An object-oriented architecture for sensorless cutting force feedback for CNC milling process monitoring and control

Roberto Augusto Gómez Loenzoa, Pedro Daniel Alaniz Lumbrerasa, René de Jesús Romero Troncosob, Gilberto Herrera Ruiz a,*

aFacultad de Ingeniería, Universidad Autónoma de Querétaro, Cerro de las campanas S/N, 76010 Querétaro, QRO, Mexico
bFIMEE, Universidad de Guanajuato, Tampico 912, Bellavista, 36730 Salamanca, GTO, Mexico

ARTICLE INFO

Article history:
Received 14 November 2008
Received in revised form 17 July 2009
Accepted 16 December 2009
Available online 21 January 2010

Keywords:
Intelligent Manufacturing System
Sensorless feedback
Process monitoring
Object oriented software design
Unified Modeling Language

ABSTRACT

Intelligent Manufacturing Systems require fault-tolerant components capable of preventing and detecting failures, and optimizing their own performance. Although there are a number of process monitoring and control systems (PMCS) available for these tasks, most of them require special equipment that increases cost and setup time. This work proposes a standard software architecture and methodology for implementing a sensorless PMCS which separates data representation from its algorithmic processing. Two different case studies are introduced showing the feasibility and applicability of the proposed design, one for cycle time optimization and another one for online tool condition detection. A good performance was obtained in both case studies based on the proposed PMCS architecture, which can be implemented using different distributed software technologies.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction

An Intelligent Manufacturing System (IMS) is a system whose main purpose is to automate and optimize one or more steps of the manufacturing process by means of an intelligent behavior—which is defined as a decision making process that mimics that of a human under the same circumstances. An autonomous component must be fault tolerant in order to minimize its interaction with other components of the system when minimal failures occur and must optimize the component performance in order to increase the overall efficiency of the whole system.

In the case of CNC processes there are diverse factors that are good candidates for optimization, such as cycle times, tool life and idle times. While there are factors such as idle times related to raw material availability, product demand and other plant management conditions, there is a class of process optimization problems, such as cycle time duration and tool life span, that depend on subtle relations between cutting speeds, depths and forces and the availability of proper algorithms to deal with these process variables.

There are several options at hand to obtain or compute these process variables, depending on the factor to optimize [1]. However, those methods requiring special instruments for measuring the process variables (e.g., dynamometers [2]) increase the cost of production due to the idle times involved during setup and the cost of the instrumentation process per se. For this reason, there are a number works proposing a less expensive sensorless approach; for example, this technique has been successfully applied to compute speed feedback for actuators [3,4] and to detect mechanical faults in electromechanical systems [5]. It has also been used as a means to detect tool damage under different conditions [6,7].

However, in spite this is an active research area, there is no standard framework or reference model available to try different algorithms and strategies, leading to repetition of much of the work.

This paper proposes a software reference model that can be applied to different hardware configurations and enables the researcher to test different algorithms without reimplementing most of the system. The proposed process monitoring framework is based on a sensorless approach and a PC as a data acquisition and processing system as depicted in Fig. 1. The reference model is properly described by means of the Unified Modeling Language (UML) [8] since its object oriented design (OOD) is based on common design patterns [9].

2. Sensorless intelligent control in milling

One of the main goals of an intelligent control unit is making the machining process more efficient by optimizing the cutting
conditions for the CNC system and detecting its failures. Cutting forces are one of the main concerns for milling processes since they determine power requirements—excessive cutting forces may cause high temperatures by friction or unstable vibrations, undermining the tool life; on the other hand, unnecessary low cutting forces usually increase the cycle time reducing overall productivity [10]. CNC programmers can select different cutting forces by setting the spindle speed $S$, the feedrate $F$ and the cutting depth $d$, but this requires machining expertise that takes time to learn [11]; in machining processes the tool geometry and the materials tend to change on a case by case basis, and the CNC programmer must take these factors into consideration along with the CNC machine special specifications. Most of the time the CNC programmer chooses to set conservative values to cope with different machine-tool architectures, relying on the CNC machine operator to possibly override these parameters manually during the machining process for cycle time reduction; however, this is not always the case.

Moreover, even when a part program has already been ‘fine tuned’, differences in the properties of materials used to produce different batches are a factor to reckon with, so even though offline [12] methods are available, online methods are preferred [13,14]. Thus, online methods for controlling the cutting force, in order to keep it within the bounds established by both the productivity requirements and the factors reducing tool life, are a great improvement for any machining process. In fact, online methods are also the best choice to detect tool wear and breakage timely, and this detection can take place at the same time the milling process is being optimized.

2.1. Sensorless cutting force measurement

During machining the required cutting forces depend on the material used to produce the part, the cutting depth $d$, the feedrate $F$ and the spindle speed $S$. The material used to machine the part and the cutting depth are fixed in the part requirements and the part program, respectively, so any chance to optimize the cutting force relies on the actual feedrate $F$ and the spindle speed $S$.

For milling applications, the spindle speed $S$ is usually programmed in revolutions per minute (rev/min), as an integer number. This parameter must be computed based on the peripheral (cutting) speed $V_C$ (m/min) recommended for the machined material and the cutting tool diameter $D$ [15]; it can be shown that the cutting force $F_C$ is directly proportional to $F/S$. Besides, the supplied spindle driver current $s$ is directly proportional to the spindle torque which in turn is inversely proportional to $S$ [16] when the depth of the cut $d$ is constant and the spindle entry and exit angles subtend an angle of 180°. Similarly, under these same hypotheses the supplied X-, Y- and Z-axis servoamplifiers current signals, $x$, $y$ and $z$, respectively, are directly proportional to the X-, Y- and Z-axis components of the feedrate $F$ [2], whence $F$ is directly proportional to $\sqrt{x^2 + y^2 + z^2}$. Thus,

$$\frac{F}{S} \propto s \sqrt{x^2 + y^2 + z^2}.$$ 

Now, by recalling that the supplied spindle driver current $s$ is directly proportional to the driver monitor output $s_{\text{mon}}$ and the supplied X-, Y- and Z-axis servoamplifiers current signals are directly proportional to the servoamplifiers monitor outputs $x_{\text{mon}}$, $y_{\text{mon}}$ and $z_{\text{mon}}$, respectively,

$$F_C \propto F/S \propto s_{\text{mon}} f_{\text{mon}},$$

where

$$f_{\text{mon}} = \sqrt{x_{\text{mon}}^2 + y_{\text{mon}}^2 + z_{\text{mon}}^2}.$$ 

There are three possibly concurrent scenarios when the assumptions used to deduce (1) do not hold:

1. The depth of the cut $d$ is non-constant, that is, it is varying and its nominal value can be larger or smaller than expected, causing the metal remove rate $Q$ to vary in the same proportion.
2. The spindle entry and exit angles subtend an angle less than 180°, that is, the width of the cut $w$ is smaller than expected, causing $Q$ to decrease.
3. The hardness of the material varies (locally) from its nominal value, causing the power at the spindle $P_S$ to increase.

In any case, $P_S$ varies and the cutting force $F_C$ oscillates as a consequence. Hence, having a constant $F_C$ control is equivalent to
having a constant \( P_5 \) control, which can be achieved by updating the feedrate \( F \).

### 2.2. Cycle time optimization

Notice that, once the material for the part has been chosen, the spindle speed \( S \) must be bounded according to information provided by machining data handbooks; in order to preserve tool life, increasing the spindle speed \( S \) beyond recommended limits is not an option, and does not have any good effect on reducing the cycle time. Moreover, decreasing the spindle speed on time in a controlled fashion may be unfeasible due to the spindle inertia. Hence, the spindle speed \( S \) shall remain constant during the whole cycle time. On the other hand, increasing the feedrate \( F \)—within recommended limits—does reduce cycle time. Besides, increasing the feedrate while keeping the spindle speed constant causes the same effect as decreasing the spindle speed and keeping the feedrate constant: hence, in practice, the there is no use of reducing the spindle speed.

Thus, once \( S \) has reached a constant value during the cycle time, it is the table feed \( F \), that is proportional to the feedrate \( F = F_n S \). This means, that if the programmed feedrate is overridden then the table feed \( F \) will be also overridden in the same proportion along with the cutting force \( F_C \). As illustrated in Fig. 2 from [10], the production rate depends both on the cycle time and the idle time—the cycle time decreases as the table feed \( F \) increases, but excessive feedrates can cause the tools to wear at an accelerated rate, reducing the production rate due to the idle times required for tool change and setup; notice that the unit cost decreases along with the cycle time, but if the tools wear at an accelerated rate the unit cost increases due to the tools and the setup costs. Thus, the cutting speed must lie in the range between the minimum unit cost and the maximum production rate.

### 2.3. Tool breakage detection

Sensorless tool breakage detection usually takes place by means of signal processing of the feed force monitoring signal (2), which is directly proportional to the feed cutting force \( f \); this signal contains all data related to the force exerted by each insert against the material. Inserts follow a trochoidal path generating periodic variations in the chip thickness—and thus in the cutting force \( f \)—every time the insert removes material, as shown in Fig. 3. The chip thickness \( 0 \leq h(\phi) \leq w \) varies as the insert enters and leaves the material, and a simplified model for this variation is given by \( h(\phi) = w \sin \phi \) for \( 0 \leq \phi \leq \pi \). The Cartesian components of the feed cutting force \( f_i \) for the \( i \)th insert can be computed from its radial \( f_{ri} \) and tangential \( f_{ti} \) component as

\[
\begin{align*}
  f_{xi} &= f_{ri} \cos \phi - f_{ti} \sin \phi, \\
  f_{yi} &= f_{ri} \sin \phi + f_{ti} \cos \phi,
\end{align*}
\]

for \( \phi_{ei} \leq \phi \leq \phi_{ex} \) where \( \phi_{ei} \) is the entry angle and \( \phi_{ex} \) is the exit angle, and \( f_{xi} = f_{yi} = 0 \) otherwise. Hence, the Cartesian components of the feed cutting force \( f \) for a tool with \( N \) inserts are given by,

\[
  f_x = \sum_{i=1}^{N} f_{xi}, \quad f_y = \sum_{i=1}^{N} f_{yi},
\]

whence

\[
  f = \sqrt{f_x^2 + f_y^2}.
\]

Thus, \( f \propto f_{mon} \).

### 3. Intelligent control architecture

From the previous section we can assert the data that a sensorless process monitoring and control system (PMCS) requires is provided by the monitor current signals \( x_{mon}, y_{mon}, z_{mon} \) and \( s_{mon} \). The PMCS also depends on the programmed \( F \) and \( S \) values. The monitor current signals are sampled directly from the milling process while the \( F \) and \( S \) values must be sent to the PMCS each time the milling machine controller updates these values or when the PMCS requests them.

#### 3.1. Deployment

Parameters \( F \) and \( S \) are sent from the CNC controller to the PMCS by means of a direct communication link (e.g., an RS-232-based link using the SLIP protocol), using a standardized format such as XDR; this same bidirectional data link is used to provide a recomm
mended feedrate override for cycle time optimization to the CNC controller and to warn about possible tool breakage conditions when they are detected. The monitor current signals $x_{mon}$, $y_{mon}$, $z_{mon}$ and $s_{mon}$ are sampled by the PMCS using a data acquisition (DA) card, as depicted in Fig. 4.

3.2. Architecture

Every Monitoring and Process Control System must perform three main tasks:

1. Sampling the process data.
2. Filtering the data.
3. Processing the data, that is, producing control signals.

3.3. Sampling

The main challenge for any PMCS that aims to provide a standardized interface is to provide a hardware-independent interface for gathering data samples. This involves dealing with two different issues:

1. Offering a sampling system whose interface is independent of the underlying hardware.
2. Implementing a data representation format whose interface remains the same without regard of the sampling system implementation, that is, that supports polymorphism.

A uniform interface for the sampling system, regardless of the underlying hardware, can be implemented by means of a Sampler abstract base class (also known as interface) by implementing a concrete derived class that deals with the particularities of the hardware; this can be done in two different ways, namely, by inheriting the interface from the Sampler class or by delegating the Sampler class functionality to an implementation SamplerImpl class. This second option is preferred because in the first situation, when interface inheritance is used, the implementation class memory footprint may vary, requiring a complete recompilation of the software in order to select different data acquisition hardware. Thus, by delegating the interface functionality to a SamplerImpl class and turning the Sampler class into a handler class, it is possible to provide a number of implementations even after the PMCS has been deployed (e.g., as dynamic linked libraries)—the memory footprint does not vary in this case because a handler class only requires a memory reference to the implementation class.

Now, since each data acquisition card uses its own format to represent the data set containing sample sequences from a given set of channels—i.e., the data storage is device dependent on the hardware and the way the hardware is configured—delegation is also used to provide both a common DataSet interface for any data set and a constant memory footprint; the implementation DataSetImpl for the DataSet interface, however, cannot be directly instantiated by the client code because that would require knowledge of the DataSetImpl memory layout, which is precisely what we are trying to hide from the rest of the system. Thus, the data set objects must be instantiated by an abstract factory, that is, a class that provides a DataSetFactory interface to create a DataSet and its delegate DataSetImpl for each milling machine channel. If the Sampler interface inherits from the DataSetFactory interface, the SamplerImpl

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{SamplerDesign.png}
\caption{Sampler design.}
\end{figure}
class can instantiate the data set and its implementation (see Fig. 5).

The data set implementation DataSetImpl can realize the DataSet interface in different ways, for example, as:

- **Vector** by storing in an internal buffer the channel data.
- **Iterator** by accessing a shared data buffer with channel data interleaved or divided by buffer segments, depending on the data acquisition card internal format. This pattern is preferred since no data is duplicated.

Thus, it is possible to implement a catalog of different sampler implementations without even modifying the DataSet interface.

### 3.4. Filtering

For the proposed framework we obtain four different data sets, namely:

- Spindle driver monitor current signal \(s_{\text{mon}}\).
- X axis servodriver monitor current signal \(x_{\text{mon}}\).
- Y axis servodriver monitor current signal \(y_{\text{mon}}\).
- Z axis servodriver monitor current signal \(z_{\text{mon}}\).

These signals, however contain undesired components in the form of high-frequency noise, current control commutation and ball-screw effects, so that a filtering process is required [6]. For example, a low-pass Butterworth filter could be used to extract the main components of \(x_{\text{mon}}, y_{\text{mon}}\) and \(z_{\text{mon}}\) [17] and a low-pass Savitzky-Golay data smoothing filter for \(s_{\text{mon}}\) [18]. Since there are a number of possibilities for filtering signals, depending on the situation at hand, the different filtering algorithms can be encapsulated in different classes that realize the same Filter interface but implement a different filtering strategy in the context of a SensorlessProcessor that gathers all the samples and filters the data; thus, the different strategies implement concrete filters such as Butterworth or SavitzkyGolay. Thus, the SensorlessProcessor class acts as a mediator between the data acquisition system and the filtering process, decoupling the data acquisition from its particular filtering (Figs. 6 and 7).

### 3.5. Processing

The acquireAndProcess method calls the process method defined in the Processor interface for every datum vector (one datum per channel) which is being sampled and filtered, thus allowing different Processor strategies to be implemented; the Processor interface must be realized by any class meant to implement process monitoring. In order to implement process control policies more information regarding the programmed feedrate and spindle speeds is required. Thus, a particular Kernel class implements the concrete Processor strategy and realizes at the same time the CuttingDataObserver interface is introduced, thus gathering all the required information to publish process monitoring and control data. There is the ToolObserver interface to provide tool condition data to any component performing tool condition decision making, along with the FeedrateOverrideObserver interface meant to offer information to any component implementing cycle time optimization; both observer interfaces are implemented by the Kernel class itself or by classes contained in the Kernel class by composition. In this case, the concrete observer CNCProxy implemented in the PMCS is a remote proxy component, which can be provided by a systems programmer or by a distributed computing framework such as CORBA or DCOM, or even an ad hoc distributed environment.

### 4. Experimental results

#### 4.1. Implementation

The experiments were conducted using a retrofitted 1.5 kW, 3-axis, Czechoslovakian Tos Kurim milling machine (Fig. 8) equipped with the CHROMA-II CNC open architecture control unit developed at the Mechatronics Laboratory at Universidad Autónoma de Querétaro and our experimental PMCS system over an RS-232-based SLIP link with an ad hoc packet format. The monitoring system was implemented on a Pentium-III PC using an Advantech PC-LabCard (PCL-818HG) for data acquisition and developed under a 16-bit DOS operating system using a C++ compiler. This data acquisition card has dual buffer storage, that is, while one buffer is being used by the card, the data in the other buffer is available for the user, which is signaled by means of a flag. The available buffer base address was used to implement DataSetImpl as an iterator, which is then used by the SensorlessProcessor to filter the signals in its acquireAndProcess method; the spindle signal was filtered using a Savitzky-Golay filter while the axes signals were filtered by means of a low-pass Butterworth filter; Fig. 9 shows the measured and filtered spindle driver monitor output current signal and Fig. 10 shows the measured and filtered Y-axis servoamplifier monitor output current signal. These signals, along with the cutting data provided by a CNCProxy class enable the system to monitor and control the milling process.
The results presented next are just examples of a way to use this framework to perform the respective tasks.

4.2. Cycle time optimization (face milling)

For face-milling cycle time optimization the implemented system calibrated itself by computing (1) as a reference with spindle entry and exit angles of 0° and 180°, respectively, and following the specifications given in [19] for the low-carbon 1018 steel with a constant (the steel part was end-milled prior to the test) depth of the cut of \( d = 2 \) mm a spindle speed of \( S = 800 \) rpm and a feedrate of \( F = 60 \) mm/min. The mean of criterion (1) was computed a number of times under these conditions, as shown in Figs. 11 and 12 and
the average of these measurements was used as a set-point \( r \); then, in order to optimize the cycle time and, at the same time, to protect the tool, a special-purpose controller was designed and implemented in the process method of the \( \text{Kernel} \) class to keep this cutting force set-point—by updating the feedrate override on the machine tool by means of the RS-232-based link—as follows:

1. A relative error is computed as \( e = (r - s_{\text{mon}})/r \).
2. If \( e < 0 \) then \( s_{\text{mon}} > r \), i.e., the cutting force exceeds the recommended cutting force, so that a proportional control action \( u = Pe < 0 \) is computed. If \( e > 0 \) then, a first order control action \( u \) is computed.
3. If \( U \) and \( L \) are the maximum allowed override above and below 100%, respectively, then the feedrate override \( F_{\text{ovr}} \) is computed by setting

\[
F_{\text{ovr}} = \begin{cases} 
(100 + U)\%, & 100u\% > U\%, \\
(100 - L)\%, & 100u\% \leq L\%, \\
100(1 + u)\%, & \text{otherwise}.
\end{cases}
\]

As a test to check the efficiency of the cycle time optimization method, a gap was end-milled on a block of low-carbon 1018 steel as shown in Fig. 13. The computed feedrate override for a path transversing the gap is show in Fig. 14. In this case, the cycle time was shortened up to 15%.

4.3. Tool breakage detection

For tool breakage detection the fast Daubechies wavelet transform was applied to signal (2) by using a \texttt{FastDaubechies} filter that realized the \texttt{Filter} interface; this wavelet transform requires \( N = 2^n \) samples for some integer \( n \) in order to perform the algorithm recursively on adjacent data sets. In this particular case 512...
samples per data set were taken \( (n = 9) \), that is, the DataSet method `getLength` returned 512, and the pattern these coefficients followed was analyzed to detect unusual changes in the frequencies after each revolution, evidencing unusual fluctuations in the cutting force. A neural network was used for wavelet coefficient classification, which provided a good classification performance. These results were thoroughly described in [20].

5. Conclusions

The proposed architecture is quite flexible allowing for specific and ad hoc systems to share the same structure and, thus, reducing the time for developing new implementations. This architecture is capable of accommodating different algorithms and internal data representations which can vary independently of one another. For example, it is possible to use the same algorithm even if the data representation has changed minimizing overall system changes required for portability. Similarly, no data representation changes are necessary to try different algorithms. The design accommodates different distributed technologies, i.e., it is possible to use a CORBA or DCOM based implementation or even a custom protocol as described in our case studies.

Acknowledgements

The authors would like to thank the Consejo Nacional de Ciencia y Tecnología (CONACyT) of Mexico and Universidad Autónoma de Querétaro for the invaluable support that made possible this research.

References