

**AN INTELLIGENT TUTORING SYSTEM FOR
COMPUTER NUMERICAL CONTROL**

A Thesis

by

QINBO LI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirement for the degree of

MASTER OF SCIENCE

Chair of Committee,	Yoonsuck Choe
Co-Chair of Committee,	Sheng-Jen “Tony” Hsieh
Committee Members,	Frank M. Shipman
Head of Department,	Dilma Da Silva

August 2016

Major Subject: Computer Science

Copyright 2016 Qinbo Li

ABSTRACT

In recent years, the use of Intelligent Tutoring Systems (ITS) in classrooms and communities has increased and they proved to be very effective.

For the domain of Computer Numerical Control (CNC), however, existing approaches in ITS are not applicable or will not work well. CNC programming is different from computer programming languages, and students fail to solve CNC programming problems mainly due to two reasons: (1) lack of problem solving skills and (2) misconceptions or missing facts. CNC programming requires that students master a lot of facts and concepts before they try to write a program.

We built an ITS for CNC called the “CNC-Tutor” and proposed a data-driven approach that can generate proper hints and feedback during the students’ problem solving process. This approach is based on finding the most similar past submissions with the current student’s solution. The similarity is measured by the proposed “Behavior & Machine state distance” metric. Experiments show that the generated hints can help the students solve the CNC programming problem and the generated feedback can help the students to find their misconceptions. A survey on the effectiveness of our CNC-Tutor shows a positive impact on the students.

To my family

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Dr. Yoonsuck Choe, for letting me join the Neural Intelligence Laboratory, and for his continuous guidance and advice throughout my entire Master of Science degree research.

I would also like to thank my co-chair, Dr. Sheng-Jen “Tony” Hsieh, for offering me the opportunity to work as a graduate assistant in the Rockwell Automation Laboratory and for guiding me throughout this research.

I am also grateful to my committee member Dr. Frank M. Shipman. His guidance and advice have been very helpful to me.

Finally, I would like to thank my mother and father for their love and encouragement.

NOMENCLATURE

AST	Abstract Syntax Trees
BITS	Bayesian Intelligent Tutoring System
CBT	Computer-Based Training
CNC	Computer Numerical Control
CPD	Conditional Probability Distribution
CTAT	Cognitive Tutor Authoring Tools
DAG	Directed Acyclic Graph
ITS	Intelligent Tutoring System
MDP	Markov Decision Process
NC	Numerical Control
NCG	Next Generation Controller
NIST	National Institute of Standards and Technology
NLG	Natural Language Generation
QG	Question Generation
NDCG	Normalized Discounted Cumulative Gain
XAIDA	the Experimental Advanced Design Advisor

TABLE OF CONTENTS

	Page
ABSTRACT	ii
DEDICATION	iii
ACKNOWLEDGMENTS	iv
NOMENCLATURE	v
TABLE OF CONTENTS	vi
LIST OF FIGURES	ix
LIST OF TABLES	xii
1 INTRODUCTION	1
1.1 Introduction	1
1.2 Introduction to ITS	2
1.2.1 Domain model	5
1.2.2 Tutoring model	6
1.2.3 Student model	6
1.2.4 User interface	7
1.3 ITS authoring tools	7
1.4 Introduction to CNC	8
1.4.1 Industrial applications and economic benefits	8
1.4.2 Machine functions	9
1.4.3 CNC programming language	10
1.5 Thesis organization	12
2 LITERATURE REVIEW	14
2.1 ITS for related domains	14
2.2 ITS for computer programming	14
2.3 Hint generation	16
3 THE ARCHITECTURE OF THE CNC-TUTOR	18

3.1	The learning module	18
3.2	The quiz module	20
3.3	The exercise module	22
4	THE CNC INTERPRETER AND SIMULATOR	24
4.1	The CNC interpreter	24
4.1.1	The preprocess stage	25
4.1.2	Read the code and check the syntax	26
4.1.3	Execution	26
4.2	The CNC simulator	27
5	PROPOSED METHOD FOR HINT AND FEEDBACK GENERATION	30
5.1	The difficulties of hint generation for CNC code	30
5.1.1	The choice of cutting paths	31
5.1.2	The choice of settings	31
5.1.3	One line vs. multiple lines	32
5.1.4	The order of the code	32
5.2	The behavior & machine-state distance	33
5.3	Hint generation	35
5.4	Feedback generation	38
6	SYSTEM EVALUATION	41
6.1	Pre-test and post-test	41
6.2	Writing CNC code by hand vs. writing CNC code in CNC-Tutor	43
6.3	The accuracy of the generated hints	43
6.4	The accuracy of the generated feedback	47
6.5	Learning style analysis	50
6.6	The survey	51
7	DISCUSSION AND CONCLUSION	53
7.1	Contributions	53
7.2	Limitations	53
7.3	Future works	54
	REFERENCES	55
	APPENDIX A PRE-TEST AND POST-TEST	60

A.1 The result of the pre-test and post-test	60
APPENDIX B HINT GENERATION	63
B.1 Hint generation templates	63

LIST OF FIGURES

FIGURE		Page
1.1	The two layer structure of an ITS	3
1.2	The four main components of an ITS	4
1.3	A typical CNC machine center	9
1.4	An example of hole operation	10
1.5	An example of the movement of a CNC machine	11
1.6	An example of one block in a part program	12
1.7	The execution sequence of a part program	13
3.1	The system architecture of the CNC-Tutor	19
3.2	Part of the DAG	20
3.3	User interface for the learning module	21
3.4	User interface for the quiz module	22
3.5	An example of the exercise description page	23
3.6	The problem solving environment	23
5.1	The blueprint for the example exercise	31
5.2	The options for circular profiles: the IJK word or the R word	32
5.3	Two equivalent CNC code: (a) is written in one line and (b) is written in multiple lines	33

5.4	Two equivalent CNC code written in different orders	33
5.5	CNC code example and one of its machine states	35
5.6	An example of behavior alignment.	36
5.7	The hint generation template for the deep hole operation (G83)	37
5.8	An example of how hints are generated	38
5.9	The feedback lists	39
6.1	The paired box plot of the pre-test (1) and post-test (2) score, where (a) and (c) are the first population group, and (b) and (d) are the second population group	42
6.2	The exercise problems used in the evaluation: (a) shows the problem used in the pre-test and post-test; (b) shows the problem used in CNC-Tutor	44
6.3	An example of a meaningful hint generated by the system	45
6.4	A failed case of generated hints	45
6.5	The blueprint for the two more complicate exercises	47
6.6	An example of the students' submission	48
6.7	The blueprint of the exercise problem used for the feedback evaluation .	49
6.8	An example in the dataset: (a) shows a program in dataset and (b) shows the execution result of the program	50
6.9	An example of the learning styles survey result	51
6.10	The survey of the first group (limited CNC knowledge)	52

6.11	The survey of the second group (basic CNC knowledge)	52
------	--	----

LIST OF TABLES

TABLE		Page
1.1	Addresses in the part programming language	12
4.1	A partial list of the information in the “Block” [1]	26
4.2	A partial list of the error code and error messages [1]	27
4.3	G code groups [1]	28
4.4	M code groups [1]	28
4.5	Example of the machine functions provided by the CNC simulator [1] .	29
5.1	The machine states that are used to calculate the machine state distance [1]	34

1 INTRODUCTION

1.1 Introduction

Intelligent tutoring systems (ITS) are computer-based teaching environments that incorporate mathematics, cognitive science, natural language processing, human-computer interaction, etc [2]. In recent years, the use of ITS in classrooms and communities has increased and they proved to be very effective. For example, the Cognitive Tutor developed by Carnegie Learning which aims to help students learn algebra was used by more than 1,700 schools in 2004 [3].

For the domain of Computer Numerical Control, however, existing approaches in ITS are not applicable or will not work well. Students fail to solve a CNC programming problem usually for two reasons: (1) lack of problem solving skills and (2) misconceptions or missing facts. The ITS for computer programming focus mainly on improving the students' problem solving skills by providing the next-step hint. Different from computer programming, CNC programming requires that students master a lot of facts and concepts before they try to write the program. Such concepts and facts include, for example, the physical characteristics of the machine, the usage of CNC programming terminology, the correct spindle and feed rate, etc.

We built an ITS for CNC called the "CNC-Tutor" and proposed a data-driven approach that can generate proper feedback and hints during the students' problem solving process. This approach generates hints and feedback based on similar correct solutions

from an archive of programs by former students. The similarity is measured by the proposed “Behavior & Machine state distance” metric. Experiments show that the generated hints can help the students solve the CNC problem and the generated feedback can help the students to find their misconceptions. A survey on the effectiveness of our CNC-Tutor shows a positive impact on the students.

1.2 Introduction to ITS

The history of ITS is believed to have started from the book “Intelligent Tutoring Systems” by Sleeman and Brown (1982) [4]. During the 30-year development following the publication of the book, many ITSs have been built and several of them have been adopted in classrooms and communities. Cognitive Tutor, developed by Carnegie Learning, is now used in more than 2,000 schools. The main distinction between ITS and traditional computer-based training (CBT) is that ITS knows what to teach and how to teach. The learning content presented to the students is determined dynamically. When a student gets stuck somewhere, the system can provide appropriate hints and explanations as needed. The student can then ask further questions.

Before researchers started to investigate ITS, there were already many Computer-based Training (CBT) systems. ITS go beyond traditional CBTs in that they incorporate mathematics, cognitive science, and human-computer interaction to incorporate deep domain knowledge and pedagogy. Recently, a dozen of successful ITS have been built, including systems for the field of algebra, geometry, physics, electronics, and information

technology.

Many ITSs follow a two-level structure, such as cognitive tutors, constraint-based tutors, and case-based tutors. The two-level structure refers to the outer loop and the inner loop. The outer loop is responsible for the selection of learning content to be presented, the need to give a test, the judgment of mastery of a concept, and other global aspect during the teaching. The inner loop, on the other hand, focuses on tracking the individual steps during the problem solving process. If the student has some questions or got stuck, the inner loop of an ITS can give responses such as hints and explanations. Figure 1.1 shows the two-level structure of an ITS (adapted from Koedinger et al., 2013 [3]).

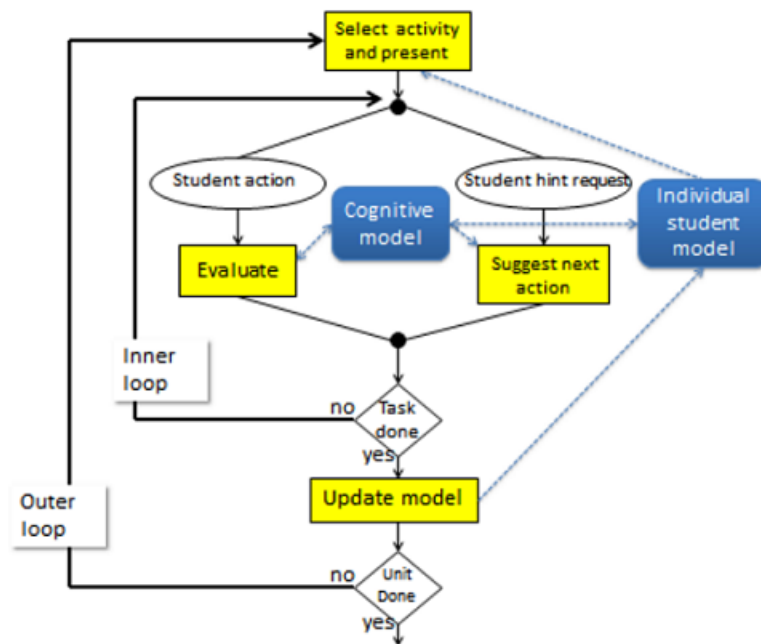


Figure 1.1: The two layer structure of an ITS [3]

There are four main components in an ITS: (1) domain model (knowledge base), (2)

tutoring model (teaching model), (3) student model, and (4) user interface (student interface). The domain model contains the representations of curriculum knowledge, problem solving expertise, and all other domain dependent knowledge. The tutoring model contains teaching strategies, for example, what is the sequence in which the learning content is presented, when to give a test, what type of feedback to give, etc. The student model encodes the status of the current student, such as the mastery of a concept and so on. The user interface determines how to present the content to the students and how to give feedback and other interactive content. Figure 1.2 shows the four main components of an ITS (adapted from Nkambou et al., 2010 [5]).

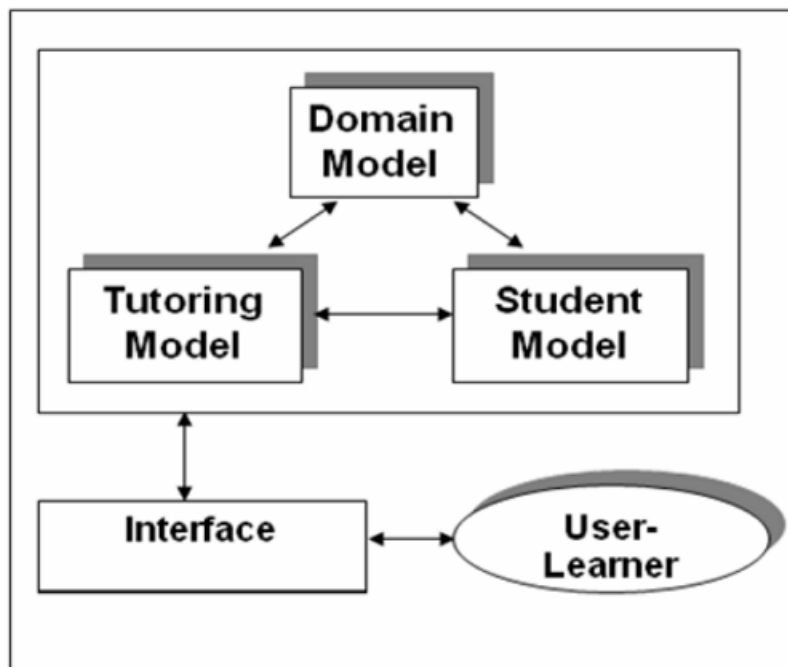


Figure 1.2: The four main components of an ITS [5]

1.2.1 Domain model

Domain model can represent different types of knowledge in the specified domain. There are mainly three categories of knowledge that can be represented by the domain model: (1) curriculum knowledge and structures, (2) simulation models, and (3) domain expertise.

Curriculum knowledge illustrates the relationships between curriculum elements such as concepts, topics, and modules. For example, XAIDA (The Experimental Advanced Design Advisor) uses semantic networks to represent the curriculum structure. Pedagogical properties, such as the importance and difficulty of a topic, can also be incorporated into the curriculum knowledge.

Simulation models provide a simulation of the world, a device, or a component. This feature is important in some ITSs, for example, RIDES [6], an application for authoring and delivering simulation-centered tutorials, models device component behavior, providing students opportunities to learn by doing.

Domain expertise includes problem solving skills, procedural skills, and domain concepts. Different systems usually have different types of domain expertise knowledge through different representations, based on the domain they are tutoring. The methods for representing knowledge include production rules, semantic networks, conceptual graphs and so on [7]. For example, XAIDA uses semantic network for maintenance procedures, causal reasoning schemes for theory of operation, and fault tree for troubleshooting. Demonstr8 [8] system uses production rules to create arithmetic cognitive tutors.

An alternative approach is Constraint-Based Modeling (CBM) [9]. Rather than modeling the procedural steps of a good solution, CBM models the declarative structure of a good solution. For example, for the problem of subtraction, the following constraints must be satisfied: “Increments and corresponding decrements must occur together”. ITS that uses constraint-based modeling includes SQL-Tutor [10], ERM-Tutor [11], etc.

1.2.2 Tutoring model

Tutoring model contains teaching strategies, for example, what is the sequence in which the learning content is presented, when to give a test, and what type of feedback to give. Many ITSs encode the tutoring model in the student model or the domain model [12].

1.2.3 Student model

Student model represents the status the student is currently in. The status can be progress, mastery, or learning achievement. There are various forms to represent the student model, for example, a numeric score of the topic, or a list of misconceptions, etc.

Many ITS use “overlay” student model. For “overlay” student model, the structure is the same with the curriculum structure in the domain model, and the difference is that there is a score assigned to each concept that represents how much mastery the student has of a concept.

1.2.4 User interface

The user interface of an ITS usually cannot compare to most computer-based training and employee training frameworks created by multimedia authoring tools. However, behind the multimedia authoring tools is a shallow representation of pedagogy and domain knowledge. Building a user friendly interface of an ITS can take more than 50% of the development time.

1.3 ITS authoring tools

Building an intelligent tutor is difficult and time-consuming, because it usually requires the cooperation between developers and domain experts. Multimedia authoring tools are widely available for building computer aided instruction and the authored learning sequences, and teaching strategies are somehow fixed. Many authoring tools for ITS has been built since the beginning of the development of ITS [13].

One of the most well-known authoring tools for ITS is Cognitive Tutor Authoring Tools (CTAT). The Cognitive Tutor is based on a cognitive model called ACT-R (adaptive control of thought - rational) [14]. ACT-R has a model to solve the problem, and it traces the students' actions while they try to solve the problem so that it can provide hints and explanations as necessary. The model that ACT-R uses is in the form of production rules. The production rule follows the format: "IF [STATE S] THEN [ACTION A]". ASTUS is another domain-independent ITS authoring tool that focuses on problem solving tasks. There are two types of knowledge that can be represented in ASTUS: declarative

knowledge and procedural knowledge. For well-defined domains, ASTUS can provide feedback or next-step hints.

SIMQUEST [15] and RIDES [6] are authoring tools for simulation-based ITS. Simulation-based learning environment is desired where the natural environment is unavailable, expensive, or dangerous. REDEEM [16] and Eon [17] are the authoring tools for ITS that focus on teaching strategies.

1.4 Introduction to CNC

Numerical control refers to an approach of operating a manufacturing machine by a code and it has been used in industry for over 40 years [18]. Numerical control systems usually have four components: tape punch, tape reader, controller, and NC machine. Tape punch is used to punch the written instructions into tape. Tape reader reads the tape and converts to an electrical signal code. Controller takes the code as input and then causes the NC machine to respond. NC machines execute motions encoded in the signal code to manufacture parts. Computer numerical control (CNC) refers to a NC machine with an on-board computer. Figure 1.3 shows a typical CNC machine center (adapted from Valentino and Goldenberg, 2012 [18]).

1.4.1 Industrial applications and economic benefits

The industrial applications of CNC include machining, fabrication and welding, presswork, inspection and measurement, assembly, etc. The economic benefits of adopting CNC includes: (1) CNC provides flexible automation where changeover from one job to



Figure 1.3: A typical CNC machine center [18]

another is rapid; (2) CNC can produce components with repeatable accuracy; (3) repeat orders require no additional work; and (4) CNC makes short production runs economical where high volume production is not required.

1.4.2 Machine functions

There are three categories of operations in CNC: (1) hole operations, (2) linear operations, and (3) circular operations. Figure 1.4 (adapted from Overby, 2010 [19]) shows a simple drilling hole operation: the tool end moves on the z-axis only and then retracts in the z-axis to clear z [19]. These operations are implemented by machine functions, which are encoded in the ROM (Read-only memory) of the on-board computer at the time of manufacture.

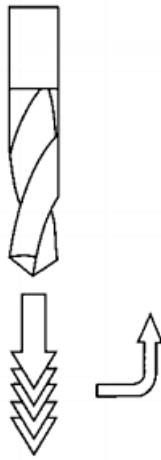


Figure 1.4: An example of hole operation [19]

CNC machines execute machining functions by performing linear motion and rotary motion. The implementation can vary from machine to machine. Figure 1.5 shows an example of the movement of a CNC machine (adapted from Valentino and Goldenberg, 2012 [18]). The table can move in the XY-axis plane and the spindle in the Z-axis plane. These movements are encoded in the machine functions and the programmers only need to program the tool end location.

1.4.3 CNC programming language

The programming language for CNC is called the part programming language (or CNC programming language). A complete part program is a set of instructions that describe the movement of the tool end. The basic element of the part programming language is a “word”. A “word” consists of a programming character following by a number. The words encode important information such the coordination of the tool end or the motion to be executed. The programming character is also called “address”, which explains the

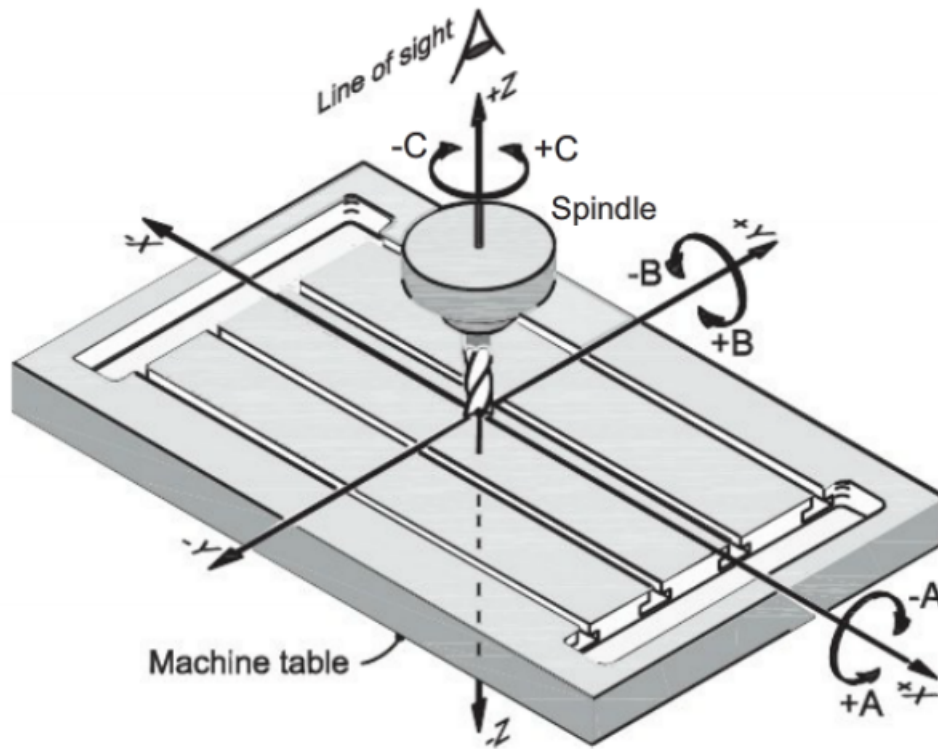


Figure 1.5: An example of the movement of a CNC machine [18]

meaning of the number followed. For example, the address “X” indicates the x-axis coordinate, and the word “X5” means “the coordinate of x-axis is 5 in the current length unit”. Table 1.1 describes the Addresses in the part programming language.

Each line of code can contain one word or multiple words, and such line of code is also called a “block”. A sequence of blocks forms a complete part program. The execution sequence of a part program is one block by another. Figure 1.6 illustrates an example of one block and Figure 1.7 explains the execution sequence (adapted form Valentino and Goldenberg, 2012 [18]).

Table 1.1: Addresses in the part programming language

Address	Description
N	sequence number
G	preparatory function
X, Y, Z	dimension words
I, J, K	dimension words
U, V, W	dimension words
A, B, C	dimension words
P, Q, R	dimension words
F	feed rate
S	spindle function
T	tool function
M	miscellaneous function
H, D	auxiliary input function

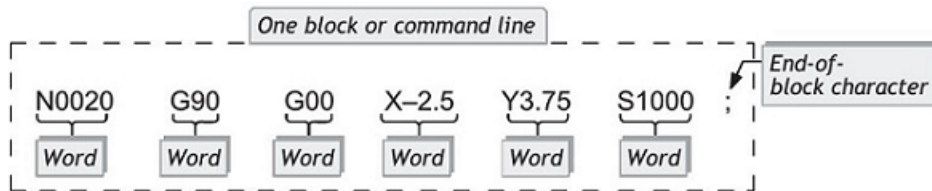


Figure 1.6: An example of one block in a part program [18]

1.5 Thesis organization

The rest of this thesis is organized as follows.

The 2nd chapter is the literature review.

The 3rd chapter explains the overall system architecture of the CNC-Tutor.

The 4th Chapter introduces the CNC interpreter that is used by our system.

The 5th Chapter details the proposed hint and feedback generation approach.

The 6th Chapter is the evaluation of our CNC-Tutor.



Figure 1.7: The execution sequence of a part program [18]

The 7th Chapter is the contributions, limitations at and future works.

2 LITERATURE REVIEW

2.1 ITS for related domains

To the best of our knowledge, by the time of this thesis there is no ITS for Computer Numerical Control. Perhaps the most related work in terms of the domain is XAIDA – an ITS authoring tool for equipment maintenance [13]. XAIDA uses semantic networks to represent the physical characteristics of equipments and procedures; causal reasoning schemes to represent theory of operation; and fault trees to represent troubleshooting. XAIDA uses the “overlay” student model. That is, the structure of the student model is the same as the domain model, and the difference between the domain model and the student model is that each node in the semantic network of the student model has a label representing whether the student has mastered the concept or not. In addition, the student model has a list of misconceptions. When a student gives a wrong response to a question, a misconception is inserted into the list. However, there is still a great difference between equipment maintenance and CNC. In addition, XAIDA does not involve the problem solving procedure.

2.2 ITS for computer programming

BITS [20] is a Bayesian intelligent tutoring system for computer programming. It uses Bayesian network to represent the structure of the problem as well as the students’ knowledge. A Bayesian network consists of a directed acyclic graph (DAG) and a condi-

tional probability distribution (CPD) table. Each node in the DAG is related to a concept in the domain. A directed link from a node to another node, say, node A to node B, indicates that node A is the prerequisite to learn node B. The DAG is constructed manually and the CPD table is obtained by previous exams in that course. The main functions that BITS provides are navigation support, prerequisite recommendations, and learning sequence generation. The navigation support function classifies each node into three categories: already known, ready to learn, and not ready to learn, based on the students' feedback and the pre-defined CPD table. Next, the student can start to learn the recommended "ready to learn" nodes. Prerequisite recommendations provide recommended concepts when the student gets stuck on the current node. The last function enables students to choose a particular lecture they want to learn without learning all the lectures before it. This is done by finding the minimum prerequisite based on the CPD table and the students' knowledge and then generating the learning sequence.

Another tutoring system for programming, QuizJET [21], is a system that supports authoring, delivery and assessment of parameterized questions for Java programming. Parameterized questions are a pattern of questions that are created initially by domain experts, and the pattern is replaced by randomly generated parameters at the presentation time. This way, a lot of similar questions can be used to evaluate the students' performance. The questions can cover critical topics in Java such as objects, classes, interfaces, inheritances, and exceptions. The advantage of this approach is that the authoring cost is significantly reduced and the possibility of plagiarism is also reduced. Students who solved more ques-

tions using this system showed better overall performance in the final exam. QuizPACK (Quizzes for Parameterized Assessment of C Knowledge) [22], is a similar system for the C programming language.

JO-Tutor [23] is another ITS for JAVA. It can generate problems automatically based on randomly instantiated templates. The topics of the problem include functions, classes, inheritance, polymorphism, and so on. It also has an expert module to solve these problems so as to judge the response and provide feedback.

2.3 Hint generation

Automatically generating hints is another area in programming tutors [24, 25, 26]. Computer programming is known to be ill-defined [27] because the solution space is too large or even infinite, especially when incorrect solutions are taken into consideration. Rivers et al. proposed a data-driven approach to generate programming hints automatically [28]. They use abstract syntax trees (ASTs) [25] to reduce the solution space: different programs with the same semantics can reduce to the same abstract syntax tree using copy propagation. The next step is to find the AST in prior student data that has the closest distance to the AST of the current program. The final step is generating hints based on the difference between the AST of the current program and the ASTs of a prior correct program.

Jin et al. proposed a hint generation approach that can generate hints automatically based on the completed solutions [26]. They used the linkage graph to represent the solu-

tion space. In the linkage graph, the nodes are the program statements while the edges are the order dependencies. To generate hints, the algorithm first compares the linkage graph of the student's program with the linkage graph from the past completed programs. It will then find the closest one in the markov decision process (MDP) and generates hints based on the next best state in the MDP. Then the new linkage graph will be added to the dataset. Their experiment shows that among 16 submissions, the approach can generate meaningful hints for 14 of them.

3 THE ARCHITECTURE OF THE CNC-TUTOR

In the domain of computer numerical control, there are two types of knowledge the students need to learn. One is the concepts and facts, and the other is the problem solving skills. Our CNC tutoring system called CNC-Tutor consists of a learning module, a quiz module, and an exercise module. The learning module and the quiz module are focused on teaching concepts and facts, while the exercise module focuses on improving the students' problem solving skills. Under these three modules are two data models: one is the domain model and the other is the student model. The domain model manages all the data related to the domain knowledge, for example, what is the prerequisite of a lesson, how to generate questions for a lesson, what are the answers and feedback for a question, and so on. The student model traces the students' state, for example, whether a student has passed a quiz, how many facts a student already knows, and so on. In addition, in order to evaluate the students' CNC code and generate hints and feedback, there is a model called CNC interpreter under the exercise module. Figure 3.1 illustrates the overall system architecture of the CNC-Tutor. The rest of this section describes in detail the learning module, the quiz module, and the exercise module.

3.1 The learning module

The learning content module contains all the knowledge the student needs to know. If the student is new to the domain, he or she may follow the suggested learning sequence pre-

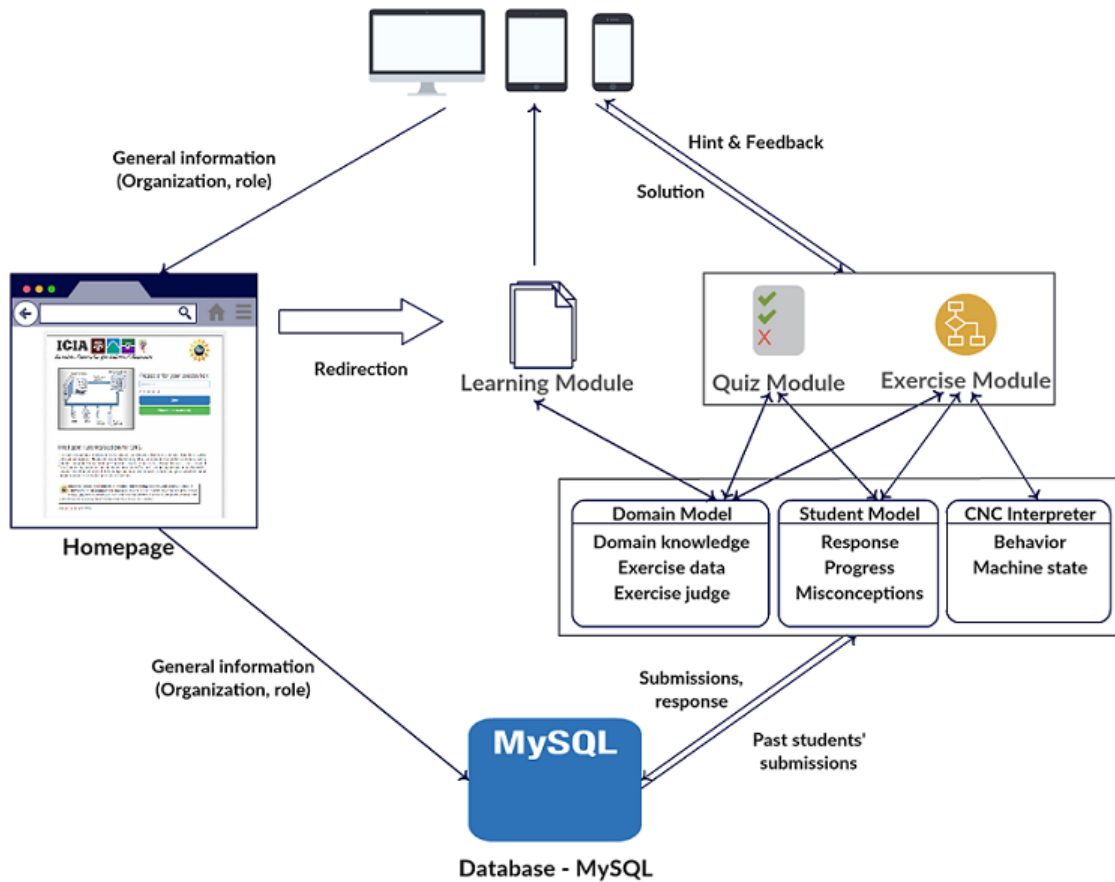


Figure 3.1: The system architecture of the CNC-Tutor

defined by domain experts. If students already has some knowledge in this domain, they can choose to learn whatever they want to. When they have difficulties in understanding the content, it is very likely that they missed some prerequisite for the content they chose. At this point, the system can recommend the prerequisite for the current learning content.

We use directed acyclic graph (DAG) to represent the curriculum structure knowledge (domain model). In the DAG, each node represents a piece of learning content. An edge from a source node to a destination node indicates that the source node is a prerequisite to learn the destination node. Currently, the system has 49 nodes and 65 edges. Figure

3.2 shows a portion of the DAG and Figure 3.3 shows the user interface for the learning module.

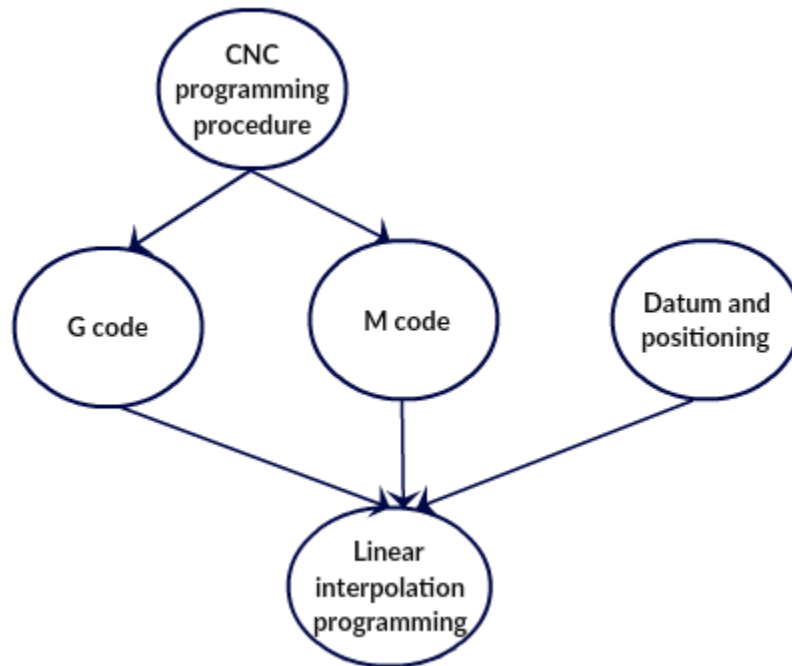


Figure 3.2: Part of the DAG

3.2 The quiz module

The second module of the system is the quiz module. Although students can choose any sequence to learn the content, they need to follow a fixed sequence to pass the quizzes. The fixed sequence is predefined based on the DAG such that students have to pass the easier quizzes to get access to more challenging ones.

The quiz module uses the student model to generate questions. The student model basically is a copy of the DAG with an additional attribute for each node, whether the

Intelligent Tutoring System for CNC Search...

- ▶ Computer Numerical Control
- ▶ Quiz for CNC
- ▶ Exercises for CNC programming

Open loop systems

An open loop system utilizes stepping motors to create machine movements. These motors rotate a fixed amount, usually 1.8 degree, for each pulse received. Stepping motors are driven by electrical signals coming from the MCU. The motors are connected to the machine table lead screw and spindle. Upon receiving a signal, they move the table and/or spindle a table lead screw and spindle. Upon receiving a signal, they move the table and/or spindle a fixed amount. The motor controller sends signals back indicating the motors have completed the motion. The feedback, however, is not used to check how close the actual machine movement comes to the exact movement programmed. Also, the lead screws used in these systems tend to generate friction and backlash. Backlash can cause positioning errors if the motions required to machine a part require a reversal in axis direction.

Open-Loop-System

Figure 3.3: User interface for the learning module

student has mastered the learning content or not. There are many questions for each piece of learning content, and the system will randomly select one or more questions. If the student answers all the questions correctly, the system will mark the learning content as learned. Otherwise, if the student answers one of the questions incorrectly, the node will be marked directly as unlearned.

The quiz module generates questions in the following way. If the student already learned the content, questions related to that content will not be asked, otherwise, it will randomly pick some questions relate to that content. In such a way, the student will always get questions that they have not seen before, or the questions that they answered incorrectly. If a student answered a question incorrectly, the system will not generate the same question again, instead, similar questions will be presented to the student. Figure 3.4 shows the user

interface for the quiz module.

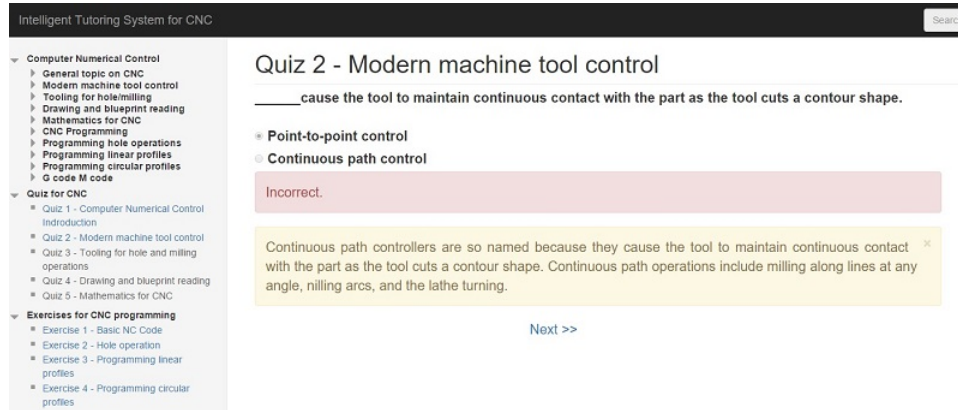


Figure 3.4: User interface for the quiz module

3.3 The exercise module

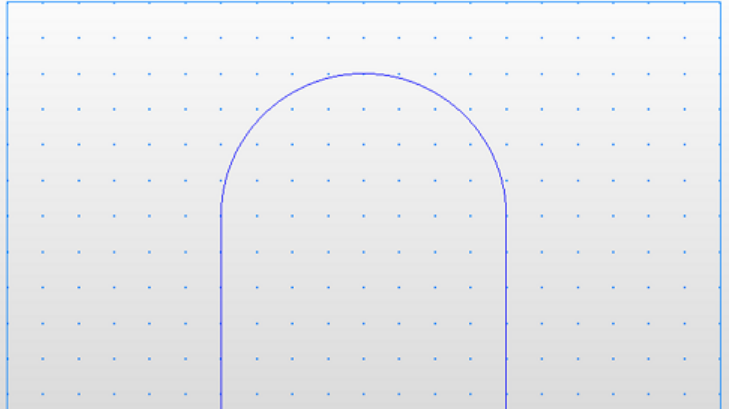
The exercise module mainly focuses on improving the students' problem solving skills. It provides a part's blueprint and a list of parameters to cut the part, and asks the students to write CNC code to solve the problem. Figure 3.5 and Figure 3.6 show the exercise description page and the problem solving environment, respectively.

Intelligent Tutoring System for CNC

Exercise 1 - Basic NC Code

<< Back Next >>

The following exercises will use the tombstone shape below.



- Computer Numerical Control
 - General topic on CNC
 - Modern machine tool control
 - Tooling for hole/milling
 - Drawing and blueprint reading
 - Mathematics for CNC
 - CNC Programming
 - Programming hole operations
 - Programming linear profiles
 - Programming circular profiles
 - G code M code
- Quiz for CNC
 - Quiz 1 - Computer Numerical Control Introduction
 - Quiz 2 - Modern machine tool control
 - Quiz 3 - Tooling for hole and milling operations
 - Quiz 4 - Drawing and blueprint reading
 - Quiz 5 - Mathematics for CNC
- Exercises for CNC programming
 - Exercise 1 - Basic NC Code
 - Exercise 2 - Hole operation
 - Exercise 3 - Programming linear profiles
 - Exercise 4 - Programming circular profiles

Figure 3.5: An example of the exercise description page

m for CNC Search... Home Dashboard About Help Log out

Please enter your NC code below.

NC Code

```

N0010 G90
N0020 G91 G28 X0 Y0 Z0
N0030 G90 G1 X1.5 Y1.5 Z0 S1600 F5
N0040 G90 X1.5 Y3.5 Z0
N0050 G90 G2 X3.5 Y3.5 Z0 R1 I1 J1 K0
N0060 G90 X3.5 Y1.5 Z0
N0070 G90 X1.5 Y1.5 Z0
N0080 G28 X0 Y0 Z0
  
```

Need Help?

Hint

<< Back Submit

Figure 3.6: The problem solving environment

4 THE CNC INTERPRETER AND SIMULATOR

Different from computer programming languages such as C++ and Java, CNC programming does not have variables, complex mathematic operations, functions and so on. However, evaluating CNC code is not necessarily easier than judging C++ code. Judging a C++ code, in most cases, is to run the program and compare the output with the desired output. This is not applicable for judging a CNC code, because a CNC program is run on CNC machine and the output goes directly to the part in the real world. Therefore, we built a CNC program interpreter and simulator to run the programs and judge the code. Since the CNC interpreter is developed in PHP and it is heavily based on the National Institute of Standards and Technology (NIST) “RS274NGC” project [1], we named this interpreter as “RS274NGC-PHP”. RS274 is a CNC programming language standard from NIST and “NGC” means the Next Generation Controller.

4.1 The CNC interpreter

Our CNC program interpreter follows the RS274 standard. It is designed for a three axes CNC machine. The axes are X, Y and Z axes which form a right-handed coordinate system of orthogonal linear axes. The rest of this section describes the detail of the interpreter.

Algorithm 1 describes the overall process of the CNC interpreter and simulator. There are mainly three stages during the whole process of the interpreter and simulator:

(1) preprocess, (2) read the code into Block and check the syntax, and (3) execution, detailed below.

Algorithm 1

```
CNC_interpreter_simulator(code)
  Pre-process(code)
  for each line of code
    Read the code and save the information into Block
    if there exists syntax errors
      return the error code
    end
    Call CNC simulator to execute
    return the trajectories and the machine states
  end
```

4.1.1 The preprocess stage

In the preprocess stage, a list of meaningless character is removed, which includes: whitespace, tab, new line, carriage return, NULL-byte, and vertical tab. Then, the interpreter will check whether the comments in the code are closed. If it finds any unclosed comments, it will generate a syntax error and report it. Otherwise, all the comment will be removed from the code. The last step is to turn the code into lowercase, since the CNC code is case-insensitive.

4.1.2 Read the code and check the syntax

The interpreter will then read the code and save the information to a data structure called “Block”. The information in the “Block” contains all the information for the CNC simulator to execute the code. Table 4.1 shows a partial list of the information in the “Block” [1].

Table 4.1: A partial list of the information in the “Block” [1]

Block element	Description
f number	the feed rate
x number	the x axis coordination of the machine
y number	the y axis coordination of the machine
z number	the z axis coordination of the machine
g modes	the entered G code in the current block
m modes	the entered M code in the current block

While the interpreter reads the code, it checks the syntax of the CNC code. If it detects a syntax error in a function, it will save the error message and its function name in a stack for debugging purpose and return the error code to the calling function, and the calling function will recursively record the function name and return the error code. The interpreter will then report the error message and stop. Table 4.2 shows some typical error code and its corresponding error messages. In total, there are 197 error code messages [1].

4.1.3 Execution

After the interpreter finished reading one Block and no syntax error is reported, it will call the Application Program Interface (API) provided by the CNC simulator to execute

Table 4.2: A partial list of the error code and error messages [1]

Error code	Error message
Error 5	All axes missing with G92
Error 84	G code out of range
Error 88	I word with no G2 or G3 or G87 to use it
Error 101	Mixed radius ijk format for arc
Error 120	Must use G0 or G1 with G53

the Block.

In CNC, some commands remain in effect once they are set until they are canceled or replaced by other commands. These commands are called modal commands, while other commands are non-modal commands.

In CNC, the G code and the M code are arranged in groups. The commands in the same group usually have similar function, so they cannot be in effect at the same time. The code in the same group are either all modal or all non-modal. Table 4.3 describes G code groups and Table 4.4 describes M code groups [1].

4.2 The CNC simulator

The CNC simulator simulates a CNC machine with all the key functions and all the key machine states. Table 4.5 shows the functions provided by the CNC simulator.

While the interpreter executes the code, it generates a message about the change of the machine state. This message will be return to the students as feedback to help them understand the execution process of a CNC program.

The correctness of the CNC program is judged by comparing the cutting paths with

Table 4.3: G code groups [1]

Group	G code	Description	Modal
0	G4, G10, G28, G30, G53, G92, G92.1, G92.2, G92.3	none	non-modal
1	G0, G1, G2, G3, G38.2, G80, G81-89	motion	modal
2	G17-19	plane selection	modal
3	G90, G91	distance mode	modal
5	G93, G94	feed rate mode	modal
6	G20, G21	units	modal
7	G40-42	cutter radius compensation	modal
8	G43, G49	tool length offset	modal
10	G98, G99	return mode in canned cycles	modal
12	G54, G55, G56, G57, G58, G59, G59.1, G59.2, G59.3	coordinate system selection	modal
13	G61, G61.1, G64	path control mode	modal

Table 4.4: M code groups [1]

Group	M code	Description	Modal
4	M0, M1, M2, M30, M60	stopping	modal
6	M6	tool change	modal
7	M3, M4, M5	spindle turning	modal
8	M7, M8, M9	coolant	modal
9	M48, M49	enable/disable feed and speed override switches	modal

Table 4.5: Example of the machine functions provided by the CNC simulator [1]

Functionality	Functions
Representation	SET_ORIGIN_OFFSETS() USE_LENGTH_UNITS()
Free Space Motion	STRAIGHT_TRAVERSE()
Machining Attributes	SELECT_PLANE() SET_FEED_RATE() SET_FEED_REFERENCE() SET_MOTION_CONTROL_MODE() START_SPEED_FEED_SYNCH() STOP_SPEED_FEED_SYNCH()
Machining Functions	ARC_FEED() DWELL() STRAIGHT_FEED()
Probe Functions	STRAIGHT_PROBE()
Program Functions	OPTIONAL_PROGRAM_STOP() PROGRAM_END() PROGRAM_STOP()

the desired cutting paths. If all the cutting paths match with the desired cutting paths and no syntax or run-time error is reported, the judging result of the program is set as correct.

5 PROPOSED METHOD FOR HINT AND FEEDBACK GENERATION

The key feature of our CNC-Tutor is that it can generate proper hints or feedback when the students are writing the CNC code. In the domain of ITS, the term “feedback” represents all kinds of messages provided by the system (examples, hints, correctness, etc.) [29]. However, in our system, hint is a message of the next-step action that leads to the correct solution, while feedback is a list of messages about the missing facts or misconceptions that the students might have. Feedback is provided only when the students failed to solve the problem.

Generating hints for programs is difficult because of the large solution space. We propose a data-driven approach to generate hints and feedback for CNC program by using past correct solutions.

5.1 The difficulties of hint generation for CNC code

Although the complexity of CNC code cannot compare to the computer programming code like C++, it is still very difficult to analyze and generate hints for CNC code. This is mainly because of four characteristics of CNC code: (1) The number of paths to cut a part is large; (2) The number of setting options is large; (3) One line of code can be written in multiple lines; and (4) The order of the code can vary, with the same outcome.

5.1.1 The choice of cutting paths

There are many choices of cutting paths to cut a part, as long as the outcome matches with the blueprint. Consider the following example: the student is asked to write a program to cut the shape in Figure 5.1 (adapted from lab manual of MMET 181, Texas A&M University). The student can choose any of the four key points to start the cutting, and he/she can also choose to cut the part clockwise or counterclockwise. Actually, the student can choose any point on the curve, as long as the outcome is the same.

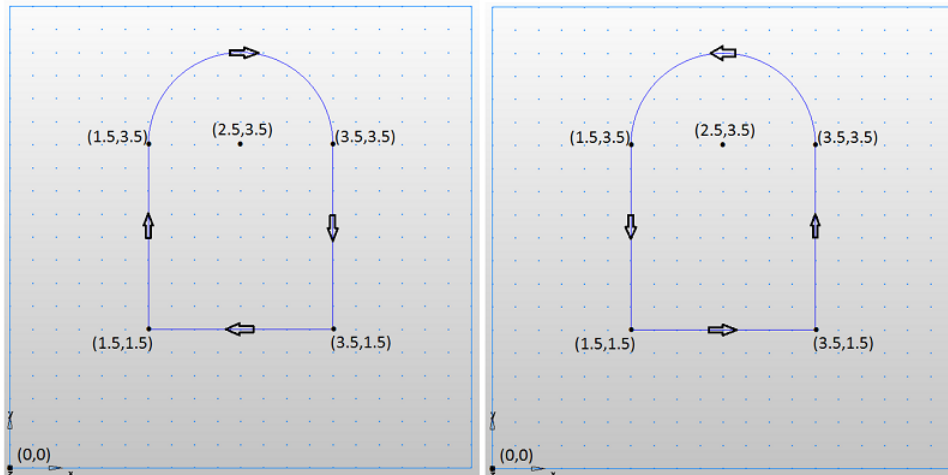


Figure 5.1: The blueprint for the example exercise

5.1.2 The choice of settings

Even if the cutting paths are fixed, the students can choose different settings to cut the part. For example, they can choose either G20 (inch mode) or G21 (millimeter mode). For circular profiles, the options are the IJK word or the R word, illustrated in Figure 5.2 (adapted from Valentino and Goldenberg, 2012 [18]). The choice of the parameter settings

will affect the code that follows. For example, all the coordinates will be related to a fixed zero point if the distance mode is G90 (absolute), while the coordinates will be related to the last position if the distance mode is G91 (incremental).

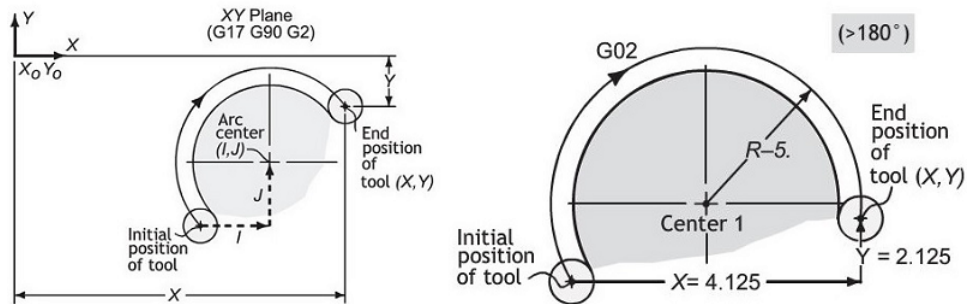


Figure 5.2: The options for circular profiles: the IJK word or the R word [18]

5.1.3 One line vs. multiple lines

As stated in Chapter 4, if G code and M code belong to modal groups, they will remain in effect once it is set until it is canceled or replaced by another command. Therefore, many NC code can be written in one line or in multiple line with the same semantics. For example, the code in Figure 5.3 (a) and the code in Figure 5.3 (b) are equivalent.

5.1.4 The order of the code

There is no fixed order to write CNC code, as discussed in the “CNC interpreter” section. For example, the code in Figure 5.4 (a) has the same effect with the code in Figure 5.4 (b).

N00 G0 G90 X-0.25 Y-0.25 Z0. S1200 M3 (a)
N00 G90 N10 M3 N20 S1200 N30 G0 X-0.25 Y-0.25 Z0. (b)

Figure 5.3: Two equivalent CNC code: (a) is written in one line and (b) is written in multiple lines

N0000 G0 G90 X-0.25 Y-0.25 Z0. S1200 M3 (a)
N50 S1200.0 M3 X-.25 Y-.25 Z0 G0 G90 (b)

Figure 5.4: Two equivalent CNC code written in different orders

5.2 The behavior & machine-state distance

Because of the characteristics of CNC code, the solution space of a CNC programming problem is very large. A solution space is a graph representation of how the students try to solve the problem [28]. The node of the graph stands for the state in the problem solving process and the edges stand for the students’ action. We notice that if the students follow the same path in the solution space (i.e. they choose the same strategy to solve the problem), the behavior (the cutting paths) of their programs will be the same, and the machine states transition during the execution in the CNC simulator will be similar, regardless of however their code differ from each other. Therefore, we propose an approach

that can compare the similarity between the current student’s solution and the past labeled data to automatically generate proper feedback. We named the distance measurement as “Behavior & Machine-state distance”.

The behavior distance is measured by the Euclidean distance between the cutting path of the current student’s solution and the past solutions. If the behavior distance is smaller than the tolerance, then these solutions follow the same cutting path. The behavior distance is calculated before the machine state distance, because it is meaningless to compare the machine state while the cutting paths are different.

The machine state is discussed in the CNC simulator section. For computation efficiency, only a portion of the machine state is recorded to calculate the machine state distance. Table 5.1 lists all the machine states that participate in the calculation of machine state distance. For example, the machine states of the code in Figure 5.5 (a) are illustrated in Figure 5.5 (b).

Table 5.1: The machine states that are used to calculate the machine state distance [1]

Machine state	Description
x	the current x-axis value
y	the current y-axis value
z	the current z-axis value
distance	distance mode: absolute or incremental
length units	inch or millimeter
feed mode	the current feedmode
feed rate	the current feed rate in ipm
motion mode	the motion to be executed
spindle	the current spindle mode
flood	whether the flood is on
tool	the selected tool number

N0010 G90 G20 G40 G80	line_number : 90
N0020 G91 G28 X0 Y0 Z0	x : -0.25
N0030 G92 X-10. Y5. Z0	y : 2.5
N0040 T1 M6	z : -0.25
N0050 G0 G90 X-0.25 Y-0.25 Z0. S1200 M3	distance : 0 (absolute)
N0060 G43 Z.1 H1	length_units: 1 (inch)
N0070 M8	feed_mode : 0 (units per minute)
N0080 G1 Z-.25 F6	feed_rate : 6 (ipm)
N0090 Y2.5	motion_mode : 10 (linear interpolation)
N0100 G2 X2.5916 Y4.3143 I2 J0	speed : 1200 (rpm)
N0110 G1 X5.2364 Y3.0875	spindle : 2 (clockwise)
N0120 G2 X4.5 Y-0.25 I-0.7364 J-1.5875	flood : 1 (on)
N0130 G1 X-0.25	tool : 1 (tool number)
N0140 G0 G90 Z.1 M5	
N0150 M9	
N0160 G91 G28 X0 Y0 Z0	
N0170 M30	
(a)	(b)

Figure 5.5: CNC code example and one of its machine states

The machine state distance is calculated by comparing the difference of each state. Because the parameters, settings, and machine states are used to cause an effect on the parts, so before comparing the distance between the machine state, their behavior must be the same. That is, the current coordination of the tool end and the motion to be executed must be the same. Therefore, each line of code in one program is aligned to the code that has the same “behavior” of another program. This process is called “Behavior alignment”. Figure 5.6 shows an example of behavior of “Behavior alignment”.

5.3 Hint generation

During the problem solving process, the students can request for hints by click the “hint” button in the left-bottom column of the page. The system will then find the closest code in the database in terms of the “Behavior & Machine-state distance”. If the behavior and machine state of the current code is in accordance with the past code, the next line of

N1 G91 G20 G40 G80	
N2 G28 X0 Y0 Z0	G20
N3 G92 X-10 Y5 Z0	T1 M6
N4 G0 G90 X0Y0Z0	G0 G90 X0 Y0
N5 T1 M6	
N6 M8 M3	M3
N7 G0 X-0.35 Y0.25	G0 X-.35 Y0.25
N8 S1500	
N9 G1 Z-0.4 F6	
N10 X2.09	
N11 X5.25 Y-2.4	
N12 Y-5.25	
N13 X-0.25	
N14 Y0.25	
N15 G0 G90 Z0.1 M5	
N16 M9	
N17 G28 X0Y0Z0	
N18 M30	

Figure 5.6: An example of behavior alignment

machine state of the past code is used to generate hint: for each different machine state, a natural language hint is generated.

Natural language generation (NLG) is another research area that is broadly used in question generation (QG), which generates natural language questions from unstructured text [30]. There are three categories of methods for NLG: (1) syntax-based method, (2) semantics-based methods, and (3) template-based methods. Syntax-based methods parse the sentence to determine the syntactic structure, and then apply syntactic transformation rules and question word replacement to generate questions. Semantics-based methods change declarative sentences into questions by transformations as syntax-based methods. However, semantics-based methods use semantic analysis instead of syntactic analysis.

Template-based methods, on the other hand, use question templates to generate questions, such that the language generation can leverage human expertise.

For CNC programming hints, because it is highly domain specific, we use a template-based method to generate natural language hints. For example, Figure 5.7 shows the template for the deep hole operation (G83).

```
<template>  
  Apply deep hole operation at (_x_, _y_) , to  
  (_z_) , with return distance (_Rn_) , and peck (_Qn_) .  
</template>
```

Figure 5.7: The hint generation template for the deep hole operation (G83)

Algorithm 2 describes the procedure of hint generation and Figure 5.8 shows an example of how the hints are generated. In Figure 5.8, (a) and (b) are the current student's code and the past code, respectively; (c) is the machine state of the last line of the code in (a); (d) is the machine state of the 9th line of the code in (b); and (e) is the generated hints based on the machine state difference between (c) and (d).

Algorithm 2

```
Hint_generation(code A)  
  
  CNC_interpreter_simulator(code A)  
  
  for each current code in the database  
    calculate the distance with A  
  
  end
```

Find the code (B) with the smallest distance with A

Calculate the machine state difference between A and B

Generate natural language hints using the templates

<pre>N10 G20 N20 T1 M6 N30 G0 G90 X0 Y0 N40 M3 N50 G0 X-.35 Y0.25</pre> <p>(a)</p>	<pre>N1 G91 G20 G40 G80 N2 G28 X0 Y0 Z0 N3 G92 X-10 Y5 Z0 N4 G0 G90 X0Y0Z0 N5 T1 M6 N6 M8 M3 N7 G0 X-0.35 Y0.25 N8 S1500 N9 G1 Z-0.4 F6</pre> <p>(b)</p>
<pre>line_number: 50 x: -0.35 y: 0.25 z: 0 feed_rate: 0 speed: 0 motion_mode: 0</pre> <p>(c)</p>	<pre>line_number: 9 x: -0.35 y: 0.25 z: -0.4 feed_rate: 6 speed: 1500 motion_mode: 10</pre> <p>(d)</p>
<div data-bbox="589 1230 1066 1459"><p>Need Help?</p><p>Hint</p><p>Linear motion - move to z:-0.4. Set feed rate to 6(ipm). Set speed to 1500(rpm).</p></div> <p>(e)</p>	

Figure 5.8: An example of how hints are generated

5.4 Feedback generation

After the student submits a CNC code, the exercise model will call the CNC interpreter to execute the code and judge whether the result matches with the requirement. An

incorrect submission may be caused by syntax error or behavior error. The exercise will generate a candidate list of learning content, which might contain the concepts or facts that the student missed. For example, if the CNC interpreter returns a syntax error - “error 7: arc radius too small to reach end point”, then the learning content in Figure 5.9 (a) will be added into the candidate list. If the CNC interpreter does not return a syntax error but the cutting path does not match with the blue print, the learning content in Figure 5.9 (b) will be added into the candidate list.

(1) Learning content 39 - Circular interpolation commands;
(2) Learning content 40 - Circular interpolation via direct radius specification;
(3) Learning content 43 - G20 and G21 (length units);
(4) Learning content 45 - G90 and G91 (distance mode).
(a)
(1) Learning content 14 - Datum;
(2) Learning content 15 - Coordinate positioning;
(3) Learning content 16 - Determining sides of right triangles;
(4) Learning content 18 - Determining angles of right triangles;
(5) Learning content 23 - IJK circular address;
(6) Learning content 37 - Cutter offsets for inclined line profiles;
(7) Learning content 42 - Determining cutter offsets for line-arc profiles;
(8) Learning content 43 - G20 and G21 (length units);
(9) Learning content 45 - G90 and G91 (distance mode).
(b)

Figure 5.9: The feedback lists

The label of an incorrect CNC code in the database is a list of learning contents that can help the student to fix the error in the code. Assume the database contains some labeled data, the exercise model will then search for the past labeled data to find the most similar code based on the “Behavior & Machine-state distance” metric and use the labels to rank the candidate list. The goal is to rank the most related learning content as high as possible,

because the students might only click the items near the top. If multiple code have the same “Behavior & Machine-state distance” to the current code, the system will vote to rank the candidate list. Then the feedback list is returned to the student as feedback. Algorithm 3 describes the procedure of the feedback generation process.

Algorithm 3

```
Feedback_generation(code)
    CNC_interpreter_simulator(code)
    if the result is incorrect
        for each incorrent code in the database
            calculate distance with A
        end
        Find a list of code with the smallest distance
        Generate the feedback list
        Rank the list using the label of the closest code
        return the feedback list
    end
```

Our assumption is that the learning contents in the learning module contain all the information the students need to master to solve the exercise problems. The students can click on the feedback items to review the learning content and then come back to solve the problem.

6 SYSTEM EVALUATION

The system is evaluated in five ways: (1) pre-test and post-test; (2) writing CNC code by hand vs. writing CNC code in CNC-Tutor; (3) accuracy of the generated hints and feedback; (4) learning style evaluation; and (5) the students' opinion survey.

6.1 Pre-test and post-test

There were 93 students who participated in the evaluation of the CNC-Tutor. We designed a pre-test and a post-test that focus on CNC concepts and facts, as well as CNC programming. The students were asked to take the pre-test first. After they finished the pre-test, they were asked to go through the selected topics in the CNC-Tutor, and try to write CNC code to solve an exercise problem. The average time that they spent on the CNC-Tutor was around one hour. In the end, they were asked to take the post-test. Based on the students' initial knowledge on CNC, they were divided into two population groups: the first group had 51 students who had limited knowledge on CNC while the second group had 42 students who had some basic knowledge on CNC.

For the first population group, the average points of the pre-test and the post-test were 30.08 and 62.45, respectively. For the second population group, the average points for the pre-test and the post-test were 37.45 and 62.64, respectively. The two groups had similar post-test scores, while the second group had higher pre-test score. The gains of the two groups were 32.37 and 25.19, respectively.

The paired T-test rejects the null hypothesis (no difference in pre-test vs. post-test) at the 1% significance level on both population groups with the T value of 10.26 and 6.99, respectively. Therefore, there is strong evidence that the CNC-Tutor leads to statistically significant improvement. Figure 6.1 shows the box plots and the paired box plots of the pre-test and post-test scores of the two population groups.

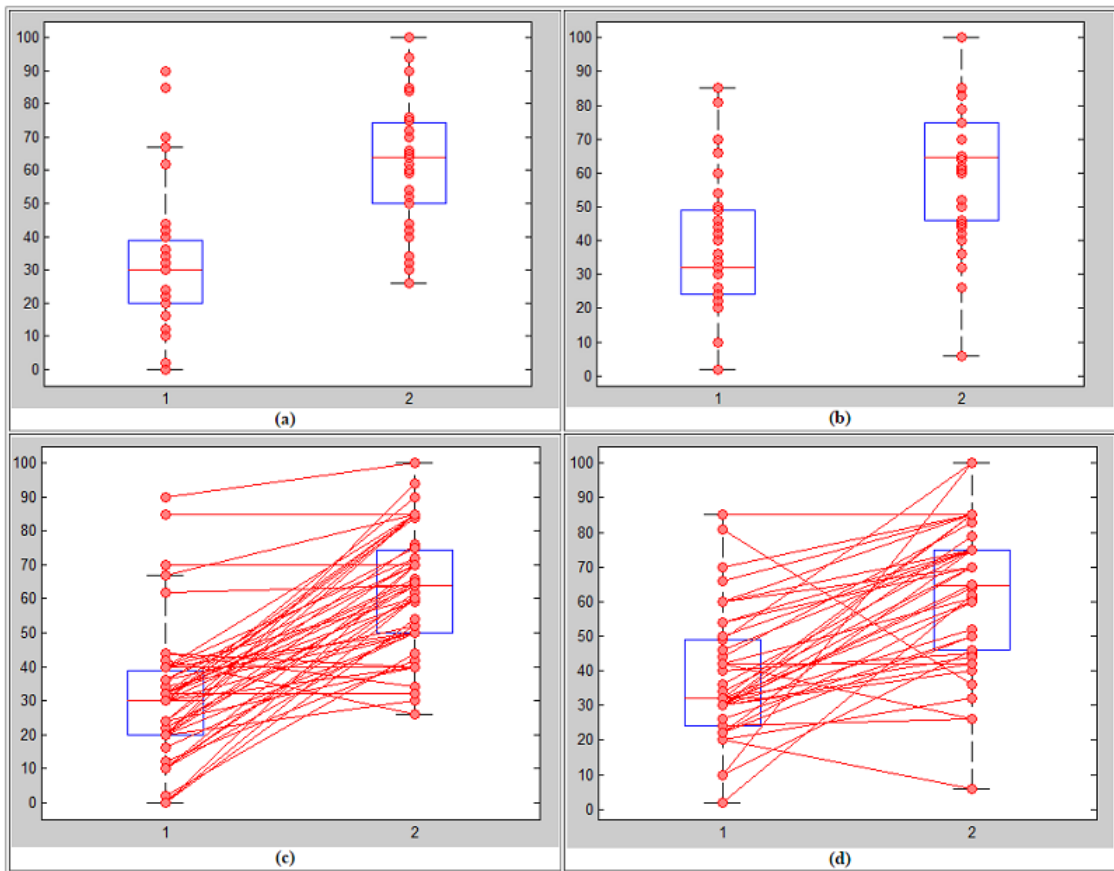


Figure 6.1: The paired box plot of the pre-test (1) and post-test (2) score, where (a) and (c) are the first population group, and (b) and (d) are the second population group

For the questions that focus on CNC programming (30 points in total), the average score of the first group increased from 1.29 to 6.12, while the average score of the second

group increased from 1.07 to 7.5. The gains for the programming questions are 4.83 and 6.43.

The pre-test and post-test shows that the CNC-Tutor can help the students to learn the CNC concepts and the CNC programming effectively.

6.2 Writing CNC code by hand vs. writing CNC code in CNC-Tutor

In the pre-test and the post-test, there was one exercise problem that asks the students to write CNC code to solve the problem illustrated in Figure 6.2 (a). When the students use the CNC-Tutor, they were asked to solve an exercise problem illustrated in Figure 6.2 (b) in the problem solving environment where they can get hints and feedback. The two exercise problems are similar in terms of the difficulty: both of them contain simple linear and arc movement, and ignore cutter compensation. Among the 93 students that participated in the evaluation, all of them were asked to finish the exercise problem in the post-test, while only 42 students attempted the exercise problem in the CNC-Tutor. There were only 3 students who solved the problem in the post-test, while 21 students solved the problem in CNC-Tutor.

6.3 The accuracy of the generated hints

The hint generation approach is evaluated in two ways. First, we had 42 students to use our CNC-Tutor to write code for an exercise. Every time the students request hints, the system records their code and the generated hints. Among the 220 hints recorded, we found 162 hints were meaningful: the accuracy of the hint generation approach was 73.6%.

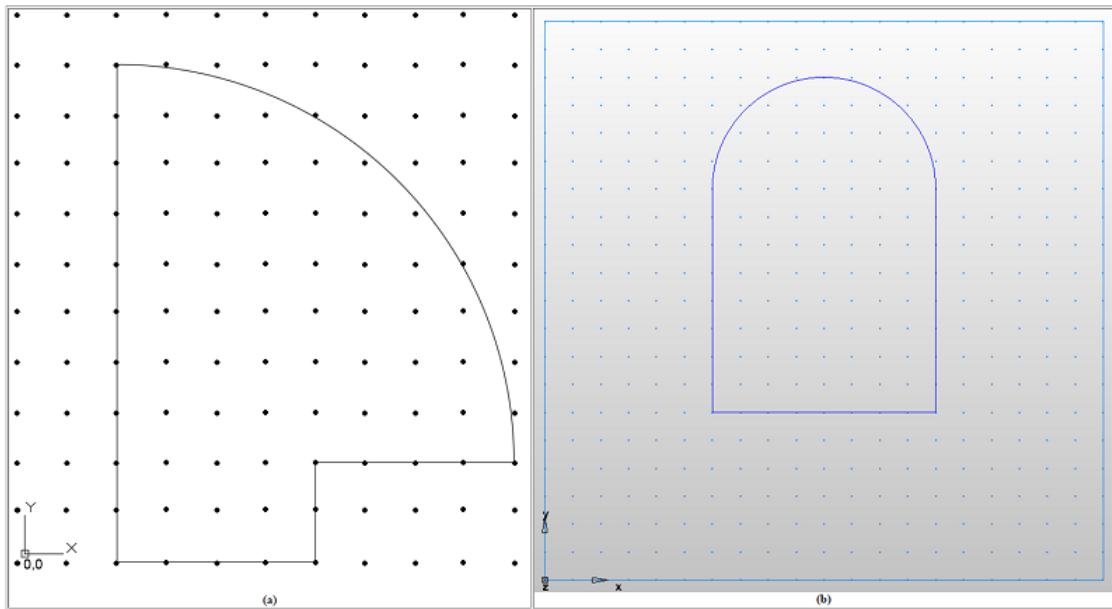


Figure 6.2: The exercise problems used in the evaluation: (a) shows the problem used in the pre-test and post-test; (b) shows the problem used in CNC-Tutor

Figure 6.3 shows an example of a meaningful hint generated by the system.

Most of the failed cases were due to the students choosing cutting paths or parameters that the database does not have yet. This can be improved as the database records additional correct solutions. Figure 6.4 shows a failed case that not caused by this reason. In this case, the student chose the cutting path in Figure 6.4 (b), however, the student used G2 (clockwise circular cutting) instead of G3 (counterclockwise circular cutting). As result, the difference between the machine state and the closest program's machine state is the motion mode, while the end coordinates are the same. Hence, the system generates meaningless hint: "Counterclockwise circular motion - move to.". Figure 6.4 shows (a) the code, (b) the generated hint, and (c) the behavior. This case failed because the code contains behavior error. If the code has syntax error, the generated hints will include the

Please enter your NC code below.

NC Code

```
N0010 G0 G20 X1.5 Y1.5  
N0015 F20 G1 Y3.5
```

Need Help?

Hint

Clockwise circular motion (G2) - move to x:3.5.

Set speed (S) to 2500(rpm).

Turn spindle on(clockwise) - M3.

Figure 6.3: An example of a meaningful hint generated by the system

syntax error message. However, if the code does not have any syntax error, but it has behavior error, the student needs to submit the code to get the feedback. In the feedback page, the message returned by the interpreter, the visualization, and the feedback message can help the student figure out the problem.

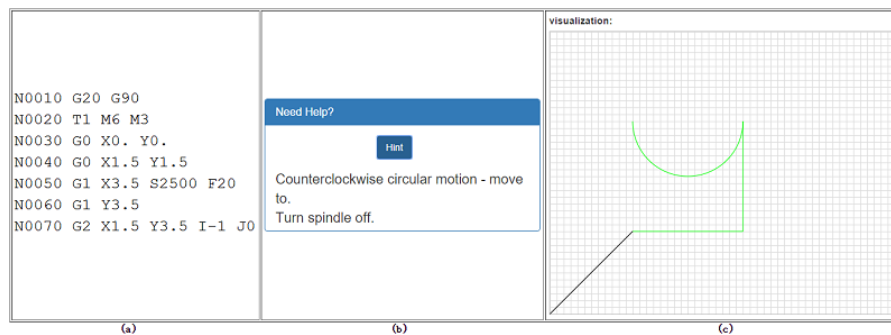


Figure 6.4: A failed case of generated hints

Because the hint generation approach is data-driven, the more data the database has,

the higher accuracy of our hint generation approach can achieve. Therefore, the second evaluation approach focuses on the adaptability of the hint generation approach. We divided 38 students into two groups, each group with 19 students. The students were asked to write CNC code to cut along a simple path. To have a better code coverage, we generated hints for each line of their code. For the first student group, the database only contained 5 correct sample codes, which were used to generate the hints. For the second group, the database contained the 5 sample codes as well as the codes from the first group. It is noteworthy that the code in the first group contained both correct codes and incorrect codes. Our approach can generate 50 meaningful hints out of 79 (65.8% accuracy) for the first group, and 66 meaningful hints out of 74 (89.2% accuracy) for the second.

Because the students that participated in the evaluation only had basic knowledge in CNC, and writing CNC code by hand is always difficult, the two experiments above used a simple exercise. This exercise focuses on the basic vocabulary used in NC code and ignores the z-axis movement and cutting compensations and so on. Therefore, we evaluated the hint generation approach on two more complicated exercises with a synthesized dataset, which contained 30 incomplete or problematic CNC code. The blueprint for these two exercises are shown in Figure 6.5 (adapted from Valentino and Goldenberg, 2012 [18]): one focuses on linear operations and the other focuses on hole operations. Our approach generated 26 useful hints out of 30 (86.67%).

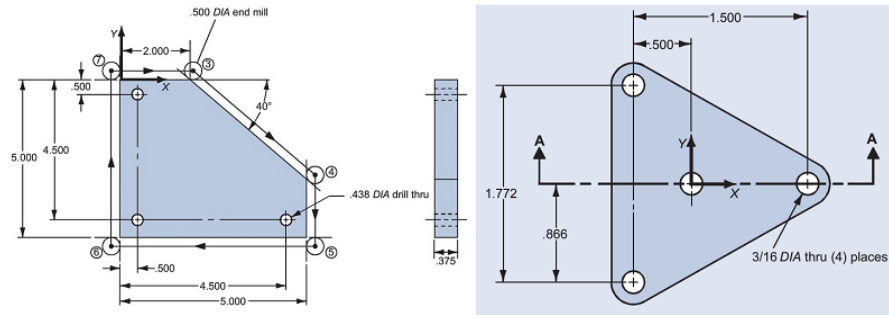


Figure 6.5: The blueprint for the two more complicate exercises [18]

6.4 The accuracy of the generated feedback

The feedback generation approach requires labeled data in the database. We took 60 of the students' code to evaluate the feedback generation approach. We labeled 30 of the code and used the other 30 for evaluation. We consider the feedback generation as a ranking problem, because our goal is to help the students find out missing facts or misconceptions by ranking the most helpful learning content as high as possible. Therefore, we use the normalized discounted cumulative gain (NDCG) to evaluate the performance of our feedback generation approach. The calculation of NDCG is shown below, where p means the number of items; rel_i means the importance of item i ; and $IDCG$ means ideal DCG value.

$$NDCG = \frac{DCG}{IDCG}$$

$$DCG = \sum_{i=2}^p \frac{rel_1}{\log_2 i}$$

We calculated $NDCG@1$ and $NDCG@3$ ($NDCG@n$ means the NDCG value for the first n items), because during the evaluation by the students, we observed that most of them

only clicked on the first three items in the feedback list. To calculate the NDCG score, we set the importance of the first three relevant items as 3, 2, and 1. The value of NDCG@1 and NDCG@3 were 0.6 and 0.662, respectively. Figure 6.6 (a) shows an example of the code submission. The student planned to use inch mode but forgot to set G20 (inch unit) word, so the system will use G21 (millimeter mode) by default. The interpreter will not report any syntax error. However, the CNC machine will cut an undersized part. Figure 6.6 (b) illustrates the execution result of this program. The feedback generated by our approach is “Lesson 43: G20 an G21 – Length Units”.

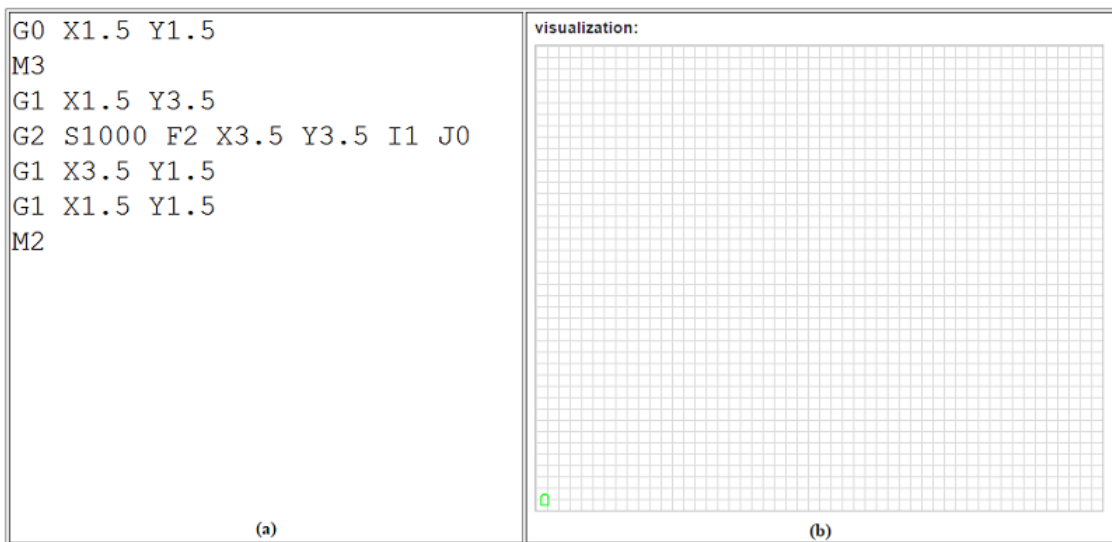


Figure 6.6: An example of the students' submission

The feedback generation approach is also evaluated on a more difficult exercise problem (illustrated in Figure 6.7, adapted from Valentino and Goldenberg, 2012 [18]) with a synthesized dataset. This dataset contains 25 CNC code, which is created based on the common mistakes the students made in the course “MMET 181 - Manufacturing and As-

sembly Processes” and “MMET 380 - Computer-Aided Manufacturing”. Figure 6.8 (a) shows an example CNC code in the dataset and Figure 6.8 (b) shows the messages from the CNC interpreter. The CNC interpreter reports a syntax error: “Syntax error 152 Radius to end of arc differs from radius to start”. However, it is incorrect not because of the wrong radius, but because of the wrong distance mode which causes the wrong starting point of the arc and thus leads to the syntax error.

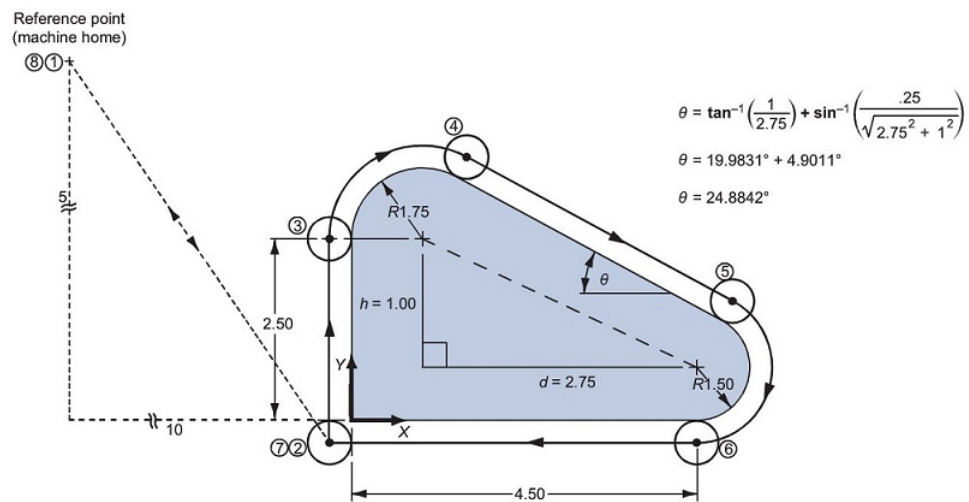


Figure 6.7: The blueprint of the exercise problem used for the feedback evaluation [18]

The dataset is randomly divided into two sets: one set contains 10 CNC programs, which is stored in the database, representing the past students’ submissions, and the other set contains 15 CNC programs, representing the new submissions. The result of NDCG@1 and NDCG@3 were 0.733 and 0.872, respectively.

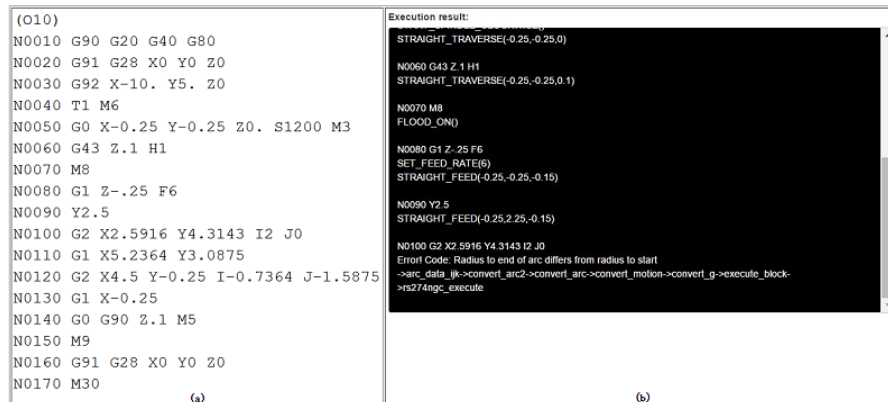


Figure 6.8: An example in the dataset: (a) shows a program in dataset and (b) shows the execution result of the program

6.5 Learning style analysis

Among the 93 students who participated in the pre-test and post-test evaluation, 40 students participated in the survey of learning styles [31]. The learning style have four dimensions: (1) active (ACT) or reflective (REF), (2) sensing (SEN) or intuitive (INT), (3) visual (VIS) or verbal (VRB), and (4) sequential (SEQ) or global (GLO). Active learners prefer to get involved in the experiments, while reflective learners prefer to think about the information first. Sensing learners like using well-established methods to solve a problem and they are patient in learning new facts, while intuitive learners usually are innovative in the problem solving process, and they dislike repetitive work. Visual learners prefer to learn by viewing diagrams, demonstrations, and so on, while verbal learners prefer to learn through text and dialogue. Sequential learners tend to absorb new information sequentially, while global learners tend to choose the learning content in the order they like. Figure 6.9 shows an example of the student's result of the survey.

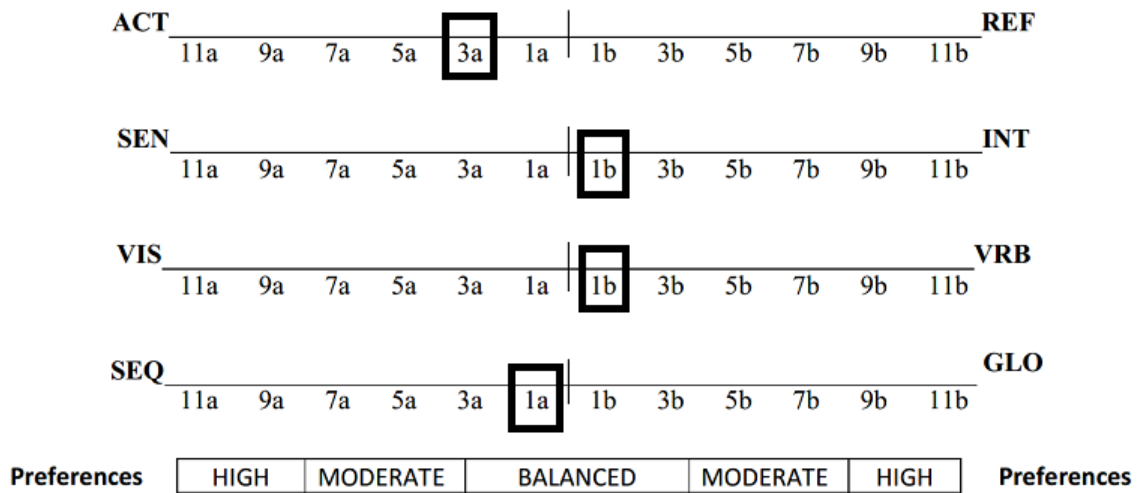


Figure 6.9: An example of the learning styles survey result

We analyzed the correlation coefficient between the learning style and the learning gain in the pre-test and post-test. For each dimension of the learning style, we set the value from -11 to 11, from left to right in Figure 6.9 (11a to 11b, respectively). For the four learning styles, the correlation coefficients were -0.21, -0.13, -0.23, and 0.09, respectively. The result shows that active learners and visual learners tend to benefit more from our CNC-Tutor, compared with reflective learners and verbal learners.

6.6 The survey

There were 93 students who participated in the evaluation survey of our system. Depending on their knowledge level on CNC, the students were divided into two groups. The students in the first group had limited knowledge on CNC while the students in the second group knew basic knowledge in CNC. Figure 6.10 shows the overall rating of the first group and Figure 6.11 shows the overall rating of the second group. The major difference of the

ratings was on the third question: “The instructional materials were easy to understand”, where the first group gave only 4.75 points, while the second group gave 5.09. Overall, the survey of the CNC-Tutor shows a positive impact of the CNC-Tutor on the students’ learning experience.

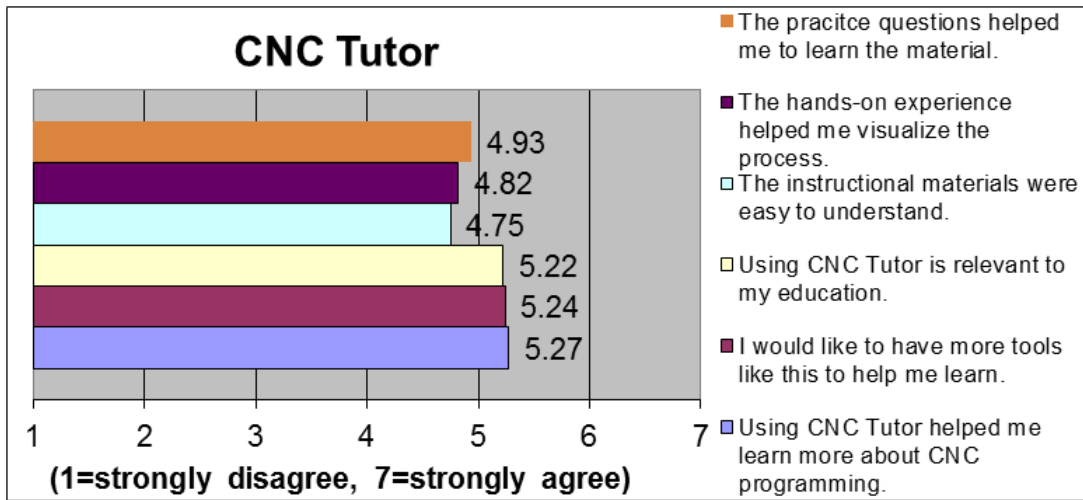


Figure 6.10: The survey of the first group (limited CNC knowledge)

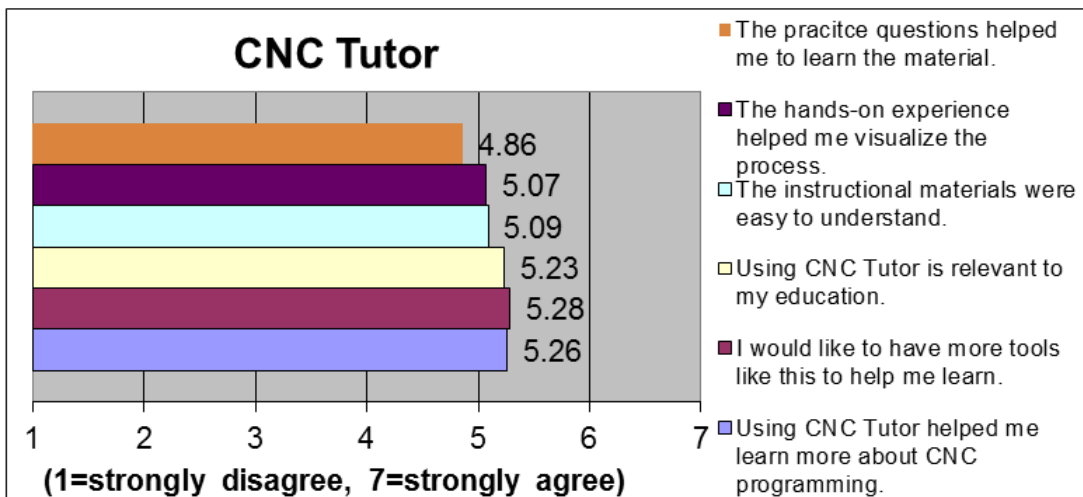


Figure 6.11: The survey of the second group (basic CNC knowledge)

7 DISCUSSION AND CONCLUSION

7.1 Contributions

We built an Intelligent Tutoring System for the domain of Computer Numerical Control. The learning module and the quiz module focus on teaching the facts and concepts, while the exercise module focuses on improving the students' problem solving skills. We proposed a novel hint and feedback generation approach that can generate proper hints and feedback based on the past data. Experiment showed that the proposed approach can be used to generate meaningful hints over 85% of the time, and the feedback can help the students find out missing facts and misconceptions effectively. The survey shows a positive impact on the students' learning experience.

7.2 Limitations

The feedback generation approach requires labeled data in the database. However, analyzing and labeling CNC code is difficult and time consuming. In addition, the feedback messages are limited to the learning content that helps the students to figure out their missing facts or misconceptions. During the evaluation, we observed that these feedback messages are more helpful for the students in finding out missing facts. For the misconceptions, however, error-specific messages might be more helpful.

7.3 Future works

Currently, our CNC-Tutor only contains basic facts, concepts and CNC programming knowledge. More lectures and exercises such as “Computer-Aided Manufacturing” can be added to the CNC-Tutor.

Because analyzing and labeling CNC code is difficult, we consider using a crowd-sourcing approach that allows students themselves to provide help messages.

REFERENCES

- [1] T. R. Kramer, F. M. Proctor, and E. Messina, “The nist rs274/ngc interpreter-version 3,” *National Institute of Standards and Technology (NIST), NISTIR*, vol. 6556, 2000.
- [2] A. C. Graesser, M. W. Conley, and A. Olney, *Intelligent tutoring systems*. Washington, DC: American Psychological Association, 2012.
- [3] K. R. Koedinger, E. Brunskill, R. S. Baker, E. A. McLaughlin, and J. Stamper, “New potentials for data-driven intelligent tutoring system development and optimization,” *AI Magazine*, vol. 34, no. 3, pp. 27–41, 2013.
- [4] D. Sleeman and J. S. Brown, *Intelligent tutoring systems*. London : Academic Press, 1982.
- [5] R. Nkambou, J. Bourdeau, and R. Mizoguchi, *Introduction: what are intelligent tutoring systems, and why this book?*, pp. 1–12. Berlin, Heidelberg: Springer, 2010.
- [6] A. Munro, M. C. Johnson, Q. A. Pizzini, D. S. Surmon, D. M. Towne, and J. L. Wogulis, “Authoring simulation-centered tutors with rides,” *International Journal of Artificial Intelligence in Education*, vol. 8, no. 3-4, pp. 284–316, 1997.
- [7] R. Nkambou, *Modeling the domain: an introduction to the expert module*, pp. 15–32. Berlin, Heidelberg: Springer, 2010.

- [8] S. B. Blessing, *A programming by demonstration authoring tool for model-tracing tutors*, pp. 93–119. Dordrecht: Springer, 2003.
- [9] A. Mitrovic, K. R. Koedinger, and B. Martin, *A comparative analysis of cognitive tutoring and constraint-based modeling*, pp. 313–322. Berlin, Heidelberg: Springer, 2003.
- [10] A. Mitrovic, “An intelligent sql tutor on the web,” *International Journal of Artificial Intelligence in Education*, vol. 13, no. 2-4, pp. 173–197, 2003.
- [11] N. Milik, M. Marshall, and A. Mitrovic, “Teaching logical database design in erm-tutor,” in *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*, pp. 707–709, 2006.
- [12] J. Bourdeau and M. Grandbastien, *Modeling tutoring knowledge*, pp. 123–143. Berlin, Heidelberg: Springer, 2010.
- [13] T. Murray, *An overview of intelligent tutoring system authoring tools: updated analysis of the state of the art*, pp. 491–544. Dordrecht: Springer, 2003.
- [14] J. R. Anderson and C. Lebiere, *The atomic components of thought*. Mathway, NJ: Lawrence Erlbaum, 1998.
- [15] W. R. V. Joolingen and T. D. Jong, *Simquest*, pp. 1–31. Dordrecht: Springer, 2003.

- [16] S. Ainsworth, N. Major, S. Grimshaw, M. Hayes, J. Underwood, B. Williams, and D. Wood, *REDEEM: simple intelligent tutoring systems from usable tools*, pp. 205–232. Dordrecht: Springer, 2003.
- [17] T. Murray, *Eon: authoring tools for content, instructional strategy, student model and interface design*, pp. 309–339. Dordrecht: Springer, 2003.
- [18] J. Valentino and J. Goldenberg, *Introduction to computer numerical control*. Upper Saddle River, NJ: Pearson, 2012.
- [19] A. Overby, *CNC machining handbook: building, programming, and implementation*. New York, NY: McGraw-Hill, Inc., 2010.
- [20] C. J. Butz, S. Hua, and R. B. Maguire, “A web-based intelligent tutoring system for computer programming,” in *Proceedings of International Conference on Web Intelligence*, pp. 159–165, IEEE, 2004.
- [21] I.-H. Hsiao, P. Brusilovsky, and S. Sosnovsky, “Web-based parameterized questions for object-oriented programming,” in *Proceedings of World Conference on E-Learning, E-Learn*, pp. 17–21, 2008.
- [22] P. Brusilovsky and S. Sosnovsky, “Individualized exercises for self-assessment of programming knowledge: An evaluation of quizpack,” *Journal on Educational Resources in Computing*, vol. 5, no. 3, p. 6, 2005.

- [23] S. Abu-Naser, A. Ahmed, N. Al-Masri, A. Deeb, E. Moshtaha, and M. AbuLamdy, “An intelligent tutoring system for learning java objects,” *International Journal of Artificial Intelligence and Applications*, vol. 2, no. 2, 2011.
- [24] W. Jin, T. Barnes, J. Stamper, M. J. Eagle, M. W. Johnson, and L. Lehmann, *Program representation for automatic hint generation for a data-driven novice programming tutor*, pp. 304–309. Berlin, Heidelberg: Springer, 2012.
- [25] K. Rivers and K. R. Koedinger, *A canonicalizing model for building programming tutors*, pp. 591–593. Berlin, Heidelberg: Springer, 2012.
- [26] T. Lazar and I. Bratko, *Data-driven program synthesis for hint generation in programming tutors*, pp. 306–311. Cham: Springer International Publishing, 2014.
- [27] N.-T. Le and W. Menzel, “Using weighted constraints to diagnose errors in logic programming—the case of an ill-defined domain,” *International Journal of Artificial Intelligence in Education*, vol. 19, no. 4, pp. 381–400, 2009.
- [28] K. Rivers and K. R. Koedinger, “Automatic generation of programming feedback: A data-driven approach,” in *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013)*, p. 50, 2013.
- [29] S. Gross, B. Mokbel, B. Hammer, and N. Pinkwart, “Feedback provision strategies in intelligent tutoring systems based on clustered solution spaces,” in *DeLFI 2012: Die 10. e-Learning Fachtagung Informatik* (J. Desel, J. M. Haake, and C. Spannagel, eds.), pp. 27–38, Köllen, 2012.

- [30] D. Lindberg, F. Popowich, J. Nesbit, and P. Winne, “Generating natural language questions to support learning on-line,” in *Proceedings of the 14th European Workshop on Natural Language Generation*, pp. 105–114, Association for Computational Linguistics, 2013.
- [31] R. M. Felder and L. K. Silverman, “Learning and teaching styles in engineering education,” *Engineering education*, vol. 78, no. 7, pp. 674–681, 1988.

APPENDIX A

PRE-TEST AND POST-TEST

A.1 The result of the pre-test and post-test

Table A.1: The result of the pre-test and post-test

Population group	Pre-test score	Post-test score	Learning gain
1	40	50	10
1	12	42	30
1	44	34	-10
1	0	60	60
1	20	30	10
1	70	70	0
1	10	62	52
1	20	72	52
1	40	72	32
1	62	64	2
1	10	52	42
1	32	32	0
1	10	52	42
1	30	50	20
1	2	44	42
1	20	50	30
1	0	70	70
1	44	40	-4
1	10	70	60
1	40	40	0
1	24	40	16
1	30	84	54
1	30	40	10
1	30	90	60
1	24	76	52
1	32	66	34
1	36	70	34
1	16	50	34
1	42	26	-16

Table A.1 continued

Population group	Pre-test score	Post-test score	Learning gain
1	32	70	38
1	34	50	16
1	22	60	38
1	30	64	34
1	20	94	74
1	36	52	16
1	10	60	50
1	22	64	42
1	67	85	18
1	36	75	39
1	85	85	0
1	32	85	53
1	0	54	54
1	32	59	27
1	20	85	65
1	32	65	33
1	40	60	20
1	24	85	61
1	30	85	55
1	20	65	45
1	40	85	45
1	90	100	10
2	54	75	21
2	40	45	5
2	42	61	19
2	42	26	-16
2	10	46	36
2	70	85	15
2	24	26	2
2	44	83	39
2	36	75	39
2	30	40	10
2	30	64	34
2	49	100	51
2	60	85	25
2	60	75	15
2	50	75	25
2	32	44	12

Table A.1 continued

Population group	Pre-test score	Post-test score	Learning gain
2	32	85	53
2	34	70	36
2	42	42	0
2	30	42	12
2	85	85	0
2	22	75	53
2	30	52	22
2	32	60	28
2	30	70	40
2	66	85	19
2	20	6	-14
2	22	60	38
2	30	79	49
2	46	75	29
2	10	100	90
2	22	50	28
2	30	85	55
2	22	60	38
2	20	32	12
2	30	65	35
2	81	36	-45
2	54	70	16
2	26	50	24
2	60	70	10
2	2	62	60
2	22	60	38

APPENDIX B

HINT GENERATION

B.1 Hint generation templates

Rapid positioning: G0

```
<template>
```

```
    Rapid positioning (G0) - move to (_x, _y)
```

```
</template>
```

Linear interpolation: G1

```
<template>
```

```
    Linear motion (G1) - move to (_x, _y)
```

```
</template>
```

Circular interpolation: G2 and G3

```
<template>
```

```
    Clockwise circular motion (G2) - move to (_x,_y)
```

```
</template>
```

```
<template>
```

```
    Counterclockwise circular motion (G3) - move to (_x,_y)
```

```
</template>
```

Hole operation

```
<template>
```

```
    Simple drilling at (_x,_y)
```

```
</template>
```

```
<template>
```

```
    Counterbore cycle at (_x,_y)
```

```
</template>
```

```
<template>
```

```
    Deep hole operation at (_x,_y)
```

```
</template>
```

Length Unit

```
<template>
```

```
    Change length unit to: (_length_unit)
```

```
</template>
```

Distance mode

```
<template>
```

```
    Change distance mode to: (_distance_mode)
```

```
</template>
```

Feed rate

```
<template>
```

```
    Set feed rate (F) at (_feed_rate) ipm
```

```
</template>
```

Spindle speed

```
<template>
```

```
    Set speed (S) at (_speed) rpm
```

```
</template>
```

Spindle

```
<template>
```

```
    Turn spindle off
```

```
</template>
```

```
<template>
```

```
    Turn spindle on (clockwise) - M3
```

```
</template>
```

```
<template>
```

```
    Turn spindle on (counterclockwise) - M4
```

```
</template>
```

Tool number

```
<template>
```

```
    Select tool number (T): (_tool_number)
```

</template>

Program end

<template>

Check program end (M2 or M30)

</template>

Correct

<template>

Looks good, you can submit it.

</template>