View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by CiteSeerX

An Experimental Setup for Modelling, Simulation and Fast Prototyping of Mechanical Arms

B. Bona, M. Indri, N. Smaldone

Politecnico di Torino
Dip. di Automatica e Informatica
Corso Duca degli Abruzzi 24, 10129 Torino, Italy
E-mail: smaldone@polito.it

Abstract

This work presents some results on a fast prototyping system used to model, simulate and control a planar manipulator at the Robotics Laboratory of the Politecnico di Torino. A unique environment allows to perform the entire range of procedures. A Host-Target architecture based on a PC-DSP link has been used for two basic advantages: simplicity of use, thanks to the well known PC environment hosting Matlab; real-time constraints fulfillment thanks to an independent hardware and software, devoted exclusively to the control function.
Using the same Matlab environment, various experimental tests and data analysis have been performed, to determine a good model for friction phenomena. On the basis of such a model a simulator has been built, and nonlinear control laws have been tested.

1 Introduction

Fast prototyping in control systems is usually addressed as the full process of modelling, simulation, design and testing, performed on the same architecture and in a common environment. This procedure speeds up the learning curve and simplifies the data exchange between domains.
The OpenDSP system is an open and integrated architecture that comprises a real-time software running on a DSP board, and a hardware that ensures the correct signals interchanging with the field and a PC host. This common structure is then composed by a PC host supervisor and a remote (but not very much, because of the parallel link characteristics) target board, with dedicated components for low level interaction with plant. The software residing on the host, e.g. Matlab, is used to develop the whole prototyping process and to interact with the plant from an higher abstraction level; between host and target a complex but well stratified structure allows the information to flow in both directions, respecting the real-time timing constraints.
At the Robotics Laboratory of Politecnico di Torino a planar robot for teaching activities has been reconfigured [1] and the original control system, no longer suitable for fast design and testing, has been substituted according to the above exposed guidelines. Starting from the general configuration of OpenDSP, a specific customization has been developed with additional hardware and software components. New interfaces have been proposed, by using the Matlab tools to take advantage from their elaboration features and simplicity of use.
In this paper we show in some details how this HW/SW architecture has been used to review the model of the manipulator, and to design new control algorithms based on the compensation of typical non-linearities and friction effects.
Section II presents a brief description of the plant and its control system. Section III illustrates the control system architecture, with details on the development environment and the user customizable C code.
In Section IV the model for friction phenomenon is illustrated along with the experimental results necessary to determine all the model parameters and the simulator based on it. Section V shows how the new model has been used to improve the control and adds some note on its C language implementation.

2 The robotic system

The planar manipulator used for the experiments has two revolute vertical axes joints, as sketched in Figure 1.
Both joints are moved, without using gearboxes, by *brushless NSK Megatorque* motors supplied with *resolvers* to measure angular positions. The maximum extension of the links ($L_1 + L_2$) is about 0.7 m, the angular limits are ± 2.15 rad for both joints, and the tip height is 0.45 m from horizontal plane.
The two brushless motors are managed by a couple of autonomous drives, which deal with all the complex characteristics of these actuators and manage the integrated position sensors. The drives communication

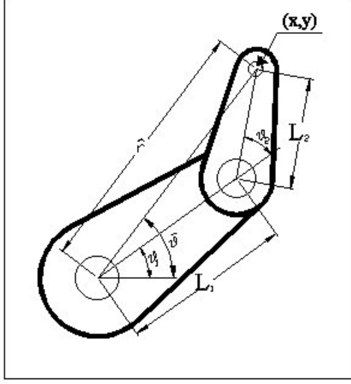

Figure 1: Diagram of IMI planar manipulator.

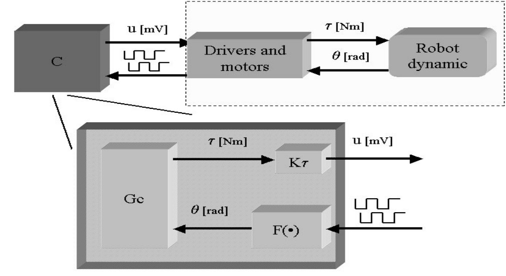

Figure 2: IMI-ODSP model

system deals, in particular, with some of the main features that are basic for control, such as the digital input/output signals interchange, the application of analog command inputs, and decoding of position information from sensors.
The drive boxes contain power electronics to manage the PWM for the motors, and a section, based on a 16 bit microprocessor, devoted to transform analog signals from resolver into digital signals of shaft encoder type. Besides, they interpret the analog signals coming from the controller as torque or velocity reference commands to be applied to the motors. Two control modes are available, the *Torque Mode* and the *Velocity Mode*: on the basis of the resolver signals, a current loop is closed to regulate the torque in the first case, whereas a further velocity loop is added in the second mode. The basic mode is the Torque mode, and it will be the only one used in this work to test different types of control algorithms, starting from the joint position information.
The inner current loop referred above is fixed, and the actuators model for control results in a simple proportional gain $K_{V2\tau}$ between the input command voltage, V_m, and the torque τ_m supplied by the motor:

$$\tau_m = K_{V2\tau} V_m \tag{1}$$

The optional Velocity Mode is useful in emergency situations, when the user wants to instantly interrupt the manipulator motion, pushing the STOP button: a digital input linked to the button lets the drive activate the velocity control loop, imposing zero velocity reference. The stopping phase will be executed as specified by the internal velocity control algorithm.
Finally, the whole plant and the controller can be modelled as shown in the summary diagram reported in Figure 2, that shows how the controller, fundamentally, receives encoders signals and gives back signals in mV proportional to required command torques.

3 The system control architecture

The original control system manufactured by the IMI Corp. has been substituted by a new one (see [1]), in which the components for real time interaction are grouped in a modular industrial standard rack. This control system environment, called *OpenDSP*, has been developed by the Mechatronics Laboratory of the Politecnico di Torino and consists of a DSP board and a programmable input/output board. A PLD (Programmable Logic Device) on the latter board allows to configure via software the digital and analog inputs and outputs, and to preprocess these signals in a customized way, before they reach the converters or the DSP. Field interfacing is obtained by means of user customizable boards, packaged with the I/O board and the DSP board in the same rack. The real-time control requirements are guaranteed by the presence of a link between the I/O and the DSP boards based on a proprietary bus (called the OpenDSP bus).
The system is linked via enhanced parallel port (EPP) protocol to a desktop PC, working as a host, and by some connections to each axes interface.
A Matlab environment with Simulink runs on the host PC. The OpenDSP system includes a new toolbox for Matlab called *MatDSP*, which allows the Matlab-code interaction with the DSP. In this way it is possible to read or change any variable processed by the DSP. For example, the parameters of a control algorithm can be changed "on fly" in the same sample time in order to guarantee a coherent switch to the new configuration (synchronous mode); or different variables, at user's choice, can be monitored without requiring a more stringent "sample by sample" acquisition (asynchronous mode). It is possible to monitor the real time variables and the drives status flags, to scope and acquire signals and make any type of mathematical operation on them. The control algorithms written in C can be compiled, downloaded and started/paused on DSP.
Some graphic user interfaces have been built in the Matlab environment by means of the GUIDE tool, to simplify testing and management of signals needed by the drives. The MatDSP commands have been

hidden by a logic construction, grouping the signals in high level functions rather than using them to perform single hardware operations. For example, a lot of cross-controls are needed to guarantee the correct and safe sequence of operations to enable and start the control task; this would oblige the user to read and change several variables using the primitive statements provided by the MatDSP toolbox. On the contrary, hiding the MatDSP commands under these GUIs allows the user to concentrate on new experiments. An example of these GUIs is shown in Figure 3.

Three tools are available to the user. The first one,

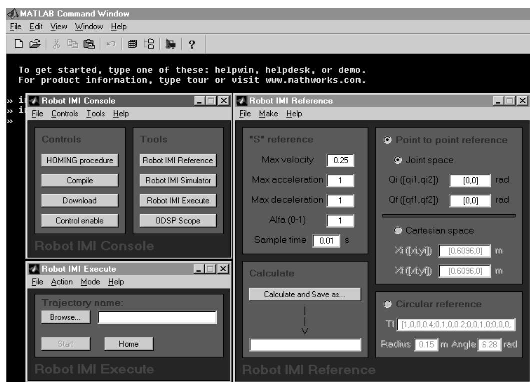

Figure 3: IMI robot GUIs

the *IMIConsole*, is a panel to perform the homing procedure, to prepare and to enable a control algorithm and, more in general, it is the entry point for the normal interaction with the control system.

The second tool is the *IMIExecute*, a panel that allows to select and execute, in single or cyclic mode, a previously planned trajectory and make a home return to the zero position. This GUI shares the same data base of IMIConsole tool to make appropriate and safety functional logic.

The third tool, the *IMIReference*, does not interact with the system because it is not related to the MatDSP toolbox, differently from the IMIConsole and the IMIExecute GUIs. It just generates some simple, basic reference functions, such as joint or cartesian point to point movements and circular trajectories and save them in a MAT file.

From the IMIConsole it is possible to open the IMIExecute or the IMIReference GUIs and to call a Simulink model of the robot to test the planned trajectories in simulation before executing them on the real plant.

The designer can compile and download his C code using the GUIs. At this point, he plans the trajectory, introducing the relevant parameters in the GUI, and enables the robot to execute it. Alternatively, he can start a Simulink model to check the effectiveness of the designed algorithm.

The OpenDSP real-time software belongs to an architecture group, known as *round-robin with interrupts*. At the beginning a main function, *Main.c*, calls some sub-functions which configure the system on the basis of a group of parameters, some of which fixed and other ones assigned by the user. Then, in an infinite loop two other sub-functions are called in turn: the first, called *Monitor*, deals with communication between the Matlab environment and the DSP; the second one, the *UserBackground*, allows to execute a user code at a lower priority level, which interprets and executes the Matlab commands and interacts with the drives logic. Both sub-functions have no particular real-time requirements and can be interrupted when the periodical axis control function written by the user has to start.

The whole user code is divided in sections and hosted in a file on the basis of a C written *template*. The initial section, the *UserInit*, contains the code to initialize the customizable characteristics of the system and the starting settings of axis control functions; it is executed one time, when the code downloaded to the DSP is launched. The variables, which must be available in the Matlab workspace, are declared and initialized within this function.

User writes in the subsequent section, the *UserISR_INT2*, the algorithm code for control and all the functions useful to close the loop: sensors reading, position reference managing and command application in the correct measure units. The UserISR_INT2 is executed every control sample time according to the following procedure:

- a timer sends a signal for Start Of Conversions (SOC) to the input and output converters (ADC and DAC);
- when the conversions finish, a signal for End Of Conversions returns, and the DSP stops the current job, i.e., one of the Monitor or UserBackground functions; note that a sample time delay is inserted by the system in the model of the plant, since the DAC uses the command computed in the previous step;
- the UserISR_INT2 is executed, and afterwards the DSP returns to the suspended job.

The sequence assumes that the control function execution stops before the next EOC signal, to respect the Shannon theorem and to execute a portion of the non real-time jobs, too. The template is ended by the UserBackground function, that contains the code executed by the DSP when the Monitor and UserISR_INT2 functions are inactive. As previously said, this code interprets the commands coming from Matlab and passed them to the DSP environment by means of the Monitor function.

To summarize, the openness of this system has allowed to configure five sections of the whole structure:

- the hardware interface toward the plant, by means of a custom electronics built on a standard development field module to be mechanically compliant with the rack and the stackthrough structure;
- the logical interface between DSP and field mod-

ules, managed by the PLD firmware. Starting from a general architecture, we filled in the PLD *user part* with opportune logic circuits devoted to group and to convert signals from and to the field module in registers, or to close faster non-clocked loop (in microseconds);

- the data base structure of the real-time signals, built in the form of registers and channel manageable by opportune macros in a pre-structured C header file;
- the Background routine that manages the communication between host and DSP, and the ISR routine to control the axes, starting from a general and strongly organized C template;
- the asynchronous communication between Matlab user and plant by means of a graphic user interface giving a logical and easier interpretation of the plant functionalities.

4 Dynamic model with friction for simulation

The well known manipulator dynamic equation is given by:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \tau_{\mathbf{f}}(\dot{\mathbf{q}}) = \tau_{\mathbf{m}} \quad (2)$$

where:

- $\mathbf{q}$ is the joint position vector;
- $\mathbf{M}$ is the inertia matrix, including both links and motors inertia;
- $\mathbf{C}\dot{\mathbf{q}}$ is the term containing Coriolis and centrifugal effects;
- $\tau_{\mathbf{f}}$ is the friction torque;
- $\tau_{\mathbf{m}}$ is the command torque.

No gravity term is present, since the manipulator is planar. The electrical dynamics of the motors is not considered, as the inner current loop, guarantees that it is much faster than the mechanical dynamics, and that, consequently, the relationship between the input voltage and the output torque is simply given by a gain, as in (1).

The friction term will be characterized taking into account two aspects of that phenomenon, *static friction* and *viscous friction*. Two different procedures should be executed to obtain a complete model of this term:

- with joints in open loop and,
- with the controlled manipulator.

4.1 Friction modelling experiments

Static friction is estimated by the following tests: each joint is set in a definite angular position, setting the drive in Torque Mode, and then supplying minimal torque increments in both clockwise (CW) and counterclockwise (CCW) directions. The friction torque prevents the joint motion until the command torque reaches the maximum static friction value.

When the joint starts to rotate, the current torque value is registered, and the procedure is repeated for various starting angular positions, to test the static friction dependency on the angular position of the joint. Trials are executed by means of a DSP code based on a fixed template, modified just in the section relative to the control function, the UserISR_INT2.

In the first two friction estimation tests the command torque increments are supplied in open loop, directly from the user. So, the DSP program receives each new torque value from PC, verifies if it is compliant with the imposed torque limits and then supplies it to the output converters by means of the macro `IOGP_FU1_WRITE_AOUT_ENGU(Channel)` directly in mV. A check on the control-enable signal is executed every sample time before refreshing the new torque value, to allow the correct sensing of the user interface inputs.

The test is executed in the Matlab environment using the IMIConsole GUI to compile and download the real-time code and to enable the axis drives; then the commands `MatDSPvariable(VarName, NewValue)` and `MatDSPupdate` allow to change the command torque reference at run-time, whenever the user needs it. The command `MatDSPupdate` allows to refresh in the same sample time all the real-time variables modified by the user with the command `MatDSPvariable`.

A unique value for τ_s to be used in the model is extracted using the mean data.

The contribution of viscous friction is evaluated letting the joints rotate freely, and using the Torque Mode functionality, to observe the phenomenon in a situation of dynamic equilibrium at constant velocity, i.e. when:

$$\tau_{\mathbf{m}} = \tau_{\mathbf{f}}(\dot{\mathbf{q}}) \quad (3)$$

The DSP code necessary for these experiments is the same used to evaluate the static friction, with the addition of the position measure by means of the macro `IOGP_FU1_READ_ENC_CURRENT(Channel)` and the acquisition data command, `Acquire()`, at the end of function UserISR_INT2. This functionality offered by the system is configurable at run-time by the Matlab command `MatDSPAcquireConfig(params)`, deciding which data are to be acquired, if data decimation is necessary and the duration of acquisition. It is not an invasive operation for the control function, i.e., it doesn't cause the violation of the sampling time, because it is executed entirely in the DSP environment to avoid a slow data exchange with the PC. The Monitor function returns data just acquired to Matlab environment, without real-time constraints, when the user invokes the command `MatDSPAcquireLoad()`. In the considered case, angular joint position values are acquired for each torque increment. A waiting time interval allows the end of the acceleration fluctuations, after which a two seconds acquisition is started.

Angular velocity data are extrapolated from the measured positions, for each joint and for each rotation direction. For every velocity sample, the correspond-

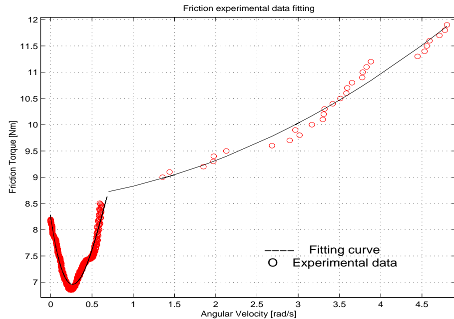

Figure 4: Joint 1 data fitting

ing friction torque is assumed equal to the command torque $\tau_{\mathbf{m}}$ according to (3).

The velocity data obtained have a lower bound value of about 2 rad/s, due to the sudden transition from stop to motion and viceversa, when the growing torque command overcomes the static friction τ_s and when the decreasing torque command reaches the minimum value, respectively.

Joint friction at low velocity is investigated by an experimental session performed with the manipulator in the operative controlled configuration. A simple PD has been used to move each joint from 0 to 1 rad/s and viceversa, using a triangular velocity profile. More code has been added at the UserISR_INT2 to supply a micro-interpolation mechanism for the user profile, and a section devoted to the position data processing needed by the PD algorithm.

This time the IMIExecute GUI has been used, with the IMIConsole, to provide the vector of position references to the DSP running code which interpolates and executes the movement. The IMIExecute asks the user for the reference vector and if the acquisition is needed; then, after a pre-positioning phase, it execute the jobs giving back the acquired data in the form of MAT file containing a matrix whose the columns are the samples of the signals acquired.

As in the previous experiments, the friction torque is deduced indirectly measuring joint positions and command torques, according to the following relation, derived from (2):

$$\tau_{\mathbf{f}}(\dot{\mathbf{q}}) + \tau_{\mathbf{err}} = \tau_{\mathbf{m}} - M(q)\ddot{q} - C(q,\dot{q})\dot{q} \qquad (4)$$

in which a new term, $\tau_{\mathbf{err}}$, representing all modelling errors and measurement disturbances, has been introduced. This term is disregarded, repeating several times the same motion and filtering the measured data to extract the mean values.

The points so obtained, along with those from the open loop experiments, give a diagram like the one presented in Figure 4, which shows the fitting curves too, for positive velocity of the first axis. The functions used to approximate experimental data are second order polynomials for the first part and third order for the second part of the characteristics.

4.2 Simulation environment

A Simulink model, based on the parameters obtained in the friction modelling experiments, is used to perform design and verification sessions.

A subsystem block collects all the components involved in the modelling of the manipulator. The inputs of the subsystem go through the command torque limits to another subsystem block devoted to evaluate the friction contribute and to return the residual torque commands for the dynamics of the manipulator. A suitable S-function calculates the friction torque distinguishing three different areas in which the model is partitioned: static friction phase with a spring-like behavior, low-velocity phase with a third order function behavior and high-velocity phase with a second order function behavior.

The IMIConsole GUI is the starting point of the design process which involves simulation and experimental tests. By means of a push-button on the GUI the Simulink model opens and the user can test his control setup; then he/she is requested to modify the C code of the control algorithm and return on the IMIConsole where he/she can compile and download that code.

5 Control design

A classical model based inverse dynamics feedback algorithm with a PD outer loop compensation has been implemented, by coding in C a simplified version of the friction model. Two different control laws have been tested, both in the simulation environment and on the real plant: in the first one only the inertial torques computed by the robot dynamic model are compensated, whereas in the second one friction compensation is added. Figures 5 and 6 show the simulated and the experimental joint angle errors obtained for the same circular trajectory by using the two control solutions. Figure 6 is divided in two sub-plot to clearly show Joint 1 and Joint 2 behaviors.

In the case of friction compensation, only the phases two and three of the model have been considered, i.e. the viscous friction modelled by two polynomials of third and second order, whereas the static friction has been neglected.

In both cases circular cartesian trajectories are used, each one lasting about 4.5 seconds, and planned by means of the IMIReference GUI.

A further test with a slower trajectory has evidenced a lower-bound limit of error compensation for these algorithms and a refinement of the friction model is required to reach extreme precision in positioning at low speed.

It can be noted some facts:

- the simulation data are in good agreement with experimental data, considering that the order of magnitude of the position error is $1 \cdot 10^{-3}$ for the first joint

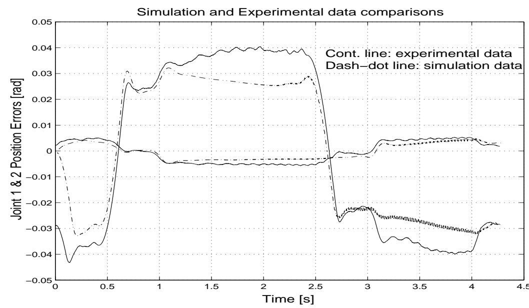

Figure 5: Error comparison when no friction model compensation is applied

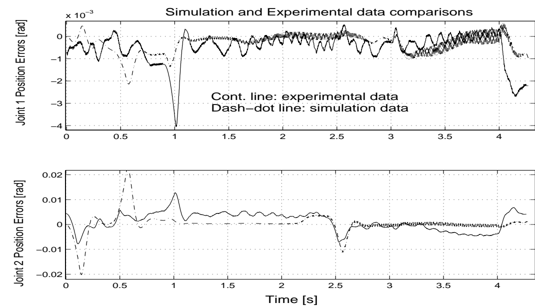

Figure 6: Error comparison when friction model compensation is applied

and $1 \cdot 10^{-2}$ for the second one;
• the improvement obtained with friction compensation based on viscous friction model only is about 75%, and this fact confirms the good approximation of the phenomenon by means of a simple analytical model;
• a better control strategy should take into account the pre-sliding phase to improve the arms behavior for very low velocity case.

The DSP code for the two cases is very similar to that one used to determine the friction parameters at low speed and using a PD control (see Section IV). This time the UserISR_INT2 contains a sub-section including the algorithm representing the inverse dynamics of the manipulator and the PD outer loop; in case of friction compensation, a simplified version of the friction model, taking into account just the viscous friction, takes place in a further sub-section. In this manner the arrangement of the UserISR_INT2 allows an easy insertion, if desired, of the friction compensation term.

6 Conclusions

A fast prototyping control system has been used to show all implementation steps involved in a procedure aimed to redefine the model and the control law of a manipulator. Starting from a basic C template for the real-time code, simple customizations allow to perform experiments aimed at determining the parameters for a simplified classical friction model. These parameters are used to design two types of model based control laws, that are first simulated and then downloaded on the DSP board for experimental validation. All these steps take place in a Matlab environment, which, by means of GUIs interacting with a dedicated toolbox, manages the control system, and allows other common operations like data analysis, trajectory planing, simulation, design and application of control laws directly on the plant.

Future works will be devoted to a strong integration between the Simulink model and the C code for control, a sort of (but not properly) *code generation*, to reflect in the DSP code in a parameterized mode each change the user would apply to the model. A visualization tool can be added to show in a structured way the data acquired by the IMIExecute GUI.

From the modelling point of view it will be necessary to investigate friction torques in pre-sliding conditions to improve the positioning of the manipulator at low speed.

6.1 Acknowledgements

The authors acknowledge the financial support of the Italian MIUR under “RAMSETE” and “MISTRAL” National Research Projects.

References

[1] B. Bona, M. Indri, N. Smaldone, *Open System Real Time Architecture and Software Design for Robot Control*, 2001 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM ’01), Como 2001, pp. 349-354.

[2] A. Delmastro, et al.,*OpenDSP Manuals and Applications*, AMET Srl, Turin, Italy, 2000.

[3] B. Armstrong-Helouvry, P. Dupont, and C. Canudas de Wit, *A survey of models, analysis tools and compensation methods for the control of machines with friction.*, Automatica, vol. 30, no. 7, pp, 1083-1138, July 1994.

[4] C. Canudas de Wit, P. Noel, A. Aubin, and B. Brogliato, *Adaptive friction compensation in robot manipulators: Low velocities*, Int. J. of Robotics Research, vol. 10, no. 3, pp. 189-199, June 1991.

[5] C. Canudas de Wit, H. Olsson, K.J. Astrom, and P. Lischinsky, *A new model for control of system with friction*, IEEE Trans. on Automatic Control, vol. 40, no. 3, pp. 419-425, March 1995.

[6] C. Canudas de Wit and P. Lischinsky, *Adaptive friction compensation with partially known dynamic friction model*, Int. J. of Adaptive Control and Signal Processing, vol. 11, pp. 65-80, 1997.