Automatic Model Identification of ABB Industrial Robots

G.E. Hovland[*], S. Hansen[+], O.J. Sørdalen[*], T. Brogårdh[+], S. Moberg[+] and M. Isaksson[+]

[+]ABB Robotics, Västerås, Sweden
[*]ABB Corporate Research, Oslo, Norway

Abstract

In this paper we present an automatic model identification tool of ABB's industrial robots. The tool has been developed entirely in Matlab and has made extensive use of the graphical-user-interface options and the object-oriented design philosophy made possible in Matlab versions 5.0 and newer. The tool is able to automatically identify rigid-body robot parameters such as inertias and Coriolis forces and flexible robot parameters such as joint stiffness and damping. The identified parameters are fed back into the robot controller to achieve a near optimal path following performance of the robot's tool.

1. Introduction

Automatic model identification of ABB's industrial robots is important for several reasons. First, the path following performance of the robots tool is directly linked to the quality and accuracy of the dynamic model used inside the real-time robot controller. The path tracking performance of industrial robots is one of the most important sales arguments and often customers compare directly ABB's path following performance with the competitors before a major order is decided. Second, in the past, a large amount of man-hours has been put into the manual model identification. By making the identification process automatic, significant savings are achieved in the development process of new robot products and the time-to-market for new products is reduced.

In this paper we present the identification tool used at ABB for the automatic model identification. The tool has been written entirely in Matlab using the graphical-user-interface options and the object-oriented design philosophy made available in Matlab 5.0 and newer versions. The Matlab tool is able to communicate directly with ABB's real-time controller to exchange raw data measurements with no manual interference. The raw data is analyzed in an external PC running Matlab. The PC and the robot controller communicate directly through an Ethernet connection. In our design process, Matlab was chosen instead of other development environments because of the extensive collection of signal-processing functions and the short development time from new ideas to working Matlab code.

Figure 1 shows a 6-degree-of-freedom industrial robot from ABB Robotics. The automatic identification tool is able to automatically identify several different parameter types from such robots. Examples of identified parameters are gravity forces, inertia terms, Coriolis forces, static and viscous friction and motor joint stiffness and damping for all 6 motors and links. The operator input and the final identification results are made available to the end-user by Matlab's graphical user interface.

Figure 1: Industrial robot from ABB Robotics.

In the following sections we present the user interface,

the object-oriented design of the identification tool, the communication link between Matlab and the robot controller and we briefly describe the identification methods. Finally, we discuss some of the advantages of the Matlab solution and a new development option using ActiveX technology for the communication link between the PC and the robot made available in the Matlab 5.2 release.

2. Using Matlab's Graphical User Interface

The layout of the menu system of the identification tool is shown in Figure 2.

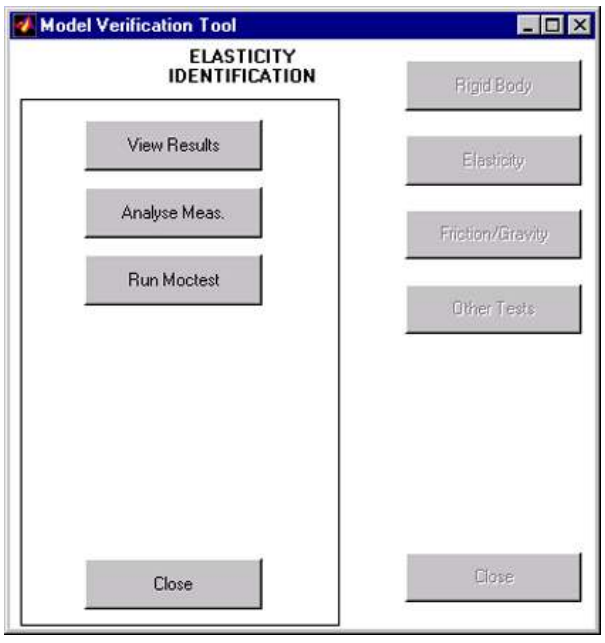

Figure 2: Menu system in identification tool.

On the right-hand side of the menu in Figure 2, the general options are given. The user has available several different identification methods, including rigid-body identification, joint elasticity identification, friction/ gravity identification and other tests. Two of these identification methods will be discussed in more detail in sections 5 and 6.

On the left-hand side of the menu system, a new set of options appear after the user has selected one of the four identification methods on the right-hand side. In Figure 2, the left-hand side appeared after the user selected the Elasticity identification method. By selecting "Run Moctest", Matlab will generate a robot motion program and communicate with the robot controller as described in section 4. When the robot has finished executing the motion program, raw data measurements will be saved on the PC's hard disk. By selecting "Analyse Measurements", Matlab will process the raw data measurements using mainly the signal processing toolbox and our own identification functions. By selecting "View Results", the final identification result will be presented to the user. An example of a final identification result is given in Figure 3.

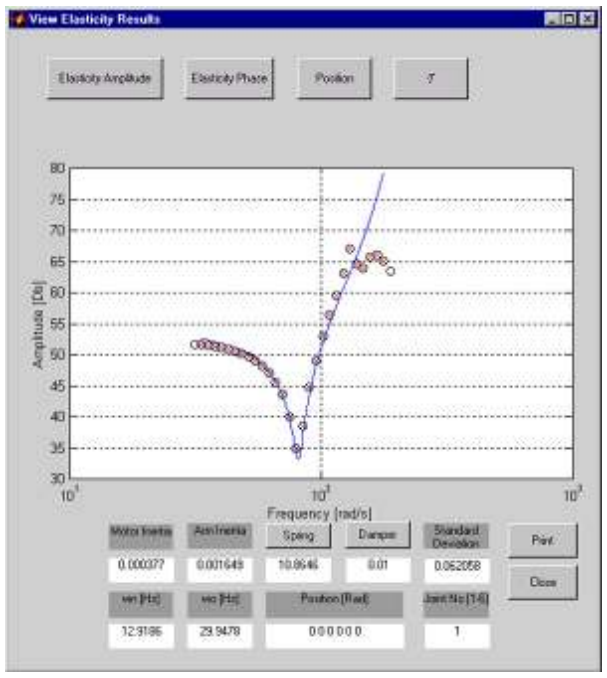

Figure 3: Example of final identification result.

In Figure 3 the sinusoidal frequency response of one of the robot's 6 motors is shown. When the user selects "Run Moctest", the robot controller commands a series of sinusoidal motions with different frequencies. The motor position and torque will behave as sinusoidal functions and are recorded and saved on the PC's hard disk. The circles in Figure 3 show the logarithmic amplitude difference between the motor acceleration sinusoidals and the motor torque sinusoidals. The frequency response curves can be generated automatically for all of the 6 motors of the robot.

Given the frequency response measurements, Matlab will identify the dynamic elasticity parameters using a least-squares technique. The solid curve in Figure 3 illustrates the modelled transfer function between motor acceleration and motor torque. At the bottom of Figure 3, the model parameters such as motor inertia, arm inertia, joint spring stiffness, joint damper and resonance frequencies are displayed.

The modelled parameters from the identification tool can be fed back into the model-based robot controller to achieve an improved path-tracking performance of the robot's tool. The main benefit of having an accurate elasticity model, is to avoid oscillatory behaviour of the robot during fast and repetitive motions. Other model parameters, such as friction, also affect the low speed and

high-precision motions of the robot. To achieve a robot capable of meeting the strict path requirements from industrial customers, a large amount of identification experiments have to be run. The automatic identification tool significantly reduces the amount of man-hours needed to fine-tune all of the robot's model parameters.

3. Using Matlab's Object-Oriented Design Options

The identification tool has made use of Matlab's object-oriented design options.

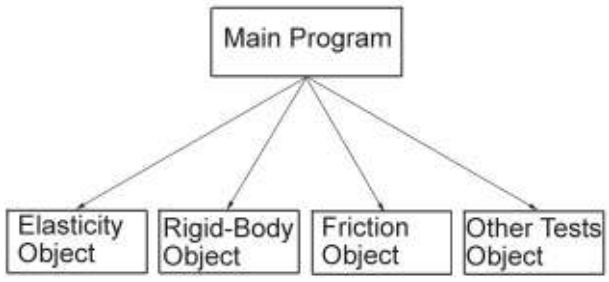

Figure 4: Using Matlab's object-oriented design options for separating identification results.

In Figure 4 the different objects created by the main identification program are illustrated. Each of the objects for elasticity identification, rigid-body identification, friction/gravity identification and other tests are responsible for maintaining its own raw data measurements, identification functions and final identification results. The other tests object has been written such that it is easy for an operator to add his own tests and robot motion programs. The advantage of the object-oriented design is the separation of functions and member variables. Any of the identification classes can be maintained individually without affecting the others.

One brief example of the actual implementation is given below. The following code is the Elasticity class constructor. The class constructor is located in a sub-directory named `@elast`, and the main program creates objects from this class by calling `myObject=elast`.

```
%%%%%%%%%% FUNCTION/METHOD HEADER %%%%%%%%%%
%   NAME:     elast
%   TYPE:     Class Constructor
%   PURPOSE:  Initialise Elasticity objects

function y = elast(f)
if nargin == 0
  %Object variables go here ...

  y = class(y, 'elast');
elseif isa(f,'elast')
  y = f;
else
  error('Input is not an elast object!');
end;
```

4. Communication Interface Between Matlab and Robot Controller

The components in our system consist of an ABB industrial robot, an S4C robot control cabinet and a PC running Matlab as illustrated in Figure 5.

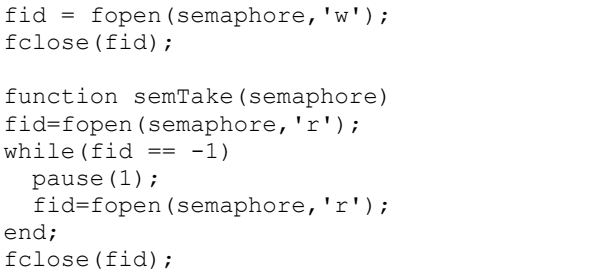

Figure 5: Communication between robot, robot controller and PC running Matlab.

The data transfer between the PC and the robot controller is made using a common file system. The robot controller is able to access the PC's hard disk across an Ethernet connection. Since the two systems run completely in parallel, a synchronization method is necessary. To show how the problem is solved using binary semaphores, we introduce the primitives, **`semTake`** and **`semGive`**. The semaphore can be seen as a binary value. When calling **`semGive`**, the binary value is set to one. When calling **`semTake`**, the calling function is blocked if the semaphore is equal to zero. The primitive **`semTake`** will wait indefinitely until the semaphore becomes equal to one. When the semaphore becomes equal to one, the semaphore is reset to zero and the calling function continues. In Matlab a simple synchronization solution is implemented using files in the common file system on the PC's hard disk. The code for **`semGive`** and **`semTake`** is given below.

```
function semGive(semaphore)
fid = fopen(semaphore,'w');
fclose(fid);

function semTake(semaphore)
fid=fopen(semaphore,'r');
while(fid == -1)
  pause(1);
  fid=fopen(semaphore,'r');
end;
fclose(fid);
delete(semaphore);
```

Similar functions are implemented in the robot controller. The complete execution of the model identification tool in Matlab and the robot controller is shown below.

```
Program flow in Matlab:
do
     generate_robot_motion_program;
     semGive(semA);
     semTake(semB);
     load_raw_data_measurements;
while(identification not ready);

Program flow in Robot controller:
do
     semTake(semA);
     execute_robot_motion_program;
     save_raw_data_measurements;
     semGive(semB);
while(identification not ready);
```

Note that there are no parameter identification routines in the program flow in Matlab when communicating with the robot controller. All the analysis is performed off-line after all the raw data measurements have been generated and saved on the PC's hard disk. The advantage of the off-line analysis, is the fact that the user can compare new identification results with previously stored identification results from other robots.

5. Identification of Rigid-Body Robot Parameters

In this section and the following, we give a more detailed description of the identification methods and results and the possibilities of the tool. When identifying the rigid-body dynamics, all elastic behaviour is ignored. The gear-box transmissions and the robot arms are considered as completely stiff and the motor inertia is added to the total arm inertia. The rigid-body dynamics and the elasticity dynamics are separated to simplify the identification methods. Care must be taken, however, when generating the robot motion programs for the rigid-body dynamics. To avoid errors in the rigid-body identification results, the motion programs should only contain frequencies well below the arms natural resonance frequencies. The rigid-body dynamics can be written as follows:

$$J(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \qquad (1)$$

where $q, \dot{q}, \ddot{q}$ are 6x1 vectors of joint positions, velocity and acceleration, respectively. J is a 6x6 position dependent inertia matrix, C is a 6x6 position and velocity dependent Coriolis and centripetal torque matrix, g is a 6x1 position dependent gravity vector and τ is a 6x1 vector of motor torques. The automatic model identification tool identifies all position and velocity dependent parameters of the J and C matrices and the gravity vector g.

The complete rigid-body dynamics may be optimised and reduced to depend on a minimum set of unknown parameters, see for example [1]. In the model identification tool, we have found the general optimised equations for a range of ABB's robots. At the bottom of Figure 6, the optimised equations for the diagonal terms of J (J_{11}, J_{22} and J_{33}) are displayed. Note that for the particular robot being identified in Figure 6, the inertia terms J_{22} and J_{33} are constants (P_{12} and P_{13}), while the expression for J_{11} is more complicated and depends on 11 linearly independent parameters. The general expression for J_{11} is given as:

$$J_{11} = s_2^2 P_1 + s_2 P_2 + s_2 c_3 P_3 + c_2 s_2 P_4 + s_2 s_3 P_5 + c_2 P_6 + s_3^2 P_7 + s_3 P_8 + s_3 c_3 P_9 + c_3 P_{10} + P_{11}$$

where ci=cos(q_i) and si=sin(q_i). We determine the general expressions for J_{11} and other dynamic terms using Maple V Release 5.0 and a tool called Sophia (http://www.mech.kth.se/~mlesser/html_lib/sophia.html) developed at KTH, Stockholm. We use Kane's method for determining the dynamic equations and Sophia is used for setting up the various co-ordinate frames. The details of the dynamic modelling are outside the scope of this paper.

In Figure 6 all the 11 independent parameters for J_{11} are identified and displayed. For example P_1=34.1072 and P_2=12.8839. These 11 parameters completely describe the inertia term J_{11} for all possible arm configurations of the robot.

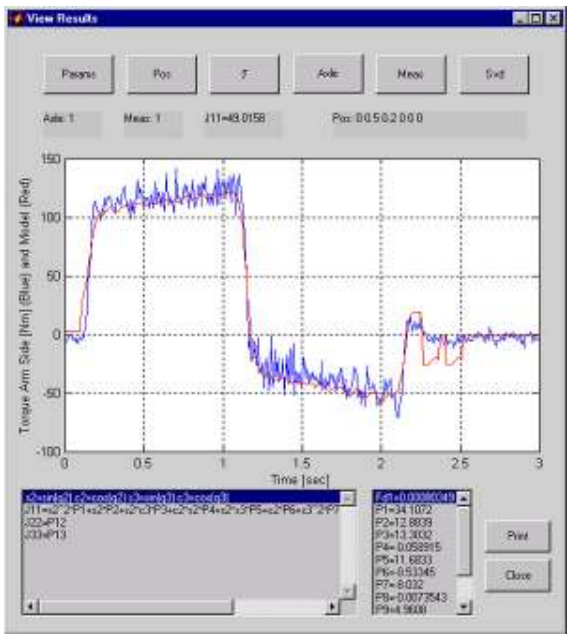

Figure 6: Example of rigid-body identification result.

Similarly, the model identification tool is able to identify all linearly independent parameters of the cross-inertia terms in the matrix J, all the terms in the Coriolis

and centripetal torque matrix C and all the terms in the gravity vector g.

The noisy curve in Figure 6 shows the measured raw data for the motor torque of motor 1 while the other curve in the Figure shows the dynamic model (first row of the left-hand side of equation (1)). The identification method performs a least-squared optimisation method on the raw data measurements to find the 11 independent parameters for J_{11}. The Matlab tool is responsible for creating robot motion programs such that at least 11 independent equations from the raw data measurements are generated before solving the linear optimisation program

At the top of Figure 6 the user can click different push-buttons to display the measured joint positions or view the dynamic model, data measurements and identified parameters for the other 5 joints. By clicking the "Param" button, the user can select between different parameters from the inertia matrix J, the Coriolis and centripetal matrix C or the gravity vector g. In this way, the user can quickly examine the identified 6-degree-of-freedom dynamic model. The accuracy of the identification methods can be verified by comparing the raw measurements of the motor torques with the curves for the identified dynamic model.

In the equation for J_{11} above, note that we have only included the parameters of J_{11} which depend on the joint angles q_1, q_2 and q_3. There is no dependency on the wrist angles q_4, q_5 and q_6. The reason for this simplification is the fact that the S4C controller can only log 6 signals simultaneously. Hence, we can only log the position and the torque for three motor joints. The next release of the robot controller (4th quarter 1999) allows for 12 signals to be logged simultaneously. Consequently, the complete 6-degree-of-freedom dynamics can be identified in one experiment. In the current version of the identification tool, we have separated the dynamic model of the first three axles from the dynamic model of the 3-axle wrist. Hence, two different experiments are needed to identify the models of the main axles and the wrist. Moreover, the coupling effects between the main axles and the wrist are currently not identified.

To summarise, complex rigid-body dynamic models for 6-degree-of-freedom industrial robots can be efficiently identified using an external PC running Matlab. The graphical-user-interface allows for portions of the dynamic model to be studied at the time. The complete identified model is available with a few mouse clicks.

6. Identification of Elastic Robot Dynamics

The identification of the elastic robot dynamics is more difficult than the rigid-body dynamics. Most industrial robots have only position measurements of the motors, while the arm positions are unknown. Hence, one important requirement of the identification algorithms is the fact that only motor torques and positions are available.

The dynamic model of a joint with elastic gear-box transmission is given below:

$$\begin{aligned} J_m \ddot{q}_m + D(\dot{q}_m - \dot{q}_a) + K(q_m - q_a) &= \tau \\ J_a \ddot{q}_a + J_{cop} \ddot{q}_{a2} + D(\dot{q}_a - \dot{q}_m) + K(q_a - q_m) &= 0 \end{aligned} \qquad (2)$$

where qm and qa are motor and arm positions, J_m and J_a are motor and arm inertias and D and K are joint damping and stiffness, respectively. Equation (2) is valid for a single arm with inertia-coupling, J_{cop}, to another arm q_{a2}. The identification algorithm implemented in the automatic tool works in the case where both q_a and q_{a2} are unknown. The actual implementation of the 2-axle identification algorithm is quite complex and a detailed description is beyond the scope of this article. For a detailed description of related work, see for example [2] and [3].

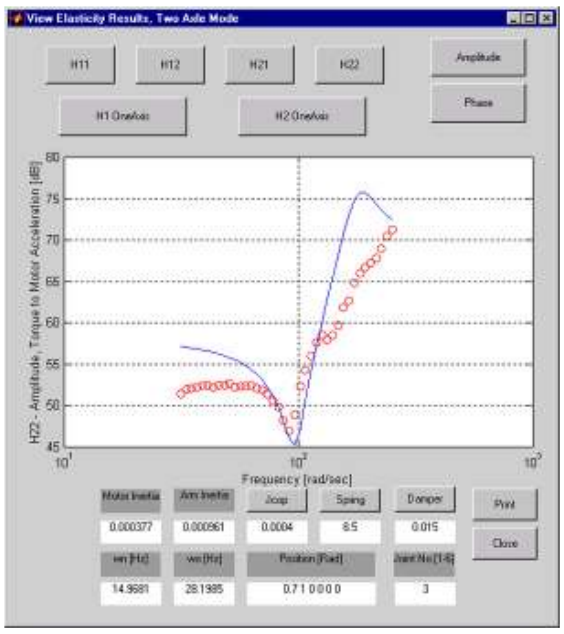

Figure 7: Example of joint elasticity identification result.

Figure 7 shows an example of the final results from the elasticity identification. The bottom of the Figure displays the identified parameters for the motor inertia J_m, the arm inertia J_a, the coupled inertia J_{cop}, the spring constant K and the damper constant D. In addition, the arms natural frequency is displayed. The circles show measurements of the logarithmic difference between the measured motor acceleration and the measured motor torque. The solid curve shows the transfer function of the identified model.

At the top of the Figure, the user has several push-buttons available for modifying the view. By clicking on

"Phase", the phase information is displayed instead of the default amplitude information. The buttons H_{11}, H_{12}, H_{21} and H_{22} will display four different transfer function elements of a multivariable 2-axle elastic system. For example, H_{11} displays the transfer function from motor 1 torque to motor 1 acceleration, H_{12} displays the transfer function from motor 2 torque to motor 1 acceleration, while H_{22} displays the transfer function from motor 2 torque to motor 2 acceleration.

To summarise, the results of quite complex identification algorithms are presented in a relatively easy way. After a short training period, non-experts in the field of system identification and control theory can understand the identification results and feed the identification results back into the S4C robot controller. When using the final identified parameters, the path tracking performance of the robot is improved. In particular, the fine-tuning of the elasticity parameters K and D reduces the oscillatory behaviour of the robot's tool.

7. Discussions and Conclusions

An automatic model identification tool has been developed by ABB. The tool is used internally at ABB to aid in achieving a near optimal path tracking performance to strengthen the position of ABB robots as high-accuracy machines.

The tool has been developed entirely in Matlab and is running on an external PC connected to the S4C robot controller through an Ethernet connection. Matlab was chosen instead of other development environments (such as C/C++) mainly because of the large amount of available toolboxes and Matlab's elegant vector/matrix operations. These features of Matlab greatly reduce the time involved from a new idea to working code.

The main limitation of the model identification tool is the communication link between the PC running Matlab and the S4C robot controller. The current version of the tool uses the PC's hard disk as a common file system for creating and deleting binary semaphores. A more elegant solution would be to use ActiveX components. ABB has already developed an interface between an external PC running Windows NT and the S4C controller. The interface is called RobComm and is currently installed on more than 2000 machines in over 15 countries. RobComm supports 32-bit Windows applications created with either Microsoft Visual Basic or Visual C++. The RobComm interface consists of a set of OLE Custom Controls known as OCXs. The OCX interface mechanism is a Microsoft standard utilised in all Microsoft visual programming environments.

The Matlab 5.2 release has support for two new ActiveX technologies: ActiveX control containment and ActiveX Automation client capabilities. The ActiveX control containment allows RobComm to be used in Matlab applications in a similar fashion to Visual Basic and Visual C++. To use RobComm in Matlab, we simply type `h=actxcontrol('RIMBASE.HelperCtrl.1');` and access RobComm through the `set`, `get` and `invoke` commands. The new ActiveX features of Matlab 5.2 are currently being tested by ABB for the model verification tool to overcome the limitations of the file-based communication channel. The ActiveX features of Matlab 5.2 means that complete robot application solutions can be written efficiently in Matlab code, converted to stand-alone applications using the Matlab Compiler and offered to external customers.

References

[1] C.G. Atkeson, C.H. An and J.M. Hollerbach, " Estimation of Inertial Parameters of Manipulators Loads and Links", International Journal of Robotics Research, Vol. 5, No. 3, pp. 101-118, 1986.

[2] A. De Luca, "Trajectory Control of Flexible Manipulators", Control Problems in Robotics and Automation, pp. 83-104, Editors: B. Siciliano and K.P. Valavanis, Springer.

[3] J. Swevers, et.al., "Optimal Robot Excitation and Identification", IEEE Transactions on robotics and automation, Vol. 13, No. 5, pp. 730-740.

Address for correspondence.

Dr. G.E. Hovland
ABB Corporate Research
Bergerveien 12
PO Box 90
N-1375 Billingstad
NORWAY
geir.hovland@no.abb.com