EMPLOYING REAL-TIME LINUX IN A TEST BENCH FOR ROTATING MICRO MECHANICAL DEVICES

Peter Wurmsdobler
Centre de Transfert des Microtechniques
39, avenue de l'observatoire, 25000 Besançon, FRANCE
peter.wurmsdobler@ctm-france.com

Abstract

This paper describes a testing stage based on RTLinux for characterising rotating micro mechanical devices in terms of their performance, quality and power consumption. In order to accomplish this, a kernel module employs several real time threads. One thread is used to control the speed of a master rotating up to 40000 rpm by means of an incremental coder and a PCI counter board with the corresponding interrupt service routine. Another thread controls the slave motor to be tested, synchronised to the coder impulses using voltage functions saved in shared memory. The measurement thread is then responsible to acquire date synchronously to the rotor angle and stuffs data on voltages, currents, torque and speed into different FIFOs. Finally, a watchdog thread supervises timing and wakes up a users space program if data have been put in the FIFOs. This GTK+ based graphical users space application prepares control information like voltage functions, processes data picked up from the FIFOs and displays results in figures.

1 Introduction

Rotating mechanical devices such as micro motors and micro actuators, or passive components like micro ball bearings and spiral springs can nowadays be found in a variety of consumer goods. Before they can be used and implemented, however, their performance and quality has to be assessed, already during development and later on, in the production line. In the framework of a European project, CTM (Centre de Transfert des Microtechniques), a research centre in the field of micro technology, has developed a testing stage for characterising such micro mechanical devices.

Generally speaking, characterising a micro motor or any electric motor is based on a profound understanding of the electro-mechanical system in terms of the energy flow and transformation. Depending on the motor principle different approaches have to be applied, e.g. a piezoelectric motor has to be supplied with sinusoidal voltages near its eigen frequency whereas the rotor speed will be a function of the voltage amplitude. The measurement principle for an asynchronous motor is different again, and for a DC or a synchronous motor, too. How can the torque-speed characteristic be determined for a variety of different motors?

The common sense approach will be to control the motor with the appropriate voltage and to use a brake to achieve a certain speed, i.e. the rotor speed is a result of the brake effort applied. Then the motor torque, speed and current can be measured, which yields the power consumption and hence the efficiency if the entire speed domain is scanned by changing the brake effort. This approach shows already that there must be a constitutive relation between speed and torque one of which can be fixed. The torque testing stage presented here employs the inverse principle to accomplish this characterisation task: it imposes a speed whereas the resulting torque depends on the the motor control. Here Real Time Linux comes into play, because this type of motor control has to be done in real time. Before the measurement principle is explained in detail, however, the test system used is presented.

2 Experimental setup

Figure 1 shows how the experimental setup looks like. The essential parts of the system, the computer, a rack containing control electronics, and the mechanical setup itself are shown.

FIGURE 1: *Micro torque testing stage with electronic rack (left), mechanical setup (middle) and computer.*

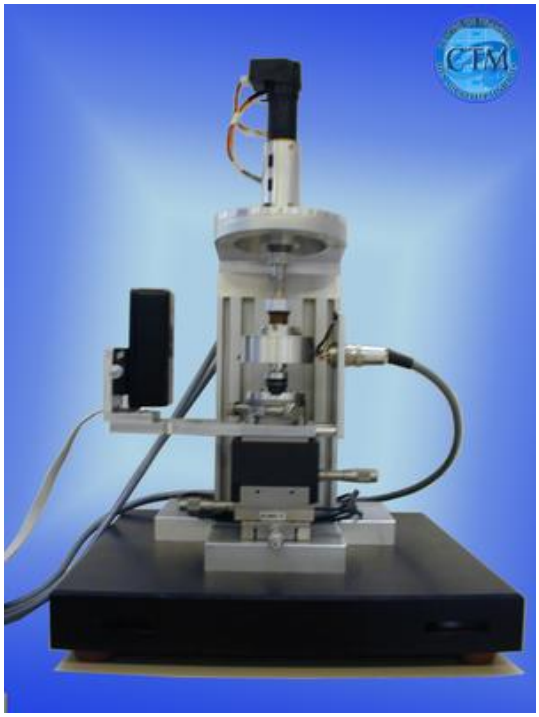

FIGURE 2: *The test rig with its vertical setup.*

2.1 Mechanical setup

Figure 2 gives a more precise idea of the mechanical setup. The micro mechanical device to be tested is held by a clamping device which is directly mounted on the torque sensor (a cylindrical shiny box). This torque sensor maps a torque of ± 50 μNm applied to it to a linear displacement by means of a linear silicon spring inside the shiny box and laser triangulation. This sensor is then mounted on another sensor, the ATI force sensor. The entire setup is fixed on a xyz translation table to make the correct alignment to the master motor on top of the setup. This master motor is equipped with an incremental coder of 360 impulses per revolution and drives the micro mechanical device using a special micro coupling being rigid for torsion, but elastic for all other degrees of freedom. In case of an active probe device, a connector for 3 channels is mounted on the right of the setup.

2.2 Electronic interface

All electronics necessary to drive the sensors and actuators is built into a rack. It contains the amplifier of the torque signal converting the laser spot displacement to ± 5V, an amplifier for the ATI torque sensor mapping forces to $0 \rightarrow 5$V. Additionally, the coil currents of a micro motor are measured mapping ± 1000 mA to ± 5V, or the induced voltages with a gain of 1. Furthermore, the rack comprises the power stage for the master motor mapping ± 5V to ± 4A as current setpoint, and the power stage for the micro motor with three channels of ± 12V output voltage mapped to a ± 5V setpoint. All signals are available on connectors on the rack.

2.3 Computer requirements

In order to output 4 signals, one for motor current setpoint and three for the coils of the micro motor, respectively, the ICPDAS 12bit output board pioda4 has been used with approximately 4.8 μs time

consumption per channel output. A ICPDAS pci-das1800 DAQ board has been employed to measure the torque, three currents, three voltages, the motor speed and the ATI forces at 12bit resolution and with 3 μs conversion time. Unfortunately, each time a single value is measured, the board has to be programmed with the channel and the channel's gain which takes approximately 30 μs. Without changing channels and gain getting a single value takes 5.6 μs. Concerning the computer, a P200 was used with 64MB RAM, a 3Com3C905 Ethernet adapter, Matrox Mystique PCI, and everything usually built in a desktop computer as mouse, etc. On the OS side, FSMLab's RTLinux `2.0` using kernel `2.2.10` has easily been compiled for this system, with `xsvga3.3.3` and `fvwm2` running as graphical user interface.

3 Measurement Principle

Figure 3 gives a simplified view of what is going on in the test bench. At a given rotor speed the incremental coder will produce TTL impulses (1) which are conveyed to the counter board (2). Note that the resulting frequency is 240 kHz at 40000 rpm for 360 impulses per revolution which is too much to be treated by the computer in real time, at least for the computer used. For this reason the counter board is programmed to count from a speed dependent value (later on called "down sampling factor") down to zero and triggers an interrupt afterwards (3). The computer measures the time between two such interrupts which is a measure for the rotor speed, and outputs the necessary master motor current (4) by the analog output board (5) and the amplifier which finally drives the master motor (6). This is the master motor speed control loop which guarantees a desired motor speed.
The measurement principle will then depend on the motor type. For example, a synchronous motor is usually driven by sinusoidal functions creating a rotating magnetic field which takes the permanent magnet with it. The angle between both, the so-called load angle, will determine the torque produced. If the computer outputs these voltage functions perfectly synchronised to the rotor angle during rotation (7,8,9), the load angle will remain constant. Thus, a load to the probe motor is simulated and this angle can simply be changed by changing an offset of a pointer to the functions to be output. At the same time the coil currents can be measured which together with the output voltage gives the power input into the motor.
In the case of an evaluation of a passive component which is driven by the master motor, a friction torque will make the sensor to output a signal proportional to the torque, too. In any case, the produced torque can be measured synchronously to the external coder impulses (10,11,12) which are equivalent to the rotor angle. The power losses and hence the efficiency can then be calculated.
It can easily be seen that this kind of testing approach is versatile and can be adapted to many types of motors and measurement, like the measurement of the EMF for synchronous motors. However, the concept needs to be implemented in a real time operating system in order to guarantee synchronisation and predictability. How this is done using RTLinux is explained in the next section.

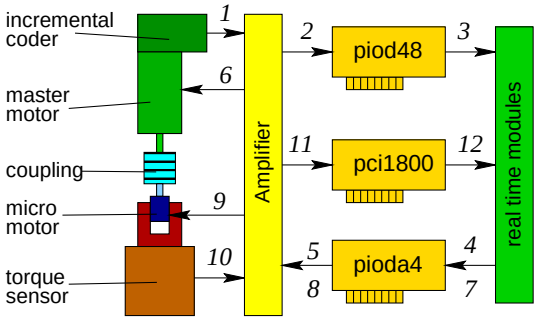

FIGURE 3: *The measurement principle.*

4 The test bench software implementation

The software for the torque testing stage is broken into two parts, a graphical interface as user space application, and a real time module responsible for measurement and control (Micro-Couple-Mètre = Micro torque testing stage):

- `xmcm`, the X-Windows Micro-Couple-Mètre interface,
- `rtl_mcm.o`, the RTLinux Micro-Couple-Mètre kernel module.

The logical structure from a software point of view is explained in Figure 4 with the user space application `xmcm`, the board specific objects, `rt_pci1800.o` for the ICPDAS pci1800L DAQ board, `rt_pioda4.o` for the ICPDAS pioda4 output board, `rt_piod48.o` for the ICPDAS piod48 counter board, and the real time measurement functions in `rt_mcm.o`.
Given that the kernel has been compiled with the real time option by FSMLab's RTLinux [1], four modules are insmoded, the shared memory module `mbuff.o` [2] for all data being output, `rtl_fifo.o` for FIFO buffers, `rtl_time.o` and `rtl_sched.o` for timing and scheduling, respectively.

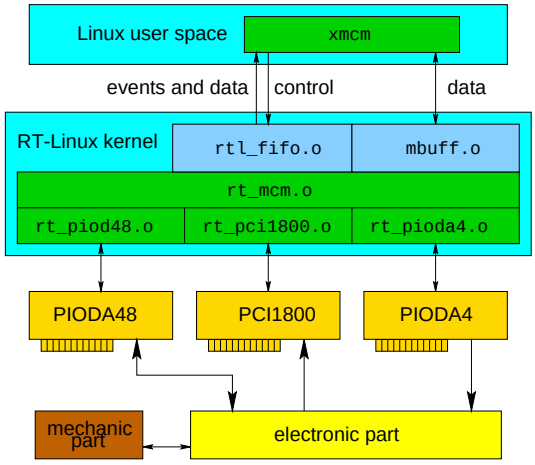

FIGURE 4: *Principle structure of the torque measurement system with flow of data and signals.*

4.1 Real time module - user space application interface

The only way the user space application can talk to the kernel modules is by means of device files, i.e. `/dev/mbuff` concerning shared memory and `/dev/rtf*` as for the FIFO buffers. In fact, it is the kernel module allocating these resources, and the user space application mapping it. Concerning shared memory data structures with its variables are defined in a header file for the use of both user and kernel space.
FIFOs are used for both messages and data. Messages defined by the appropriate header file can be passed to and from the real time module causing a message handler to be executed both in the real time module and the user application. Data acquired in the real time module are stuffed into the FIFOs and picked up from by the user space application.

4.2 Real time module

As the DAQ board related modules (`rt_pci1800.o`, `rt_piod48.o` and `rt_pioda4.o`) are concerned, the basic functions being called from the measurement object `rt_mcm.o` are:

```
extern void rt_pci1800_aget(
    unsigned short int channel,
    unsigned short int *value );
extern void rt_pioda4_aset(
    unsigned short int channel,
    unsigned short int value );
extern void rt_piod48_bitset(
    unsigned short int channel,
    unsigned short int value );
```

Based upon these functions, `rt_mcm.o` is responsible for the data and control flow by employing four threads and an interrupt service routine with descending priority:

- `isr`, the interrupt service routine calculates the time difference between two interrupts used by the probe motor control thread and wakes up some threads if necessary, especially the probe motor control thread for a synchronous motor.
- `watchdog`, a watchdog to check "down sampling factor", update certain values and send `UPDATE` to the user space if a certain number of values are in the FIFOs.
- `pm_control`, the probe motor control thread which outputs voltage functions either woken up by the interrupt service routine or scheduled in order to generate a rotating field of a given speed at a certain sample frequency in the case of an asynchronous motor. Additionally, it measures voltages and currents if desired by the user space application.
- `mm_control`, the master motor control thread running at constant sampling time for open and closed loop master motor speed or brake control. If wanted for display, the speed is put into the corresponding FIFO.
- `measure`, the measurement thread woken up by some instance if timing can afford it, i.e. if there is sufficient time available. This thread will stuff torque and force values into the corresponding FIFOs.

If the module receives a `START` message, the message handler will prepare all initial data, start the necessary threads and enable the interrupt on the coder interrupt line. The combination and interaction of all threads depends on the master motor, the probe type which can also be a either an active or a passive component, the signals to be measured or output, like torque, forces or induced voltages and currents. For the synchronous motor mentioned above Fig. 5 explains the necessity of synchronisation and real time interrupt treatment. The CPU is loaded either with the interrupt service routine waking up the probe motor thread `pm_control`, or the measurement task `measure`, or if there is some time left with other kernel or user tasks, `OTHER`. With RTLinux, the treatment of the interrupts for staying synchronised can be guaranteed to high frequencies, even at high load or if some measurement is going on. This is necessary in order to output the driving signal for the micro motor synchronously.

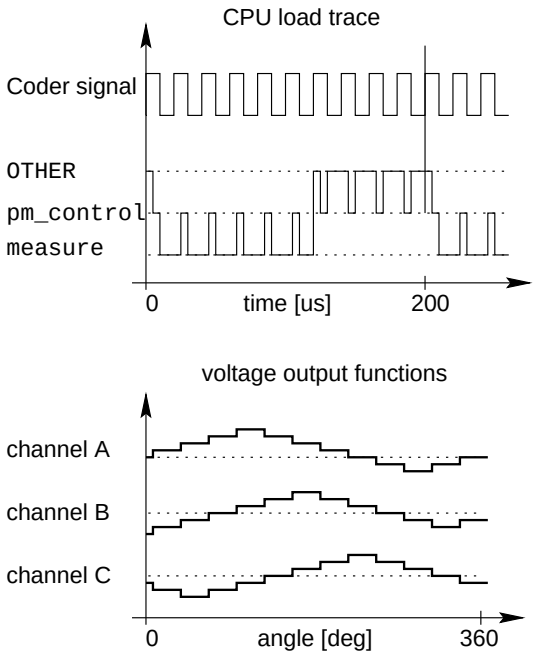

FIGURE 5: *Principle of measurement and output synchronisation, in terms of time (top) and angle (bottom).*

4.3 User space application

As the visible main application, `xmcm` (Fig. 6) provides in general a main window with some subwindows showing the inputs and outputs of the micro torque testing stage, some control buttons in order to start and stop measurement and to start and stop calibrating, and some text fields in order to enter rotation speed, select motor types etc. This X-Windows interface is programmed using the GTK+ widget set library [3] in combination with a scientific plot widget [4]. In addition a widget for displaying scientific style figures has been programmed for all inputs and outputs, like voltage functions (Fig. 7). All language specific texts are defined in a separate header file such that other languages can be compiled easily (necessary, because this is a European project funded by the European government).

At start-up this applications maps the shared memory in its space, and opens the FIFO buffers. After this, all GTK+ widgets are created, like the start button or the figures for the torque to be measured. The user can now calibrate the sensors, enter some values like the voltage amplitude or the speed setpoint, and select the channels he is interested in, torque, currents or more. If the user then presses the start button, some values are calculated and put into shared memory. Afterwards, a `START` is sent to the real time module and the user application is waiting.

The watchdog of the real time module sends a message to the non real time process by an event FIFO saying `UPDATE`. From this point, a routine gets the values form the FIFOs, converts them to physical values and calculates some mean values for a given speed. These values are appended to a already measured values and displayed on the screen in "human scale real time", i.e. the time constraints or deadlines imposed by the human eye are met.

If the user presses the stop button, a stop message `STOP` is sent. This message will be received by the real time module and the appropriate function will stop interrupt generation an disable motors and the like. Then the user can print the deserved torque versus speed plots.

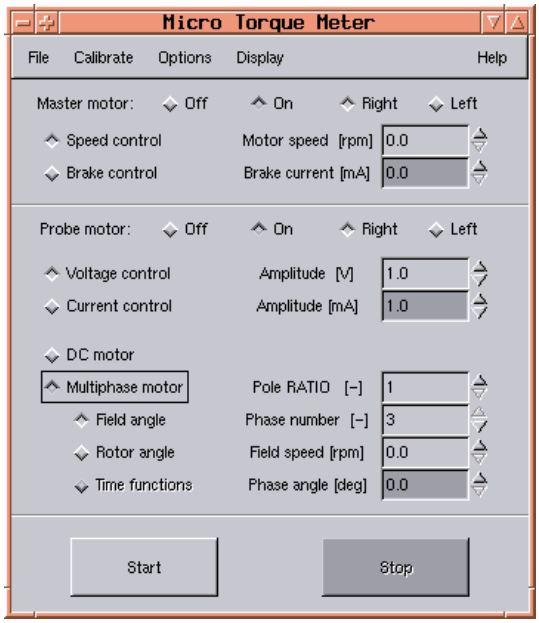

FIGURE 6: *Test bench user interface.*

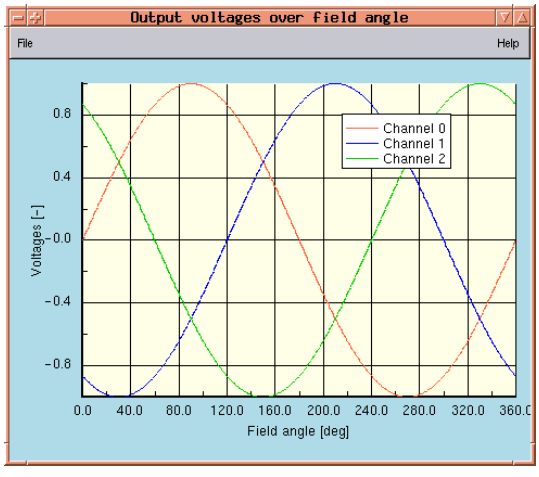

FIGURE 7: *Voltage functions.*

5 Experimental results

Some tests for assessing the performance of all threads and the interrupt service routine have been carried out before any real measurements. For example, a signal generator with a square wave of 5 V was used to simulate coder impulses. Using a FIFO for passing the time differences to user space for debugging, the maximum interrupt frequency was approximately 50 kHz, i.e. 20 μs. In terms of rotor speed, this would be 8333 rpm for 360 points per revolution, or the necessary down sampling factor of approximately 5 which means 72 points for one revolution for the maximum speed of 40000 rpm. Of course, taking into account the measurement time per channel, the number of measured values will decrease depending on the number of channels selected.
Another test was performed with creating a rotating field using three sinusoidal voltage functions with 360 interpolation points per revolution. Running the simple probe motor thread without any measurement, the CPU could schedule this task at 20 kHz, i.e. 50 μs. Using no down sampling, this yields 55.5 Hz or 3333 rpm. In order to obtain the maximum speed of 40000 rpm, a down-sampling factor of 12 is needed, i.e. 30 interpolation points per revolution. Since the time for one single output is approximately 5 μs, this time delay could be seen between the channels on the scope.
Finally, for the use in real measurements, the sampling frequency for the watchdog thread was set to be 10 Hz, the master motor control frequency to 200 Hz.

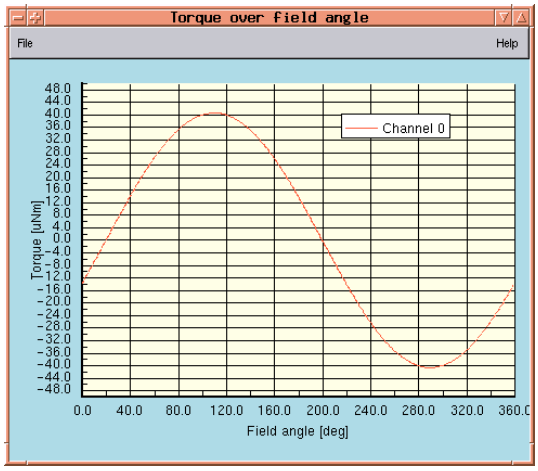

FIGURE 8: *Measured torque over field angle for speed equal to zero.*

As an example for a measurment, the test bench has been used for testing a synchronous motor contributed by a project partner within the HAFAM project. The setup as described above creates a rotating field at 60 rpm, but controls the rotor speed to be zero. With a typical tri-phase voltage functions of 1 V amplitude applied to it, the torque depending on the phase angle between permanent magnet and field vector can be seen in Fig. 8 for a rotor speed equal to zero.

6 Conclusion

Rotating micro mechanical devices as they are widely used in consumer goods have to be evaluated, a task which can easily be carried out using common proprietary software for simple types of measurement, because real time is done by the the DAQ board. If more sophisticated measurement techniques are used, however, e.g. accurate synchronisation of input and output and single measurements on demand, then RTLinux constitutes a good and reliable means to accomplish the task. Especially, programming time critical control and measurement tasks being very close to computer hardware as small and simple threads running on top of a real time executive, makes the entire application predictable and transparent. The measurement software shown here uses four small of these thread with very little code. Most data pre and post processing can be deferred into user space.

Acknowledgements

This work has been carried out within the HAFAM project (Handling and Assembly of Functionally Adapted Microcomponents), funded by the European Government in the Research Network of Training and Mobility for Young Researchers, contract NR ERB FMRX-CT97-0141.

References

[1] RTLinux, FSMLab's Real Time Linux.
`http://www.rtlinux.org`,
`http://www.fsmlabs.com/`

[2] mbuff, the shared memory module.
`http://crds.chemie.unibas.ch/`

[3] GTK+, the graphical toolkit.
`http://www.www.gtk.org/`

[4] GTKplot, a widget for scientific plots.
`http://www.ifir.edu.ar/grupos/gtk/`