Bondlab,
A Toolbox for Mechatronic Design Embedded in Matlab 5

The 1997 BENELUX MATLAB User Conference, October 27-28, Amsterdam, The Netherlands.
Hosted by Scientific Software Benelux.

Willem Minten, Sven Vranckx, Bart De Moor, and Joos Vandewalle
ESAT - Katholieke Universiteit Leuven
Kardinaal Mercierlaan 94, 3001 Leuven (Heverlee), Belgium
E-mail: {Willem.Minten,Sven.Vranckx,Bart.Demoor,Joos.Vandewalle}@esat.kuleuven.ac.be

Keywords
GUI, bond graph methodology, physical systems modeling, software, interactive programs, object oriented models.

ABSTRACT

Bondlab has been set up as a platform independent toolbox in Matlab for mechatronic systems design. Powerful data structure constructs and object oriented facilities of the latest Matlab release are used to guarantee code transparency and maintenance efficiency. In order to retain fast execution of the code the classical vectorization of critical parts is still necessary.
This new tool has the ability to model systems in a behavioral sense, i.e. the modeler does not have to take into account computational considerations of the problem formulation (as is the case with causally constrained block diagram editors like Simulink). To cover this behavioral framework, bond graphs are chosen as the modeling language. This choice makes it possible to (1) exploit the human perception of graphs, (2) to cover the bond graph related analysis aspects of a design, and (3) to connect seamlessly the behavioral and causal model description formalisms that occur in the design of mechatronic systems.
It is the first Matlab based toolbox which makes it possible to integrate the whole design process, from conceptual modeling to product design, in a single software environment.

1. INTRODUCTION

The design of mechahytronic products is characterized by an attempt to consider simultaneously the *mecha*nical, *hy*draulic, electrical/elec*tronic*, and control aspects of the final product during the design cycle. Today the importance of the first step of the design cycle, namely the modeling phase, is largely underestimated. This is due to the extensive use of 'causal' model descriptions. Such causal models are necessary at the end of the design, where control and implementation aspects pop up. In order to manage a design also in the first steps, behavioral models should be used. In such a behavioral framework the designer is not focussed on a certain mathematical realization of a problem, and hence attention can be directed towards a proper model formulation. Bond graphs [2] belong to this family of 'behavioral' model description languages. Moreover, they also possess the property of exploiting the human perception (since they are graphical in nature), and they reveal already in early stages of a design structural properties of the final design. And since the translation of a bond graph model to a causal model can be automated, it is a good modeling tool candidate in the design of mechahytronic products. In this paper a new mechahytronic design environment, named *Bondlab* (an acronym of *Bond* Graph *Lab*oratory), is proposed. It embeds entirely bond graphs in Matlab, hence this environment facilitates the use of both the behavioral and causal model description languages during a design.
The need for such an integrated design environment came up some years ago [7]. Since even now there exists no excellent *integrated* software dealing with (behavioral) modeling, simulation analysis and design together, the need for such a tool was, and still is high. During that time a lot of useful stand-alone software packages (e.g. Camp-G, CAMAS -and his successor 20SIM-, MS1, ... [1, 3, 5]) became available. However they lack integration of all the aspects of a mechahytronic design[1]. During that time the first version of Bondlab was created in Matlab4. Due to the lack of rich data structures, symbolic formula manipulation facilities, and object orientation of the code this task was very cumbersome. With the advent of the Symbolic Math toolbox and release 5 of Matlab [6] the software development became much more transparent. This paper is organized as follows: In Section 2 some implementation aspects of Bondlab are covered. In Section 3 some current features are illustrated with an example. In the last Section some conclusions are made.

[1] In late releases of these packages they aim some port to Matlab. However it is left for the designer to verify the statement that the embedding of a behavioral modeling tool within Matlab is more preferable than several stand-alone software packages.

2. BEHIND THE USER INTERFACE

Two constraints are focussed while embedding Bondlab in Matlab: *fast* and *powerful* code. The term 'powerful' is used for specifications like code transparency, compactness and extensibility. The combination of fast and powerful code gave rise to specific data constructs in Bondlab. The most important ones are the `pointer` type and several kinds of `graph objects`. All this is outlined shortly in the sequel.

2.1. Powerful code

The introduction of enhanced data types in Matlab5 provides structural means for constructing a proper database in Bondlab. Especially the `structure` and the `cell` seemed to be very useful for the construction of transparent, extensible and compact code. Also the new possibility of object orientation improved the readability of the code. However it must be said that the Matlab implementation of object properties like *inheritance* differ considerably of the common usage like in e.g. C++. This difference is shown in Table 1.

parent (MATLAB 5)		*parent* (C++)	
▷	parentfield	▷	parentfield
▽		▽	
child (MATLAB 5)		*child* (C++)	
▷	**parent**	▷	parentfield
▷	childfield	▷	childfield

Table 1: the parent- *and* child-*object type in Matlab5 and C++*

In C++ a method of the *parent*-type has only access to the *parentfield*. When a *child*-object of a *parent*-type is passed to such a method, then the *childfield* is invisible such that the *child*-object behaves just like a *parent*-object. The Matlab implementation however is very awkward: a *child*-object has two fields for a *parent*-method: **parent** and *childfield* (this last one is only *visible*, it can not be manipulated). Consequently, the *parent*-method has to change its behavior with regard to the objects on which it is used. Especially this difference has influenced the implementation and use of objects in Bondlab: Bondlab objects are not inferred by a parent object, they will create a parent object.

2.2. Speed

An important feature of Bondlab is its user-friendly graphical interface. The basic graphical manipulations like *create, move, copy, paste, cut, flip horizontal, flip vertical, rotate, and resize* are provided in such a way that minimal key strokes or mouse manipulations are needed.
In order to be *interactive*, not only the code but also the data structures have to be optimized with regard to speed. It is well known that Matlab code can be speed up by *vectorization* of loops or algorithms. This vectorization treats whole rows of data with a single build-in function. It has been found that inheritance must be abandoned in favor of speed optimization of the code. The most important reason for this is that different types of objects cannot be placed in one row on which vectorization can be applied.

2.3. Pointers

There is a strong need for pointers in many aspects of the implementation of the Bondlab database. Bondlab defines its own pointer type, since there is no pointer mechanism in Matlab. *Pointer* objects, together with their *methods*, are a convenient way to acces a global *cell array* in which the Bondlab database is stored (In Figure 1 this *cell array* is part of the *structure* `pool`, namely `pool.data`). A *pointer* object has a single data field, which is an index for that global *cell array*. The implementation of the *subsref-* and *subsasgn- methods* of the *pointer* objects causes a *pointer* to act as the *cell* itself.

2.4. Graph objects

Bondlab takes advantage of certain *object oriented programming* facilities of Matlab, but does not use a straightforward inheritance mechanism. As already mentioned the main reason for this is that inheritance prevents the usage of vectorization, which is the only way to achieve fast Matlab code.
Bondlab objects are not inherited from a parent object type, they create a parent object. Both the parent object and child object have a pointer to each other (Figure 1 (b)). The parent object stores all the graphical information, no matter what type of object its child is. This means that any graphical manipulation can be done on the parent level, without any effect on the child level. Manipulations on the parent objects can be vectorized since they all belong to the same object type, resulting in a fast implementation.

2.5. Two-layer implementation of the database

The Bondlab database has been implemented as two layers. Figure 1 *(b)* shows these layers containing the objects of the bond graph shown on top of Figure 1 (a), `[db_example]`. The graphical 'parent' objects (nodes and bonds) are placed above the dashed line, the bond graph 'child' objects (0, 1, R, C, I, TF GY) are placed below of it. These objects are all implemented as cells. As already mentioned this layer implementation is beneficial for the manipulation speed of *either* the graph itself *or* the bond graph technical aspects, which in most cases is a separate event.

2.6. The bondgraph object

The most important object in Bondlab is the *bondgraph* object (Figure 1 (b)). The main purpose of this object is being the parent of all the objects that together form a bond graph. However, a *bondgraph* object does not contain any of those objects. Objects belong to the same bond graph when they all refer to the same *bondgraph* object as their parent.
Any function that acts on an entire bond graph needs a *bondgraph* object as a parameter. The objects of that

bond graph are then found as those who have the same parent. The function then continues using those objects.

2.7. The databases `pool` and `db`

All these objects are stored in the first field of the structure pool: `pool.data` (see Figure 1 *(c)*). In order to improve the search speed for the objects in `pool.data`, the other pool fields serve as an interface on which optimized code is used. Beside this single bond graph database, every window is managed by its own *window data base* `db(i)`. This is shown in Figure 1 *(a)*. This database is implemented as a structure array. It contains information about e.g. the grid (size, visibility) and the selected graph objects in the respective window.

3. DESIGNER'S VIEW ON THE INTERFACE

Bondlab solves mechahytronic problems in a very elegant way:

- The model entry is supported by a highly perceptive Graphical User Interface (GUI) (Subsection 3.1.).
- This GUI supports the investigation of the model quality with respect to the final desired design [11].
- Another corner stone in this efficient design environment is the ability to automate the subsequent model transformations throughout a design (Subsection 3.2.).

Parts of this are shown by means of an example, the physical model of which is shown in Figure 2 [8].

3.1. The User Interface

In order to prevent modeling errors, Bondlab is designed according to the standards described in [10]. The interface is extremely user-friendly [12, 13], and, as already shown, the Bondlab database is designed for speed and code transparency. The result is a compact, powerful and highly perceptive interface in which the designer feels comfortable.
Figure 3 shows the main window of Bondlab. In order to minimize the manipulations during the creation or modification of the bond graph by the designer, several push-buttons are provided under the pull-down menus. Keyboard shortcuts for creation and modification purposes are also available (*create, move, copy, cut, paste, delete, flip horizontal and vertical, rotate, and resize*). Every additional setting of Bondlab can be accessed by means of the pull-down menus. The push buttons are organized in 4 groups: (1) the *mode selection*, (2) the *I/O*, (3) the *graph related*, and (4) the *bond graph related* group. The Behavior of Bondlab is governed by the mode selection group. It determines 6 different modes in which the bond graph editor can be set: `selection mode`, `bond mode`, `signal mode`, `element mode`, `zoom mode`, and `properties mode`). In Figure 4 Bondlab is shown in element mode. This figure shows the placement of a transformer while taking advantage of the `AutoConnect` sub-mode. This sub-mode invokes an intelligent routine that automatically connects the transformer to the relevant junctions. Below the drawing window there is a possibility to enter or change the most important properties of the selected element.
Nonlinear behavior can be modeled in several ways, e.g. by means of direct input of relevant Matlab code right away in the value field of the elements. Even user defined `.m` function files can be used in the same straightforward manner. Using the properties mode it is possible to have direct access to all the properties (graphical and bond graph related) of a selected component.
While (or after) drawing the bond graph, parameter labels and values can be entered, and a simplification routine can be invoked to remove e.g. redundant junction structure constructs or to create equivalent components of parts of the bond graph. Thereafter a consistency check can show questionable bond graph parts [9] and remove disconnected elements. Finally, upon designer demand, the causal assignment routines assigns the element statements from the equations.

3.2. Automatic Model Transformations

Several causal analysis procedures can be invoked. The two most important ones are named SCAP (for derivation of the set of first order differential equations, and other related explicit forms like transfer functions, block diagrams, ...) or D-SCAP (for system inversion and additional bond graph related analysis). A symbolic or numerical output can be saved to the Matlab workspace from the *Transformations Tab* in the *Analysis* window, shown in Figure 5.
This output can be a (non-)linear state space form or transfer matrix form, shown in Table 2.
Remark that all this is integrated in Matlab in a flexible way, e.g. simple LaTeX code [4] can accompany the element labels resulting in very readable graphs (This LaTeX mark up code is removed automatically when a symbolic form of the equations is desired), and the derived equations can be visualized as LaTeX code or as Matlab relevant code.
Bondlab also provides internal facilities for direct numerical simulation. Therefore, the set of ODE's can be simulated from within the *Simulation Tab* of the *Analysis* window, shown in Figure 6.
If unavoidable derivative causalities occur, or if some unsolvable nonlinear algebraic loops appear, then Bondlab can automatically create an optimal Simulink2 `.mdl` block diagram, which consequently can also be simulated (from within Bondlab).

3.3. Mechatronic Design Support

At last one basic tool, the energy based model reduction, will be explained shortly.
After simulation, the power distribution in the bond graph can be visualized in the bond graph [8, 5]. Bondlab achieves this e.g. by changing the width of the bonds according to the instantaneously transmitted power. This is especially useful for the reduction of nonlinear systems, in which the nonlinear behavior in time as function of an

$$\begin{bmatrix} \dot{p}_3 \\ \dot{p}_7 \\ \dot{q}_{11} \\ \dot{q}_{12} \\ \dot{q}_{13} \end{bmatrix} = \begin{bmatrix} -\frac{R2}{J2} & 0 & 0 & 0 & \frac{kr\ \cos(th)\, r2}{2} \\ 0 & -\frac{R1}{J1} & 0 & 0 & -\frac{kr\ \cos(th)\, r1}{2} \\ 0 & 0 & -\frac{k_t2}{R_t2} & 0 & \frac{kr\ \cos(th)}{2\ R_t2} \\ 0 & 0 & 0 & -\frac{k_t1}{R_t1} & \frac{kr\ \cos(th)}{2\ R_t1} \\ -\frac{r2\ \cos(th)}{2\ J2} & \frac{r1\ \cos(th)}{2\ J1} & \frac{k_t2\ \cos(th)}{2\ R_t2} & \frac{k_t1\ \cos(th)}{2\ R_t1} & -\frac{kr\,(\cos(th))^2}{4}\left(\frac{1}{R_t2}+\frac{1}{R_t1}\right) \end{bmatrix} \cdot \begin{bmatrix} p_3 \\ p_7 \\ q_{11} \\ q_{12} \\ q_{13} \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} T_2 \\ T_1 \end{bmatrix}$$

$$\begin{bmatrix} e_9 \\ f_9 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1/2\, kr\ \cos(th) \\ 0 & \frac{r1}{J1} & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_3 \\ p_7 \\ q_{11} \\ q_{12} \\ q_{13} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} T_2 \\ T_1 \end{bmatrix}$$

$$\frac{f_9}{T_1} = \frac{num(s^4)}{den(s^5)} = \frac{51/100 * (s+10) * (544 * s^3 + 6522 * s^2 + 10742 * s + 9945)}{17/400 * (s+10) * (4 * s + 7) * (1088 * s^3 + 13044 * s^2 + 23473 * s + 39780)}$$

Table 2: State space form (symbolic) and transfer function description (numerical) of the bond graph of Figure 2. These equations (LaTeX format) are copied straight from the Matlab workspace.

input shape should be evaluated. In Figure 7 a snap-shot of such an animation based analysis is shown. In this example it turned out that 96% of the power-picture remains the same when the tape dynamics are neglected. This observation makes it possible to reduce the bond graph from a fifth to a third order system, while maintaining the most important dynamics. For this (linear) model it turns out that the fastest poles are omitted in this reduction step. Notice that the power of this model reduction approach originates from the fact that it has been executed in the physical domain and hence, does not depend on a certain mathematical description of the problem. Therefore this method can be directly applied to nonlinear systems in which e.g. the input shape dependency has to be taken into account.
Classical methods (i.e. by means of minimal rank approximations with singular values or by means of eigenvalue decomposition) are not appropriate for this.

4. CONCLUSIONS

`Bondlab` is an integrated tool in Matlab 5 for the design of mechatronic products. Its flexibility and user-friendliness are two of its most appreciated aspects. The used concepts are based on bond graphs, which makes it possible to exploit the nature of physical systems (behavioral) and the human perception (graphs). Other functionalities include automatic computer support for the model transformations and the ability to interrogate the models with regard to the final design (mechatronics). The seamless implementation of this tool is done as Matlab 5 `.m` files, which makes this tool platform-independent (PC, UNIX workstation, ...) From an implementational point of view one can summarize that the data structures are created with regard to available data structures, code transparency, and speed considerations. Especially for this last constraint it turns out that a pure object oriented solution seems at this time not applicable in Matlab5.

ACKNOWLEDGMENTS

This work is supported by several institutions: the Flemish Government (Concerted Research Action GOA-MIPS, the FWO (Fund for Scientific Research - Flanders: G.0292.95, G.0256.97), the FWO Research Communities (ICCoS, Advanced Numerical Methods for Mathematical Modeling), the IWT (Flemish Institute for Science and Technology in Industry: IWT/VCST project (1992-1997), IWT Action Program on Information Technology (ITA/GBO/T23) - IT-Generic Basic Research: ISIS)), the Belgian Government (Inter-university Attraction Poles: IUAP P4-02 (1997-2001) and IUAP P4-24 (1997-2001)), and the European Commission (Human Capital and Mobility Network SIMONET; SCIENCE-ERNSI SC1-CT92-0779; TMR Network ALAPEDES; Keep In Touch SYSIDENT-KIT124). The scientific responsibility is assumed by its authors.

REFERENCES

[1] Granda J.J., and F. Pourrahimi, *Computer Graphic Techniques for the Generation and Analysis of Physical System Models*, Artificial Intelligence, Graphics, and Simulation, Proceedings of the Western Simulation MultiConference, (G. Birtwistle, ed.), SCS Publishing, pp.70-75, 1985.

[2] Karnopp D.C., Margolis D.L., and Rosenberg R.C. 1990, *System dynamics: a unified approach. Second edition.*, John Wiley & Sons, New York, USA, (first print in 1975).

[3] Kleijn C., "Twente Simulation Homepage", Controllab Products B.V., `http://www.rt.el.utwente.nl/20sim/clp.htm`, (1996).

[4] Lamport L., *LaTeX: a document preparation system*, 2nd Edition, Addison-Wesley Publishing Company, (1995).

[5] Lorenz F., Goffart M., *Direct representation of Simulation results on the Bond Graph Model*, Proc. of the 3rd International Conference on Bond Graph Modeling and Simulation, Part of the WMC'97 Multiconference (ICBGM'97), Phoenix, Arizona, Jan., pp. 20-25. (1997).

[6] The Mathworks Inc, "Matlab release 5", 24 Prime Park Way, Natic, MA 01760-1500, USA, `http://www.mathworks.com`, (1996).

[7] Minten W., Vanvuchelen P., De Moor B., *Modeling of an electronically driven continuously variable transmission*, Proc. of the 1993 International Conference on Bond Graph Modeling and Simulation (ICBGM'93), San Diego, California, Jan. 1993, pp. 223-228.

[8] Minten W., De Moor B., and Vandewalle J. *Power and Power Variables in Graphical Modeling of Physical Dynamic Systems*, ESAT-SISTA, report TR96-92, K.U.Leuven, Belgium, (1996).

[9] Minten W., De Moor B., and Vandewalle J. 1996 *How Can Different Interpretations and Use of Bond Graph Parameters be Avoided.*, J. Franklin Institute, Vol334b, No3, pp.389-406, (1997).

[10] Minten W., De Moor B., Vandewalle J., *Condensed Bond Graph modeling*, Proc. of the 3rd International Conference on Bond Graph Modeling and Simulation, Part of the WMC'97 Multiconference (ICBGM'97), Phoenix, Arizona, Jan., pp. 50-55. (1997).

[11] Minten W., Vranckx S., De Moor B., Vandewalle J., *Bondlab, a Matlab Based GUI for Bond Graph Modeling*, Accepted for the special issue of Journal A, September, (1997).

[12] Preece J., Keller L., *Eds*, *Human-Computer Interaction*, Prentice Hall, UK, (1994).

[13] Tognazzini B., *Tog on Interface*, Addison Wesley, New York, USA, (1991).

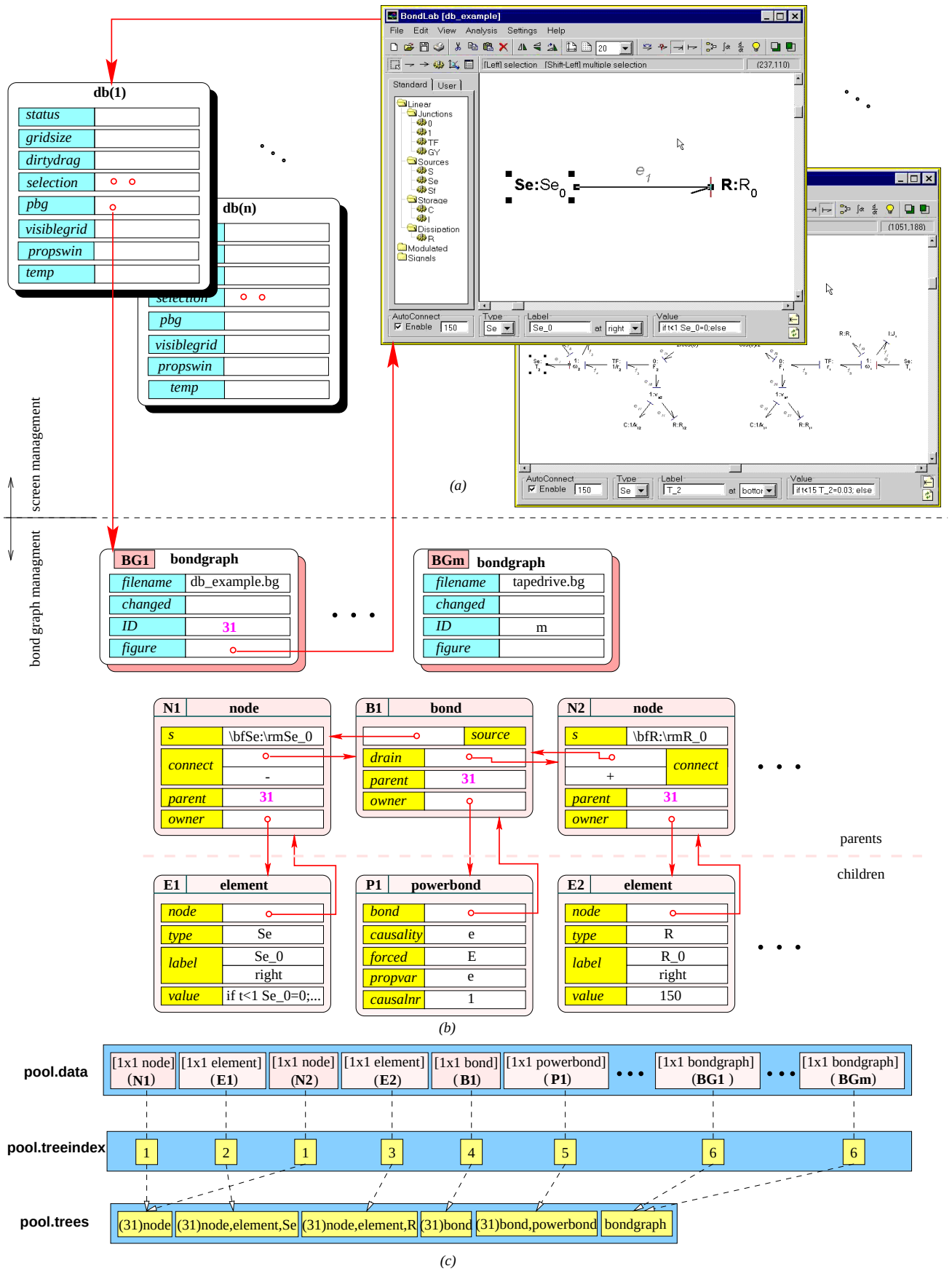


Figure 1: Internal structure of Bondlab. (a) Window manager databases, (b) partial view on the cells in `pool.data`*. These cells cover graphical aspects (nodes and bonds) or bond graph related aspects (elements, powerbond, bondgraph). (c) View on the* `pool`*. The arrows visualize pointers.*

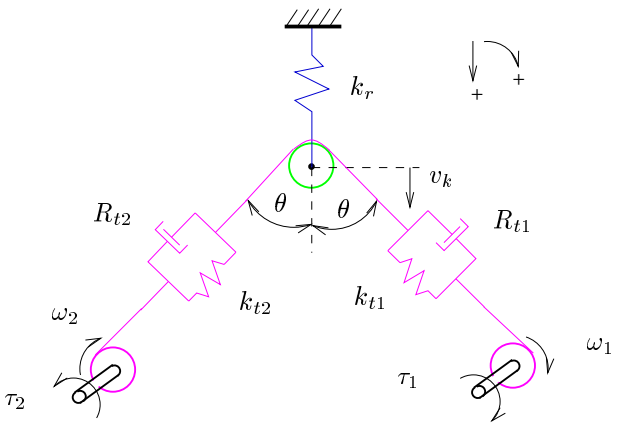


Figure 2: Physical model of two coupled drives, the drive side is located on the right hand side. A tape (index t) is transported from the left to the right. The tape tension is partially determined by the spring k_r.

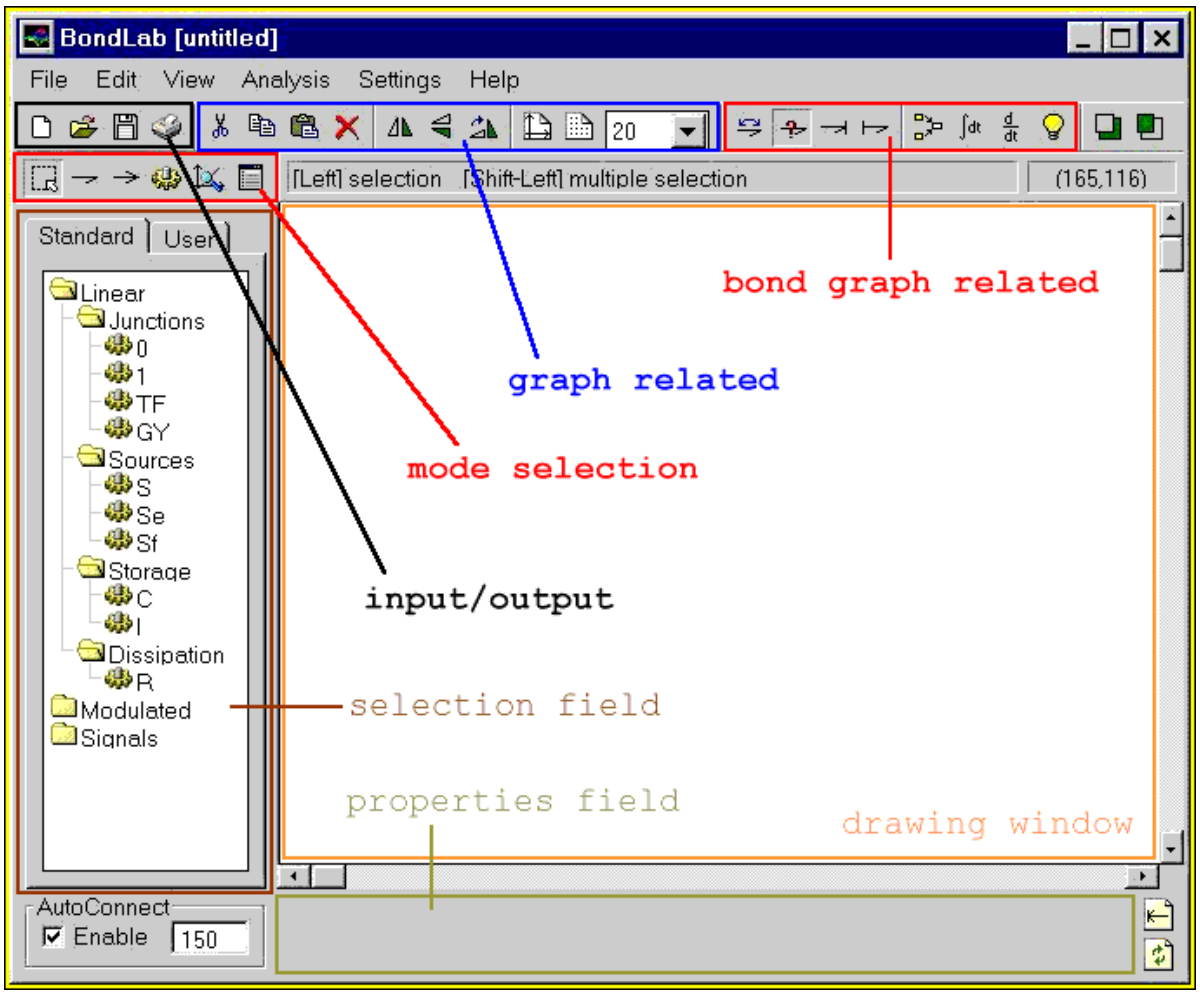


Figure 3: Screen-shot of the main window of Bondlab (reduced color-map).

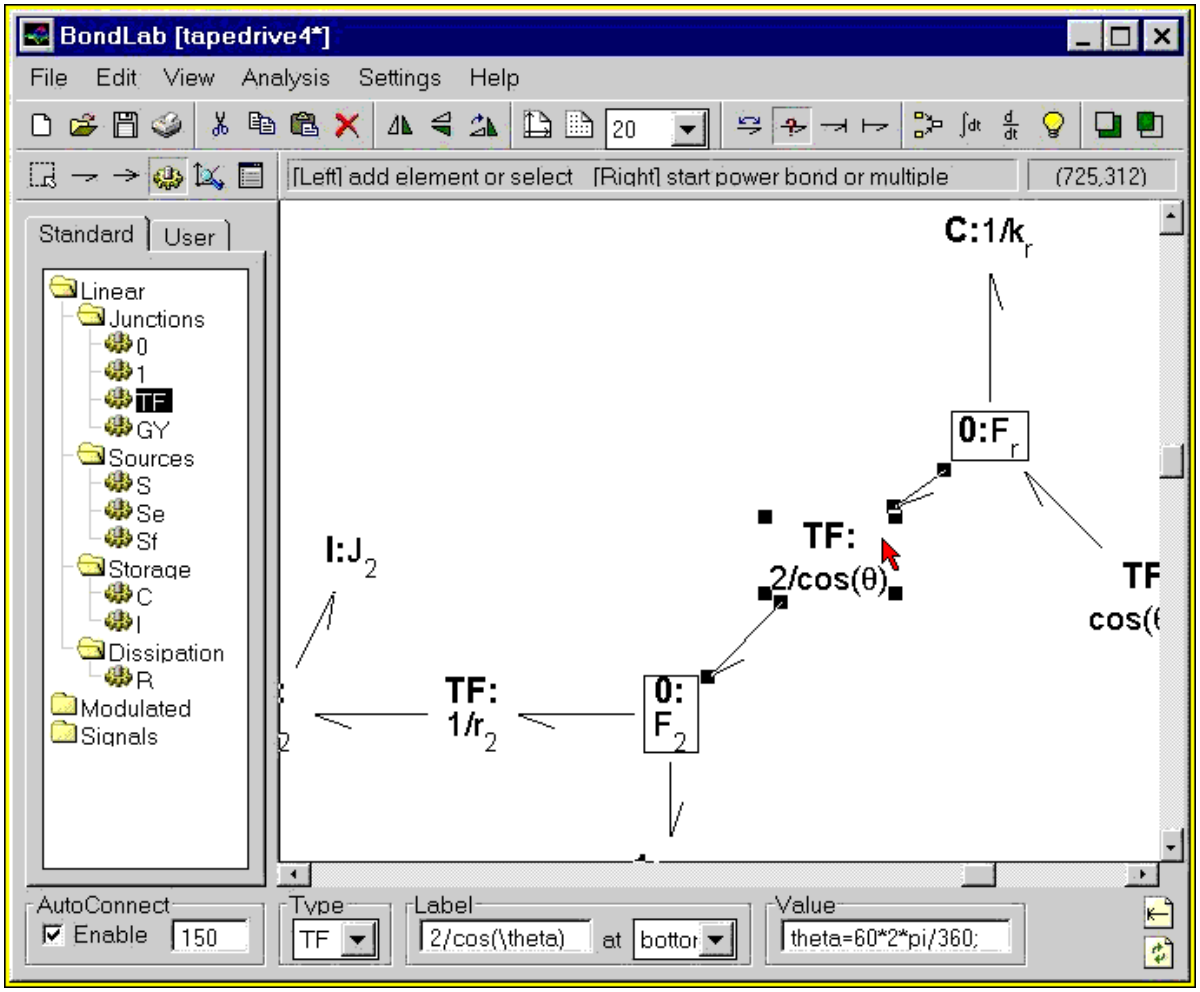


Figure 4: Placement of a transformer in `AutoConnect` *mode.*

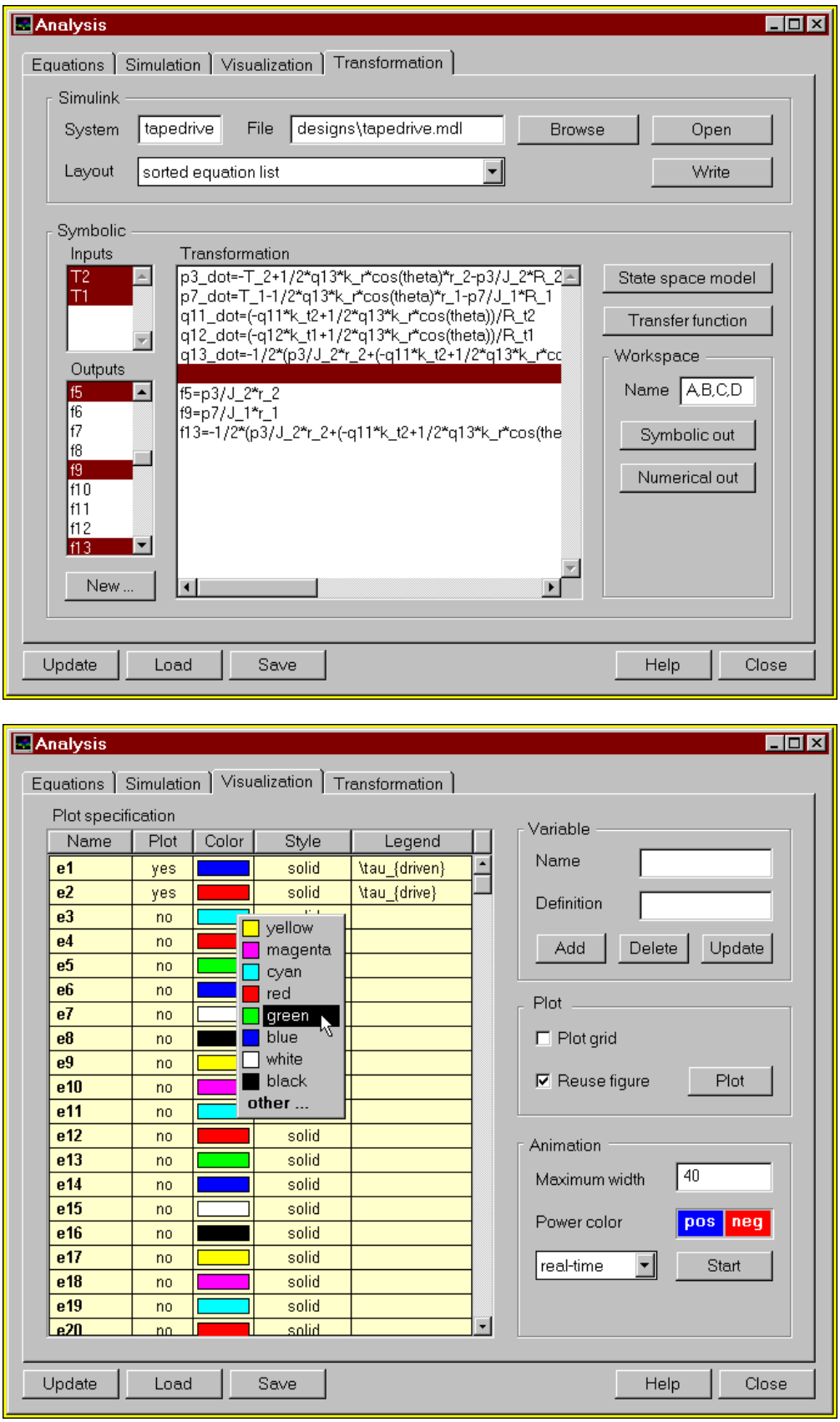
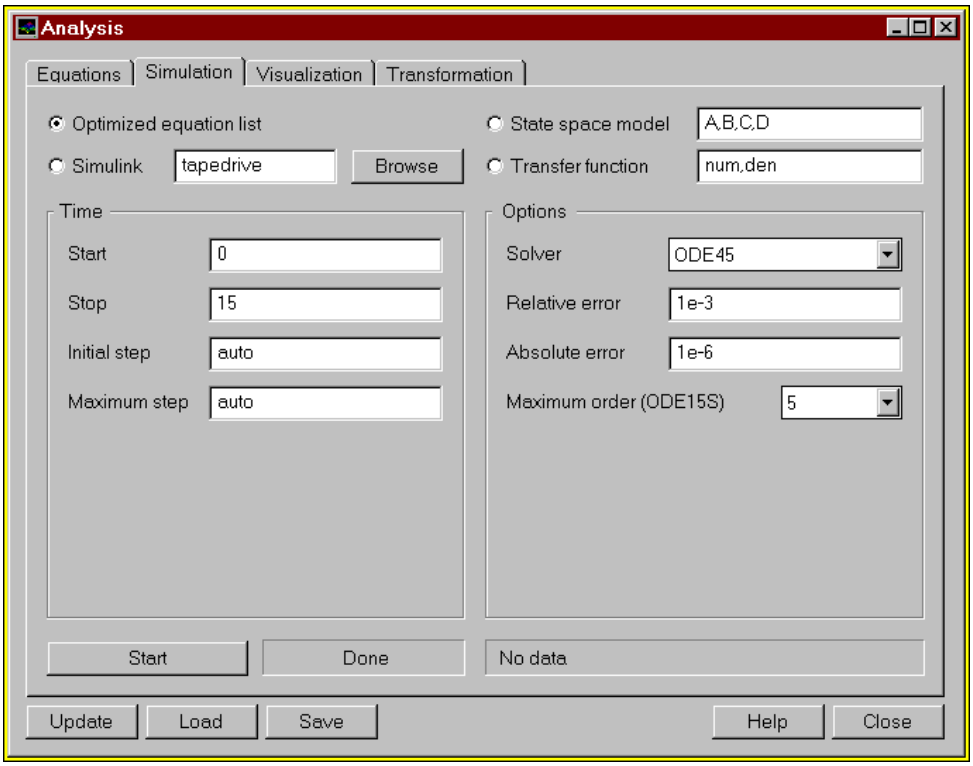


Figure 5: (top) The Analysis window, Transformations Tab. Input and output variables can be chosen freely from the two lists on the left. The state and output equations and the transfer functions are calculated accordingly. The push-buttons to the right of the window are used for symbolic or numerical export of the equations into the Matlab workspace.
(bottom) The Analysis window, Visualization Tab.



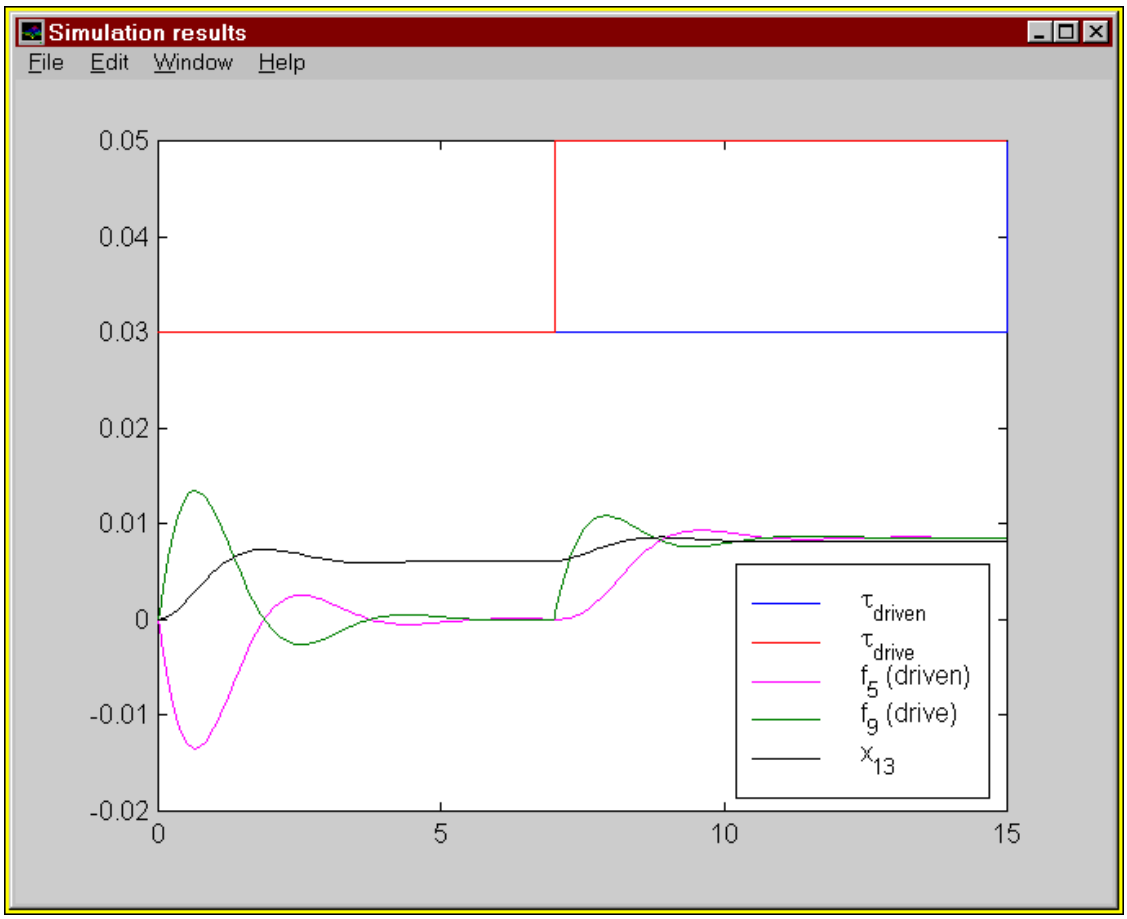


Figure 6: (top) The Analysis window, Simulation Tab.
(bottom) Simulation results of the coupled drives: The tensioning of the tape $(0 - 7sec)$ and the roll up $(7 - 15sec)$ simulation results are visualized in this plot window.

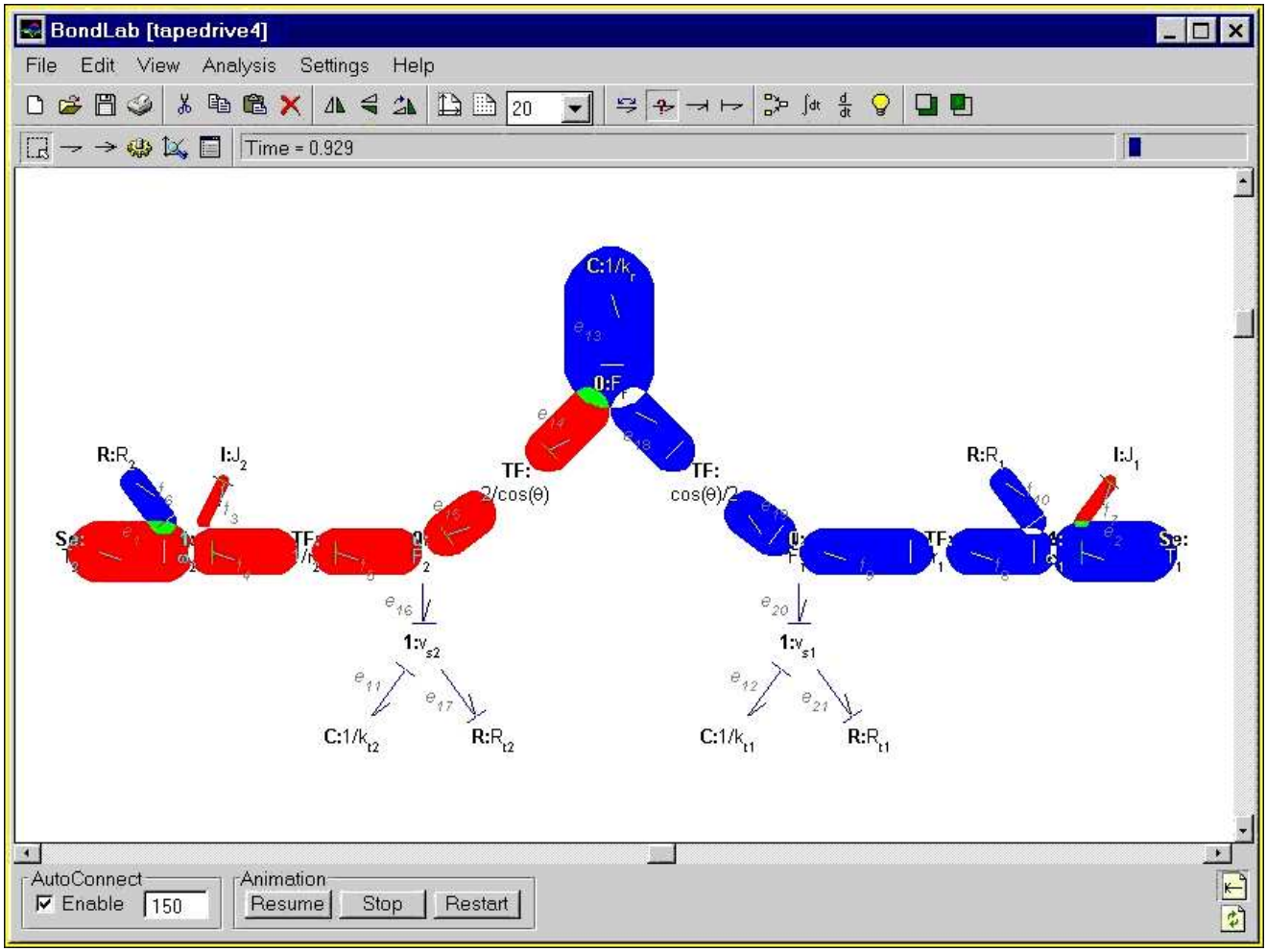


Figure 7: Snap-shot at 0.929sec of the animation of the energy flow. This figure shows the power transmitted in each bond. The width of each bond is proportional to the instantaneously transmitted power. A red (bright) line depicts negative power with respect to the power arrow reference, a blue (dark) one depicts positive power.