Expert System to Real Time Control of Machining Processes*

Luis Carlos González Valencia[2], and Rafael Serrano Bello[2]

[1] Computational Logic Group
Dept. of Computer Science and Artificial Intelligence
University of Sevilla
E.T.S.I. Informática, Avda. Reina Mercedes, s/n. 41012 Sevilla, Spain
[2] Dept. de Diseño Industrial
Instituto Andaluz de Tecnología, Parque Tecnológico "Cartuja 93"
c/ Leonardo da Vinci, 2. 41092 Sevilla, Spain

Abstract. Industrial machining processes use automated milling machines. These machines are connected to a control device that provides the basic instructions used to obtain a piece. However, these processes depend on the human decision to diagnose and correct in real time the inaccuracies that can occur. In this work we present an expert system to real time control of machining processes using the information provided by sensors located on the machine. This system has been implemented as a prototype in a Kondia 600 milling machine with a FAGOR 8025-MG control device.

1 Introduction

Industrial machining processes use automated milling machines. These machines are connected to a control device that provides the basic instructions used to obtain a piece. However, these controls are not capable of checking and correcting in real time the imprecisions that happen in the process; for its usual operation, the milling machines still depend on the human decision for the production and modification of its processes. This dependence implies a high consumption of technical resources, affects the quality of manufactured products and the manufacturing and fine tuning process time. All these circumstances motivate that the current work on optimization of industrial machining processes remains a partially solved problem. To deal with this limitation, it has been proposed as a solution the use of artificial intelligence techniques.

According to Kyung [6], Artificial Intelligence applications to machining processes can be grouped into the following categories: knowledge-based expert systems, neural networks, probabilistic inference methods and fuzzy logic and genetic

* This work has been developed under the INFAERO project, supported by the "Orden de Incentivos a los Centros Tecnológicos de la Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía".

algorithms. The difference between these approaches is the way how to represent the knowledge base in the system and its inference engine.

Early work on rule-based expert systems to control the machining process come from the late eighties, as in the case of Bohez [2], who developed the first prototype of an expert system based on 500 rules. Today they are also successfully implemented rule-based expert systems with the genetic algorithms approach [5].

Unlike expert systems where knowledge is made explicit, systems based on neural networks generate their own knowledge from case studies of network training. This means that such systems build the knowledge base through learning, and do not require additional processes of acquiring knowledge [3].

The probabilistic approach uses an influence diagram model with a probabilistic reasoning engine. The influence diagram is developed for the representation of complex problems where the decision is based on incomplete information belonging to several sources [8].

Another line of research are the adaptive controls. The aim of these devices is to control the process input variables using another ones that show the state of the process and must be monitored at all times. However, this architecture can not act in general on external parameters of the machine.

In this work we present an expert system based on production rules to control machining processes. This system is integrated into an interface from which information is collected from the machining process control device itself and sensors on the machine. The system diagnoses the state of the machining process and decides corrective actions to keep it within optimal parameters. The system performs a separate treatment of each datum provided, this allows both deactivation, such as adding new data sources. The system design minimizes the number of changes between successive runs, thereby reducing its response time.

2 Pilot Implementation

The system showed here was implemented on a Kondia 600 milling machine. This machine has a table that can move horizontally in both directions of the XY plane, with a swing jaw to fix the block to be processed, and a spindle that moves vertically, where the cutting tool is placed. It is equipped with a control unit 8025-MG FAGOR from which the cutting instructions needed to process a piece are provided. The machining process instructions are provided through a machining program that is stored in the control unit. While the machining program can not be modified during processing, the behavior of the machine can be altered modifying the machining table motion (also knows as *feed*) and the spindle speed, indicating a reduction over the values provided by the machining program, or ultimately stopping the machine.

To capture real-time information about the process, the machining center has been equipped with several sensors: an infrared temperature sensor providing the cutting tool temperature; two vibration sensors that provide information about the spindle vibration in X direction and the milling table vibration in Y and Z directions; and a strength sensor located under the swing jaw, that provides information about the vertical strength exerted by the cutting tool on the block

to be processed. This information is transmitted to the Sensor-IA application through a data acquisition system.

The neural center of the Sensor-IA application is a user interface implemented in Visual Basic from which data are received from the sensors. These data are sent to the expert system, which generates a set of actions to modify the behavior of the machining center. There are three kinds of actions: acting on the two speed controls of the control device (feed and spindle speed) and acting on a coolant flow directed to the cutting tool. Once generated the set of actions, the expert system returns the control to the interface that is responsible to enforce these actions, and waits for new data. We call an expert system *run* to the process of receive data from the interface, decide the actions and return them to the interface.

In the implementation of this expert system we have pursued the following objectives: A design based on production rules in a shell integrated into Visual Basic, to obtain a good degree of efficiency in processing large sets of rules. A separate treatment of each datum provided from the interface. In this way, the expert system behavior is as expected whether it receives only a datum, or several. This also facilitates the incorporation of new data sources (e.g. from new sensors), because they do not affect or are affected by the treatment of the other data. Finally, a design that minimizes the number of changes between successive queries to the expert system, reducing its response time.

3 Production Systems

A production system is a computational mechanism based on rules. This mechanism has two components: a set of *facts* (*database*) and a set of *rules* (*knowledge base*). The facts are data describing a concrete situation and the rules describe how this situation could change. Every rule has two parts, the *antecedent* and the *consequent*. The antecedent is a set of conditions about the data, mainly the existence or absence of some datum matching a given pattern. The consequent of a rule is a set of actions by means of which some existent data are deleted or new ones are included.

In every moment there are a set of *active rules*, that is rules with their conditions accomplished, obviously this depends on the state of the database. In every step of computation an active rule is selected using a *conflict resolution strategy* and its actions are executed, we say that the rule has been *fired*. The firing of a rule changes the state of the database and, as a consequence, the set of active rules. This process continuates until the set of active rules is empty.

There are several languages designed to define production systems and provide an inference engine in which a production system could be evaluated. A common reference point between all of them is CLIPS [7], used in this work.

3.1 Facts in CLIPS

Basic facts in CLIPS have the following syntax: **`(name v1 ... vn)`**, where **`name`** is a symbol identifying the fact and **`v1`**, ..., **`vn`** are the information stored in the fact. In this case we say that **`name`** is the *type* of the fact.

Facts represents information known in a concrete instant and can change at any time. The elimination of a fact would suggest that the information it represents is no longer relevant or true. In the other hand, new situations generate information represented by new facts.

3.2 Rules in CLIPS

As we have mentioned before, rules have two parts: antecedent and consequent. The antecedent is a set of conditions about the facts, establishing the existence or absence of facts matching a pattern. The consequent is a set of actions associated with the rule, the most common are the removal of existing facts and the inclusion of new ones.

The conditions of the antecedent of a rule are expressed by means of *patterns*. The syntax of these patterns is similar to the syntax of the facts, but we can use variables to specify unknown values and additional conditions about these variables. We can use two kinds of variables in a pattern fact: *simple* and *multiple*. Simple variables can store a simple value and have the syntax **`?name`**. Multiple variables can store a sequence, maybe empty, of values and have the syntax **`$?name`**. In both cases **`name`** is the *name* of the variable. When the value of a variable is not interesting, we can use the *unnamed variables* **`?`** or **`$?`**.

Actions in the consequent of a rule depend on the conditions in its antecedent. So in order to delete a fact, there should be a condition in the antecedent ensuring the existence of that fact. Furthermore, to include a new fact, the associated information must be provided in a explicit way, perhaps from the values of the variables used in the antecedent. It is also possible to use *global variables* in the consequent of a rule. Global variables have the syntax **`?*name*`** and must be previously declared. These variables are used to store values that do not change during the execution of the expert system. In our work, we use them as a communication channel between the expert system and the Sensor-IA application.

3.3 The Inference Engine in CLIPS

CLIPS uses a *forward chaining* inference engine, based on removing or deriving data from a set of facts, following the guidelines specified in the rules. This process has the risk of entering infinite loops of derivation: the simplest example is a rule without conditions or actions, this rule could be fired indefinitely. To avoid such situations, CLIPS exhibits a property called *refraction*, which prevents from firing a rule more than once for a specified set of facts. To improve performance in the rule pattern match stage, a optimized version of Rete algorithm by Forgy [4] is used: instead of testing the conditions of all rules to compute the active rules, it only checks changes in the rules that may be affected by the last rule fired.

4 Description of the Expert System

The expert system integrated into the Sensor-IA application receives as input a data set from the interface. As a result, it generates a set of actions on the manual

controls of the control device. The objective of these actions is to safeguard the proper operation of the machine, and to fit it to a predeterminated level.

The data provided from the interface are considered in the following stages: setting the data level; setting the data state; setting the actions associated with the data; and finally combining the actions associated with the data. In the first three stages the treatment is independent for each datum, and the latter is the only stage in which all the data provided are related in some way.

4.1 Setting the Data Level

The data received from the interface have the form: **(value ?datum ?v)** where **?datum** is a unique identifier for each datum and **?v** is the value of this datum in its range of values.

Since the range of values of each datum is very broad, we considered subdividing it into six intervals or numbered levels. Thus, the *level* of a datum is the number of the interval in which its value is located. The boundaries of these levels are stored as: **(risk-levels ?datum ?l1 ?l2 ?l3 ?l4 ?l5 ?l6 ?l7)** where **?datum** is the identifier of the datum considered and **?l1**, **?l2**, **?l3**, **?l4**, **?l5**, **?l6** and **?l7** are the boundaries of the six levels.

For every datum we take into account its evolution over the last 5 runs of the expert system. We call this information *history* of the datum and it is stored as: **(history-level ?datum ?n4 ?n3 ?n2 ?n1 ?n0)**, where **?datum** is the identifier of the datum considered, **?n0** is the current level of this datum, **?n1** is the level of the datum in the previous run of the expert system, **?n2** is the level of the datum two runs ago, and similarly **?n3** and **?n4**.

The expert system rules determining the data history (and of course, the data level) have the following form:

```
   | (defrule update-history-level-data-2
 1 |   ?h1 <- (value ?datum ?v)
 2 |   (risk-levels ?datum ? ?l2&:(<= ?l2 ?v) ?l3&:(< ?v ?l3) $?)
 3 |   ?h2 <- (history-level ?datum ? ?n3 ?n2 ?n1 ?n0)
   |   =>
 4 |   (retract ?h1 ?h2)
 5 |   (assert (history-level ?datum ?n3 ?n2 ?n1 ?n0 2)))
```

This rule updates the history of a datum whose level is 2. Condition #1 gets the current value **?v** of a datum **?datum**. Condition #2 gets the boundaries of the levels established for the datum and checks that the value **?v** is in the second level (between the boundaries **?l2** and **?l3**). Condition #3 gets the history of the datum in the previous run of the expert system, where the current level (2) should be included in the last position and the oldest level (just the one after the datum identifier) should be removed. The actions of this rule are the elimination of the facts with the current value of the datum and the history of the datum in the previous run (action #4), and the inclusion of a fact with the history of the datum for the current run (action #5).

Once we have updated the information about the data history in the current run, the facts about the current values of the data provided from the interface

are no longer needed and they are removed. The elimination of these facts avoids that they could be used several times in the update rules of the data history, in the same run of the expert system.

4.2 Setting the Data State

The *state* of a datum represents the last change of level in this datum over the last 5 runs of the expert system. This information is stored in independents facts (one for each datum) as: **(state ?datum ?v1 ?v0)**, where **?datum** is the identifier of the datum, the value **?v0** is the current level of this datum and **?v1** shows the trend of change of the datum in the last 5 runs of the expert system in the following way: if the datum comes from a lower level, then the value **?v1** is **?v0** minus 1; if the datum has been stable (that is, in the same level), then the value **?v1** is equal to **?v0**; and if the datum comes from a upper level, then the value **?v1** is **?v0** plus 1.

The state only changes when the current level of a datum is different from the previous one (in this case the state stores the current level and a value computed from it) or when the datum becomes stable in the same level over the last 5 runs of the expert system (in this case the state stores the current level repeated).

The rules to establish the state of a datum are two: the first one computes the changes of level of a datum in the last 5 runs and the second one the stability of a datum in the last 5 runs.

```
   | (defrule set-datum-state-1
 1 |   (history-level ?datum ? ? ? ?v1 ?v0&~?v1)
 2 |   ?h <- (state ?datum ? ?v1)
   |   =>
 3 |   (retract ?h)
 4 |   (if (< ?v1 ?v0)
   |       then (assert (state ?datum (- ?v0 1) ?v0))
   |       else (assert (state ?datum (+ ?v0 1) ?v0))))
```

This rule starts with the history of a datum **?datum** where the current level **?v0** is different from the previous one **?v1** (condition #1). Condition #2 ensures that the state of the datum does not correspond to the current history and therefore it must be changed. Action #3 removes the incorrect state and action #4 inserts in the database the new state computed from the current level **?v0** and the previous one **?v1**.

```
   | (defrule set-datum-state-5
 1 |   (history-level ?datum ?v0 ?v0 ?v0 ?v0 ?v0)
 2 |   ?h <- (state ?datum ~?v0 ?v0)
   |   =>
 3 |   (retract ?h)
 4 |   (assert (state ?datum ?v0 ?v0)))
```

This rule starts with the history of a datum **?datum** where the level has been stable for the last 5 runs (condition #1). Condition #2 ensures that the state of this datum does not correspond to the current history because the values it stores are not equals. Action #3 removes the incorrect state and action #4 inserts in the database the new one.

These rules only change the data state when there are any modifications on it, this minimizes the number of changes between successive runs of the expert system, improving its performance.

4.3 Setting the Actions Associated with the Data

The actions associated with the data depends on the data state, the risk level assumed in the process and the status of the manual controls.

The risk level assumed in the process indicates the kind of use we want of the machine. We have considered three situations: *quality job mode*, where we focus on quality over time; *balanced job mode*, where we seek a balance between quality, time and half-life values of tools and equipment: and *time job mode*, where we focus on time over quality. This information is stored in facts as: **(assumed-risk ?v)**, where **?v** takes the values 1, 2 or 3, respectively.

The status of the manual controls indicates the degree of interference that has already been done in the machining program. These controls are the feed, the spindle speed and the coolant flow. This information is stored in facts as: **(control ?control ?v)**, where **?control** could take the values **feed**, **spindle** or **coolant**.

The rules establishing the actions associated with the data are the core of the expert system, constituting the 85% of it. An example of such rules, with medium complexity, is the following:

```
   | (defrule cut-strength-7[3]
 1 |   (state cut-strength 4 4)
 2 |   (assumed-risk 3)
 3 |   (control spindle ?vs&:(< ?vs 105))
 4 |   (control coolant ?vc)
   |   =>
 5 |   (assert (action cut-strength 4 4 spindle (+ ?vs 5)))
 6 |   (if (< ?vc 10)
   |       then (assert (action cut-strength 4 4 coolant (+ ?vc 5)))))
```

This rule establishes that if the cut strength is stable in the level 4 (condition #1), the assumed risk is 3 (condition #2), the level of the spindle control is less than 105 (condition #3) and the coolant flow level is **?vc** (condition #4), then the spindle control should be adjusted increasing in 5 units (action #5) and if the coolant flow level is less than 10 then it should be adjusted increasing in 5 unities (action #6).

The actions suggested by these rules are stored in facts as: **(action ?datum ?v1 ?v0 ?control ?value)** where **?datum** is the identifier of the datum in the state described by **?v1** and **?v0** that suggest the action, and **?control** is the control that must be adjusted in the amount indicated by **?value**.

While each datum suggests a single action by each of the controls, when the data set is considered, they can suggest several actions on the same control. These actions must be analyzed and combined to generate the effective action.

4.4 Combining the Actions

The effective action on a control is computed as a weighted sum of all the suggested actions on this control, where the weights of this sum depends on the highest level of the data generating actions on this control. The effective action is computed as follows:

First, we compute the highest level of the data generating actions on the same control, storing this information as **(effective-level ?control ?i)**. We also build a fact to store the partial weighted sum of the adjusting values suggested on this control: **(weighted-sum ?control 0 0)**.

```
  | (defrule effective-level
1 |   (action ? ? ?v1 ?control ?)
2 |   (not (action ? ? ?v0&:(< ?v1 ?v0) ?control ?))
3 |   (not (effective-level ?control ?))
  |   =>
4 |   (assert (effective-level ?control ?v1)
  |           (weighted-sum ?control 0 0)))
```

Next, we add up all the suggested actions on that control weighted according to a table of weights that depends on the effective level calculated in the previous stage. The function **(weight ?v1 ?v0 $?weights)** returns the weight corresponding to the state described by **?v1** and **?v0** within the weight vector **$?weights** obtained from the information about the weights stored as **(weights ?level $?weights)**.

```
  | (defrule weighted-sum
1 |   (effective-level ?control ?m)
2 |   (weights ?m $?weights)
3 |   ?h1 <- (action ? ?v1 ?v0 ?control ?v)
4 |   ?h2 <- (weighted-sum ?control ?n ?s)
  |   =>
5 |   (retract ?h1 ?h2)
6 |   (assert (weighted-sum ?control (+ ?n 1)
  |                          (+ ?s (* ?v (weight ?v1 ?v0 $?weights))))))
```

When the weighted sum is finished, the result must be divided by the number of actions and approximated in terms of the minimum increase of the control adjustment. The function **(aprox ?s ?n)** approaches the value **?s** in accordance with the minimum increase **?n**. The result is stored in a global variable through which communication is established with the interface. This is done for each control.

```
  | (defrule weighted-sum-coolant
1 |   ?h1 <- (effective-level coolant ?)
2 |   ?h2 <- (weighted-sum coolant ?n ?s)
3 |   (not (action ? ? ? coolant ?))
  |   =>
4 |   (retract ?h1 ?h2)
5 |   (bind ?*coolant* (aprox (/ ?s ?n) 5)))
```

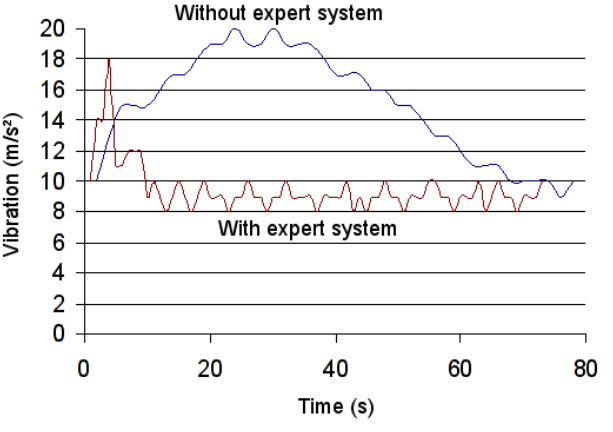
At this point the execution of the expert system ends, though preserving the data structures used to store the rules and their partial activations. This

is the starting point of the next run of the expert system, minimizing thus the number of checks to be carried out in each run. The global variables **?*feed***, **?*spindle*** and **?*coolant*** are read from the interface and are used to update the facts on the status of the controls at the beginning of the next run.

5 Experiments

The experiments developed consist in several XY rough cutting on a steel F114 block, with and without expert system, analyzing the behavior of the tool with respect to the temperature reached in the cutting tool and the vibrations in the spindle and the milling table. Let's see in more detail how they were developed and what results were obtained.

The test began with a vacuum machining (without material), for which the milling table vibration sensor gave values between 0.5 and 4 m/s^2 for spindle speeds between 500 and 3000 rpm. Machining steel without the expert system help, the vibration sensor detected values of 20 m/s^2 for machining process with 3000 rpm, 0.5 mm depth of cut and feed 200 mm/min. With these information it was determined the levels of the vibration sensor datum. The optimum vibration level that the expert system should maintain was defined between 6 and 9 m/s^2. Once these values were introduced into the system, the experiments were very positive, as it is show in the following figure:



As we can see, the expert system tries to maintain the machining vibration into the optimal interval (6-9 m/s^2), getting this aim most of the machining time. In this state, the expert system reduces to 2700 rpm spindle speed and to 180 mm/min the feed. These changes obviously increase the process time, although this consequence is assumed in the risk level considered in the process (quality job), where the quality of the result is more important than the processing time.

6 Conclusions and Further Works

We have successfully designed and tested an architecture for the optimization of a machining process in real time by means of an expert system based on

production rules. We have successfully achieved the goals outlined in the work: representation of the expert knowledge about the machining process by means of production rules; independent behaviour of the amount of input data provided to the expert system, allowing to activate or deactivate the sensors arranged in the machine or inserting new ones; and minimization of the number of changes between successive queries to the expert system, which reduces the response time.

The system can be deployed in any machining tool, whenever it is connected to a control device; can manage any information, internal from the device control or external from the sensors; and can act on serveral components of the machining process, internal as the feed or spindle speed or external as the coolant flow.

The next stage in the development of the expert system is the analysis of work sessions to adjust the rules that determine the actions associated with the data. From here there are some additional goals for the Sensor-IA system: the inclusion of new parameters in the system, such as type of material to be processed (steel, aluminum, etc.), the cutting tool, or new sensors; the automated correction of the rules from data stored about work sessions; or, in the longer term, the modification of the machining program.

References

1. Alique, J.R., Gajate, A., Novo, M.: Control adaptativo inteligente para la optimización de los procesos de fresado desatendido. CIC marGUNE, Centro de Investigación Cooperativa en Fabricación de Alto Rendimiento (2008)
2. Bohez, E.L.J., Thieravarut, M.: Expert system for diagnosing computer numerically controlled machines: a case-study. Computers in Industry 32(3), 233–248 (1997)
3. Cus, F., Zuperl, U., Milfelner, M.: Dynamic neural network approach for tool cutting force modeling of end milling operations. International Journal of General Systems 35(5), 603–618 (2006)
4. Forgy, C.: Rete: A Fast Algorithm for the Many Pattern/Many Objects Pattern Match Problem. Artificial Intelligence 19(1), 17–37 (1982)
5. Kuo, L., Yen, J.: Servo parameter tuning for a 5-axis machine center based upon GA rules. International Journal of Machine Tools and Manufacture 41(11), 1535–1550 (2001)
6. Park, S.K., Kim, S.H.: Artificial intelligence approaches to determination of CNC machining parameters in manufacturing: a review. Artificial Intelligence in Engineering 12(1), 127–134 (1998)
7. Riley, G.: CLIPS: A tool for building expert systems (2008), `http://clipsrules.sourceforge.net/`
8. Shachter, R.D.: Probabilistic inference and influence diagrams. Operations Research 36(4), 589–604 (1988)