Proceedings of the 2012 IEEE
International Conference on Robotics and Biomimetics
December 11-14, 2012, Guangzhou, China

User Friendly Matlab-Toolbox for Symbolic Robot Dynamic Modeling used for Control Design

Emmanuel Dean-Leon†, Suraj Nair†, Alois Knoll‡, *Member, IEEE*

***Abstract*— In this paper a new Robot Modeling/Simulation Toolbox for Matlab is presented. The primary purpose of this toolbox is to generate all the common equations required for robot control design. It can compute the kinematic and dynamic equations of a serial robot in closed-form. The toolbox generates codes for the most representative matrices of the robot dynamics. For example, the Inertia Matrix, Coriolis Matrix, Gravitational Torques Vector and most important the Robot Regressor can be computed in closed-form with symbolic representation. This toolbox uses the Denavit-Hartenberg (DH) and Euler-Lagrange Methodologies to compute the Kinematic and Dynamic models of the robot. Furthermore, it automatically generates useful code for these models, such as M-Files, Simulink model and C/C++ code, allowing easy integration with other popular Matlab toolboxes or C/C++ environments. The only requirement from the user are the DH parameters, making it an easy to use tool. For 3D visualization, the toolbox supports different methods. The primary contribution is the automation and simplification of the robot modeling process which is important for correct robot design and control. In addition, the easy to use GUI and simplified models allow rapid prototyping and simulation of robots and control design/validation. As a proof of concept, validation of the computed models of a real industrial robot is included, where the toolbox was used to compute all the robot models. Thereafter, using the motion equations generated by this toolbox, a Dynamic Compensation Control was designed and implemented on a Stäubli TX-90 industrial robot in order to demonstrate how this toolbox simplifies the process.**

I. Introduction

Robotics is one of the most important research topics in engineering. In the past decades, the potential applications of robots in different fields, such as industry, education, research and entertainment have attracted the attention of many researchers. Along with advances in robotic design and control, simulation of robots is also gaining increasing importance. Simulation engines are capable of evaluating different such as, kinematics, dynamics, control approaches, etc. Simulation is of fundamental importance during each design phase such as: analysis, evaluation, planing, modeling and implementation. It allows the design and development of new control algorithms, mechanical structures of robots and helps to define the optimal parameter specifications of a system (control gains, link lengths, masses, etc.). As the complexity of the system under investigation increases the role of modeling/simulation becomes more important. For example, when the system is in design stage, it is not feasible to build the robot without validation in simulation due to financial constraints.

Similarly, robot simulators are important for educational purposes, where in many cases access to real robots is not possible. In such situations, simulation is a good replacement for a physical robot, provided the robot model is close to the real one. Therefore, simulation softwares have become strategic tools in robotics and are being used by many developers and manufacturers. In the control design field, its is well known that model-based control strategies (e.g. Adaptive Controllers, Regressor Based Sliding Mode Controllers, etc.) for robot manipulators are very effective since they take into account modeling uncertainties thereby, enhancing robustness by making the robot track a time-varying reference trajectory [1]. However, the principle drawback of these approaches is to obtain the dynamic motion equations. The toolbox introduced in this paper addresses this problem and offers an automatic solution to compute these complex equations in a symbolic form.

In the specific area of robot modeling, there are iterative methods. For example, DH for the kinematics [2], Euler-Lagrange for the dynamics [2] and recursive methods for the robot regressor [3], [4]. In these methods, the mathematical background required is not complex. However, it is time consuming. These approaches can be implemented using programing languages such as, C++, Java, Python or programing environments such as, Matlab/Simulink, Dymola/Modelica, Mathematica, 20-Sim, Scilab/Scicos, etc. In particular, Matlab is platform independent and widely used in the robotics community for modeling and simulation/evaluation.

Although Matlab provides helpful functions, the user has to compute and implement the mathematical models for his/her robot. In order to overcome this problem, several authors [5]–[7] have worked to obtain a general purpose robot modeler/simulator, where the creativity of the designer is required only in the design process and a set of tools are provided which significantly increases his/her efficiency. Robotic tools such as Peter Corke's Robotics Toolbox (RT) [7] provides useful applications for robot modeling/simulation, where the most important features are the simplicity and an easy to use API. These qualities are important while selecting a software tool. If the tool is not user friendly, then the user has to spend more time to solve problems not related to the main issue of robot/control and design/implementation.

In this article, we present a new Matlab toolbox which supports robot modeling and simulation. It mainly focuses

Authors Affiliation:†Cyber-Physical Systems, fortiss - An-Institut der Technischen Universität München. ‡Technische Universität München, Fakultät für Informatik.
Address: †Guerickestr. 25 80805 München (Germany). ‡Boltzmannstrasse 3, 87452 Garching bei München (Germany).
Email: †{dean, nair}@fortiss.org, ‡knoll@in.tum.de

978-1-4673-2127-3/12/$31.00 © 2012 IEEE

on the automatic code generation of the principal robot models as closed-form equations to be used in model-based control design, i.e. symbolic equations of Position Kinematics, Velocity Kinematics, Dynamics and Robot Regressor. Therefore, the primary goal of this toolbox is to generate the robot motion equations, where all the robot parameters (dynamic and kinematic) and joint states (positions, velocities and accelerations) are represented in a symbolic form (i.e. literals). This means, the tool box is not a physics engine simulator but rather a user friendly automatic generator of complex robot motion equations. The output code of this toolbox can be directly connected with popular Matlab toolboxes, e.g. RT [7] or with other programming languages such as, C/C++ language.

The remainder of this paper is organized as follows. In Section II, a brief survey on robot modeling/simulation software is given. Section III explains the main functions of the toolbox and the basic knowledge required by the user. Section IV introduces the GUI using a robot example. Section V shows a potential application of this toolbox in a Stäubli TX90 Industrial Robot. Finally, Section VI formulates the conclusions and future work.

II. Prior Art

An extensive review on robot simulation tools can be found in [8], [9] and [10]. On one hand, the authors in [8], [9] classify the robot simulators in 3 different classes: *1) MATLAB Based Simulation Systems*, e.g. [11]–[15], *2) General Simulation Systems*, e.g. [16]–[20] and *3) Multibody Dynamic Engines*, e.g. [12], [21]–[23]. On the other hand, in [10], the classification is given as *1) Generic Robotic Software*, e.g. [24]–[26], *2) Robot Software from Robot Manufacturers*, e.g. [27], [28] and *3) Academic/Open Source Robotic Software*, e.g. [5], [13], [29]–[31].

In our case we will divide the robot modeling/simulation in two main streams: *1) Numeric computation* and *2) Symbolic computation*. The former is the most common type and within this class, the most popular is RT [7]. Similar to RT is the Robotics Toolbox for Scilab/Scicos [5], which is an open-source robot modeler/simulator. In [32] the authors present a toolbox for modeling, design and simulation of robots based on actor-oriented modeling. This toolbox works in its own environment called Ptolemy II. In [6] a C++ library for kinematics and dynamics computations in real-time robot control is introduced. This library was developed with the goals of flexibility and high computational efficiency. The work in [33] shows how MAPLE can be used to model a robot system. RobotiCAD, described in [34], is a user friendly Matlab/Simulink toolbox which can model robotic system using the DH parameters and similar to [35], it is mainly focused on educational proposes. The common denominator of the above mentioned robot modeling tools is that they represent the robot model in a numeric form. These numeric models are suitable for simulation and model-free control design. However, these toolboxes have limitations when the exact robot model is required, e.g. for model based control design. An example of this case will be elaborated in Section V.

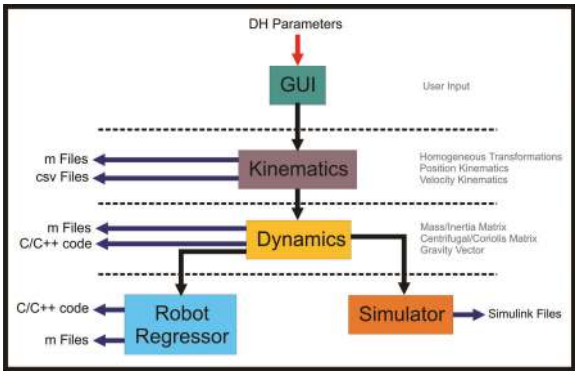

Fig. 1. Structure of the Robot Modeling/Simulation Toolbox.

In the second stream, we can find the robotic tools that use symbolic computation. The pioneer work of [20] presents *Robotica*, which is a software package for robotic manipulators based on Mathematica and it can calculate the symbolic and numeric equations of the kinematic and dynamic models for multiple degrees of freedom robots. Another interesting work is depicted in [36], where the authors illustrate a methodology for automatically deriving the Lagrangian equations based on Organic Computing. The work presented by [37] introduces a symbolic Matlab based toolbox called DAMARob which is capable of computing the kinematics and dynamics in symbolic-math matrix form.

Although many robot modelers/simulators are available, to the best of our knowledge, there is no robot modeling/simulation tool that can automatically generate the robot dynamic symbolic equations, including the Inertia Matrix, Coriolis/Centripetal Matrix, Gravity Vectors and specially the Robot Regressor, with the aim of control design. In addition, the simplicity and the user friendly interface allow any user to compute the motion models of any industrial serial-link robot without the need of deep knowledge on the toolbox. Furthermore, the toolbox is capable of generating Matlab/Simulink files and C/C++ code. The generated Simulink block model allows the user to quickly modify/validate the control structure without altering the simulation system. The computed matrices provide a concise means of describing the robot model and could facilitate the sharing of models across the robotics research community. These features makes this toolbox unique and a useful software tool. In the next section we introduce the structure of the toolbox.

III. Robot Modeling/Simulation Toolbox

The structure of the toolbox is depicted in Figure 1. It consist of four modules: 1) Kinematics, 2) Dynamics, 3) Robot Regressor and 4) Simulator. The toolbox also includes a GUI to simplify usage.

A. Kinematics

This low level module handles the symbolic computation of the robot kinematic models. It receives the DH table and

performs the following tasks.

1) Extracts the parameters from the DH table.
2) Computes the Homogeneous Transformations for each center of mass and joint. These transformations are calculated using the DH methodology [2]. This function generates CSV (Comma Separated Values) files for all homogeneous transformations. These files can be accessed using any common spreadsheet application. Several M-Files are generated containing the axis of motion and position vectors of centers of mass and joints. A special M-File with the Forward Kinematics of the robot end effector is also generated. Rotation matrices for each center of mass are computed and saved. These matrices are used in the dynamic model calculus.
3) Computes the Robot Jacobians. The symbolic equations of the velocity kinematics of each joint and each center of mass are calculated and saved in CVS and M files. In this case, the Jacobian of the end effector is also stored in a special M-file.

B. Dynamics

This module generates the dynamic motion equations of the robot. It is based on the Euler-Lagrange methodology [2] and obtains the dynamic model in the form:

$$M(q,\dot{q})\ddot{q}+C(q,\dot{q})\dot{q}+G(q)=\tau-B\dot{q} \tag{1}$$

Where, $M(q,\dot{q})\in\Re^{n\times n}$ represents the Mass/Inertia Matrix with n as the robot degrees of freedom. $C(q,\dot{q})\in\Re^{n\times n}$ is the Centripetal and Coriolis effects matrix. The vector of gravitational toques/forces is given by $G(q)\in\Re^{n\times 1}$. τ is the input joint torques/forces and $B\in\Re^{n\times n}$ is the viscous friction matrix. The vectors $q,\dot{q}\in\Re^{n\times 1}$ represent the joint position and velocity respectively.

The inputs to this module are the Homogeneous Transformations and the Jacobians obtained previously by the Kinematics module.

The actions performed by this module are:

1) Computes the Inertia Matrix $M(q)$. There is an option in the toolbox to validate the symmetric property of this matrix.
2) Computes the Centripetal and Coriolis effects matrix $C(q,\dot{q})$. This is done using the Christoffel symbols of the inertia matrix $M(q)$. There is also a skew-symmetric proof for the matrix $N=\dot{M}(q)-2C$ in order to validate the matrix $C(q,\dot{q})$.
3) Calculates the vector of Gravitational Torques $G(q)$.
4) Finally, it generates a M-File with the robot dynamics. This file is used in a Simulink block model to simulate the robot. This block model is generated automatically by the Simulator module. All the above matrices are used by the Robot Regressor Module to compute the linear regressor in terms of parameters.

C. Robot Regressor

The automatic computation of the robot regressor is one of the main features of this toolbox. The robot dynamic model can be expressed in the form:

$$M(q)\ddot{q}+C(q,\dot{q})\dot{q}+G(q)=Y(q,\dot{q},\ddot{q})\Theta \tag{2}$$

Where $Y(q,\dot{q},\ddot{q})\in\Re^{n\times p}$ is a matrix of robot states and $\Theta\in\Re^{p\times 1}$, is a vector of parameters. This equation represent a linear regression in terms of parameters of the form:

$$Y(q,\dot{q},\ddot{q})\Theta=\tau \tag{3}$$

Obtaining this matrix requires time and effort, especially when the robot has more than 3 dof and is not redundant[1]. This module overcomes this problem and provides an easy solution for the user. The final output is a M-file[2] with the state matrix and the vector of parameters. The tasks performed by this module are:

1) Simplify the symbolic expressions of each matrix/vector of the robot dynamic model.
2) Extract the robot parameters, e.g. masses, lengths, inertia terms, etc.
3) Generate the M-file with the state matrix and parameters vector. A validation of the regressor is performed with the equation: $M(q)\ddot{q}+C(q,\dot{q})\dot{q}+G(q)-Y(q,\dot{q},\ddot{q})\Theta=0\in\Re^{n\times 1}$.
4) If the C/C++ option is enabled, the module will generate C/C++ code for all the representative matrix/vectors of the robot dynamics and the robot regressor. This is useful for integrating the model in an environment different from Matlab.

The robot regressor plays a pivotal role in the design of model-based control strategies, since it encapsulates all the dynamic information in a linear form.

D. Simulator

This module generates a Simulink block model to simulate the robot dynamics. The unknown robot parameters are generated as a Simulink *Constant Block*. The user can input/modify the desired parameters while running the simulation. The functions of this module are:

1) Write the Simulink block diagram model. The M-file with the robot dynamics[3] will be included as *Matlab Fcn Block*.
2) If Peter Corke's 3D visualization is chosen, it generates a M-file to create the robot in the RT environment and includes a *plot Block*[4] in the Simulink model.
3) If Coach visualization (a 3D visualization module based on the Robotics Library [38] and Coin3D [39]) is selected, it generates a special *Matlab Fcn Block*. This block receives the joint positions from the simulator and sends them via TCP/IP to the Coach server.[5]

The last two features are optional and can be configured in the GUI. In the next section we will demonstrate the user friendly interface and the generated files.

[1]For example, a 6 dof articulated robot with spherical wrist can exhibit more than 70 parameters and the dimension of $Y(q,\dot{q},\ddot{q})$ could be 6×70.

[2]Also a C/C++ code will be generated if the option is selected.

[3]Generated by the Dynamics module.

[4]*Graphical robot display block* from RT toolbox.

[5]It requires XML files with the robot information and the scene.

IV. User Friendly Interface

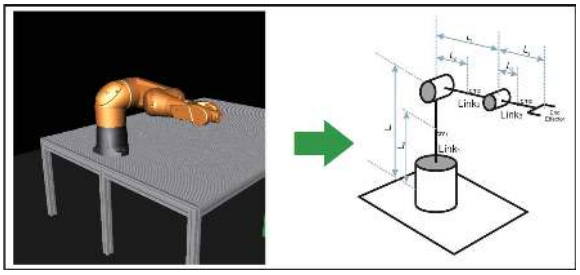

Fig. 2. Stäubli TX90 Robot. In this case only the first 3 dof will be analyzed.

We will show the simplicity of the GUI through a 3 dof robot. In this case we have selected the Stäubli TX90 industrial robot. This is a 6 dof robot, however, for simplicity, we only take into account the first 3 dof. Figure 2 shows the simple representation of the robot in the Coach environment. The only input needed from the user is the DH table of the robot and the joint configuration. In concrete, the DH table for the joints and for the centers of mass is required[6]. All the robot parameters are expressed in a symbolic form. This means, the user does not need to input the real kinematic and dynamic values to compute the motion equations.

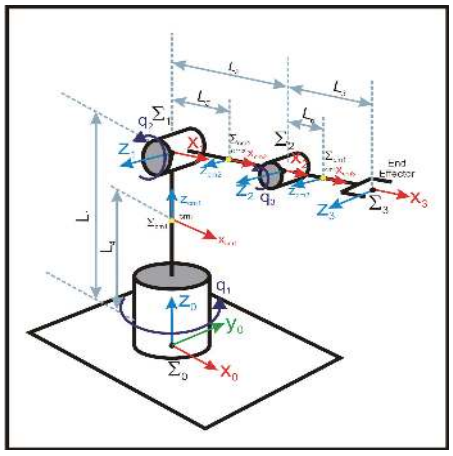

Fig. 3. Robot coordinate frames. The image shows 7 coordinate frames, 3 for each link, 3 for each center of mass and 1 for the robot base (reference frame), q_1, q_2 and q_3 are the joint positions.

The DH table of the robot shown in Figure 3 is defined in Table I, where all the parameters are represented in symbolic form.

The toolbox contains a GUI where all the settings can be defined. The first window consists of the general settings, see Figure 4. Here the user can define the settings related to the output files. The fields are:

1) **Robot Name**: This field defines the robot name for all the files created during the complete process. In Figure 4 (a), we can see the name 'ArticulatedRobotRRR'. The toolbox will generate files in the form: *_*ArticulatedRobotRRR.**, e.g. 'DSimulator_ArticulatedRobotRRR.mdl'.
2) **Degrees of Freedom**: This is the number of degrees of freedom for the robot.
3) **Joint Configuration**: The number of options will be defined by 'Degrees of Freedom', see Figure 4 (a) and (b). For each joint, the user can specify two types of joints: a) Revolute or b) Prismatic.
4) **Simulink Visualization**: Currently two options are supported i.e Peter Corke's RT Visualization and Coach. In the first option, the toolbox will create a M-File with the specifications of the robot in the RT format and a *RT plot Block* will be included in the Simulink block model. In the second option, a special M-function block will be generated in the Simulink model. This block sends the joint position of the robot to the TCP/IP Coach server. The user can select one, both or non of the options. When neither of the options are selected, the user has the standard *Simulink scope* to visualize the data, see Figure 7.
5) **Extra Options**: These options are related with the output files and the optimization level.
   - Validate Dynamics: if this option is enabled, the mathematical validations described in Section III-B for $M(q)$, $C(q,\dot{q})$ and $Y(q,\dot{q},\ddot{q})\Theta$, are evalu-

[6]If the user desires a simplified model, he/she can use the DH of the links for both.

TABLE I
DH Parameters of Joints and Centers of Mass

i	θ	d	a	α
1	q_1	L_1	0	$\frac{\pi}{2}$
2	q_2	0	L_2	0
3	q_3	0	L_3	0
cm_1	q_1	L_4	0	0
cm_2	q_2	0	L_5	0
cm_3	q_3	0	L_6	0

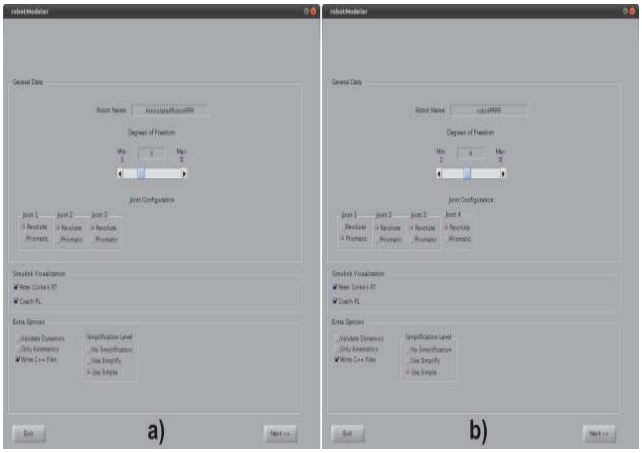

Fig. 4. Main settings windows. The image shows the different fields which the user can configure in the toolbox. a) Setting for the robot in Figure 3, b) The same robot with an extra degree of freedom, e.g. the robot mounted in a prismatic joint.

ated.

- Only Kinematics: only the Forward Kinematics (position and velocity) models are generated. This is when the dynamics are not needed.
- Write C++ files: this flags controls whether the output files are written as M-Files only or both M-Files and C/C++ files.
- Simplification Level: this option controls the optimization level for computing the symbolic equations. There are three levels: a) No simplification (this option will generate extremely large equations), b) Use the Matlab function *simplify()*, and c) Use the Matlab function *simple()*. This level depends on the number of degrees of freedom, since for more than 5 degrees of freedom the computation using *simple()* could take a few hours[7].

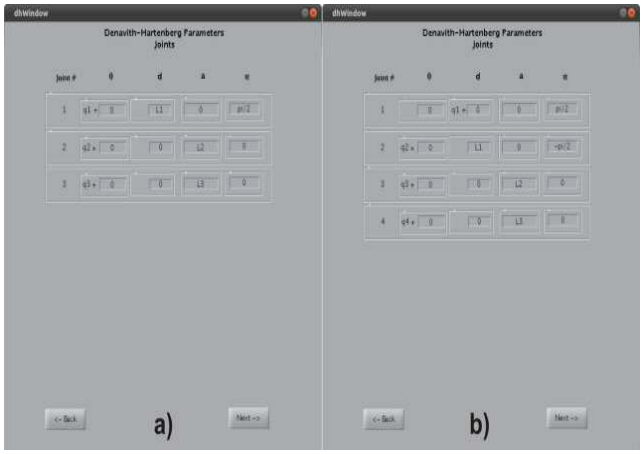

Fig. 5. Denavit-Hartenberg parameters of each Link. a) The parameters of robot in Figure 3 are presented, b) Parameters of the same robot with the extra dof. The number of rows and the column of the joint position variable are automatically defined by the toolbox

The next step is to input the specific data of the robot. This has been divided into 3 windows. The first one is the DH parameters of each joint. The DH parameters of the example robot are depicted in Figure 5 (a), while in Figure 5 (b), parameters of the 4 dof robot are shown. It is important to note that the toolbox automatically selects the place of the joint position variables q_i in the table, i.e. in θ_i or in d_i. Thereafter, the user only needs to input the offsets, which are specific for his/her robot. In general, these offsets can be either real numbers or variables, but the basic idea is to generate a **parameterized model**. Hence, it is recommended to use variables to describe these offsets.

In the next window, the user is requested to input the DH parameters of the centers of mass. This data is used to compute the robot dynamic model (and the regressor). If the *Only Kinematics* option is selected, then this window is omitted. The user can use the same DH parameters of the joints. This implies that the centers of mass of the robot are located exactly at the joints.

[7]This is not a problem, because the robot modeling should be done one time and it is out-of-line. Thereafter, the generated model can be used in a real-time application.

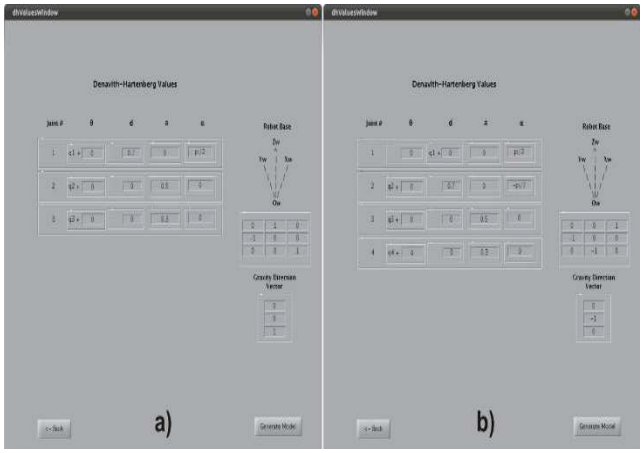

Fig. 6. DH values. These values are needed to visualize the robot in the *RT* environment. a) 3 dof robot, b) 4 dof robot

The last window is optional. If the *Peter Corke's Visualization* is selected and *Only Kinematics* option is set to 0, then the options shown in Figure 6 are available. The window is divided into three sections: 1) DH Values: these are the real robot lengths, 2) Robot base: the user needs to provide the rotation matrix between his/her robot base-frame and the world coordinate frame proposed in RT, 3) Gravity Vector: a unitary vector describing the gravity direction. Thereafter, the toolbox will compute the robot models and generate the output files. An interesting feature of this toolbox is the output Simulink model. This model will be launched after all computations are performed. Thereafter, the user can start simulating and visualizing his/her robot.

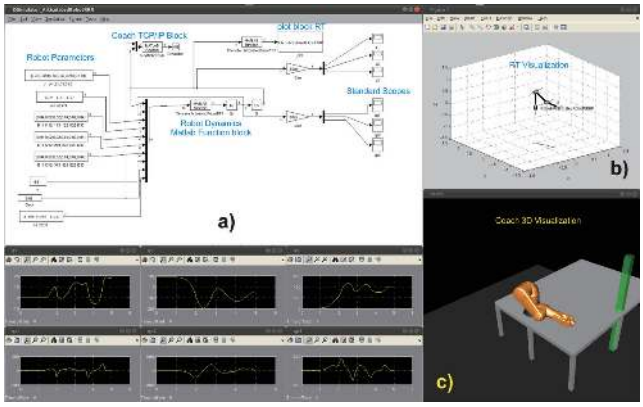

Fig. 7. Automatically generated Simulink block model. It consists of (a) the block model of the robot, (b) RT robot visualization and c) Coach 3D visualization. The user can modify the parameters of the robot and observe the different robot dynamic behaviors.

The automatically generated Simulink block and the visualization of the 3 dof robot are demonstrated in Figure 7. In this figure, three different parts are illustrated: (a) Simulink block diagram: This model is automatically generated by the toolbox and allows the user to set and change the robot

parameters, i.e. the user can test the robot model using different values. The available parameters are: Robot offsets (lengths), masses, inertia tensor of each link (this tensor is calculated with respect to its center of mass and can be obtained with a CAD software), gravity acceleration and the viscous friction values. b) RT Visualization: This is the visualization provided by RT. A robot configuration file is generated for this propose. c) Coach 3D visualization: Currently, the toolbox only generates the TCP/IP block connection. In the near future, a tool to generate XML files of the robot and the scene will be included.

The Simulink model is designed in a manner to improve its usability, i.e. only minimum knowledge in Matlab/Simulink is required.

Thereafter, the user can simulate the robot with different conditions in Matlab/Simulink or use the generated dynamic model files in a C/C++ application.

A video with a real time robot model generation can be seen under the link:
http://youtu.be/3pV6tELV5ig?hd=1

The generated model files of both examples can be downloaded under the link:
http://www6.in.tum.de/~dean/GeneratedRobotModels.zip

In the next section we demonstrate a potential application of this toolbox in a real industrial robot.

V. Experimental Validation

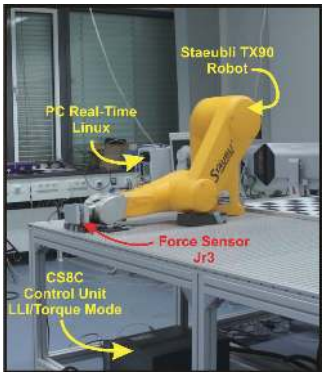

Fig. 8. Experimental Validation. The figure depicts the robot setup used to validate the toolbox. The system consists of an industrial robot with an open architecture control unit, a real time Linux workstation and a force sensor.

In order to validate the toolbox we use the industrial robot system shown in Figure 8. The robot system consists of a Stäubli TX90 Robot with a CS8C control unit running with a Low Level Interface library (LLI) in torque mode, a real time Linux workstation with a Intel Core i7 processor, 8Gb RAM and a JR3 force sensor attached to the robot tool-tip. The LLI provides a C++ Application Program Interface (API), to control the robot at low-level. More information on the LLI can be found in [40].

The procedure of the experiment is as follows:

A) Compute the robot dynamic model (matrices M, C, Y and vectors G, Θ) using the toolbox and the DH parameters shown in the Table II. Thereafter, the parameter vector Θ is estimated.

TABLE II
DH parameters of the Stäubli TX90 robot.

i	θ	d	a	α
1	q_1	d_1	L_1	$-\frac{\pi}{2}$
2	$q_2-\frac{\pi}{2}$	d_2	L_2	0
3	$q_3+\frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
4	q_4	d_4	0	$-\frac{\pi}{2}$
5	q_5	0	0	$\frac{\pi}{2}$
6	q_6	0	0	0
cm_1	$q_1+\alpha$	d_7	L_3	0
cm_2	$q_2-\frac{\pi}{2}$	d_8	L_4	0
cm_3	q_3	0	L_5	0
cm_4	q_4	d_9	0	0
cm_5	$q_5-\frac{\pi}{2}$	0	L_6	0
cm_6	q_6	d_{10}	0	0

B) In order to estimate Θ, we exploit the linear property of eq. (3), which relates measurements of the trajectories q, $\dot{q}$, $\ddot{q}$ and the non-conservative torque τ to the vector of parameters Θ. The robot should move with trajectories having persistence of excitation and non-singularity of the innovation terms. With the LLI, it is possible to read the joint positions/velocities and the applied joint torques every 4ms. In this case, we assume that the noise in all measurements has the same standard deviation. Therefore, the standard Linear Least Square Estimation results in the form:

$$\hat{\Theta} = \left(\Upsilon^T \Upsilon\right)^{-1} \Upsilon^T \varsigma \tag{4}$$

with,

$$\Upsilon = \begin{bmatrix} Y\left(q(t_0), \dot{q}(t_0), \ddot{q}(t_0)\right) \\ Y\left(q(t_1), \dot{q}(t_1), \ddot{q}(t_1)\right) \\ \cdots \\ Y\left(q(t_m), \dot{q}(t_m), \ddot{q}(t_m)\right) \end{bmatrix}, \text{ and } \varsigma = \begin{bmatrix} \tau(t_0) \\ \tau(t_1) \\ \cdots \\ \tau(t_m) \end{bmatrix}$$

where, m is the number of samples. In this manner, Θ can be computed. More about parameter identification can be found in [41].

C) Design and implement a control strategy using the robot regressor to compensate the robot dynamics in real-time. The control law is given by

$$\tau = Y\left(q, \dot{q}, \ddot{q}\right) \hat{\Theta} \tag{5}$$

D) For safety reasons, the robot starts in a stable position, i.e., lying on the table. Thereafter, the joint torques are computed using the control law in eq. (5) and (4). The desired torque is sent via TCP/IP to the CS8C control unit every 4ms. This controller transforms the reference signal into voltage for each motor.

E) The JR3 force sensor is only used to measure the force exerted by the user. Figure 9 shows the results of the experiment. In the first case [(a) and (b)], no dynamic compensation was implemented, i.e. $\tau = 0 \in \Re^{n\times 1}$ and in the second case [(c) and (d)], the control law (eq. (5)) was applied.

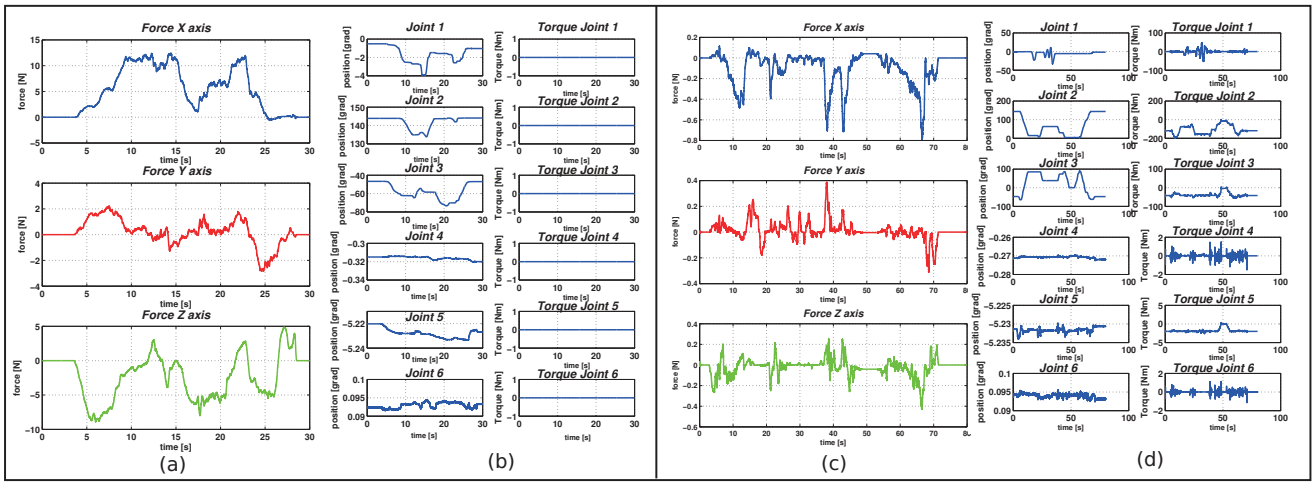

Fig. 9. Experimental Results without dynamic compensation ($\tau = 0$) [(a) and (b)] and with dynamic compensation ($\tau = Y(q, \dot{q}, \ddot{q})\hat{\Theta}$) [(c) and (d)]. The figure shows (a) the force exerted on the tool-tip in X-Y-Z axes without dynamic compensation, (b) The joint positions q and joint torques τ without dynamic compensation, (c) The force in the tool-tip with dynamic compensation and (d) The joint positions and applied joint torques with dynamic compensation. Notice that in the first case $\tau = 0$, while in the second case, τ is computed on-line and depends on the joint position.

In Figure 9, the experimental results are exhibited. Figure 9 (a) shows the exerted force by the user on the robot tool-tip. This force is captured by the force sensor. In this case, no dynamic compensation is executed. Figure 9 (b) illustrates the joint positions for each link and the applied joint torques under the same situation. It can be observed that the joint torques are always zero, which means the user is trying to carry the complete weight of the robot[8]. Therefore, the obtained motion is very low. This is more evident in the case of joint 2 (the joint with the heaviest link in the robot), where the maximum motion was around 12°. On the other hand, Figure 9 (c) and (d) shows the forces applied by the user on the tool-tip and the joint position/torques in the robot respectively. In this case, the dynamic compensation control approach is activated. It can be seen from Figure 9 (c) that the applied force is reduced almost to a one tenth as compared to the case without dynamic compensation. Furthermore, the generated motion naturally achieves bigger dimensions. In the case of link 2, the motion is almost 150°. Figure 9 (d) illustrates the applied joint torques. We can see how the torques change in time and as expected, depend on the joint position. A video with both experiments is available under the link:
http://youtu.be/hQrt7YpOnts?hd=1. It is important to note that the design of this controller was completely simplified using this toolbox where the most complicated part is to compute the dynamic equations, which is done automatically by the toolbox. This is the main objective of the toolbox.

VI. Conclusions and Future Work

A. Conclusions

In this article, we presented a new Matlab toolbox to compute the motion equations of serial-link industrial robots. The toolbox automatically generates files of the most representative models of a robot. This includes Forward Kinematics (position and velocity), robot dynamics (Mass matrix, Centripetal and Coriolis effects matrix, Gravitational Torques vector and the robot Regressor). These matrices can be used to design robust control laws. The code generation can be selected either as Matlab or Matlab and C/C++. The toolbox has a user friendly GUI, which allows the user to easily configure the different options. It is capable of generating Simulink models where the user can validate/simulate the robot in a closed loop with his/her controller under different conditions. It also offers two different methods for 3D visualization based on popular and available software. The knowledge required to use the tool is minimum (only DH parameters are required and basic linear algebra). This toolbox is accessible for a wide range of users. Furthermore, a experimental validation was given, where the code generation was demonstrated. The validation is carried on a real (Stäubli TX90) industrial robot, where the obtained models were used to control the dynamic behavior of the robot.

[8]The total robot weight is 110 Kg.

B. Future Work

The future work can be divided into four sections:

1) Visualization: The immediate next step is to implement the automatic generation of 3D visualization environment. Currently, if the user selects Coach, he/she needs to generate the XML files. This process can be done automatically using the DH parameters. One interesting feature of Coach is that it has the capability of detecting collisions. This information can be used in the Simulink model to generate more realistic simulations.

2) Robot Configuration: Presently, the toolbox can generate code only for serial-link robots. We are working to include parallel robots in the toolbox. The basic problem to solve is maintaining the user friendly API.

3) Control Options: We plan to include pre-defined control approaches such as, PD, PID, Computed Torque, Adaptive Control, etc. The idea is to provide a selection box, where the user can include any of these popular control laws in his/her simulation. Another important issue is the friction model. In the current state, only viscous friction is available. Therefore, we are planning to include more complex friction models, which include dynamic effects [42], e.g. Stick-Slip effects, Hysteresis, Coulomb Friction, etc. Thereafter, dynamic friction compensation can also be included in the control approaches.

4) Parameter Identification: Automatic identification of robot parameters, not limited to the linear but also including the non-linear parameters, is an interesting and challenging task which need to be analyzed in detail.

This Matlab toolbox will be soon freely available for the public domain. Currently, we are finalizing the License.

REFERENCES

[1] H. G. Sage, M. F. De Mathelin, and E. Ostertag, "Robust control of robot manipulators: A survey," *International Journal of Control*, vol. 72, no. 16, pp. 1498–1522, 1999. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/002071799220137

[2] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. John Wiley & Sons Hoboken, NJ, 2006.

[3] W. Lu and Q. Meng, "Regressor formulation of robot dynamics: computation and applications," *Robotics and Automation, IEEE Transactions on*, vol. 9, no. 3, pp. 323–333, 1993.

[4] J. Yuan and B. Yuan, "Recursive computation of the Slotine-Li regressor," in *American Control Conference, 1995. Proceedings of the*, vol. 3. IEEE, Jun. 1995, pp. 2327–2331.

[5] J. Liu, R. Su, and S. Liu, "A Scilab/Scicos-based modeling and simulation toolbox for flexible manipulator," in *Open-source Software for Scientific Computation (OSSC), 2009 IEEE International Workshop on*. IEEE, 2009, pp. 30–36.

[6] R. Hopler and M. Otter, "A versatile C++ toolbox for model based, real time control systems of robotic manipulators," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 4. IEEE, 2001, pp. 2208–2214.

[7] P. Corke, "A robotics toolbox for MATLAB," *Robotics & Automation Magazine, IEEE*, vol. 3, no. 1, pp. 24–32, 1996.

[8] L. Zlajpah, "Robot Simulation for Control Design," in *Robot Manipulators Trends and Development*, ser. Mobile Robotics, A. J. Basil and M. A. Hadithi, Eds. InTech, Mar. 2010, ch. 3, pp. 43–72.

[9] L. lajpah, "Simulation in robotics," *Mathematics and Computers in Simulation*, vol. 79, no. 4, pp. 879–897, 2008.

[10] Z. Pan, J. Polden, N. Larkin, S. Van Duin, and J. Norrish, "Recent progress on programming methods for industrial robots," *Robotics and Computer-Integrated Manufacturing*, 2011.

[11] L. Zlaipah, "Planar Manipulators Toolbox: User's Guide," Jozef Stefan Institute, Tech. Rep. DP-7791, 1997.

[12] M. Hollars, D. Rosenthal, and M. Sherman, "SD-Fast Users's Manual, Symbolic Dynamics," *Inc., Mountain View, CA, b*, vol. 1, 1990.

[13] P. Corke, "MATLAB toolboxes: robotics and vision for students and teachers," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 4, pp. 16–17, 2007.

[14] G. Wood and D. Kennedy, "Simulating mechanical systems in Simulink with SimMechanics," *The MathWorks inc*, 2003.

[15] C. Kleijn, *Getting Started with 20-sim 4.1*. http://www.20sim.com/downloads/files/20sim41GettingStartedManual.pdf: 20-sim, 2009.

[16] "RoboWorks," http://www.newtonium.com/.

[17] "ROBCAD/Workcell, User's manual," Tecnomatix.

[18] J. Jackson, "Microsoft robotics studio: A technical introduction," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 4, pp. 82–87, 2007.

[19] O. Michel, "Webots: Professional Mobile Robot Simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.

[20] J. Nethery and M. Spong, "Robotica: a Mathematica package for robot analysis," *Robotics & Automation Magazine, IEEE*, vol. 1, no. 1, pp. 13–20, 1994.

[21] J. Jarez and A. Suero, "Newton Game Dynamics," http://newtondynamics.com/, 2008.

[22] R. Smith, "Open Dynamics Engine," 2008, http://www.ode.org/.

[23] A. Miller and P. Allen, "Graspit! a versatile simulator for robotic grasping," *Robotics & Automation Magazine, IEEE*, vol. 11, no. 4, pp. 110–122, 2004.

[24] Z. Bi and S. Lang, "A Framework for CAD-and Sensor-Based Robotic Coating Automation," *Industrial Informatics, IEEE Transactions on*, vol. 3, no. 1, pp. 84–91, 2007.

[25] R. Brown, "Driving digital manufacturing to reality," in *Simulation Conference Proceedings, 2000. Winter*, vol. 1. IEEE, 2000, pp. 224–228.

[26] D. Lee and W. ELMaraghy, "ROBOSIM: a CAD-based off-line programming and analysis system for robotic manipulators," *Computer-Aided Engineering Journal*, vol. 7, no. 5, pp. 141–148, 1990.

[27] C. Connolly, "Technology and applications of ABB RobotStudio," *Industrial Robot: An International Journal*, vol. 36, no. 6, pp. 540–545, 2009.

[28] K. Vollmann, "A new approach to robot simulation tools with parametric components," in *Industrial Technology, 2002. IEEE ICIT'02. 2002 IEEE International Conference on*, vol. 2. IEEE, 2002, pp. 881–885.

[29] W. Dai and M. Kampker, "PIN-a PC-based robot simulation and offline programming system using macro programming techniques," in *Industrial Electronics Society, 1999. IECON'99 Proceedings. The 25th Annual Conference of the IEEE*, vol. 1. IEEE, 1999, pp. 442–446.

[30] A. Jaramillo-Botero, A. Matta-Gomez, J. Correa-Caicedo, and W. Perea-Castro, "ROBOMOSP," *IEEE Robotics & Automation Magazine*, p. 2, 2006.

[31] M. Association *et al.*, "Modelica-A Unified Object-Oriented Language for Physical Systems Modeling–Language specification," *URL: http://www. modelica. org/documents/ModelicaSpec22. pdf*, 2005.

[32] N. Le and T. Nguyen, "A robotics modeling, design and simulation toolbox in Ptolemy II," in *Control and Decision Conference (CCDC), 2010 Chinese*. IEEE, 2010, pp. 2297–2301.

[33] J. Wallén, "On robot modelling using Maple," Technical Report LiTH-ISY, Tech. Rep., 2006.

[34] R. Falconi and C. Melchiorri, "Roboticad: An educational tool for robotics," in *Proceedings of the 17th IFAC World Congress, Seoul*, vol. 17, 2008.

[35] H. Arshad, J. Jamal, and S. Sahran, "Teaching Robot Kinematic in a Virtual Environment," in *Proceedings of the World Congress on Engineering and Computer Science*, vol. 1, 2010.

[36] M. Wenz and H. Wörn, "Automatic Configuration of the Dynamic Model for Common Industrial Robots," *GI-Edition Lecture Notes in Informatics (LNI)*, pp. 137–144, 2006.

[37] M. Caputano and D. Bellicoso, "DAMAROB: Dario and Marco's Robotics Symbolic Toolbox for Matlab," http://www.damarob.altervista.org.

[38] M. Rickert, "Robotics Library," http://sourceforge.net/apps/mediawiki/roblib.

[39] "Coin3D: 3D Graphics Development Tools," http://www.coin3d.org/.

[40] F. Pertin and J. B. des Tuves, "Real time robot controller abstraction layer," in *Proceedings of the International Symposium on Robotics 2004*, 2004.

[41] L. Lennart, "System identification: theory for the user," *PTR Prentice Hall, Upper Saddle River, NJ*, 1999.

[42] C. de Wit, H. Olsson, K. Astrom, and P. Lischinsky, "Dynamic friction models and control design," in *American Control Conference, 1993*. IEEE, 1993, pp. 1920–1926.