Flexible application development and high-performance motion control based on external sensing and reconfiguration of ABB industrial robot controllers

Blomdell, Anders; Dressler, Isolde; Nilsson, Klas; Robertsson, Anders

2010

Flexible Application Development and High-performance Motion Control Based on External Sensing and Reconfiguration of ABB Industrial Robot Controllers

Anders Blomdell, Isolde Dressler, Klas Nilsson and Anders Robertsson

***Abstract*— New robot applications increasingly require external sensing to accomplish robustness and performance despite the variations and uncertainties that come with the increasing demands on flexibility for manufacturing of customized products. The demands on productivity then require high performance, which for well known feedback-control reasons means short response times from external process event to reaction on the robot motor control. The need for high-performance sensor interfaces connected to the motion control without excessive buffering and computations has therefore gained increasing attention over the last decade.**

Apart from high sampling frequencies and short latencies, there are also engineering aspects to consider. Control engineering tools should be possible to utilize, additional computing power is better connected via real-time Ethernet connections, and all interfaces should check compatibility without sacrificing performance. Furthermore, all that should preferably coexist with the original system, superimposing the effects on external sensing while still maintaining safety and consistency of data. The reason is that the original control software is model based and built on millions of lines of C-code, and tested beyond the possibilities of a research lab.

Such a system has been accomplished for control and application development using ABB robots. A short description of the system is given, and its use for application/control development is exemplified. User specific feedback control can be done easily with short sampling periods and less than 1ms of latency from sensing to motor reaction.

I. Introduction

As explained in [1], most robot control systems of today support some type of user IOs connected on local networks or buses. A crucial issue is the achievable bandwidth for the control loops with the external feedback. For many applications the effect of the bandwidth limitations only show up as longer duty cycles, whereas for some applications, like contact force control between the robot and the environment/workpiece, stability problems and severe performance degradation may result [2],[3],[4].

From a control perspective of robotics research, direct access to the driving torques of the manipulator and fast feedback is very valuable, or even crucial for algorithm evaluation and implementation of high-performance control schemes. This made early robot systems like PUMA 560 popular. Unfortunately, this kind of low-level access is not present in commercial robot control systems of today.

For demanding new applications we both need to close fast feedback loops at a low level, and to do so in a consistent way, supporting supervision and coordination with the application oriented programming on higher hierarchical levels. Therefore, alternative ways to obtain high-bandwidth control based on external sensors, which maintain the existing supervision and coordination functionality, are necessary [5], [6].

II. Network communication

In a research lab with rapid application development control modules are often changed without properly changing all dependent modules. Therefore it has been considered important to connect the different parts of experiments in such a way that changes interfaces are detected and precludes operation with incompatible versions of software.

To handle this versioning, we use a binary protocol (LabComm) [7] where a data source starts with a one-time transmission of an indexed list with the layout of all possible subsequent samples. This list is then used by the data sink to verify that all possible samples has a layout which is identical to that of the data source. All data actually sent is then prefixed by the proper index into this list, followed by a binary representation of the sample data. LabComm is capable of sending samples consisting of primitive types as well as fixed and variable size arrays and arbitrarily nested structures. In contrast to C, LabComm supports variable sized arrays and in contrast to Java it supports multidimensional arrays (Java only supports arrays of arrays).

Data which is to be sent by LabComm is described by a simple Data Description Language, which is used to generate the proper marshalling/demarshalling code for a number of languages (currently C, Java, Python and C#).

Part of a description file could look like

```
sample float fswitch;
sample float fromFile[3];
sample float heidenhain;
```

where `sample` refers to a snapshot of an object ("an instance of an instance of a type", in contrast to remote objects).

An important design feature of the LabComm protocol, is that only one-way communication is needed, which means that it is possible to store a LabComm data stream to a file for later interpretation. This is furthermore used for general

Anders Blomdell, Isolde Dressler, and Anders Robertsson are with the Department of Automatic Control, LTH, Lund University, SE-22100 Lund, Sweden, mailto:anders.robertsson@control.lth.se

Klas Nilsson is with the Department of Computer Science, LTH, Lund University, SE-22100 Lund, Sweden, mailto:klas.nilsson@cs.lth.se

data logging and logging can be implemented in separate log program (or used by an online data viewer) and need thus not be integrated with the other applications.

To guarantee that each sensor data network package arrives on time to the low-level motion control, each communication node obeys a data bandwidth limitation that we refer to as *throttling* [8].

A. Orcinus/ORCA

Built on top of the LabComm protocol we have a two-way protocol nicknamed ORCA[1] that divides samples into the four classes *inputs*, *outputs*, *parameters* and *log signals*. Parameters can be gains or tuning parameters in the low-level controller which can be changed via an operator interface.

III. Simulink → RTW → ORCA

Due to historical reasons, we use Matlab/Simulink Real Time Workshop (RTW) for code-generation for our robot controllers, but unfortunately the code generated by these tools is not very well suited for direct integration with other software, since the model input/output structure is only exposed as C-code structs with name-mangled fields. In order to remedy this flaw, we have developed a tool **`rtw2orca`** that uses an undocumented temporary file (*modelname*.rtw) and the *.lc*-files given as build parameters to generate a main program that properly handles marshalling/demarshalling of LabComm data as well as running the code generated by RTW in an orderly manner. Since we use signal names in the Simulink model and match them with the declarations in the *.lc*-files, we get a reasonably good typechecking of the data sent to and from the generated code.

As an example, we can look at the internal position reference signal generated by the ABB controller. This is fed into a shared memory ORCA connection which is described by a *.lc*-file containing the following lines:

```
typedef struct {
  ...
  double posRef;
  float  velRef;
  ...
} irb2ext_joint;
sample irb2ext_joint irb2ext[6];
```

If we want to use the vector of posRef signals, we just put an input signal named `irb2ext[i].posRef` in the Simulink diagram (if we had wanted the second element of the vector, we could have used `irb2ext[2].posRef` instead). When the `rtw2orca`-tool then parses the following from the totally undocumented .rtw file (internal representation from MathWorks):

```
ExternalInput {
  BlockName   "<Root>/irb2ext[i].posRef"
  Identifier  "irb2extiposRef"
  TID         0
  CGTypeIdx   14
  VarGroupIdx  [4, 13]
}
```

the tool infers that the Simulink diagram contains an input named `irb2ext[i].posRef` and uses this together with the .lc file to generate the following C-code:

```
for (i = 0 ; i < 6 ; i++) {
  (*U).irb2extiposRef[i] =
    controller->v_irb2ext.a[i].posRef;
}
```

and hence a name-matched connection with less dependence on Simulink/RTW versions is accomplished.

Before we had the `rtw2orca`-tool, all Simulink models had a common structure, where inputs and outputs were coupled to (almost) arbitrary signal names, which was both hard to understand and maintain, and hence very error prone.

IV. Usage of the Extended Control and Sensor Interface

Before an experiment can be carried out, the controller needs to be implemented in Matlab/Simulink and eventually, drive routines need to be programmed for additional input/output signals of the controller, i.e. signals other than the standard input/output exchanged with the ABB IRC 5 controller.

For the Simulink controller, any commonly used blocks are supported by the Matlab RTW C-code generation. S-functions or embedded Matlab code blocks can be used for implementing complex relations, e.g. forward or inverse kinematic functions. Whenever possible, the controller should be tested in simulations (with a model of the robot) before applying it to the real robot. Parameters (e.g. a switch parameter to turn the controller on or to start an experiment) can be changed from the user interface during the experiment if they are declared as *inline parameter* in the Matlab/Simulink RTW options. Any signal in the controller block diagram can be logged by ticking it as *test point*. After making sure that the LabComm files required for the inputs/outputs are available, the Simulink controller can be translated to C-code using Matlab RTW and downloaded to the G4 Power PC.

If input or output signals have been added to the Simulink controller, a driver routine is needed to receive/send the data using the LabComm protocol. Typically, additional inputs are added to read data from an external sensor.

Fig. 3 shows the overall structure of the extended control system and sensor interface. Ongoing work withing the EU-FP7 project ROSETTA aims at further developing this architecture to a network based extension with even faster update rates in the low-level control architecture.

[1]Since other platforms also use that name, our original platform name ORCA (Open Robot Control Architecture) has been replaced by the Latin first name of the orca animal; Orcinus. ORCA is still used for symbols and in names of tools, as for the protocol in this case. Orcinus stands for Open Robot Control using Integrated Networked Ubiquitous Sensing.

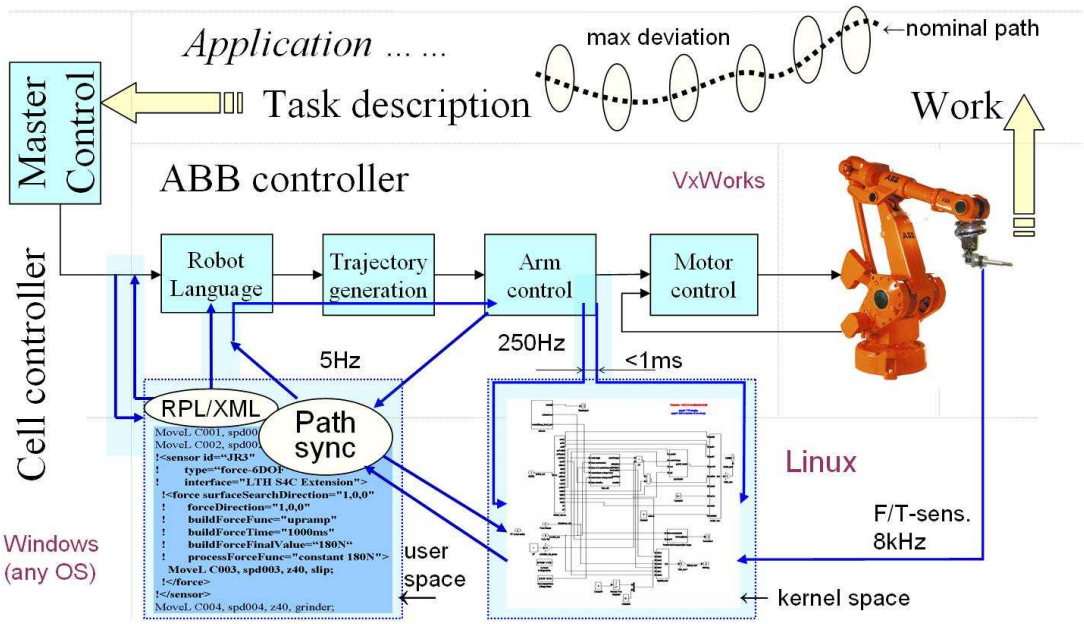

Fig. 1. Extension of industrial robot controller with sensor interface and support for external computations and synchronization, using a Motorola PPC-G4 PrPMC-800 processor board mounted on an Alpha-Data PMC-to-PCI carrier board with a local PCI bus.

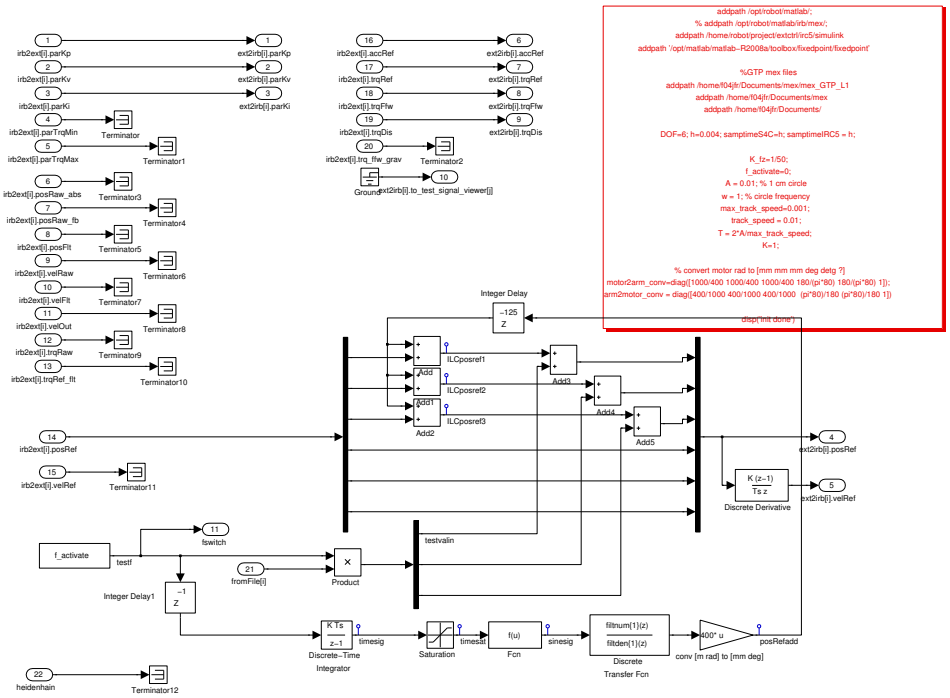

Fig. 2. Example: Simulink model for iterative learning control (ILC): The model contains the standard input and output signals from and to the IRC5 system and an input from an external linear encoder (`heidenhain`). The reference signal is generated from the user interface input `f_activate`, and the correction terms are read from a file, generated by the ILC-algorithm.

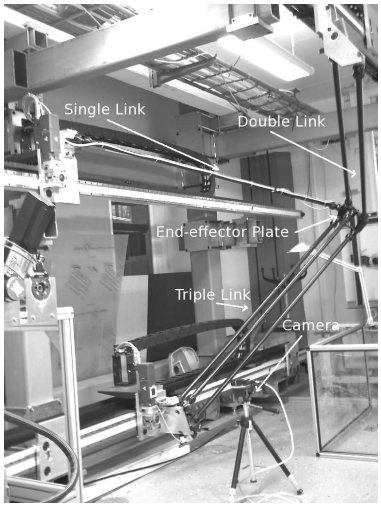

Fig. 5. The prototype of a Gantry-Tau PKM at the Robotics Lab at LTH, Lund University; A camera and a high resolution linear encoder along the rail is used to measure the cart motion on the arm-side.

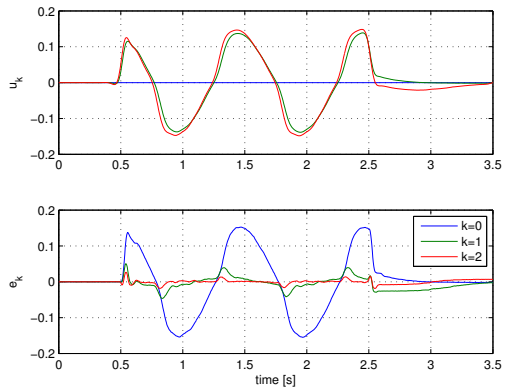

Fig. 6. ILC-experiments: Sequences of signals added to position reference (*upper*) and position error (*lower*) for each iteration $k = 0, 1, 2$. Note that already after one iteration the tracking error is significantly reduced.

The opcom user interface (Fig. 4) communicates with the G4 Power PC and the IRC 5 system. The opcom interface is implemented as a finite state machine (FSM). The Simulink controller can be loaded from it (*load* and *unload* state). In the *submit* state, is starts to receive the data the IRC 5 main controller sends to the axis controllers, but only when activating *obtain*, the axis controller receives the modified data from the Simulink controller instead of the data from the main controller. The communication with drive routines or the logging routine is started separately from a PC. In principle, each drive routine as well as the opcom interface can run on a different PC which has the appropriate hardware.

The controller should be tested in the *submit* state before running the experiment to ensure that the controller is stable and safe to use.

V. Application Example: Iterative Learning Control

Below, the presented controller extension is illustrated by the application of iterative learning control (ILC) for a parallel kinematic robot, see Fig. 5.

Even if this does not fully reflect the possibility to add complex controllers including e.g., stateflow diagrams for mode changes it captures the main communication flow and relates to the described modularity of the system.

The robot is to "learn" a specific motion by decreasing the deviation from the reference trajectory with every iteration in which the movement is performed. The position references of the robot joints are updated with a correcting term calculated from the deviation from the reference trajectory between the iterations.

Figure 2 shows the Matlab/Simulink implementation for iterative learning control. There are several features necessary for that purpose:

- The position and velocity references of the joints should be updated with the reference trajectory (still without the correction terms).
- The updated correction terms for the current iteration should be added to the reference values sent to the robot system.
- Additional sensors, in this example a linear encoder, to measure position at the arm side, can be included.
- It should be possible for the user to manually start an ILC experiment in a synchronized way via a switch or similar.

The switch (`f_activate`) is implemented as a parameter that can be changed and the experiment thus started from the opcom user interface. The reference trajectory and correction terms are synchronized with this parameter.

For the reference trajectory, here a filtered sine signal was chosen. The synchronization with the `f_activate` switch is here done by integrating `f_activate`, which is set to 1 to start the experiment. This creates a time signals `t` which is fed into a `sin(t)` block. A saturation block on the time signal terminates the sine after an appropriate number of cycles. A more flexible way would be to read the reference from a file, as described below for the correction terms.

After calculating the correction terms, they are written to a file. A C-program was written that uses the LabComm protocol to connect to the orca client. It receives the `f_activate`/ `f_switch` signal, and when it switches to 1, it opens the specified file and starts to send the correction values to the orca client with the same sampling interval it receives `f_activate` from the orca client.

Any additional sensor, such as the high-resolution linear encoder (1 μm) in this example, can communicate with the orca client in a similar way. A C-program sets up the connection and sends the measurement data to the orca client. To ensure equal sampling intervals, the sending can be synchronized with data received from the orca client.

Finally, Fig. 6 shows details of the ILC experiments.

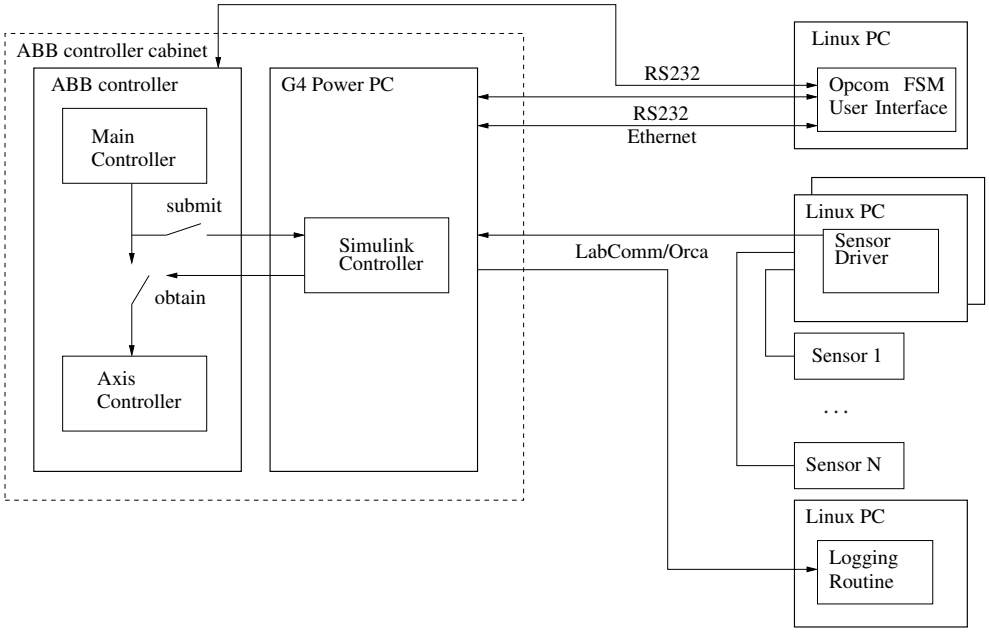

Fig. 3. Structure of the extended ABB IRC 5 control system.

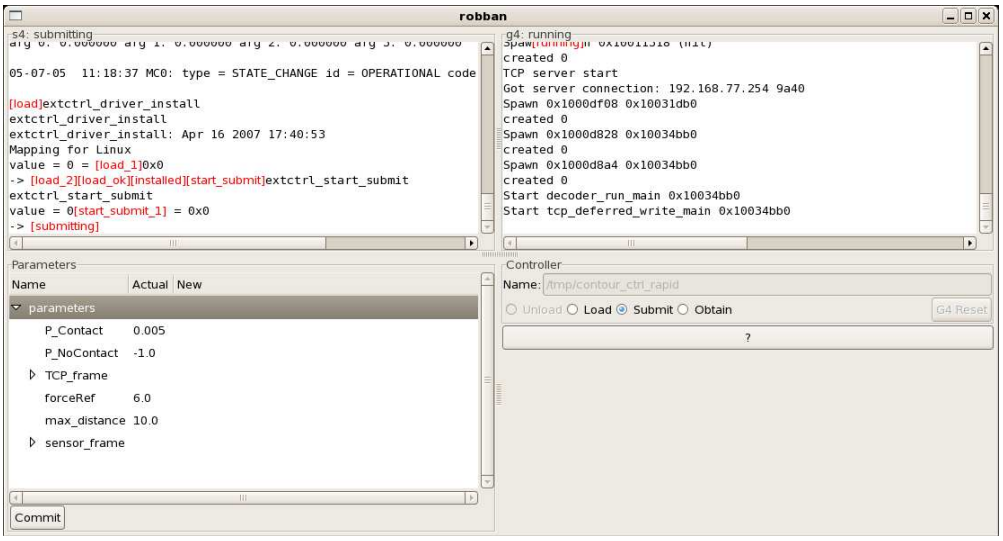

Fig. 4. Screenshot of Opcom interface: Connection to IRC 5 (top left), G4 Power PC (*top right*), parameter interface (bottom left) and controller interface (*bottom right*).

REFERENCES

[1] A. Blomdell, G. Bolmsjo, T. Brogardh, P. Cederberg, M. Isaksson, R. Johansson, M. Haage, K. Nilsson, M. Olsson, T. Olsson, A. Robertsson, and J. Wang, "Extending an industrial robot controller: implementation and applications of a fast open sensor interface," *IEEE Robotics & Automation Magazine*, vol. 12, no. 3, pp. 85–94, 2005.

[2] B. Siciliano and L. Villani, *Robot Force Control*. Kluwer Academic Publishers, 1999.

[3] D. Gorinevsky, A. Formalsky, and A. Schneider, *Force Control of Robotic Systems*. CRC Press, 1997.

[4] T. Yoshikawa, "Force control of robot manipulators," in *Proceedings of IEEE Int. Conf. on Robotics and Automation*, April 2000, pp. 220–226.

[5] A. Robertsson, T. Olsson, R. Johansson, A. Blomdell, K. Nilsson, M. Haage, B. Lauwers, H. de Baerdemaeker, T. Brogårdh, and H. Brantmark, "Implementation of industrial robot force control - case study: High power stub grinding and deburring," in *Proc. 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2006)*, Beijing, China, Oct. 2006, pp. 2743-2748.

[6] T. Olsson, M. Haage, H. Kihlman, R. Johansson, K. Nilsson, A. Robertsson, M. Björkman, R. Isaksson, G. Ossbahr, and T. Brogårdh, "Cost-efficient drilling using industrial robots with high-bandwidth force feedback," *Robotics and Computer-Integrated Manufacturing*, vol. 26, pp. 24–38, Jan. 2010.

[7] LabComm, http://torvalds.cs.lth.se/moin/LabComm.

[8] A. Martinsson, "Scheduling of real-time traffic in a switched ethernet network," Department of Automatic Control, Lund University, Sweden, Master's Thesis ISRN LUTFD2/TFRT--5683--SE, Mar. 2002.