R’) of this height to the maximum height (i.e., Max_height in fig. 6, which is instantiated by 50 in arguments of workpiece data ‘wp’) of the fin-block. After all the leaf nodes of the partial explanation tree is instantiated, the values of ‘H’ and ‘R’ can be obtained, which are respectively 6 and 0.12. In this way, this partial explanation tree explains that the robot will release its tool at a height of 6 mm above the top orifice of the hole while inserting the fin-block into the target hole to avoid a collision between the tool and the hole. The ratio of this height to the maximum height of the fin-block is 0.12.

Then, the subsequent generalizing process is accomplished by selecting several key arguments that appear in the instantiated explanation tree and transforming them into operable and understandable forms according to the new operationality criterion.

The examples discussed in this section illustrate how the human expertise embedded in the robot program of the assembly task ‘assemble fin-block’ is analyzed by the improved EBL method. To make the human expertise both operable to generate robot programs for similar ‘simple block-hole insertion’ assembly tasks and understandable by human workers, the operationality criterion (i.e., a set of rules and language processing functions) generalizes the assembly task description and the instantiated explanation tree as:

Assembly plan

ID: ap00011

Name: simple_block_hole_insertion

%pickup

Sub ID: 01

Table 4.1 Assembly plan of picking up the first fin-block

Primitive robot	Command explanations	Parameter explanations

command schema		
Ovrd M_NOvrd HOpen 1	Preparation(set normal speed, open tool);	
Mov Phome	Move to safe point;	
Mov Ps,-Ds	Approach seizing point at a distance of Ds to it;	Approach distance Ds = 50 mm, Ds >= fin-block_max_height.
Ovrd 10 **Dly Ts**	Slow down speed, delay Ts=0.1 second;	
Mvs Ps	Arrive at seizing point;	Seizing point is at 0.76*height of the fin-block.
Dly Tc1 **HClose 1** **Dly Tc2**	Seize the target fin-block, Delay Tc1 before and Tc2 after closing the robot tool, Tc1 and Tc2 should be longer than 0.2 second;	
Mvs Ps,-Ds	Retreat to approach point;	
Ovrd M_NOvrd **Dly Tr**	Recover normal speed, delay Tr=0.1 second;	
Mov Pphome	Move to safe point near destination.	

%place

Sub ID: 02

Table 4.2 Assembly plan of placing the first fin-block

Primitive robot command schema	Command explanations	Parameter explanations
Mov Pphome	Move to safe point near destination;	
Mov Pp,-Dp1	Approach placing point at a distance of Dp1 to it;	The first approach distance Dp1=150, Dp1 >= tallest_obstacle_height + fin-block_max_height.
Mov Pp,-Dp2	Approach placing point at a distance of Dp2 to it;	The second approach distance Dp2=35.
Ovrd 10 **Dly Tsp**	Slow down speed, delay Tsp=0.1 second;	

Mvs Pp	Arrive at placing point;	Placing point is at H=6 mm, about 0.12*max_height of the fin-block, above the top orifice of the hole.
Dly To1 **HOpen 1** **Dly To2**	Release the fin-block, delay To1 before and To2 after opening the robot tool, To1 and To2 should be longer than 0.2 second;	
Ovrd M_NOvrd	Recover normal speed;	
Mvs Pp,-Dp1	Retreat to the first approach point;	
Mov Pphome	Retreat to safe point near destination.	

Indexing rule

assembly_task(Task_ID,block,two-finger,simple_peg_hole_insertion,
[obstacle(front,_,_),_]):- assembly_plan(ap00011).

Notes

While the fin-block is being picked up, the approach distance, Ds, should not be shorter than the maximum height of the fin-block to prevent the seized fin-block and other nearby fin-blocks from colliding.

While the fin-block is being placed, the first approach distance, Dp1, should not be shorter than the sum of the height of the tallest obstacle and the maximum height of the fin-block to prevent the fin-block and the obstacle from colliding.

The above, including how to determine the parameters of the heights for picking up and placing down the fin-block, the time delays, and speed changes (i.e., Table 1-2) represent the human expertise acquired from the robot program of the assembly task ‘assemble fin-block’. This is saved in the planning knowledge base together with its assembly task description.

The assembly plan can be reused to generate robot program for a similar assembly task. The command and parameter explanations in the assembly plan can help human workers understand how the generated robot program works.

The indexing rule is used by the planning function to retrieve the saved assembly plan. In other words, the indexing rule defines the generality boundary of the generalized assembly plan. The indexing rule in the above example means:

If the type of workpiece is **block**, the type of robot tool is **two-finger**, the assembly operation is **simple_peg_ hole_insertion**, and the obstacle is in **front** of the destination; **then** the assembly plan that can be reused is one whose ID is **ap00011**.

The notes will be output by the planning function for workers along with explanations of the robot program.

4.4 Acquisition of Error Recovery Knowledge

4.4.1 Error Occurred in Reusing the Acquired Knowledge

In Figure 4.2, there are a total of three fin-blocks to be assembled into three square holes (i.e., the left, the middle, and the right holes) in the body of the circuit breaker. The two squares in Figure 4.2 to the right of the assembled fin-block indicate the middle and the right holes, which are the destinations for the other two fin-blocks to be assembled. The initial state of the other two fin-blocks is almost the same as that of the first except for their initial positions. The robot tool and assembly operation are the same for assembling all three fin-blocks. Thus, the assembly task description for assembling a fin-block into the middle hole is as:

Task ID: 29.
Task name: assemble fin-block.
Workpiece: id(10), name(fin-block).
Tool: id(6), name(gripper2).
Assembly operation: simple_peg_hole_insertion.
Initial position:
p_ini(209.260,-585.400,228.720,-180.000,0.000,72.810).
Destination position:
p_des(600.050,-308.610,272.310,179.980,-0.200,157.490).
Destination environment:
hole_width(25),hole_depth(33),obstacle(front,5,69), obstacle(left,18,50).

Here, the differences between this assembly task description and the one in the last section are: the task ID, the initial position, the destination position, and particularly the destination environment. Because the fin-block assembled in the left hole is taken to be an obstacle, there is ‘obstacle(left,18,50)’ in the destination environment, which means the obstacle is 18 mm to the left of the destination and its height is 50 mm (i.e., the assembled fin-block).

The new assembly task description is input into the planning function. By analyzing the above new assembly task description, the planning function can obtain the facts that include the type of the second fin-block (i.e., block), the type of the robot tool used (i.e., two-finger), the assembly operation (i.e., simple_peg_hole_insertion), and the obstacles in destination environment (i.e., obstacle(front,5,69), obstacle(left,18,50)). These facts satisfy the precondition features of the head of the indexing rule learned from assembling the first fin-block, and activate this indexing rule. Therefore, according to this indexing rule learned in Section 4.3.2, the assembly plan ‘ap00011’ is reused to generate a robot program for this new assembly task as:

%pickup
Commands:

Ovrd M_NOvrd
HOpen 1
Mov p5
Mov p6,-50
Ovrd 10
Dly 0.1
Mvs p6
Dly 0.3
HClose 1
Dly 1
Mvs p6,-50
Ovrd M_NOvrd
Dly 0.1
Mov p7

Parameters:

p5(344.060,430.570,423.240,106.130,-20.520,154.730).

p6(209.260,-585.400,260.810,-180.000,0.000,72.810).

p7(368.950,31.110,505.450,-179.800,-0.030,-109.290).

%place

Commands:

Mov p7
Mov p8,-150
Mvs p8,-35
Ovrd 10
Dly 0.1
Mvs p8
Dly 0.3
HOpen 1
Dly 0.3
Ovrd M_NOvrd
Mov p8,-150
Mov p7

Parameters:

p8(600.050,-308.610,311.310,179.980,-0.200,157.490)).

As the robot program above is generated for the robot, the explanations for the commands and parameters and the notes in the assembly plan are also output for human workers through an

interface. Then, workers test the above program in playback mode. In the test, the robot picks up the fin-block and successfully moves it to the second approach point of the placing position. However, while the robot is moving the fin-block to its placing point ‘p8’, the robot tool collides with the assembled fin-block, i.e., an error occurs.

The reason for this collision error is that the robot has stretched its two fingers left and right while placing the fin-blocks in the destination holes. Thus, collisions can be avoided if obstacles are located at front (i.e., the black part above the fin-block in Figure 4.2) or back of the destination. However, collisions cannot be avoided if the obstacles are located to the left or right of the destination. While experienced workers were assembling the first fin-block into the left hole, because there was no obstacle located at the left or right side of the destination, they made the placing point ‘pp’ to be as near as possible to the top orifice of the hole, i.e. H=6 mm in assembly plan ‘ap00011’ to make the insertion stable. In contrast, while the robot was assembling the second fin-block into the middle hole, the assembled fin-block became an obstacle that was located at the left of the second fin-block’s destination. Moreover, the distance between the assembled fin-block and the destination was 18 mm, which was shorter than half the tool’s outer open width of 22.5 mm, and consequently collision error occurred.

4.4.2 Learning from Error Recovery

To correct the error in assembling the second fin-block, the human expert revises only one item in the generated program. That is to raise the placing point ‘p8’ of the second fin-block to make it higher than the top of the assembled fin-block. In this way, the collision between the tool and the assembled fin-block can be avoided.

The specific revision made by the human expert was to change the coordinate of the placing point ‘pp’ of the second fin-block to:

p8(600.050,-308.610,327.260,179.980,-0.200,157.490).

The human expertise in this revision can be acquired by the second learning mode of the improved EBL method, which is learning from a revision in error-recovery if an error occurs in reusing the acquired human expertise. The second learning mode is the most important distinguishing feature of the improved EBL method. The original EBL doesn’t have this learning mode.

Different from the first learning mode, in the second learning mode the robot program input into the learning function includes both the revised robot program and the robot program generated by the planning function. In addition, the assembly task includes both the new assembly task description of assembling the second fin-block and the previous assembly task description of assembling the first fin-block of the reused assembly plan. Another difference between the two learning modes is that in learning from the direct-teaching example, the learning function just analyzes the input

assembly task description and the corresponding robot program. However, in learning from the error-recovery example, apart from analyzing the newly input assembly task description and its corresponding revised robot program, the learning function also compares the newly input assembly task description with the previous assembly task description in the reused assembly plan, and compares the revised robot program with the reused assembly plan to find differences between them. Human expertise on revising robot program is contained in the differences of the assembly task descriptions and the robot programs. Then, the operationality criterion generalizes the differences from the comparisons.

The key to acquiring human expertise from the error- recovery example is to find the factors that lead to the error in the new assembly task description and the recovery measures for this error in the revised program. In the example of assembling the second fin-block, the factor that leads to the collision is that the obstacle located to the left of the destination (i.e., the first assembled fin-block). And, the recovery measure given by the human expert was to raise the placing point 'p8' of the second fin-block to make it higher than the top of the first assembled fin-block.

The human expertise acquired from this error-recovery training example is:

Assembly plan

ID: ap00012

Name: simple_block_hole_insertion

%pickup

Sub ID: 01

Table 4.3 Assembly plan of picking up the second fin-block.

Primitive robot command schema	Command explanations	Parameter explanations
Ovrd M_NOvrd HOpen 1	Preparation(set normal speed, open tool);	
Mov Phome	Move to safe point;	
Mov Ps,-Ds	Approach seizing point at a distance of Ds to it;	Approach distance Ds = 50 mm, Ds>=fin-block_max_height.
Ovrd 10 Dly Ts	Slow down speed, delay Ts=0.1 second;	
Mvs Ps	Arrive at seizing point;	Seizing point is at 0.76*height of the fin-block.
Dly Tc1 HClose 1 Dly Tc2	Seize the target fin-block, Delay Tc1 before and Tc2 after closing the robot tool, Tc1 and Tc2 should be longer than 0.2 second;	

Mvs Ps,-Ds	Retreat to approach point;	
Ovrd M_NOvrd **Dly Tr**	Recover normal speed, delay Tr=0.1 second;	
Mov Pphome	Move to safe point near destination.	

%place

Sub ID: 02

Table 4.4 Assembly plan of placing the second fin-block.

Primitive robot command schema	**Command explanations**	**Parameter explanations**
Mov Pphome	Move to safe point near destination;	
Mov Pp,-Dp1	Approach placing point at a distance of Dp1 to it;	The first approach h distance Dp1=150, Dp1>= tallest_obstacle_height + fin-block_max_height.
Mov Pp,-Dp2	Approach placing point at a distance of Dp2 to it;	The second approach distance Dp2=35.
Ovrd 10 **Dly Tsp**	Slow down speed, delay Tsp=0.1 second;	
Mvs Pp	Arrive at placing point;	Placing point is at H=22 mm, about 0.44*max_height of the fin-block, above the top orifice of the hole, H > left_obstacle_height – hole_depth.
Dly To1 **HOpen 1** **Dly To2**	Release the fin-block, delay To1 before and To2 after opening the robot tool, To1 and To2 should be longer than 0.2 second;	
Ovrd M_NOvrd	Recover normal speed;	
Mvs Pp,-Dp1	Retreat to the first approach point;	
Mov Pphome	Retreat to safe point near destination.	

Indexing rule

assembly_task(Task_ID,block,two-finger,simple_peg_ hole_insertion,[obstacle(front,_,_),obstacle(left,LD,_), tool(_,_,_,_,Outer_open_width,_,_), LD <= Outer_open_width/2, _]):- assembly_plan(ap00012).

Notes

While the fin-block is being picked up, the approach distance, Ds, should not be shorter than the maximum height of the fin-block to avoid collision between the seized fin-block and other nearby fin-blocks.

While the fin-block is being placed, the first approach distance, Dp1, should not be shorter than the sum of the height of the tallest obstacle and the maximum height of the fin-block to avoid collision between the fin-block and the obstacle.

While the fin-block is being placed, if there is an obstacle to the left of the destination, the placing point 'Pp' should be higher than the obstacle to the left of the destination.

The human expertise acquired in assembling the second fin-block (i.e., Table 3-4) is almost the same as that acquired in assembling the first one (i.e., Table 1-2). The only difference is in determining the position of the placing point 'Pp' in Table 4. The specific differences have been marked with a grey background. The new generalized indexing rule means that:

If the type of workpiece is block, the type of robot tool is two-finger, the assembly operation is simple_peg_hole_ insertion, there are two obstacles, one is in front of the destination, the other is at the left side of the destination, and half the outer open width of the robot tool (i.e., Outer_open_width/2) is bigger than or equals the distance (i.e., LD)between the destination position and the left obstacle; then, the assembly plan that can be reused is one whose ID is ap00012.

Compared with the indexing rule learned in assembling the first fin-block, this indexing rule requires more precondition features in its head. In other words, the generality boundary of assembly plan 'ap00012' is smaller than that of assembly plan 'ap00011'.

In the planning part, there is a rule defines that the assembly plan of smaller generality boundary should be activated first, this is because such assembly plan has stricter reusing precondition. Therefore, when generating robot program for assembling the third fin-block, assembly plan 'ap00012' will be selected. The generated robot program can be executed successfully in the test.

4.5 Saving Evaluation and Reusing Selection

As more and more assembly tasks are learned by the proposed method, there should be evaluation criteria to decide whether the acquired human expertise for a new assembly task is worth being saved in the planning knowledge base. And the evaluation criteria are:

If an error occurs in the execution of a robot program generated by the planning function, and the function asks human workers to revise the robot program; then the human expertise acquired from this error-recovery assembly task must be saved in the planning knowledge base.

If any one (or more than one) of the following four points: 1. name of workpiece, 2. name of

tool, 3. assembly operation, or 4. destination environment in the new assembly task description is different from that (or those) in the assembly task descriptions of the saved human expertise; then the human expertise acquired from this new assembly task is worth saving in the planning knowledge base.

For example, when the planning function reused the assembly plan of the first fin-block for assembling the second fin-block, a collision occurred during the execution of the generated program. Then, the workers revised the generated program to solve this error. Thus, the human expertise acquired from assembling the second fin-block must be saved, although the human expertise acquired from assembling the first fin-block has already been saved in the planning knowledge base.

To assemble the third fin-block in the right hole, the assembly plan of the second fin-block is reused. The generated program can be executed successfully. Because the name of the workpiece, the name of the tool, the assembly operation, and the destination environment in the assembly task description for assembling the third fin-block are the same as those for assembling the second fin-block, the human expertise acquired from assembling the third fin-block will not be saved in the planning knowledge base for this does not satisfy either of the evaluation criterion of saving acquired human expertise.

Herein, both the assembly plans of the first and the second fin-blocks are saved in the planning knowledge base. Thus, a problem for assembling the third fin-block is that which assembly plan should be selected. The reusing selection is done by the following 4 steps:

Step1: When a new assembly task is input into the planning function, there is a set of general rules used to analyze features of the new task.

Step 2: An indexing rule will be activated if the features in its head can be satisfied/matched by the analyzed features of the new input task.

Step 3: The result of Step 2 has 3 possible cases: 1. No indexing rule can be activated. Then, the workers have to teach the robot. 2. Only one indexing rule is activated. Then, the planning part reuses the corresponding assembly plan to generate program for the new input task. 3. More than one indexing rules are activated. Then, go to step 4.

Step 4: If there are more than one indexing rules are activated in Step 3, then the indexing rule that has the largest number of features in its head will be selected.

Compared with the indexing rule of the first fin-block, the indexing rule of the second fin-block has more features (i.e., obstacle(left,LD,_), tool(_,_,_,_,Outer_open_width,_,_), LD <= Outer_open_width/2) in its head. Thus assembly plan of the second fin-block is selected for assembling the third fin-block. This is because the more features an indexing rule has in its head, the more preconditions the indexing rules requires. In other words, the indexing rule that has more features has a smaller generality boundary and is more similar as the new input task.

4.6 Conclusion

The modified EBL method proposed in the Chapter 3 is improved in this chapter. The improved EBL method has two learning modes. The first learning mode is a hierarchical use of the modified EBL presented in Chapter 3 for higher level tasks. The second learning mode can learn from error recovery training examples, which is the novel proposal of the improved EBL method. Saving evaluation criteria of the acquired knowledge and its reusing method are also discussed in this chapter.

It is worth noting that the acquired knowledge is represented as different levels of knowledge modules in the assembly plan. The knowledge modules in an assembly task can not only be reused in whole, but also can be flexibly reused in part. The method for flexibly reusing the knowledge modules will be given in the next chapter.

APPENDIX

Explanation for Terminology in Chapter 4

There are two kinds of terminology in this chapter. One is used in robot programming. The other is developed by the authors, and used in the proposed method. Table 4.5 and Table 4.6 in the appendix respectively explain the two kinds of terminology.

Table 4.5 Explanation for terminology used in robot programming.

Terminology	Explanation	Location
Mov	Robot command. Using joint interpolation movement, moves from the current position to the destination position, e.g., Mov p1 means moving to point p1 with joint interpolation operation.	Section 3 Section 4.1~4.2
Mvs	Robot command. Using linear interpolation movement, moves from the current position to the destination position, e.g., Mvs p2 means moving to point p2 in straight line.	Section 3 Section 4.1~4.2
Ovrd	Robot command. Specifies the speed of the robot movement as a value in the range from 1 to 100%, e.g., Ovrd 10 means specifying the robot speed as 10% of the maximum speed.	Section 3 Section 4.1~4.2
Dly	Robot command. Causes a wait, e.g., Dly 1 means delaying 1 second before executing the next command.	Section 3 Section 4.1~4.2
HClose/ Hopen	Robot command. Commands the robot tool to open or close, e.g., HClose 1 means closing the robot tool whose number is 1.	Section 3 Section 4.1~4.2
M_Novrd	It is a system status variable, which indicates the initial override value that is the general robot speed set in the automatic running mode.	Section 3,4
p1~p8	Point parameter. The target points defined in the robot movement path.	Section 3 Section 4.1~4.2
p2,-50	Point parameter. Approach point. ‘p2,-50’ means a point at the position retracted 50mm in the robot tool direction, as shown in Fig. 4.	Section 3 Section 4.1~4.2

Note: The explanation for the robot-programming terminology is obtained from the instruction manual of the robot controller, ‘MELFA Industrial Robots Instruction Manual CR1/CR2/CR4/CR7/ CR8/CR9 Controller’.

Table 4.6 Explanation for terminology developed in the improved EBL method.

Terminology	Explanation	Location
Phome	Safe point in the ‘pickup’ part. It is the start point used to avoid collision with obstructions.	Table 1,3
Ps	Seizing point. The point at which the robot seizes the target workpiece.	Table 1,3
Ds	Approach distance between the approach point and the seizing point.	Table 1,3
Ts	Delay time after slowing down the speed in the ‘pickup’ part.	Table 1,3
Tc1	Delay time before closing the robot tool.	Table 1,3
Tc2	Delay time after closing the robot tool.	Table 1,3
Tr	Delay time after recovering the normal speed.	Table 1,3
Pphome	Safe point in the ‘place’ part. It is the start and end points of the ‘place’ part.	Table 1~4
Pp	Placing point. The point at which the robot releases its tool to assemble the target workpiece.	Table 2,4
Dp1	The approach distance between the first approach point and the placing point. Usually, the first approach point is used to avoid collision with obstructions near the destination.	Table 2,4
Dp2	The approach distance between the second approach point and the placing point. Usually, the second approach point is used to align with the destination.	Table 2,4
Tsp	Delay time after slowing down the speed in the ‘place’ part.	Table 2,4
To1	Delay time before opening the robot tool.	Table 2,4
To2	Delay time after opening the robot tool.	Table 2,4
H	The height between the placing point and the top orifice of the destination hole.	Table 2,4
LD	The distance between the destination of the workpiece and the obstruction on the left side of the destination.	Indexing rule in Section 4.2

Chapter 5

Flexible Reuse of Hierarchical Knowledge Modules and Automatic Programming for Robots

5.1 Introduction

In this chapter, the method for flexibly reusing different levels of knowledge modules acquired in an assembly plan is presented. In addition, the KBS proposed in Chapter 3 is improved and transformed into a case-based system. In the case-based system, the past tasks and their corresponding acquired knowledge are saved as cases in a cases base. The case base and the domain theory of the KBS make up the knowledge base of the case-based system. The learning function developed in Chapter 3 and Chapter 4 is embedded in the retaining agent of the case-based system. The planning function introduced in the last 2 chapters is assigned to and improved by the retrieving and reusing agents of the case-based system.

Section 5.2 introduces the structure of the case-based system and the case representation form.

Section 5.3 presents how the retrieving and the reusing agents generate program for new tasks by flexibly reusing the knowledge modules in past cases.

Section 5.4 gives the experiments and related analysis.

Section 5.5 compares the proposed method with other off-line automatic programming methods.

Section 5.6 gives the conclusion.

5.2 The Case-based System for Flexibly Reusing Hierarchical Knowledge Modules

5.2.1 The System Structure

As Figure 5.1 shows, the case-based system is a support system for the host control system of a robot work cell (HCSRWC). Its input is an assembly task description. Here, an assembly task refers to the process (or the requirements) of assembling a single workpiece, including picking it up from its initial position and placing it at its destination. Its output is the successfully tested robot program for an input task. The HCSRWC collects the outputs for different tasks of assembling various workpieces and then integrates them with control programs of other facilities (i.e., sensors, conveyors, fixing devices, and so on) to formulate a comprehensive program for the robot work cell to assemble a product.

The core of the case-based system is its knowledge base, which is composed of the case base, the rules base, the workpiece database, and the robot tool database. The case base is the most important component, which is a reservoir of past robot-teaching experiences that can be updated as new experiences are retained as cases. The rules base contains domain knowledge on robotic manufacturing and other general rules (e.g., retrieving rules, case retaining evaluation rules, and so on) that are used in retrieving, reusing, revising, and retaining processes. The workpiece and robot tool databases provide data on workpieces and robot tools, respectively. In our case-based system, the four processes within the original CBR framework (i.e., retrieving, reusing, revising, and retaining) are considered the four agents.

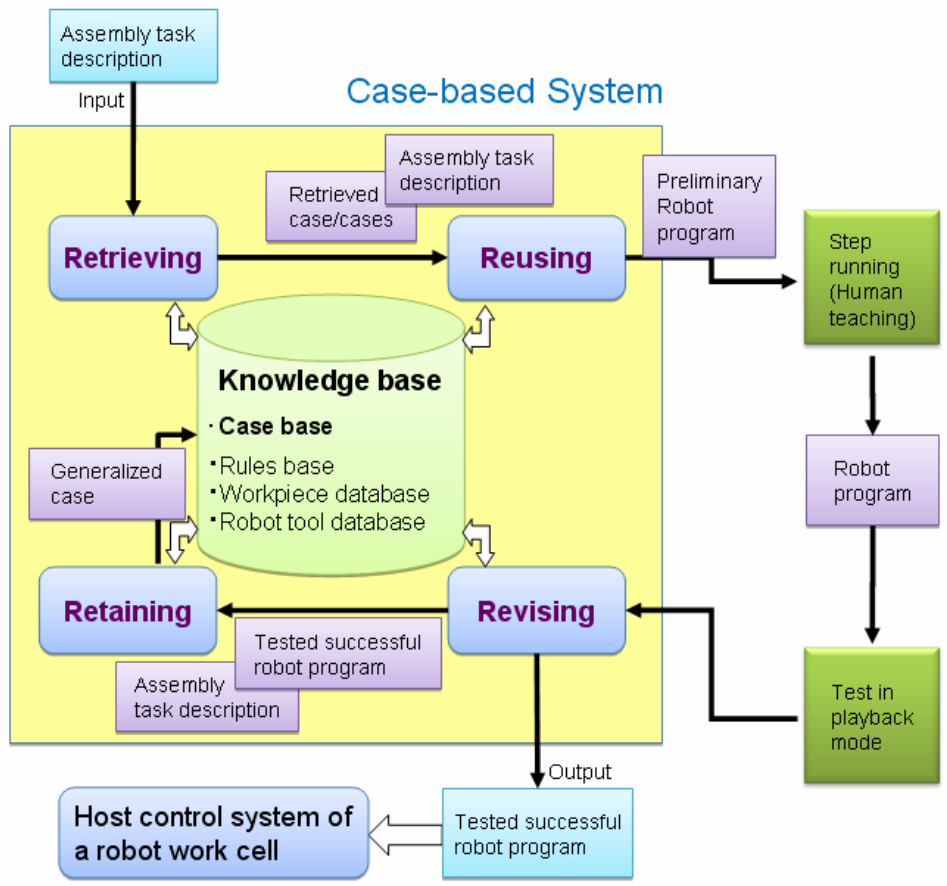

Figure 5.1 The structure of the case-based system.

The retrieving agent is in charge of selecting the appropriate case(s) for an input assembly task.

The reusing agent uses the retrieved case(s) and data that are obtained from the assembly task description and workpiece and robot tool databases to generate a preliminary robot program. Here, if the retrieved case(s) is not adequate to satisfy all the requirements of the input assembly task, then the reusing agent reports the incomplete part to the worker. The worker teaches the robot the incomplete part. In this way, tasks that have been met in past cases are addressed by the reusing agent so that the worker just needs to teach the robot new tasks. Thus, the reusing agent helps reduce

robot-teaching time.

The revising agent helps the worker revise the preliminary robot program. The worker tests the preliminary robot program in playback mode. If the worker is satisfied with the execution of the preliminary robot program, then he/she does not revise it. However, if an error occurs in the execution or the worker is not satisfied with the performance, then the worker revises the preliminary robot program. In this case, the revising agent provides error diagnosis and error recovery information to the worker.

The retaining agent takes the assembly task description and the successfully tested robot program and generalizes knowledge on this basis. The learning result is the assembly task and its corresponding generalized robot program, which is saved in the case base.

This chapter mainly presents how the retrieving and reusing agents automatically generate a preliminary robot program by using past robot-teaching cases. For this purpose, it is necessary to first describe the representation forms of the past cases.

5.2.2 Case Representation

As shown in Figure 5.2, in the case-based system, an assembly task is represented as a case with three components: 1.) the description of the assembly task; 2.) the assembly plan for the assembly task, i.e., a robot program schema; and 3.) notes about key points in operation.

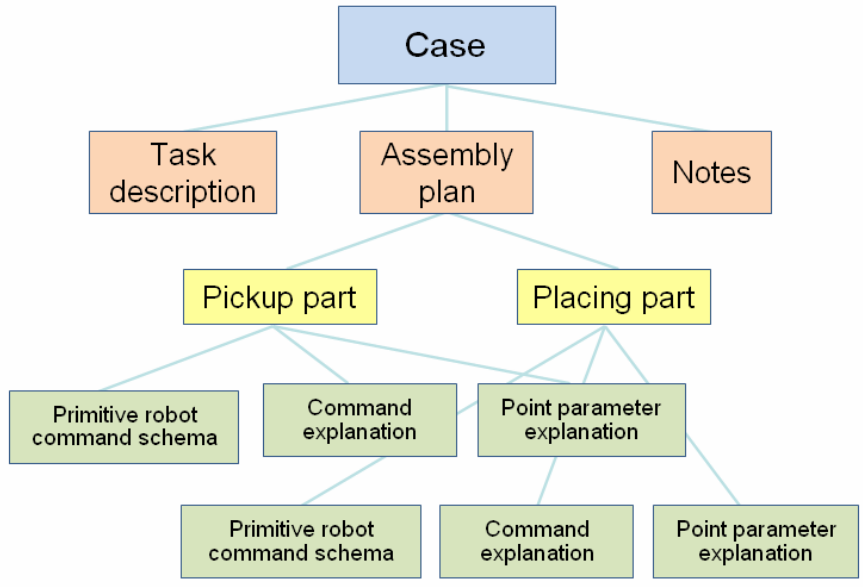

Figure 5.2 The case representation form.

The assembly task description mainly contains the workpiece to be assembled, the robot tool used to assemble it, the assembly operation, the initial position of the workpiece, the assembly destination position, and the destination environment. Here, we assume that all the workpieces are in their standard initial states. This is because it is easy to set a workpiece into a standard initial state

from its random initial states with a robot and a vision sensor. The items in an assembly task description contain features of the assembly task. Thus, they are used to retrieve past cases for a new task.

The assembly plan, i.e., the robot program schema, contains knowledge regarding the case. Because the task of assembling most workpieces can be segmented into ‘pickup’ and ‘place’ phases, most of the assembly plans in the case representations are also divided into the ‘pickup’ and ‘place’ parts. The ‘pickup’ and ‘place’ parts in each case have different preconditions to be reused. Thus, the ‘pickup’ and ‘place’ parts can partly be reused for a new task. In addition, each of the ‘pickup’ and ‘place’ parts in a case is composed of three parts, namely, the primitive robot command schema (PRCS), the command explanations (CE), and the point parameter explanations (PE). The PRCS is used for generating a robot program for a new task. The CE is a natural language translation of the PRCS, which is used to explain the operation of the PRCS to workers and also can be used to generate the robot program. The PE is used to calculate point parameters in the robot program. The preconditions for reusing the PRCS, the CE, and the PE present different constraints. As such, they can be reused separately for different new tasks. The constraints of reusing the PE are much stronger than those of reusing the PRCS and CE. How different parts of a case can be reused flexibly for different, new tasks is the core of this paper, which will be explained in detail in the next section.

The notes in a case are used to emphasize to workers the key points that may lead to errors in the execution of the robot program.

A case and its components are indexed by their respective IDs.

5.3 Case Reuse in Automatic Programming

In the case-based system, the retrieving agent and the reusing agent can automatically generate the preliminary robot program for a new assembly task in three steps. First, the retrieving agent analyses features of the input task description. Second, the retrieving agent selects the most similar case or parts of several cases by comparing features of past cases with the analyzed features of the new task and sends the relevant IDs to the reusing agent. Third, the reusing agent generates the robot program based on the selected case(s) and the data from the task description and the workpiece and tool databases.

5.3.1 Feature Analysis of a New Assembly Task

In this section, the assembly of a circuit breaker is used to illustrate the critical features in selecting a case and how the retrieving agent analyses features of the input task description. Figure 5.3 shows the circuit breaker and the workpieces used to assemble it. In Figure 5.3(b), the workpieces from up to down and from left to right are as follows: body, armature, cover, different types of bolts,

fin-blocks, l-junctions, square-junctions, z-junctions, flat-with-springs, switch-handle, and brown-squares. Figure 5 shows the sequence of the assembly tasks in assembling the circuit breaker. In Figure 5.4, the task of assembling a circuit breaker is hierarchically decomposed into several levels [29]. The underlined items are the basic-level tasks, i.e., the assembly tasks that are inputted into the proposed system.

Figure 5.3 The circuit breaker and its workpieces

(a) Circuit breaker.

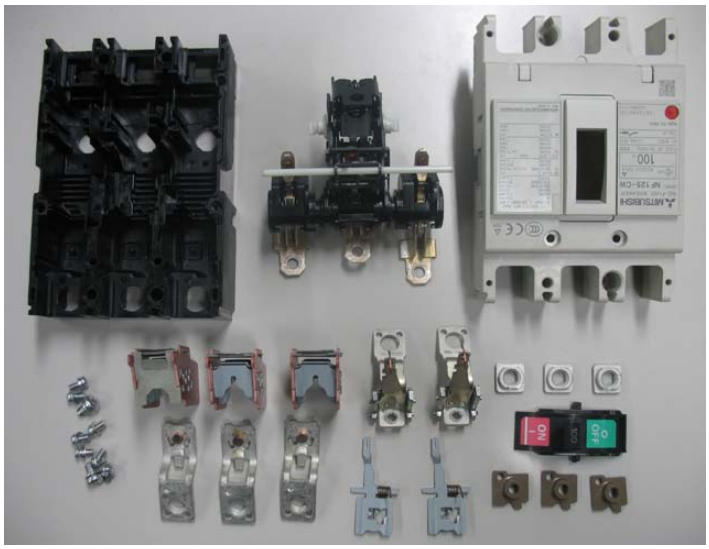

Figure 5.3 The circuit breaker and its workpieces

(b) Workpieces.

After the description of one assembly task is inputted, four main factors are considered by the retrieving agent, namely, the workpiece, the tool, the assembly operation, and the destination

environment. Features of the workpiece and the tool determine how to pick up the workpiece, while features of the assembly operation and the destination environment have impact on how to place the workpiece at the destination. Figure 5.5 shows the content of an assembly task description. In Figure 5.5, the initial environment is not considered by the retrieving agent but is used by the reusing agent to calculate the seizing point coordinates.

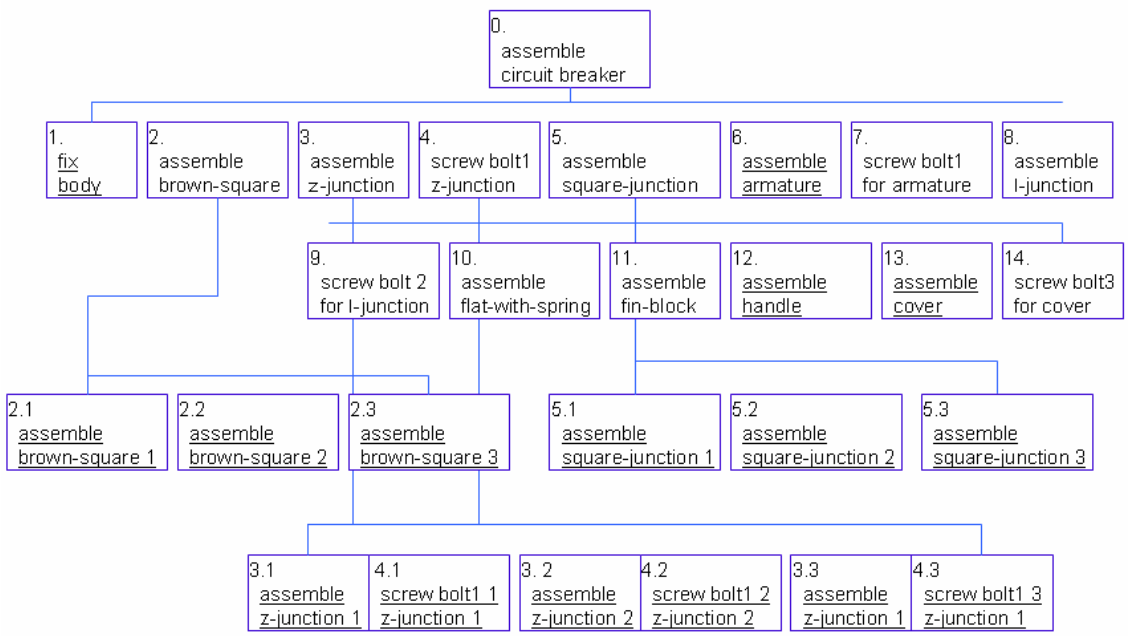

Figure 5.4 Task analysis of assembling a circuit breaker.

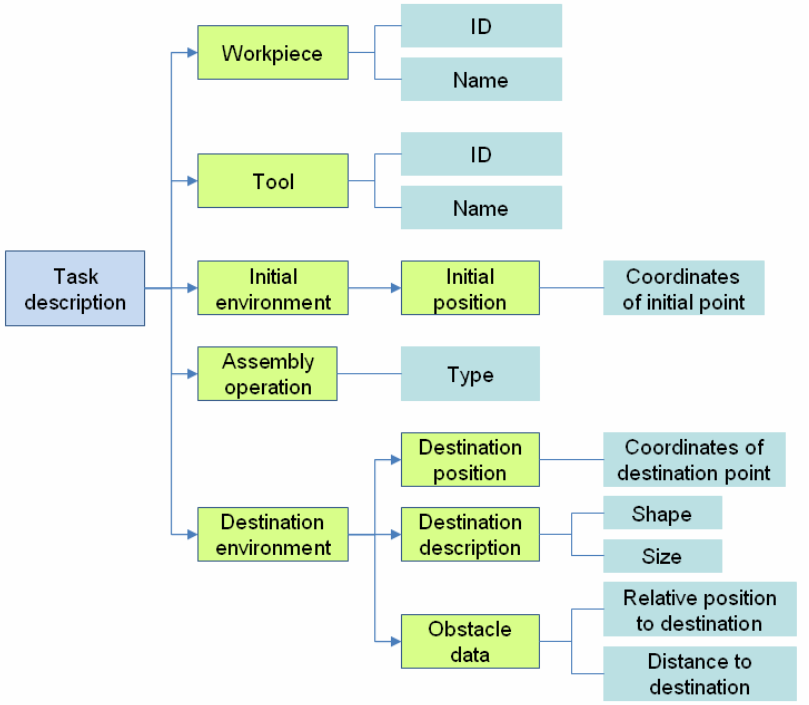

Figure 5.5 Task description.

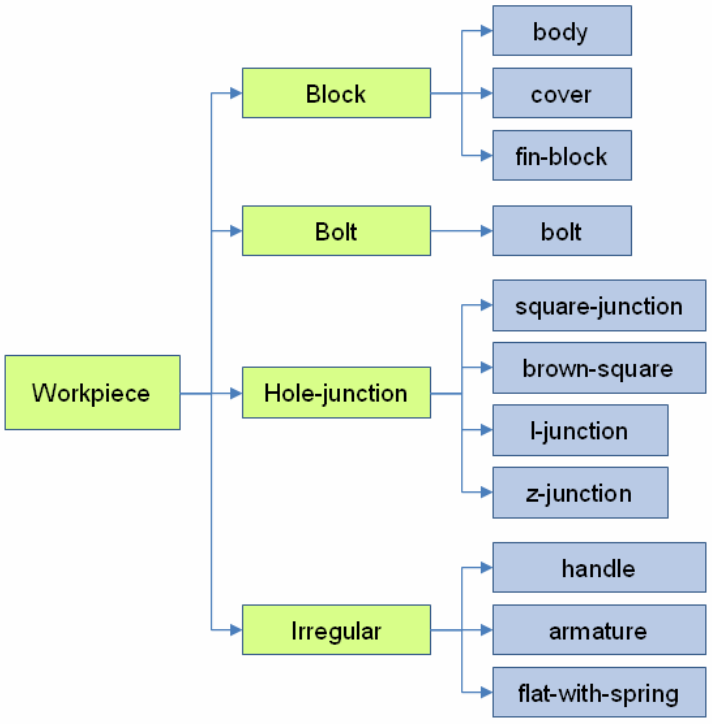

Figure 5.6 Classification of workpieces.

With the IDs of the workpiece and the tool, the retrieving agent can retrieve related data on them, which are saved in the workpiece database and the tool database, respectively. The data on a workpiece include its ID, name, type, sub-type, size, weight, and so on. The retrieving agent mainly employs the type and size of the workpieces to calculate its similarity with the workpieces in past cases. Figure 5.6 shows the classification of the workpieces of the circuit breaker, which has two levels, namely, type and sub-type. Types are classified by the shape of the workpiece, which can be classified as block, bolt, hole-junctions, and so on. For example, in Figure 5.3(b), the body, cover, and fin-blocks belong to the same type, the block. Sub-types are classified by function of the workpiece. For example, ‘fin-block’ is a sub-type of a group of workpieces that refers to a kind of arc-extinguishing devices used in circuit breakers. Different types of circuit breakers have fin-blocks with different sizes, weights, and even internal structures. However, all of the fin-blocks have the same sub-type of ‘fin-block’ because they have very similar shapes and the same functions.

The data on a tool refer to its ID, name, type, size, weight of load, and so on. Figure 5.7 shows the classification of the tools used in robotic manufacturing and the important data that should be considered. The types of tools are classified according to their functions. A tool in the ‘gripper’ category keeps its fingers open when it approaches the target workpiece and seizes the target workpiece by closing its fingers. A ‘plug’ is used to assemble a hole-junction, which keeps its fingers closed, inserts them into the hole, opens the fingers, and seizes the workpiece. A ‘vacuum’ is used to

absorb and twist workpieces; e.g., bolts are screwed by a vacuum. A ‘hook’ and ‘thruster’ are used to provide temporary support such as pulling, pushing, and so on. The retrieving agent compares the size of a tool with the size of the workpiece and data on the destination environment to select a tool. For example, when a ‘gripper’ is selected to assemble a block, the inside open width of the ‘gripper’ should be larger than the width of the workpiece to avoid a failure to pick up the block, and the outer open width of the ‘gripper’ should be smaller than the obstacle distance to prevent it from colliding with an obstacle near the destination.

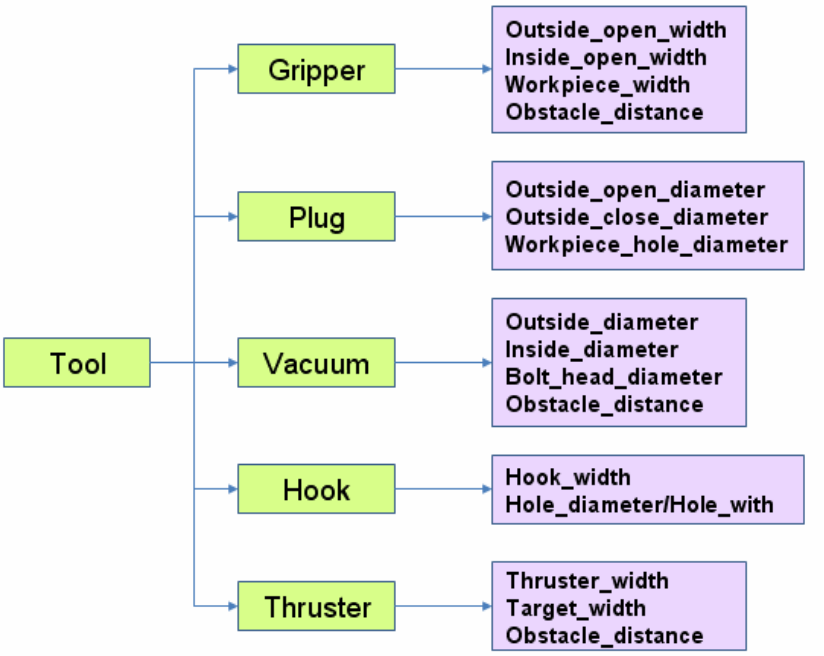

Figure 5.7 Classification of tools.

Feature of assembly operation refers to its type. Nevins and Whitney [44] have proposed 12 basic assembly operations. The commonly-used assembly operations include ‘simple-loose-peg-hole-insertion,’ ‘simple-tight-peg-hole-insertion,’ ‘multiple-loose-pegs-holes-insertion,’ ‘multiple-tight-pegs-holes-insertion,’ ‘screw,’ ‘push,’ ‘pull,’ and so on.

Features of the destination environment include the destination position, the destination description, and nearby obstacles. The destination description indicates the shape and the size of the target area; e.g., when a block is inserted into a hole, the destination description includes the shape and size of the orifice of the hole and its depth. The obstacles can be classified according to their relative positions to the destination, which include front-obstacle, back-obstacle, side-obstacle, above-obstacle, and so on. The distance between the obstacle and the destination should also be given in the task description.

This section has introduced the features of the input task description that should be analyzed by

the retrieving agent. The next section presents how the retrieving agent selects cases for the new task.

5.3.2 Case Selection

The case base in the proposed system is a sorted base; i.e., past cases are classified by key features, such as workpiece type and sub-type, tool type, operation type, and destination environment, when they are saved in the case base. Therefore, after analyzing the features of the new task description, the retrieving agent selects a past case or parts of cases by comparing the features of the new task with the features used in classifying the case base to find whether there exists a case that satisfy certain requirements. Figure 5.8 shows the case selection method used by the retrieving agent. In Figure 5.8, the same destination environment implies that the destination descriptions and the obstacle data are the same. A similar destination environment refers to a situation in which only the relative positions to the destinations in the obstacle data are the same. The detailed case selection procedure is as follows.

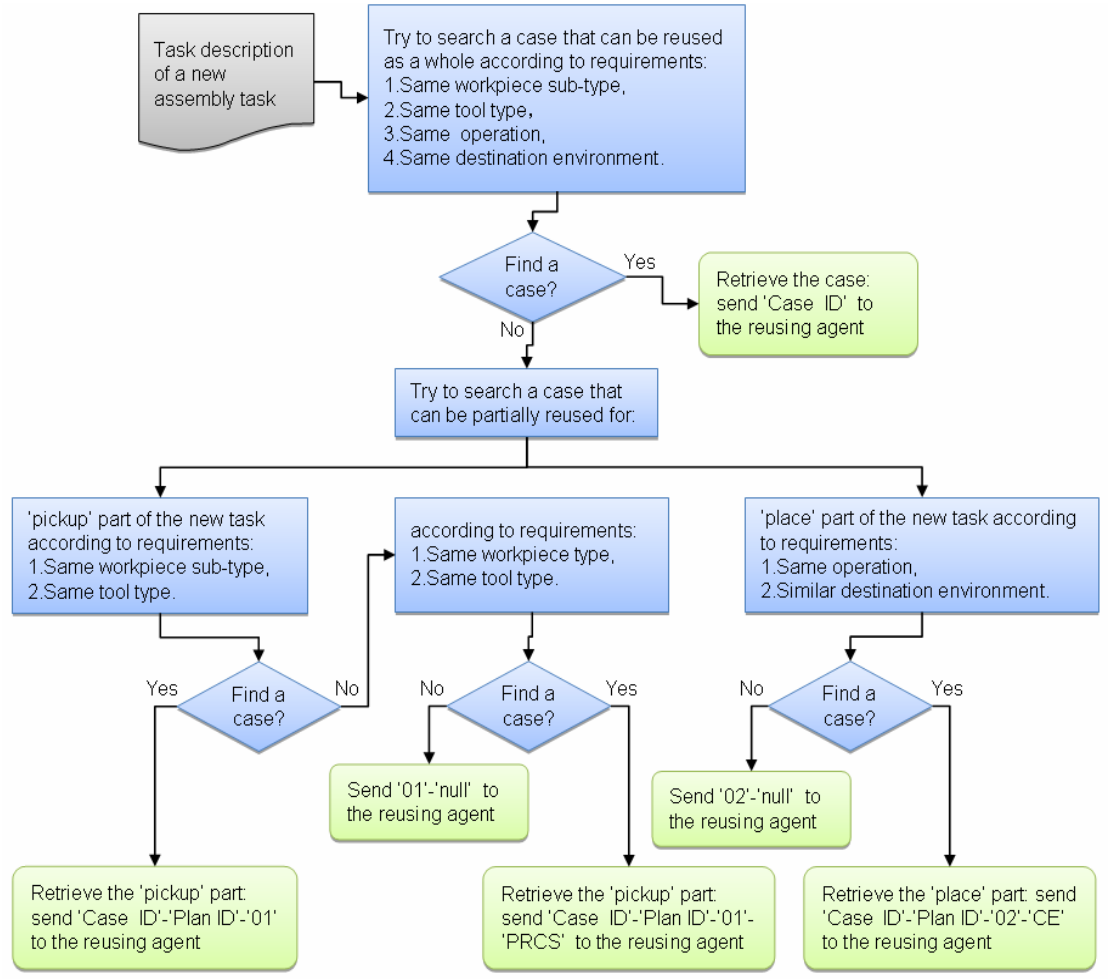

Figure 5.8 The case selection procedure.

Step 1: If all the four features, i.e., the workpiece sub-type, the tool type, the assembly operation, and the destination environment of a case are the same as those of the new input task, then the case is selected, and its 'Case ID' is sent to the reusing agent. This selected case can be reused in whole.

Step 2: If there is no case in the case base that can satisfy the four requirements outlined in Step 1, then the retrieving agent looks for cases that can be reused in part for the new input task. If the workpiece sub-type and tool subtype of an existing case are the same as those of the new input task, then the PRCS and PE of the 'pickup' part of this case can be reused for the new task, and a message 'Case ID'-'Plan ID'-'01' of this case is sent to the reusing agent. In the case base, there may be several cases in which the workpiece sub-type and tool subtype are the same as those of the new task. In this case, the first-met case is selected by the retrieving agent. In the following steps, the same 'first-met first selected' strategy is used.

Step 3: If there is no case that can satisfy the two requirements outlined in Step 2, then the retrieving agent relaxes its requirements. If the workpiece type and tool sub-type of a case are the same as those of the new task, then the PRCS of the 'pickup' part of this case can be reused for the new task, and a message 'Case ID'-'Plan ID'-'01'-'PRCS' of this case is sent to the reusing agent.

Step 4: If there is no case that can satisfy the two requirements outlined in Step 3, then there is no case in the case base that can be reused for the 'pickup' part of the new task, and a message '01'-'null' is sent to the reusing agent. In this case, the retrieving agent tries to find a case that can be reused for the 'place' part of the new task.

Step 5: If the assembly operation of a case is the same as that of the new task, and their destination environments are similar, then the CE of the 'place' part of the case can be reused for the new task, and a message 'Case ID'-'Plan ID'-'02'-'CE' is sent to the reusing agent. Here, two destination environments are similar if the number of obstacles near the destinations and the relative positions between the obstacles and the destinations are the same.

Step 6: If there is no case that can satisfy the two requirements outlined in Step 5, then there is no case in the case base that can be reused for the 'place' part of the new task, and a message '02'-'null' is sent to the reusing agent.

The above presents how the retrieving agent retrieves cases from the case base. Generally, the features of the workpiece and tool determine whether the 'pickup' part of a case can be reused, and the features of the assembly operation and destination environment influence whether the 'place' part of a case can be reused.

5.3.3 Case Reuse for Automatically Generating Robot Programs

By using the message sent from the retrieving agent, the reusing agent recalls the corresponding case or parts of cases to generate a program for the new task. Figure 5.9 shows how the reusing agent deals with this message. Note that there are seven possible cases of this message.

Case 1: The message is ‘Case ID.’ Then the reusing agent can use the PRCS in the retrieved assembly plan to generate commands of the robot program for the new task and use the PE in the retrieved assembly plan and the initial position coordinate and the destination position coordinate in the task description of the new task to generate point parameters of its robot program. In this way, the entire preliminary robot program is generated by the reusing agent.

Case 2: The message is ‘Case ID1’-‘Plan ID1’-‘01’ and ‘Case ID2’-‘Plan ID2’-‘02’-‘CE.’ Then the reusing agent can use the PRCS and PE in the ‘pickup’ part of ‘Case ID1’ to generate robot commands and point parameters for the ‘pickup’ part of the new task, respectively. It also can generate robot commands for the ‘place’ part of the new task by reusing the CE of ‘Case ID2.’ The reusing agent presents the generated part of the robot program to the worker through the human-system interface of the system and tells the worker that the point parameters for the ‘place’ part of the new task should be taught by the worker. Then, the worker uploads the partially-completed preliminary robot program to the robot controller and teaches the uncompleted part, i.e., the point parameters for the ‘place’ part of the new task, with the teaching pendant.

Case 3: The message is ‘Case ID1’-‘Plan ID1’- ‘01’ and ‘02’-‘null.’ The reusing agent can use the PRCS and PE in the ‘pickup’ part of ‘Case ID1’ to generate robot commands and point parameters for the ‘pickup’ part of the new task, respectively. In addition, it asks the worker to teach the ‘place’ part of the new task.

Case 4: The message is ‘Case ID1’-‘Plan ID1’- ‘01’-‘PRCS’ and ‘Case ID2’-‘Plan ID2’-‘02’-‘CE.’ Then the reusing agent can use the PRCS in the ‘pickup’ part of ‘Case ID1’ and the CE in the ‘place’ part of ‘Case ID2’ to generate robot commands for the new task. It also asks the worker to teach the point parameters for both the ‘pickup’ and ‘place’ parts of the new task.

Case 5: The message is ‘Case ID1’-‘Plan ID1’- ‘01’–‘PRCS’ and ‘02’-‘null.’ The reusing agent can use the PRCS in the ‘pickup’ part of ‘Case ID1’ to generate robot commands for the ‘pickup’ part of the new task. It also asks the worker to teach the incomplete parts, i.e., the point parameters for the ‘pickup’ part, and the robot commands and the point parameters for the ‘place’ part of the new task.

Case 6: The message is ‘01’-‘null’ and ‘Case ID2’- ‘Plan ID2’-‘02’-‘CE.’ Then the reusing agent can use the CE in the ‘place’ part of ‘Case ID2’ to generate robot commands for the ‘place’ part of the new task. It also asks the worker to teach the uncompleted parts, i.e., the robot commands and the point parameters for the ‘pickup’ part, and the point parameters for the ‘place’ part of the new task.

Case 7: The message is ‘01’-‘null’ and ‘02’-‘null.’ Then the reusing agent asks the worker to teach all four parts, i.e., the robot commands and the point parameters for the ‘pickup’ part, and the robot commands and the point parameters for the ‘place’ part of the new task.

The above has presented how the reusing agent deals with the message sent by the retrieving

agent as well as how it reuses the PRCS, CE, and PE of a previous case in whole or in part to generate the preliminary robot program for a new task. The PRCS can be used to directly generate robot commands for the new task. The CE is transformed into robot commands by a set of language processing rules in the rules base. This set of rules can be defined by referring to the manual of the robot controller [45]. The PE is a set of rules that are used to calculate point coordinates for the new task. By substituting an initial point coordinate and a destination point coordinate in the task description and data on the target workpiece and the tool used for variables in the PE, the coordinates for the seizing point, placing point, and approach points can be calculated. The PRCS, CE, and PE in the assembly plan of a case are learned by the improved EBL method developed in Chapter 3 and 4, which is embedded by the retaining agent.

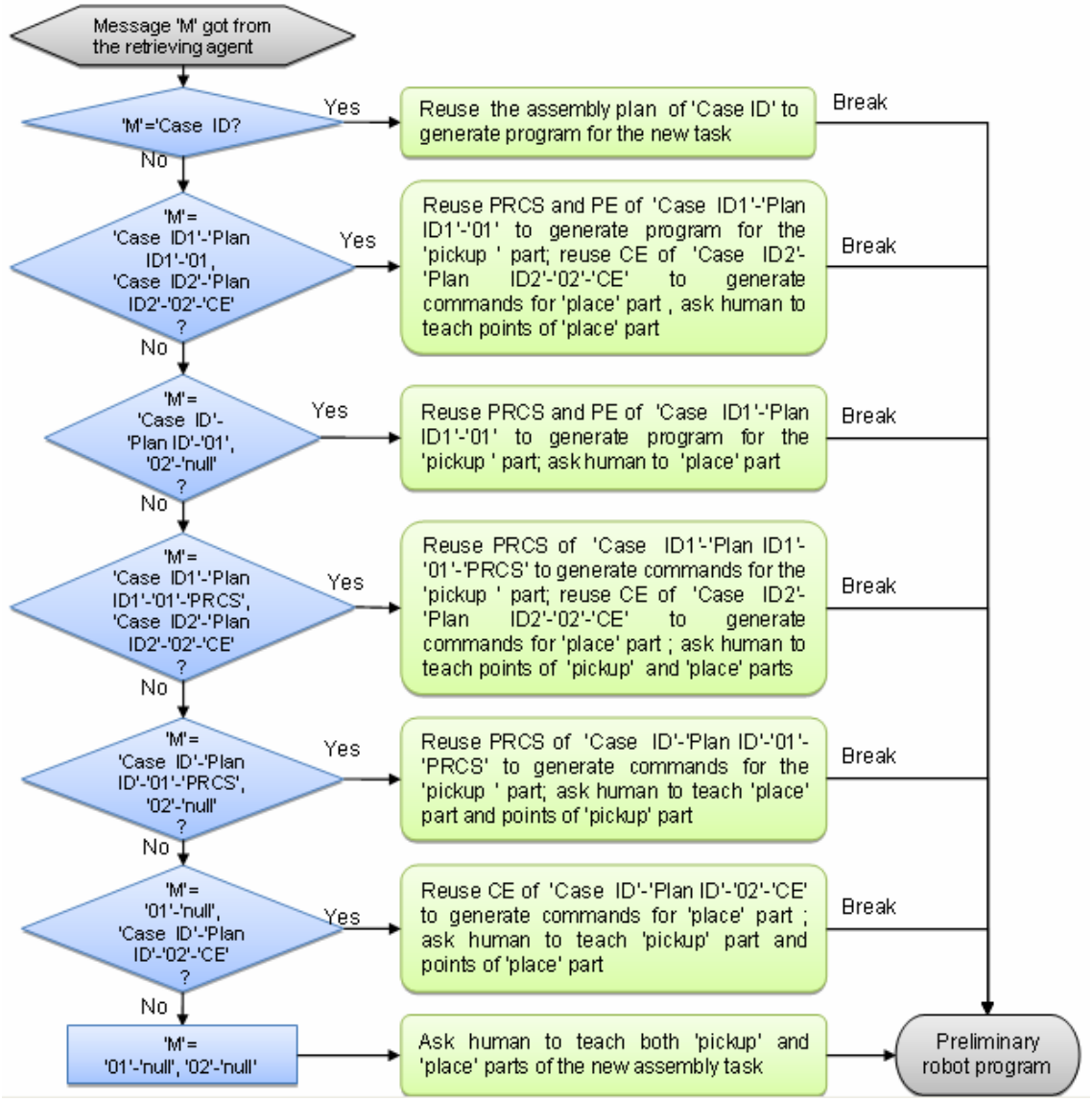

Figure 5.9 The case reusing procedure.

When there is no appropriate case available for a new task, the reusing agent cannot generate the robot program or can only generate part of the robot program. In this case, the reusing agent outputs the completed part of the program to the worker, reports which part has not been completed, and asks the worker to teach the incomplete part through the human-system interface. Then the worker uploads the generated preliminary robot program to the robot controller, teaches the incomplete part, and tests the robot program.

5.4 Experiments

In this section, two experiments are provided to illustrate how the retrieving agent and reusing agent automatically generate the preliminary robot program for a new task. The experiments were not conducted with simulation software but rather were carried out with the real robot work cell [30]. This is because the proposed system is developed to collaborate with workers in real-world robot teaching.

5.4.1 Reuse of a Single Case

The first experiment is the assembly of fin-blocks. In Figure 4.2, the left photo shows the fin-block, and the right photo shows the destination environment. In a circuit breaker, three fin-blocks are assembled. The assembly of the first fin-block was taught by workers and saved as a case in the case base.

Because the input task description of the second fin-block is the same as that of the first one except for the initial position and destination position, the proposed system uses the case of assembling the first fin-block in whole and automatically generates the program. However, an error occurs in the step-running test. The tool collides with the assembled first fin-block when it approaches the destination to place the second one. This is because during the assembly of the first fin-block, there was no obstacle near the destination. To make a stable insertion, the tool placed the first fin-block at a low position, which was about 6 mm above the orifice of the target hole. However, when assembling the second fin-bock, the assembled first fin-block becomes a side-obstacle, which has a height of about 20 mm above the orifice of the hole. Moreover, the distance between the assembled fin-block and the destination is 18 mm, which is less than half of the tool's outer open width, which is 22.5 mm. Consequently, a collision error occurs. To resolve this error, the worker raises the placing point of the second fin-block to a height of about 30 mm above the orifice of the target hole. The error recovery is learned and saved as a new case into the case base such that the task description of the second fin-block is revised by adding the obstacle data. Then, when assembling the third fin-block, the input task description includes the obstacle data. The case of the second fin-block is selected to generate the program, which can be executed successfully during

step-running.

The experiment reveals that case selection is determined by the input task description. Mistakes in the task description, e.g., oversight of an obstacle, lead to the misuse of past cases. Therefore, the use of sensors or other methods to obtain an adequate task description should be researched in future.

5.4.2 Partial Reuse of Parts of Several Cases

The second experiment is to assemble the l-junction. In Figure 5.3(b), 'z-junction,' 'square-junction,' and 'l-junction' are sub-types of workpieces. They are of the 'hole-junction' type because they all have large holes in them. Figure 5.10 provides the assembly sequence of the hole-junctions and the armature, which is 1.) z- junction, 2.) square-junction, 3.) armature, and 4.) l-junction. Figure 5.11 shows the tool used to assemble the hole-junctions, which belongs the 'plug' type and has three fingers.

By the time the new task of assembling the first l-junction is inputted into the case-based system, the cases of assembling the z-junction and square-junction have already been saved in the case base. As the task description of assembling the first l-junction is inputted, the retrieving agent analyses its features and saves them as facts in a temporary file. The analyzed feature facts are workpiece_type(hole-junction), workpiece_sub-type(l-junction), tool_type(plug), operation(tight-peg-hole-insertion), operation(push), and obstacle(above,_,_). Then the retrieving agent matches the analyzed features against the features of the cases in the case base to look for a case that can be reused. No case can be found. Afterwards, the retrieving agent tries to find whether parts of several cases can be reused.

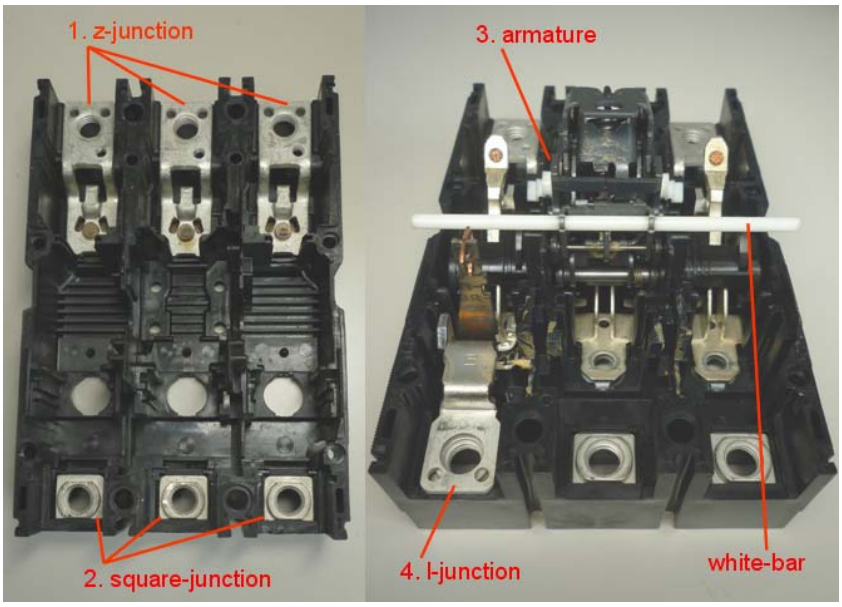

Figure 5.10 The assembly sequence of hole-junctions.

For the case ‘assemble z-junction,’ its features (i.e., workpiece_type(hole-junction) and tool_type(plug)) partly satisfy the features of ‘assemble l-junction’ (i.e., they have the same workpiece type and same tool sub-type). Therefore, the PRCS for the ‘pickup’ part of case ‘assemble z-junction’ can be reused for the new task. In fact, case ‘assemble square-junction’ also has features workpiece_type(hole-junction) and tool_type(plug). Because the retrieving agent first identifies case ‘assemble z-junction,’ it selects this case according to the first-met-first-selected rule.

For the case ‘assemble square-junction,’ its feature (i.e., operation(tight-peg-hole-insertion), operation(push), and obstacle(above,_,_)) partially satisfy the features of ‘assemble l-junction’ (i.e., they have the same operation and similar destination environment). So, the CE for the ‘place’ part of case ‘assemble square-junction’ can be reused for the new task. Because the operation for both the square-junction and the l-junction is ‘tight-peg-hole- insertion,’ a ‘push’ operation is necessary to insert them into the tight destination holes. In ‘assemble l-junction,’ the above obstacle is the with-bar of the armature, as shown in Figure 5.10. The above obstacle in ‘assemble square-junction’ is the black-overhang as shown in Figure 5.12. The above obstacle is such that there should be an intermediate point (i.e., an intermediate motion action) to bypass the above obstacle before approaching the destination.

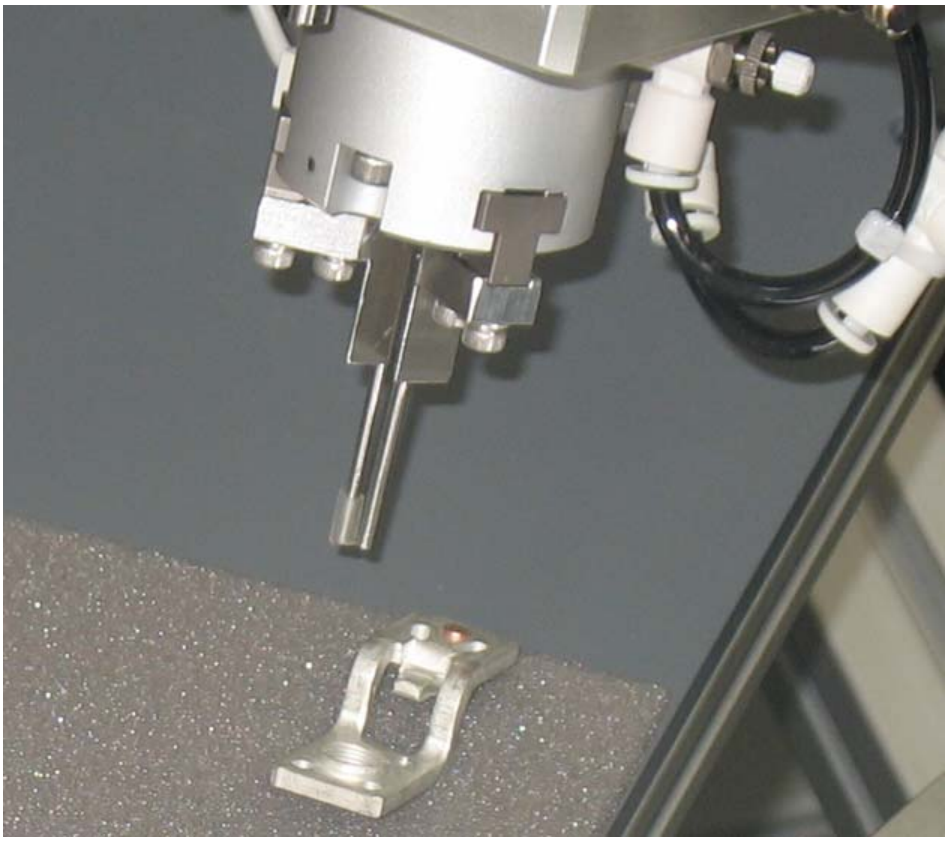

Figure 5.11 The tool 'plug'.

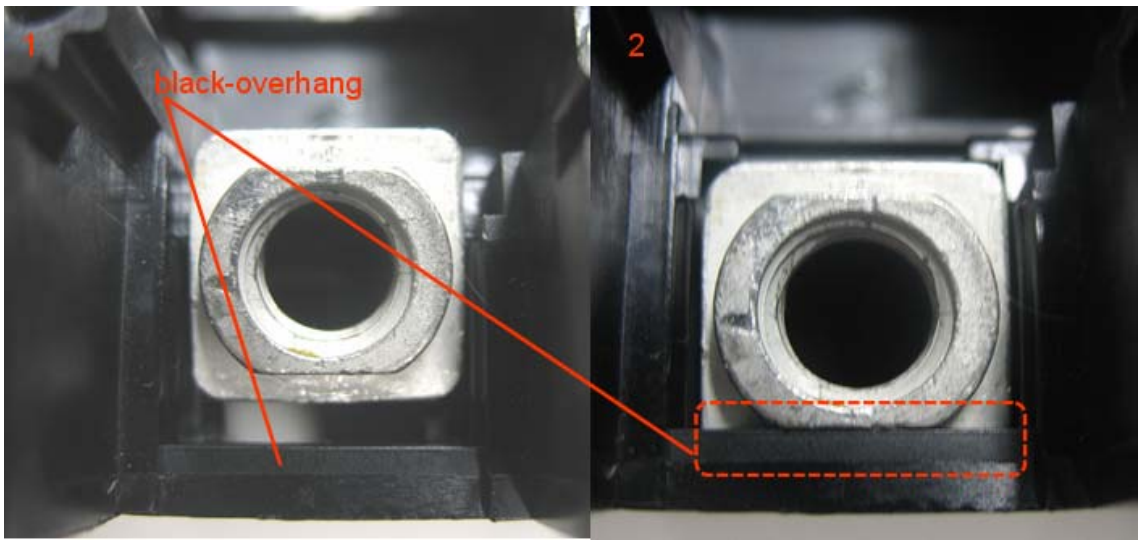

Figure 5.12 The above-obstacle black-overhang.

After receiving the message from the retrieving agent, the reusing agent transforms the PRCS in 'assemble z-junction' and the CE in 'assemble square-junction' into robot commands (i.e., a preliminary program) for 'assemble l-junction,' outputs the robot commands to workers, and reports that the point parameters in the robot program has to be taught by the workers.

This experiment illustrates the following two points. First, a past case not only can be reused in whole but also can be reused in part. Parts of several cases can be reused to synthesize the preliminary program for a new task. Second, in a given case, the robot command module (i.e., the PRCS and the CE) has a broader reusing boundary than the point parameter module (i.e., the PE). Thus, the preliminary robot program generated by the proposed system is not always a complete one. In this case, the preliminary program should be completed by human teaching; e.g., in this experiment, human workers must teach the point parameters in the generated program.

5.5 Discussion

In this section, the proposed method is compared with other off-line automatic programming methods to show its novelty and its potential applicability to other related systems.

Many off-line automatic programming methods have been developed based on simulation environments by using CAD data to generate motion paths for robots. Hiraoka et al. [46] developed a model-based off-line programming system in which robot program for assembly operations could be generated by utilizing the information of an environment model and its high-level description. Czarnecki [47] designed a programming system for a dual robot manufacturing cell to perform the stripping operation in garment manufacturing. In Czarnecki's system, the task description obtained from CAD data was converted into an executable program. Mitsi et al. [48] constructed an off-line programming system for welding operations, which included the graphical simulation of the robot

and its workcell, a kinematic model of the robot, motion-planning and creation of the NC code for the manufacturing process. Webb et al. [49, 50] created a system capable of assembling fuselage components using standard industrial robots and relatively low-cost metrology systems. In their system, the acquired data could be processed through a mathematical algorithm that calculates the relative component positions required for optimal assembly. The data could also be used to check gross distortion of components and to reject those outside the specification limits. Reinhart and Tekouo [51] proposed a system for automated programming of robot-mounted optical scanning devices and algorithms to automatically extract measurement features and generate collision-free robot scanning paths from CAD models.

Similar to related methods, in this study, the proposed method also uses data on workpieces and tools as well as environment data obtained from sensors to calculate the point parameters using the PE. The PE includes the rules or algorithms used to calculate coordinates of seizing points, placing points, approaching points, and so on, which are acquired through knowledge-based deductive learning of past robot teaching programs. On the other hand, the originality of the proposed method is as follows.

1. The proposed method is not to design rules or algorithms to plan robot motion path (i.e., point parameters), as is the case with previous, related methods. Rather, it is developed to generate robot programs that emphasize both robot commands and point parameters. In the author's view, point parameters are important in that they affect the accuracy of the robot motion trajectory. That is why so many methods are developed to generate optimal and accurate robot motion paths. However, this is not enough for robot programming because robot commands in a robot program are not just used to move a robot from one point to another. More importantly, robot commands determine the stability of the real-world execution of a robot program by setting speed changes, delays, the number of intermediate points, and communication signals with other robots or auxiliary devices. With this in mind, the author proposes to include both the robot command schema (i.e., the PRCS and CE) and the point parameter calculation rules (i.e., the PE) of a given case and generate programs for the new task by reusing past learned cases.

2. The main concept behind the proposed method is modular programming. Humans implement their own knowledge when they teach robots manufacturing operations. Such knowledge can be learned and generalized as knowledge modules (KMs) and then can be saved in a knowledge base and reused for similar manufacturing tasks. In this research, the core problems involve how to acquire the KMs, how to represent the KMs and define the granularity of them, and how to retrieve and reorganize the KMs for new tasks. In Chapter 3 and 4, the acquisition method of the KMs was introduced. In this study, the author proposes to represent the KMs hierarchically in terms of cases, such as the PRCS, CE, and PE (i.e., the hierarchical granularity of KMs). This granularity definition enables the KMs to be reused flexibly with relatively low retrieving cost. Retrieving and reusing KM

methods have also been presented in this chapter. However, the proposed system has only recently been developed, and so the case base is still small. Therefore, the current proposed retrieving method will improve as the case base expands in the future.

5.6 Conclusion

This chapter presents the method for flexibly reusing different levels of knowledge modules saved in the past cases. In addition, the KBS proposed in Chapter 3 is improved and transformed into a case-based system. In the case-based system, the acquired knowledge is saved as cases in a cases base. The improved EBL method developed in Chapter 3 and Chapter 4 is embedded in the retaining agent of the case-based system.

Chapter 6

Discussion

6.1 Integration of EBL and CBR for Reducing Utility Cost

In this study, the EBL method is used in acquiring knowledge modules and the CBR method is used in flexibly reusing the acquired knowledge modules. In this section, the merit of integrating EBL with CBR is presented.

CBR is a problem solving paradigm that in many respects is fundamentally different from other major AI approaches. Instead of relying solely on general knowledge of a problem domain, or making associations along generalized relationships between problem descriptors and conclusions, CBR is able to utilize the specific knowledge of previously experienced, concrete problem situations (cases). A new problem is solved by finding a similar past case, and reusing it in the new problem situation. A second important difference is that CBR also is an approach to incremental, sustained learning, since a new experience is retained each time a problem has been solved, making it immediately available for future problems [54].

The problem in applications of CBR is its utility problem, that is, the cost of retrieving the most appropriate case from the case library for a new given problem and the cost of adapting (i.e., reusing and revising) the retrieved case for solving the new given problem. Mantaras, et al. [57] regard the utility problem as a natural trade-off between the benefits of speed-up knowledge and the cost of its application. In their view, the utility problem in CBR systems is caused by the conflict between: 1. the average savings in adoption effort due to the availability of a particular case, which tends to increase efficiency as the case base grows, and 2. the average retrieval time associated with a given case base size, which tends to decrease efficiency. Moreover, as new cases are added retrieval costs become progressively greater but adaption savings progressively less. Therefore, most researchers on CBR focus on developing new retrieval and adaption methods. There are also researchers who have discovered the importance of maintaining the case library to solve the utility problem of CBR [60, 61].

However, in the author's opinion, learning (i.e., retaining) is very important for CBR with regard to solving its utility problem. This is because the retrieval and adaption costs are not solely depend on the amount of cases, but also rest with the representation forms and contents of the cases. In CBR, learning decides the representation forms of cases and the contents that can be learned from cases. Therefore, the basic proposed idea is to save retrieval and adaption costs by making more efforts on post-processing of cases. The aim of emphasizing learning is to post–process cases to make them easier to be retrieved and to be adapted for new problems. The method used to learn cases is EBL.

The idea of EBL is much like that of CBR. Both of them are to acquire knowledge from a single problem-solving example and to reuse the acquired knowledge to solve new problems. The difference between the two is that EBL uses a domain theory to explain why the example is a positive example of the goal concept and generalizes the explanation to form an operational knowledge that can be generally reused for a type of new problems, while CBR doesn't analyze the example, but just directly saves the example as a case into the case library. Therefore, CBR and EBL can be integrated with each other. Armengol, et al. [62, 63] have applied EBL in retrieving appropriate cases in CBR.

In this study, EBL is integrated with CBR to construct the case-based system that is applied in the robotic manufacturing. The proposed system works as a co-worker of human workers to assist them in their task of teaching robots. The main functions of the proposed system are: 1. automatically generating robot programs for new assembly tasks by reusing past learned experiences; 2. providing suggestions and hints for human workers when human workers revise the robot programs or teach robot new assembly skills; 3. acquiring knowledge from revising and teaching demonstrations of human workers. The most distinctive feature of the proposed system is that it uses CBR to help human workers in teaching robots new assembly tasks by reusing past experiences and applies EBL to learn assembly knowledge by observing robot teaching demonstrations of human workers. The EBL learning process can be regarded as a post-processing process of cases before retaining them in the case base. The post-processing process is generalizing the acquired knowledge as a hierarchy of knowledge modules. Its aim is to reduce the retrieving and adapting cost of cases.

6.2 Integration of Manufacturing and Designing for Reducing Lead Time

A problem in current industry is that isolation of designing and manufacturing makes it difficult to achieve essential rapid-response manufacturing. There also exists such a fact that: the problems that are difficult to be solved by tuning robots in the manufacturing phase can be easily solved by making some changes on the design of a product. Thus, it is important to make designers of products know and consider the problems that have happened in the manufacturing phase during they design the products. Because the case-based system proposed in this study saves the tasks and their corresponding knowledge modules, it has the potential to become the connecting bridge between the manufacturers and the designers.

The proposed system can help share knowledge between product designers and manufacturing workers. The purpose is that once the designers finish the design of a product and provide its related information such as the assembly sequence and data of the workpieces, the proposed system can automatically generate the program for the robot work cell to assemble the product. In this way, the

time to market and the manufacturing cost can be dramatically reduced.

Figure 6.1 shows the architecture of a transformation of the proposed system. The transformation makes the designers can interact with the proposed system, and thus connects designing with manufacturing. It is composed of the learner, the planner, and the knowledge base. The designers of products interact with the knowledge-based system through a human-system interface to know the related assembly environment and the problems occurred in the robotic assembly implementation. On the other hand, the designers provide their design results for the knowledge-based system through the interface.

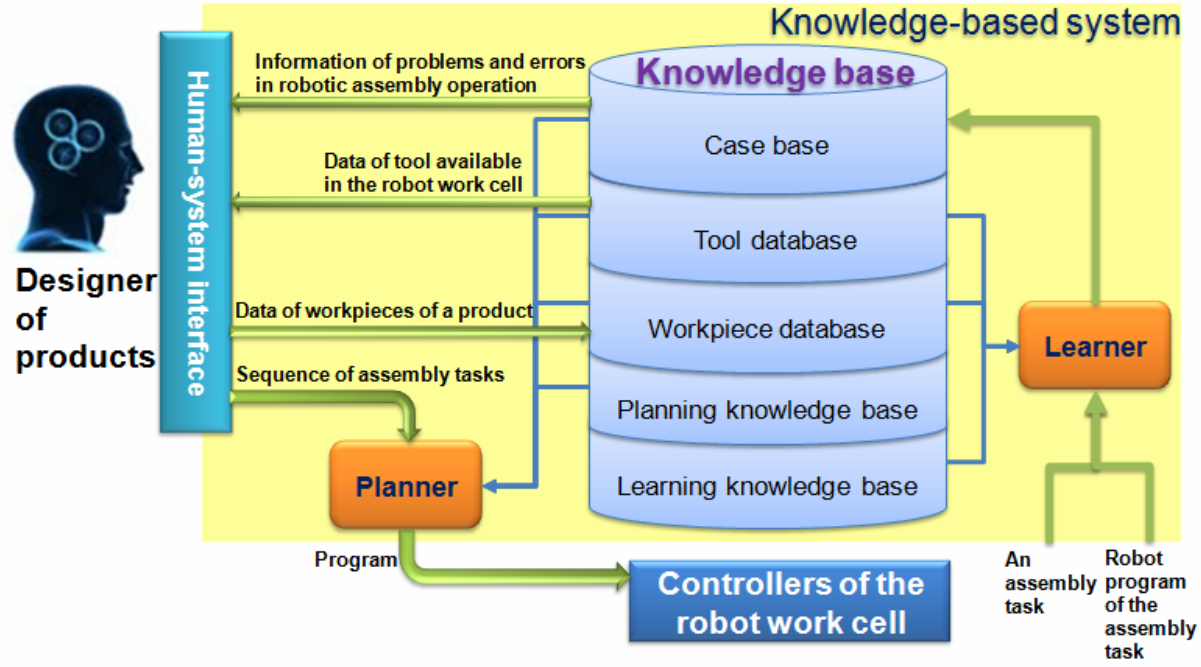

Figure 6.1 The Architecture of the transformed system.

The learner acquires assembly knowledge by analyzing and generalizing an assembly task and its corresponding robot program. The learning result is saved as a case in the case. The learner can learn from both successful and failed examples. The cases learned from successful examples can be reused for making programs for new similar assembly tasks. The cases learned from failed examples can be reused in error recovery in the implementation process of the robot work cell. Moreover, the failed cases learned from the robotic assembly implementation of a product will be fed back to the designers. This helps the designers improve their design with considering the assembly problems.

The planner is in charge of automatic programming. After the designers finish the design of a product, they input the assembly sequence of the product and the relevant requirements into the planner. By retrieving and reusing past similar successful cases from the case base, the planner first makes a robot program for each of the assembly tasks. Then it synthesizes the robot programs to generate the preliminary program for the robot work cell. Workers in the robot work cell test the

preliminary program, and revise it if they are not satisfied with some parts of it to make the final program for the robot work cell. The revising demonstrations will be learned by the learner. Thus, the ability of the planner can be updated and improved as the learner acquires more and more robotic assembly knowledge. In this way, the planner helps shorten the time of robot-teaching, and thus assists a smooth transition from design to assembly.

The knowledge bas has been introduced in Chapter 5.

The designers design the products with considering the manufacturing factors in the procedure as that shown in Figure 6.2.

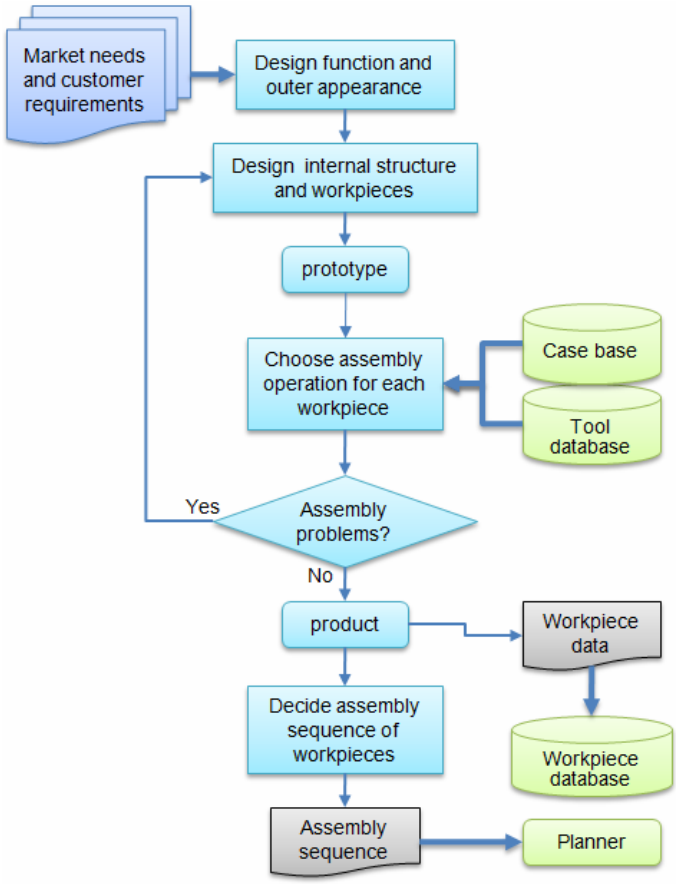

Figure 6.2 The design procedure with considering manufacturing factors.

First, the designers analyze the needs of the market and the requirements of customers, and design the necessary functions and outer appearance of the product. Then, the designers devise the internal structure and workpieces of the product to realize the necessary functions. In this way, the prototype is created.

Next, the designers revise the initial prototype by referring to robotic assembly factors through the proposed system to make the product easy to be assembled by robots. Here, the designers should keep two criteria in mind: 1. Minimize the number of workpieces and 2. Provide sufficient geometric and joint data of workpieces. The designers must understand that it is robots not human that will

assemble their product. Thus, the designers plan the assembly operation of the workpieces by consulting the case base of the proposed system and select the proper tool used to assemble each workpiece from available tools in the tool database.

While planning the assembly operation for a workpiece, the designers should retrieve all the previous cases of workpieces that are similar to the workpiece, and pay attention to the failed cases in particular. If only failed cases are retrieved for a workpiece, the designers should consider whether the failure is caused by defective product design or whether the failure can be avoided by revising the design of the product.

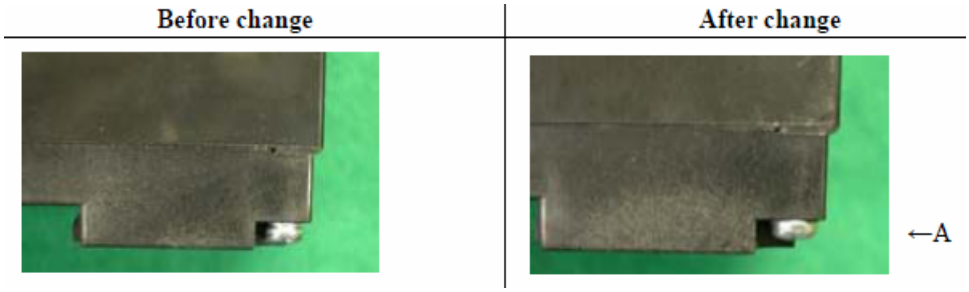

Figure 6.3 Example: delete washer [64].

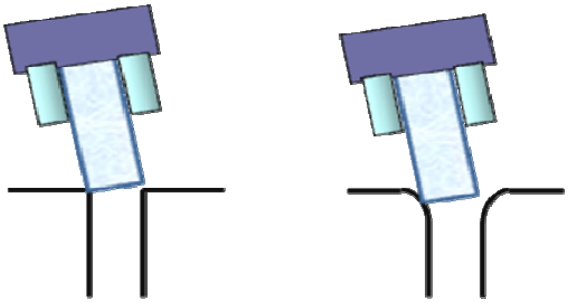

Figure 6.4 Stuck vs. slide.

For example, washer-falling errors often occur during a robot screw a pin with a washer. Thus, the designers will delete unnecessary washers from their product, as Figure 6.3 shows. Another example is shown in Figure 6.4. When a robot inserts a peg into a hole, the peg may be stuck at the sharp orifice of the hole. For this failure, the designers will design round cornered orifice for the hole. In this way, the peg can slide smoothly into the hole when it is pressured by a robot. As the examples given above, the designers re-design the prototype by referring to the proposed system until there is no assembly problem in the assembly operation design.

Finally, the designers finish the design of a product. In the following, they should define the origin point and the coordinate system for the product. Together with this, the designers input the geometric data and other data of the workpieces into the workpiece database. Then, the designers should decide the assembly sequence of the workpiece and input it into the planner of the proposed system. In this way, the planner will be able to generate the program for assembling the product with sensors in the robot work cell.

Chapter 7

Conclusion

Robot cell production is the development trend of the next generation manufacturing systems. Although robots has been deployed in more and more industrial sectors, there still exists a critical problem in robotic manufacturing that the task of teaching robots is excessively time-consuming, which keeps many industrial sectors, especially small and medium size enterprises away from robotic automation. Moreover, this problem has become the bottleneck of improve the flexibility and reconfigurability of robotic manufacturing systems and other automation systems to achieve rapid-response manufacturing. In this study, this problem is regarded as the difficulty of transferring knowledge from human to robot and from skilled workers to novice workers. Therefore, this study proposes a method knowledge acquisition, sharing and reusing in robotic manufacturing.

The originality of this study is that it proposes the concept of hierarchical knowledge modularization to realize interactive knowledge acquisition and reuse for industrial robot teaching. Herein, 'interactive' means that not only robots are able to learn from human workers, but also it is possible for human workers to learn from robots through their implementing the robot-teaching tasks. This research has become especially important in the recent years, during which more and more experienced workers were going to retire from their jobs and it takes a long time to train new workers. Hierarchical knowledge modularization means that the developed system acquires and represents the knowledge implemented in a task in terms of a hierarchical set (i.e., different levels) of modules, organizes these knowledge modules as a generalized plan for the object task, and saves the plan and the task description as case in the knowledge base of the system. In this way, the system can flexibly retrieve a case or modules of several cases and reuse it or them for a new task. In other words, the system can re-organize different levels of knowledge modules in different cases to generate the solution for a new task.

In this study, an improved EBL method has been developed and hierarchical used in acquiring knowledge from robot programs. This improved EBL method has two learning modes. The first learning mode can generalize knowledge from successful training examples. The second learning mode is able to learn knowledge from error recovery training examples. The retrieving and reusing method for the knowledge modules has also been developed based on CBR. However, the knowledge base of the current system is still small. As more and more knowledge are acquired, a more sophisticated retrieving method has to be developed in the future research. In addition, a graphical human-system interaction interface has also to be developed in the future research to assist

workers in error recovery for manufacturing tasks and in revising knowledge modules in the knowledge base.

Bibliography

[1] Gunasekaran, A. *Agile Manufacturing: The 21st Century Competitive Strategy*. Elsevier Science, February 2001.

[2] Fujita, T., et al. Robot control cell production system of senju (thousand-handed) kannon model that demonstrated optimality to the multi-product production in varying volumes for eight years. Proceedings of the 4th IEEE Conference on Automation Science and Engineering, August 2008.

[3] Sawaragi, T. Autonomous cell production robotic systems for integrative circulation of multiple resources. Proceedings of the 9th SICE System Integration Division Annual Conference, pp.175-178, 2008. [in Japanese]

[4] Kodaira, N. and Makita, H. Innovation of manufacturing related with the progress of industrial robots. Proceedings of the 9th SICE System Integration Division Annual Conference, pp.183-184, 2008. [in Japanese]

[5] Blomdell, A., et al. Extending an industrial robot controller: implementation and applications of a fast open sensor interface. *Robotics & Automation Magazine, IEEE,* 12(3), pp. 85-94, 2005.

[6] Brogårdh, T. Present and future robot control development - an industrial perspective. *Annual Reviews in Control,* 31, pp. 69–79, 2007.

[7] Haegele, M., et al. White paper - industrial robot automation. European Robotics Network, from http://www.euron.org/miscdocs/docs/euron2/year2/dr-14-1-industry.pdf , 2005.

[8] Argall, B.D., Chernova, S., Veloso, M. and Browning, B. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5), pp. 469-483, 2009.

[9] Aleotti, J., Caselli, S. and Reggiani, M. Leveraging on a virtual environment for robot programming by demonstration. *Robotics and Autonomous Systems*, 47, pp. 153–161, 2004.

[10] Li, Y.F., Ho, J. and Li, N. Development of a physically behaved robot work cell in virtual reality for task teaching. *Robotics and Computer Integrated Manufacturing*, 16, pp. 91-101, 2000.

[11] Chong, J.W.S., Ong, S.K., Nee, A.Y.C. and Youcef-Youmi, K. Robot programming using augmented reality: an interactive method for planning collision-free paths. *Robotics and Computer Integrated Manufacturing*, 25, pp. 689-701, 2009.

[12] Maeda, Y., Ushioda, T. and Makita, S. Easy robot programming for industrial manipulators by manual volume sweeping. 2008 IEEE International Conference on Robotics and Automation, Pasadena, *CA*, pp. 2234-2239, (2008).

[13] Pires, J.N. *Industrial robots programming: building applications for the factories of the future*. New York: Springer, 2006.

[14] DeJong, G. and Mooney, R. Explanation-based learning: an alternative view. *Machine Learning*, 1, pp. 145-176, 1986.

[15] Mitchell, T.M., Keller, R.M. and Kedar-Cabelli, S.T. Explanation-based generalization: a unifying view. Machine Learning, 1, pp. 47-80, 1986.

[16] Keller, R.M. The role of explicit contextual knowledge in learning concepts to improve performance. *Technical Report #ML-TR7*, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1987.

[17] Keller, R.M. Defining operationality for explanation-based learning. *Artificial Intelligence*, 35, pp. 227-241, 1988.

[18] Keller, R.M. Learning by re-expressing concepts for efficient recognition. Proceedings of AAAI-83, pp. 182-186, 1983.

[19] Dietterich, T.G. Learning at the knowledge level. *Machine Learning*, 1(3), pp. 287-361, 1986.

[20] Hirsh, H. Explanation-based generalization in a logic-programming environment. Proceedings of the 10th International Joint Conference on Artificial Intelligence, pp. 221-227, 1987.

[21] Segre, A.M. and DeJong, G. Explanation-based manipulator learning: acquisition of planning ability through observation. Proceedings of 1985 IEEE International Conference on Robotics and Automation, 2, pp. 555-560, 1985.

[22] Segre, A.M. *Machine learning of robot assembly plans*, Kluwer Academic Publishers, Norwell, MA, 1988.

[23] Segre, A.M. Learning how to plan. *Robotics and Autonomous System*, 8, pp. 93-111, 1991.

[24] Segre, A.M. On the operationality/generality trade-off in explanation-based learning. Proceedings of the 10th International Joint Conference on Artificial Intelligence, pp. 242-248, 1987.

[25] Marling, C.R., Petot, G.J. and Sterling, L.S. Integrating case-based and rule-based reasoning to meet multiple design constraints. *Computational Intelligence*, 15(3), pp. 308-332, 1999.

[26] Marling, C., PETOT, G. and STERLING, L. Adaptation-oriented retrieval in a case-based reasoning system. Technical report, Department of Computer Science, University of Melbourne, Melbourne, Australia, 1996.

[27] Kolodner, J. *Case-based reasoning*. Morgan Kaufmann, San Mateo, CA, 1993.

[28] Carbonell, J. G. and Veloso, M. M. Integrating derivational analogy into a general problem solving architecture. Proceedings of the Workshop on Case-Based Reasoning (DARPA). Morgan Kaufmann, San Mateo, CA, 1988.

[29] Wang, L., Sawaragi, T. and Tian, Y. A hierarchical knowledge based system for assembly tasks in human-robot cell manufacturing. Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing, pp. 127-132, 2009.

[30] Noda, A., Nagatani, T., Nagano, H., Okuda, H., Kitaaki, Y., Domae, Y. and Tanaka, K. System integration of intelligent robotic technologies in robotic cell assembly systems. Proceedings of the 10th SICE System Integration Division Annual Conference, pp.753-756, 2009. [in

Japanese].

[31] Nuttin, M. and Brussel, H. Learning the peg-into-hole assembly operation with a connectionist reinforcement technique. *Computers in Industry*, 33, pp. 101-109, 1997.

[32] Prabhu, S.M. and Garg, D. P. Fuzzy logic based reinforcement learning of admittance control for automated robotic manufacturing. Proceedings of the 1st International Conference on Knowledge-Based Intelligent Electronic Systems, Adelaide, South Australia, 1997.

[33] Lopez-Juarez, I. and Howarth, M. Knowledge acquisition and learning in unstructured robotic assembly environments. *Information Sciences*, 145, pp. 89-111, 2002.

[34] Chen, Y. and Naghdy, F. Skill acquisition in transfer of manipulation skills from human to machine through a haptic virtual environment. Proceedings of the 2002 IEEE International Conference on Industrial Technology, Bangkok, Thailand, 2002.

[35] Noda, A. and Nagatani, T. Motion learning for industrial robots with an active search algorithm, Proceedings of the 26th Annual Conference of the Robotics Society of Japan, 2008. [in Japanese]

[36] Noda, A., Nagatani, T. and Nagano, H. Active search algorithm on motion learning of industrial robots, Proceedings of the 9th SICE System Integration Division Annual Conference, pp. 211-212, 2008. [in Japanese]

[37] Hovland, G., Sikka, P. and McCarragher, B. Skill acquisition from human demonstration using a hidden markov model. Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Minneapolis, MN, 1996.

[38] Friedrich, H. and Dillmann, R. Robot programming based on a single demonstration and user intentions. Proceedings of the 3rd European Workshop on Learning Robots EWLR-3, Heraklion, Crete, Greece, 1995.

[39] Dillmann, R., et al. Learning robot behaviour and skills based on human demonstration and advice: the machine learning paradigm. Proceedings of the 9th International Symposium of Robotics Research, Snowbird, UT, 1999.

[40] Hwang, C.P. and Ho, C.S. A knowledge-based task-level programming and execution environment for robots. *Robotics and Computer-Integrated Manufacturing*, 12(4), pp. 329-351, 1996.

[41] Fujita, T., et al. Robot control cell production system of senju (thousand-handed) kannon model that demonstrated optimality to the multi-product production in varying volumes for eight years. Proceedings of the 4th IEEE Conference on Automation Science and Engineering, Key Bridge Marriott, DC, 2008.

[42] Braverman, H. *Labour and Monopoly Capital: The Degradation of Work in the Twentieth Century*, New York: Monthly Review Press, 1974.

[43] Nonaka, I. and Takeuchi, H. *The Knowledge-Creating Company*, New York: Oxford University

Press, 1995.

[44] Nevins, J.L. and Whitney, D.E. Computer- controlled assembly. *Scientific American*, 238(2), pp. 62-74, 1978.

[45] Mitsubishi Electric Corp. *Mitsubishi Industrial Robot Instruction Manual - detailed explanations of functions and operations*, Art. No. 132315, Version K, BFP-A5992, Mitsubishi Electronics Corporation, 14, July, 2005.

[46] Hiraoka, H. et al. Development of off-line programming system of robot operations based on environment models, *Journal of the Japan Society of Precision Engineering*, 53(1), pp. 137-143, 1987. [in Japanese]

[47] Czarnecki, C.A. Design and off-line programming of a dual robot workcell for garment manufacture. *Microprocessors and Microsystems*, 23, pp.225-234, 1999.

[48] Mitsi, S., et al. Off-line programming of an industrial robot for manufacturing. *International Journal of Advanced Manufacturing Technology*, 26, pp. 262-267, 2005.

[49] Webb, P., et al. Automated aerostructure assembly. *Industrial Robot*, 32, pp. 383-387, 2005.

[50] Jayaweera, N. and Webb, P. Adaptive robotic assembly of compliant aero-structure components. *Robotics and Computer-Integrated Manufacturing*, 23, pp. 180-194, 2007.

[51] Reinhart, G. and Tekouo, W. Automatic programming of robot-mounted 3D optical scanning devices to easily measure parts in high-variant assembly, *CIRP Annals – Manufacturing Technology*, 58, pp. 25-28, 2009.

[52] Schank, R. *Dynamic Memory: A Theory of Learning in Computers and People*. New York: Cambridge University Press, 1982.

[53] Hammond, K.J. Case-based planning: a framework for planning from experience. *Cognitive Science*, 14, pp. 385-443, 1990.

[54] Aamodt, A. and Plaza, E. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1), pp. 39-59, 1994.

[55] Leake, D.B. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press/MIT Press, 1996.

[56] Vong, C.M., et al. Case-based reasoning and adaptation in hydraulic production machine design. *Engineering Applications of Artificial Intelligence*, 15(6), pp. 567-585, 2002.

[57] Mantaras, R.L., et al. Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, 20(3), pp. 215–240, 2006.

[58] Perner, P. Case-based reasoning and the statistical challenges. Proceedings of the 9th European conference on Advances in Case-Based Reasoning, Lecture Notes In Artificial Intelligence, Vol. 5239, pp. 430–443, 2008.

[59] Schank, R. and Jona, M. Issue of psychology, AI, and education: a review of Newell's unified theories of cognition. *Contemplating Minds — A Forum for Artificial Intelligence*, New York:

The MIT Press, 1994.

[60] Iglezakis, I., Reinartz, T. and Roth-Berghofer, T. Maintenance memories: beyond concepts and techniques for case base maintenance. Proceedings of the 7th European Conference on Case-Based Reasoning, pp. 227–241, 2004.

[61] Wilson, D. and Leake, D. Maintaining cased-based reasoners: dimensions and directions. *Computational Intelligence*, 17(2), pp.196–213, 2001.

[62] Armengol, E., Ontanon, S. and Plaza, E. Explaining similarity in CBR. Proceedings Of ECCBR 2004, 2004.

[63] Armengol, E. Usages of generalization in case-based reasoning. *Lecture Notes In Artificial Intelligence, Vol. 4626, Proceedings of the 7th international conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, 2007.

[64] http://www.mitsubishielectric.co.jp/haisei/lvs/news/pdf/discon/yama099.pdf .

Published Papers

Journal Papers

1. Wang, L., Tian, Y., Sawaragi, T. and Horiguchi, Y. "A knowledge-based system for sharing and reusing tacit knowledge in robotic manufacturing", International Journal of Knowledge and System Sciences (accepted for publication in 2010).

2. Wang, L., Sawaragi, T., Tian, Y. and Horiguchi, Y. "Acquisition of human expertise in robotic assembly", SICE Journal of Control, Measurement, and System Integration, Vol.3, No.3, 2010 (accepted for publication).

3. Wang, L., Tian, Y. and Sawaragi, T. "Case-based automatic programming in robotic assembly production", Industrial Robot: An International Journal, Vol. 37 (accepted for publication in 2010).

International Conference Papers

4. Wang, L., Tian, Y. and Sawaragi, T. "Explanation-Based Manipulator Learning: Acquisition of Assembling Technique through Observation", Proceedings of the 17th IFAC World Congress, Vol.17 (1), pp. 2412-4417, 2008.

5. Wang, L., Sawaragi, T. and Tian, Y. "A hierarchical knowledge based system for assembly tasks in human-robot cell manufacturing", Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing, pp. 127-132, 2009.

6. Wang, L., Sawaragi, T., Tian, Y. and Horiguchi, Y. "Integrating CBR and EBL in an apprentice agent", Proceedings of the 2nd International Conference on Agents and Artificial Intelligence, Vol. 1–Artificial Intelligence, pp. 667-670, 2010

Japanese Conference Papers

7. Wang, L., Sawaragi, T., Tian, Y., Horiguchi, Y. and Nakanishi, H. "Acquiring human experts' tacit knowledge from observed robotic commands satisfying the criterion of reusability and understandability, SICE Symposium on Systems and Information 2008, pp.497-502, 2008.

8. Wang, L., Sawaragi, T., Tian, Y. and Horiguchi, Y. "Apprentice learning for interface agents in factory automation: a case study of a teaching task for assembling robots", Proceedings of the 158th Iron and Steel Institute of Japan (ISIJ) Meeting, pp.1045-1049, 2009.

9. Wang, L., Sawaragi, T., Tian, Y. and Horiguchi, Y. (2010) "A knowledge-intensive architecture for integrating designing and manufacturing", SICE Symposium for Young Researchers in Kansai, pp. 49-52, 2010.