Development of machining topology knowledge base

Yeung, Millan K.; Wang, Lihui; Orban, Peter

Publisher's version / Version de l'éditeur:

ICME 2003 Conference proceeding, 2003

Development of Machining Topology Knowledge Base

Millan K. Yeung[1], Lihui Wang[1] and Peter Orban[1]

[1]*Integrated Manufacturing Technologies Institute, National Research Council Canada*

Keywords

Knowledge base, machining topology, object-oriented paradigm, CNC programming, process planning, artificial intelligence.

Abstract

CNC machining is one of the widely used manufacturing process. Its flexibility and efficiency allows the production of low quantity or high volume parts cost effectively. However, the set up of machining topology – the logical sequence of machining operations for different features in a part, is mostly relied on the skill and experience of the CNC programmer. The experience and knowledge could diminish or even disappear if the programmer leaves. An intelligent flexible knowledge base would be an ideal tool to preserve this knowledge and experience for the utilization of CNC programming. This paper describes the development of a machining topology knowledge base of an intelligent process planning system for the automation of CNC programming.

1. Introduction

CNC machining is one of the most efficient and widely used manufacturing processes for industries. It provides high efficiency for mass production of consumer products and flexibility for low quantity production of specialized parts and components. Despite the flexibility and capability of CAD/CAM systems, CNC programming still requires a skillful programmer who should not only be a CAD/CAM and computer literate but also a machining expert to plan and generate the tool paths. Recent publications have started to address this shortcoming of CNC programming. Many research works have been conducted to find ways to reduce this deficiency but they mostly dealt with specific environments and conditions or in higher-level generality [1-9]. In a recent publication, Yeung [10] described the development of an intelligent process planning system (IPPS) for CNC programming and machining. The development was divided into three modules, the feature extraction, the tool competition and the tool optimization. The feature extraction module captures the topology of machining of the given part while the tool competition and tool optimization modules provide an optimal tool and condition for

[1] 800 Collip Circle, London, Ontario, N6G 4X8, Canada. Email: Millan.Yeung@nrc-cnrc-gc.ca

machining these topological features. This paper describes the development of the feature extraction module and gives an overview of its design and implementation.

2. Topological machining feature extraction

Part features for machining are well defined by CAD systems and extensive work on feature recognition for CAD/CAM has been pursued [11-12]. The focus of this development is aimed at storing and extracting these features in a logical sequence according to their machining order. Output of the development would be a topological machining feature extraction system that can interact with other modules in Yeung's [10] IPPS but also work independently to define and extract topology of machining parts. The core design of this system is divided into two working sets and a topological ordering engine. The primitive set is a set of elementary features such as holes, wedges, bosses, slots, islands, threads, etc. that their machining operations are well defined. The feature set consists of machining features that are composed by a number of primitives or sub-features. A given part is subdivided into a number of machining features. Each of these features is further divided into a number of machining primitives or sub-features. Figure 1.0 shows the decomposition of a part in the machining operation.

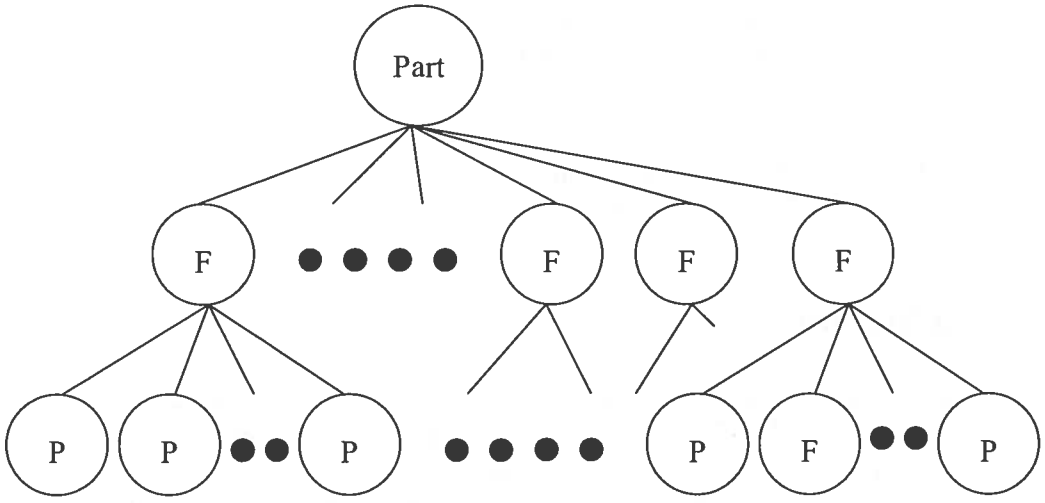

Figure 1.0 Representation of a part to be machined.

The system then determines the topological order of these features and returns it in a logical sequence for CNC programming. The core of this system is the knowledge base that consists of data structure and operational requirements and constraints of these features. Emphasis of the implementation lies heavily on the architect of the data structure. Object-oriented programming paradigm gives a convenient way to facilitate the implementation.

2.1. The primitive

There are a number of commonly used machining primitives and their corresponding machining operations are well defined by logics and past experiences. The essence of machining primitive structure is the geometries of the primitive and the blank configuration for that primitive. This information is used for the definition of the machining operations for the primitive. Primitives should be unique and mutually exclusive from each other. To illustrate the data structure of machining primitives, a primitive example 'hole' is used. The data structure of 'hole' will consist of dimensions of the hole as well as the dimensions of the blank. It can be represented in the form of an object in the object-oriented programming paradigm. Following is the pseudo code of the structure.

```
Object hole
{
      hole( parameter-list );
      references(start-point, direction-cosine);  //identify the
         location and orientation
      hole-diameter;
      hole-depth;
      hole-type(blind, through);
      stock-inside-diameter;
};
```

Most of these variables are self-explanatory and their structures are not unique. The 'stock-inside-diameter' indicates the amount of material that need to be removed and if an existing hole is presented so to determine if drill operation is required. In general, the stock information presented in the primitive should be localized and only include the portion that concerns with the primitive. Other information such as material, tolerances, surface finishing, etc. could also be included for quality control. For clarity of illustration, these qualitative parameters and the implementation of their applications are assumed. Furthermore, constraints and requirements for machining the primitive are included. This information allows the assessment of the type of operations for the production of the primitive. For our hole primitive example, this information is represented by embedded methods.

```
Object hole
{
      hole( parameter-list );
      references(start-point, direction-cosine);  //identify the
         location and orientation
      hole-diameter;
      hole-depth;
      hole-type(blind, through);
      stock-inside-diameter;
      …
      BOOL drill-required( ) { return blank-inside-diameter <= 0.0; }
      BOOL finish-cut-required( ) { return tolerance < given-dimension
                                    || surface-roughness < given-Ra; }
};
```

The hole example demonstrated the key information required for the machining operation of primitives.

The term machining primitive gives the impression that the underlying feature is simple and easy to represent and machined. However, this is not always true for many practical and realistic machining primitives. Indeed, profiled surfaces and even free form surfaces such as spherical, parabolic, and NURBS surfaces can also be machining primitives. Their representation and operational requirements and constraints are not as obvious and easily defined. The inclusion of these complex primitives depends on the preferences of the user and the software developers.

2.2. The feature

The feature working set consists of machining features that composed by one or more primitives, other features or could even be a completed part. They are defined by the users according to their needs and frequently machined parts. For example, a brake rotor of an automobile, the part is composed by a number of primitives.

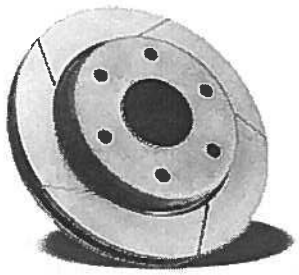

Figure 2.0 An automotive brake rotor.

It can be represented in a feature object data structure.

```
Object rotor
{
      rotor( parameter-list );
      mounting-hole1(mounting-hole1-parameter-list );
      mounting-hole2(mounting-hole2-parameter-list );
      mounting-hole3(mounting-hole3-parameter-list );
      mounting-hole4(mounting-hole4-parameter-list );
      mounting-hole5(mounting-hole5-parameter-list );
      mounting-hole6(mounting-hole6-parameter-list );
      gripping-slot1(gripping-slot1-parameter-list );
      gripping-slot2(gripping-slot2-parameter-list );
      gripping-slot3(gripping-slot3-parameter-list );
      gripping-slot4(gripping-slot4-parameter-list );
      gripping-slot5(gripping-slot5-parameter-list );
      round-boss ( round-bass-parameter-list );  //OD of the hub
      center-hole( center-hole-parameter-list );
      round-OD( round-shaft-parameter-list );  //outside diameter
      OD-groove( OD-groove-parameter-list );  //outside diameter groove
      stock-OD;
      BOOL outside-diameter-operation( … );
      •••;  //assuming primitives at the backside are defined.
};
```

This data structure represents the machining features of the rotor shown in Figure 2.0. Primitives of the structure can be represented in a similar fashion as the hole example described in 2.1. The mounting-hole and center-hole data structures could be identical to the hole example. Other primitives could be constructed in the similar way.

```
//A boss is defined as a protrusion comes out from a surface - the hub
//of the rotor.
Object round-boss
{
      round-boss( parameter-list );
      reference( start-point, direction-cosine );  //direction-cosine
         points to the boss-axis
      diameter;
      height;
      stock-OD;
      stock-height;  //measured from the flat surface and up
};

//A slot is defined as a groove on a surface - slots on the disc of the
//rotor.
Object gripping-slot
{
      gripping-slot( parameter-list );
      reference( start-point, end-point, direction-cosine );
      slot-size( width, height );
      stock-size( width, height );
      start-hole( start-hole-parameter-list );
};

//round-OD is just a round surface - the outside surface of the rotor.
Object round-OD
{
      round-OD( parameter-list );
      OD;  //outside diameter
      length;
      stock( OD, length );
};

//the groove on OD - the groove on the outside surface of the rotor.
Object OD-groove
{
      OD-groove( parameter-list );
      groove-size( width, depth );
      stock( width, depth );
};
```

Furthermore, the data-structure of rotor can be hierarchically nested with sub-features to avoid code redundancy as well as improving the clarity of the implementation.

```
Object rotor
{
      rotor( parameter-list );
```

```
        set-of-mounting-holes( repeat-mounting-hole-definition, number-
                               of-holes );
        set-of-gripping-slots( repeat-gripping-slot-definition, number-
                               of-slots );
        hub( round-boss1, center-hole );
        OD-with-groove( round-OD, OD-groove, stock-OD );
        Operational-methods( ••• );
        •••;  //assuming primitives at the backside are defined.
};
```

2.3. Topological ordering

Topology of machining is defined from pre-defined logical order of certain machining features and special order set by user preferences and past experience. Besides the logical topology such as boring hole before broaching key-slot and facing before drilling, a number of criteria or goals can be applied to determine the topological order of machining features. In general, the criteria of largest volume material removal will take precedence to set the topological order for roughing. For finishing, the common practice is starting from the interior geometries and work toward outer shape. Applying these criteria to the rotor example, we can generate a topological sequence for the machining operation.

```
Rotor = { rough-cut( hub, mounting holes, OD-with-groove ), finish-cut(
          hub, gripping-slots, OD-with-groove) };
```

These criteria are also applied to the sub-feature level and the primitive level. In our example, the topological machining sequences for the hub and outside surface of the rotor are also defined in the same manner.

```
Hub = { rough-cut( center-hole, round-boss ), finish-cut( center-hole,
        round-boss ) );

OD-with-groove = { rough-cut( OD-groove, round-OD ), finish-cut( round-
                   OD ) };
                 //note that the groove requires only rough-cut because
                   of loose tolerance.
```

With the object-oriented-paradigm, the blank configuration is updated at the object structures after their operation. This satisfies the requirement of the feature-extraction module described in the IPPS proposed by Yeung [10].

2.4. Implementation

Within this design, there are three steps to implement the knowledge base for the topology of machining features. The first step is to define the machining primitives. User and developer can determine the depth and granularity of the primitives to meet their needs. Secondly, the definition of machining features or classes of parts to be machined is developed based on the machining primitives. Finally, criteria for setting up the topological sequence are defined according to 2.3 and the knowledge base is completed. To utilize the knowledge base and to complete this topological machining feature

extraction system, two interfaces are needed for capturing the machining features from feature-recognition system and the topological preferences from user. An expert system would be an ideal complement for the user interface but is outside the scope of this development. Figure 3.0 summarizes the development.

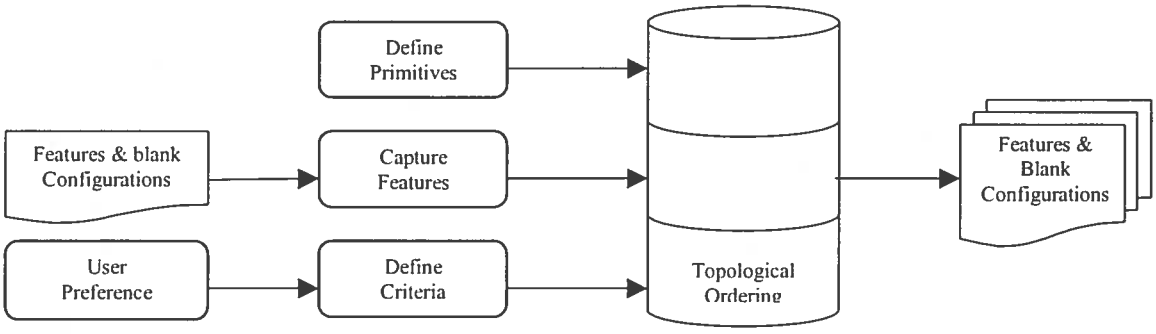

Figure 3.0 Implementation of the topological machining feature extraction

To facilitate the implementation, object-oriented programming systems such as Visual C++® and the dynamic-linked library (DLL) facility of Microsoft® would provide a great developing environment for the construction and dynamic update of the knowledge base and interfaces.

3. Summary and Conclusion

CNC machining is an efficient process but requires skillful and experienced CNC programmer to generate process plan and CNC programs. Yeung described an intelligent system IPPS for process planning of CNC programming in a recent publication [10]. This paper gives a detail description of the development and implementation of the feature extraction module of IPPS. The design of the module is divided into two working sets and a topological ordering engine. The primitive set consists machining primitives while the feature set consists of machining features that are composed by primitives, other features or classes of parts. The topological ordering engine determines the order of machining operation for these features based on criteria set by logics and user's preferences and experience. These working sets and the topological ordering engine are stored and manipulated in a knowledge base. Object-oriented paradigm is used for the implementation of this knowledge base because of its capabilities on method encapsulation (for capturing and retaining knowledge) and polymorphism (inheritance for code re-use). Tools such as Visual C++® and the DLL from Microsoft® are ideal for the development. The flexibility of this topological machining feature extraction system allows the storage and utilization of existing machining topology yet permits the user and developer to use their own special preferences. The basic architecture and structure of the feature extraction system described provides tools for knowledge acquisition and the development of machining topology for various parts. Challenges remain in the capturing and representation of machining topology because of the large variety of machining parts.

Physical machining experiments and modeling will help to build up the knowledge hence expanding the applicability of the system.

4. References

[1] L. Wang, N. Cai, H.-Y. Feng and W. Shen: Feature-Based Reasoning for Machining Process Sequencing in Distributed Process Planning, Proceedings of 13th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2003), Vol.2, pp.655-667, June, 2003.

[2] Fuh J.Y.H., Ji P. and Zhang Y.F., "Future development trends in CAM/CAPP-NC systems", International Journal of Computer Applications in Technology, vol.8, no.3-4, pp.203-210, 1995.

[3] Yeo S.H., "A multipass optimization strategy for CNC lathe operations", International Journal of Production Economics, vol.40, no.2-3, pp.209-218, Aug. 1995.

[4] Norton N., "Controlling the Shop Floor", Manufacturing-Engineer, vol. 72, no. 6, pp. 272-275. December 1993.

[5] Budde W. and Imbusch K., "EXAPT process planning and NC planning with database-supported management of production data", IFIP Transactions B (Applications in Technology), vol.B-3, pp.119-30, 1992.

[6] Pande S.S. and Prabhu B.S., "An expert system for automatic extraction of machining features and tooling selection for Automats", Computer Aided Engineering Journal, vol.7, no.4, pp.99-103, Aug. 1990.

[7] Martin J.M., "A strategy for NC programming", Manufacturing-Engineering. vol.102, no.2, pp.82-84. Feb. 1989.

[8] Mantyla M., "Feature-based product modeling for process planning", Organization of Engineering Knowledge for Product Modeling in Computer Integrated Manufacturing, 2nd Toyota Conference. Elsevier, Amsterdam, Netherlands, xii+461, pp.303-324. 1989.

[9] Chin Sheng Chen, "Developing a feature based knowledge system for CAD/CAM integration", Computers & Industrial Engineering. vol.15, pp.34-40, 1988.

[10] Yeung M., "Intelligent Process Planning System for Optimal CNC Programming – A Step Towards Complete Automation of CNC Programming", Proceedings of the International Conference on Manufacturing Automation (ICMA 2002), ISBN: 1-86058-376-8, pp.169 – 177, Dec., 2002.

[11] Christensen G.K. and Mogensen O.B., "Backward form feature recognition and removal for an automatic CNC-programming system-BCAM", Advanced CAD/CAM Systems – State-of-the-art and Future Trends in Feature Technology. Chapman & Hall, London, U.K., ISBN: 0412617307, pp. 205-216, 1995.

[12] Brun J.M., "From characteristic shapes to form features: a recognition strategy", Advanced CAD/CAM Systems – State-of-the-art and Future Trends in Feature Technology. Chapman & Hall, London, U.K., ISBN: 0412617307, pp. 179-192, 1995.