SELECTED ENGINEERING PROBLEMS
NUMBER 6
INSTITUTE OF ENGINEERING PROCESSES AUTOMATION
AND INTEGRATED MANUFACTURING SYSTEMS

Krzysztof FOIT

Institute of Engineering Processes Automation and Integrated Manufacturing Systems
Corresponding author: krzysztof.foit@polsl.pl

TASK-LEVEL PROGRAMMING OF A ROBOTIC SYSTEM USING HIGH LEVEL LOGIC LANGUAGE

Abstract: The planning of robotics task is the main problem during the creation of the code for robot's controller. In the initial phase of task designing, the program should be described on low level of details i.e. only the main actions should be concerned. This is so-called task-level approach and the program could be shown in the form of pseudo code, which is not standardised form of programming language. The analysis of the available literature points out that the formalised pseudocode could be successfully used for future processing in the field of mobile robotic, while the application in the field of stationary robotics is in early stage of development. However some task requires detailed analysis before creating the plan of program, because of complex task conditions. In such cases the operator could use the logic programming language, like Prolog, to solve the problem.

1. Introduction

Most of the effort during developing the code of a robot's program is focused on translation of the task's description to the machine-comprehensible form. The job that could be easily described in the everyday language often requires many lines of a source code. The things are even worse when the programmer must resolve some additional problems before coding – for example, to determine the correct sequence of manipulator's movements, according to the specifics of objects that should be placed at the correct position. One of the possible ways to resolve of such problem is shown in Figure 1. The method consists of four steps:

- problem analysis and solution,
- description of the solution in the form of block diagram, SFC/GRAFCET or in everyday language,
- translation of such description to the form of the source code – skeleton of the program,
- the development of the source code – procedures, functions, error handling etc.

The first step may include some elements of computer aided program development. Using one of the available logic programming languages, the operator could resolve complicated issues regarding the sequence of arm's movements.

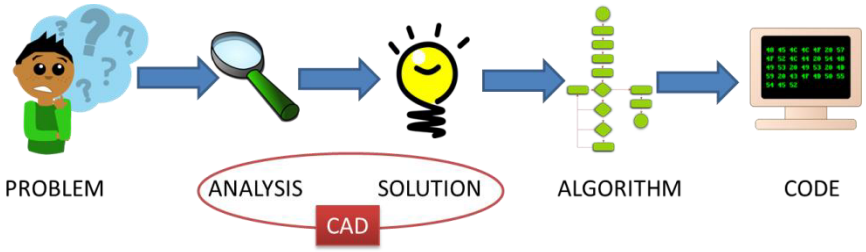

Fig.1. The process of resolving complex programming issue

The other scenario is related to the automatic task planning, where the role of the operator is limited to entering the data that describes the environment and the computer program does the rest of the job. The result should be more or less complete program that could be run on the robot's controller. Because different types of robots use different programming languages, there is almost no possibility to write the program in one, universal way. It should also be noted that the differences between the various types of industrial robots concern not only the software level, but also the hardware requirements, like controller abilities, manipulator kinematics, etc. In the literature, there are several cases where the general purpose task planning languages are used in the field of robotics [1-3], like STRIPS, ADL, UWL etc. Of course the plan is not a classical program. Lin and Levesque said that "*Clearly these [structures] would not be programs in a traditional language like C or LISP. For one thing, the primitive statements of the program would have to involve the actions […], rather than the usual variable assignment or read/write statements.*" [1]. The conclusion is that there is the gap between automatic task planning and the resultant robot's program, which is hardly filled by a proper method, because of the differences between robots.

The paper discus some problems connected with automated task planning for the industrial robot. The focus is mainly put on the use of high-level logic programming language to solve the planning problems and to obtain the code in the form of task-level description. Some aspects of obtaining the template code for a particular robot will be also discussed in the further part of the paper.

2. Development of the task-level code

As it was mentioned, the task-level code is based on actions, rather than typical variable and keywords processing. This involves the use of so-called pseudocode. Using such approach allows presenting the task in the form similar to the source code written in high-level programming language. The code written in pseudocode is often shorter and more general than source code of robot's program and therefore is more understandable by a human.

There is no clear and strict definition of the pseudocode. It could be described as a form of textual description of the algorithm that uses the syntax based on the popular high-level programming languages, but the keywords are taken from everyday language. Gilberg and For ouzan state that the pseudocode is "*English-like representation of the algorithm logic. It is part English, part structured code.*" [4] The syntax of the pseudocode is mainly based on the one of popular programming language: frequently Pascal/Modula, less often C or structural form of BASIC languages. There is no special outlines how to use the syntax or what words are acceptable. In general, the pseudocode is not intended for processing with

compilers or interpreters, although it is possible to introduce some rules in order to create pseudocode-like, high level scripting language [5].

Compared to the block diagram, the pseudocode is more concise in its form and offers better commenting of the code. Because of preserving the structural form, the pseudocode could be translated by the operator to the source code in any high-level programming language.

The very important thing is to keep in mind, that pseudocode is the mix of structural code and natural-like language. The key is the term *natural-like*, because it should be distinguished from natural language. Everyday communication between people is set on many shorthands and simplifications. This introduces the ambiguity and noise, leaving the considerable margin for interpretation. For example the sentence "*give me some milk, please*" carries only the information about what the person wants. There is no data about the amount, if the milk is fat or skim, about the container or the action. Moreover, if someone wants to drink the milk, we pour the milk into the glass or mug, but that person may need it for coffee – there is no information about intended use in the sentence. The missing knowledge could be complemented by requesting the missing data or by the interpretation of the other messages – the person that gives the milk is in certain environment and uses its own knowledge and experience. There should be noted that the knowledge comes from an external source, while the experience is based on heuristic.

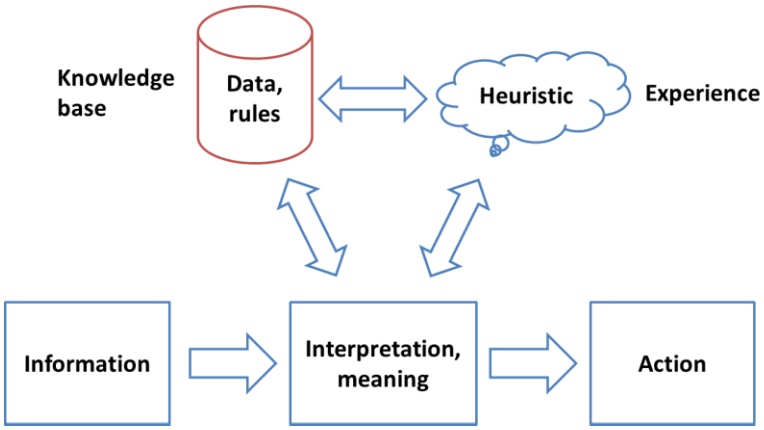

Fig.2.Data flow during interpretation of information

Figure 2 shows the data flow during interpretation of the information. Considering human abilities, it can be said that a person first of all will use the knowledge from database (own mind, books, observations). If the solution has not been found, then he will try to make interpretation based on his experience and skills to combine the owned knowledge with the new facts. The consequence is the creation of the new knowledge that could be used in the future.

During the development of the task-level pseudocode, it must be taken into consideration that it is interpreted mainly by human. Machine processing is very limited. However Blank et al. [5] show the example of pseudocode implementation using the unified framework written in Python. The Pyro (*Py*thon *Ro*botics) is composed of a set of Python classes that provides the programming interface (API). The authors explain that the API covers the low-level

implementation of "drivers" for a few robots. Although they do not directly state that it is possible to automatically generate the Pyro scripts, it seems to be feasible.

In conclusion, it should be mentioned that for further processing of the pseudocode it is necessary to implement strict syntax. All of the actions should be determined in the complete and unambiguous way, albeit it is not mean that the obtained code will be complete and ready to run.

3. Splitting the task into the smaller parts – structural programming approach

The task-level planning gives the description of the job using lowest level of details. However, each activity consists of lower-level actions that can be conceived as subroutines. In this manner, a tree structure is created, where the "branches" are lower-level subroutines, and the "leaf" is the lowest level routine, which could not be divided anymore.

```
Program Drill_Holes;

var
Pos_Array: array [1..3] of Positions;

begin
    for i:=1 to 3 do
    begin
        Move_Tool.Position(pos_array[i]);
        Drill_Hole(depth_of_hole=18);
    end;
end.
```

```
Procedure Drill_Hole(depth_of_hole);
begin
    Drill_Motor(on);
    Move_Tool.Towards_Material until contact;
    Move_Tool.Inwards until depth < depth_of_hole;
    Move_Tool.Outside_Material;
    Drill_Motor(off);
end;
```

Fig. 3. The example of the pseudocode using structural programming style

The structural approach (Figure 3) allows reusing the subroutines (procedures or functions) with different parameters, as well as to make use of code snippets and templates during pseudocode translation. On the other hand, this may lead to some problems similar to so called "copy&paste programming" [6]. The negative effects of such approach manifest themselves in the bad optimization of the code, replication of errors etc. Therefore, if the pseudocode translation involve the use of snippets and templates (for example in order to realize communication with the environment through the industrial network or to handle a tool mounted on the manipulator of the robot) the resultant code must be carefully checked before running it on the robot's controller.

4. Planning of robotics tasks using logic programming language

Some of the tasks realized by industrial robots require complicated and long plans that contain many lines of code. Considering the "Towers of Hanoi" problem [7-9] from the point of view of robotics, it could be said that it is – in general – a complicated assembly task. One of the efficient ways to solve elaborated task planning problem is the use of logic programming language – this method is frequently used in the mobile robotics, eg. [10,11]. Using the similar approach in stationary robotics is less popular but also possible..

One of the most known logic programming languages is the Prolog. It was developed in early 1970's. As opposed to structural or object-oriented languages it has declarative nature

that means there is no coded algorithm. The source code of the Prolog program consists of logical formulas describing the problem. The compiler solves it using the entered data.

As the example consider the simple robotic cell (Figure 4) that consists of robot, input storage, output storage and temporary buffer. The aim is to perform the assembly process of a car lamp. The input storage contains bodies of the lamp with mirrors, bulbs and glass covers that are arranged in the random order. The temporary buffer has the capacity of one element. The robot assemblies the lamp by manipulating the elements in the following way:

- the body is placed in the holder,
- the bulb is mounted inside the body,
- the glass is covering the body with the bulb.

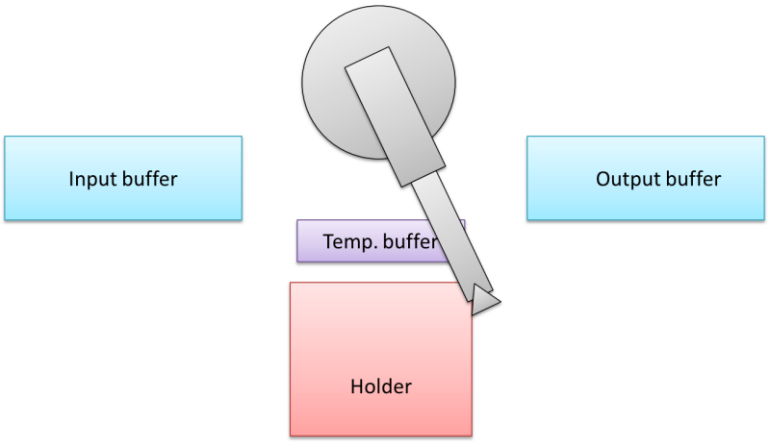

Fig. 4. The considered assembly cell

Using the Prolog, the input data must reflect the configuration of the buffer. In order to do that, the *input_buffer* predicate has been defined in the form of (1):

$$\text{input\_buffer (part1, part2, part3)} \tag{1}$$

where *part1*, *part2*, *part3* are the variables used for particular components. For the example configuration of input buffer in the form *input_buffer (body, glass, bulb)*, the resultant pseudocode is shown in Figure 5.

```
Move body from in_buffer to holder
Move glass from in_buffer to temp_buffer
Move bulb from in_buffer to holder
Move glass from temp_buffer to holder
Move lamp from holder to out_buffer
```

Fig.5. The example of the resultant pseudocode

Some sequences of parts in the input buffer give no solution because there is no possibility to store more than one element in the temporary buffer.

5. Conclusions

The development of the program at the task level significantly simplifies the process of programming because of using the high level of generality. This is possible through the use of pseudocode, which is the mixed form of a programming language syntax and natural language. Using the standardized form of pseudocode gives the possibility to translate the task-level description into the more or less complete source code for the robot's controller. On the other hand, there are kind of tasks that cannot be easily programmed and requires additional analysis before making the plan. In such case the logic programming language – like Prolog – could be used to solve the problem. As a result, the pseudocode is obtained that could be used for further processing. Future work will focus on coherent system that will assist the operator from the identification of the problem to the creation of source code for the robot's controller.

References

1. Lin F., Levesque H.J.: What robots can do: robot programs and effective achievability, ArtificalInteligence 101 (1998), pp. 201-226.
2. Fikes, R., and Nilsson, N.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, Artificial Intelligence, 2(3/4):189-208, 1971.
3. Levesque, H.: What is Planning in the Presence of Sensing?, In The Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI-96, pp. 1139-1146, Portland, Oregon, 1996.
4. Gilberg R., Forouzan B.: Data Structures: A pseudocode approach with C. Cengage Learning, 2004.
5. Blank D., Kumar D., Meeden L., Yanco H., Pyro: A python-based versatile programming environment for teaching robotics, Journal on Educational Resources in Computing (JERIC) 4.3 (2004): 3
6. Kim M., Bergman L., Lau T., Notkin D.: An Ethnographic Study of Copy and Paste Programming Practices in OOPL, Proceedings of the 2004 ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2004).
7. Momtaz A.S.Z. et al.: A Practical Solution for Robotic Arm of the Towers of Hanoi Problem, International Journal of Computer and Electrical Engineering, 2011, 3.4: 583.
8. Hoffmann, J.F.F.: The fast-forward planning system. AI magazine, 2001, 22.3: 57.
9. Benjamin D. P., Lyons, D., Lonsdale, D.: Designing a robot cognitive architecture with concurrency and active perception. In Proceedings of the AAAI Fall Symposium on the Intersection of Cognitive Science and Robotics. Menlo Park, CA: AAAI Press, 2004
10. Hähnel D., Burgard W., Lakemeyer G.: GOLEX—bridging the gap between logic (GOLOG) and a real robot. In: KI-98: Advances in Artificial Intelligence. Springer Berlin Heidelberg, 1998. p. 165-176.
11. Grosskreutz H.: Towards more realistic logic-based robot controllers in the GOLOG framework. 2002. PhD Thesis. Bibliothek der RWTH Aachen.