Numerical Test Rig for Large-Scale and Interconnected Dynamical Systems

submitted
Project Laboratory
Networked and Cooperative Control
by

cand. ing. Francisco Llobet   cand. ing. Jose Rivera

born on July 9, 1985   born on December 18, 1986
resident: resident:
Briennerstr. 39   Amalienstr. 87
80333 München   80799 München

Institute of
Automatic Control Engineering
Technische Universität München

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss
Univ.-Prof. Dr.-Ing. Sandra Hirche

Abstract

The goal of this project was to develop a test rig for large-scale and interconnected dynamical systems. The result is MTIDS or Matlab Toolbox for Interconnected Dynamical Systems, which is a mash-up that wraps different toolboxes used for graph analysis and dynamic systems simulation together. MTIDS allows the definition and analysis of graphs, where each node has a specific dynamic assign to it. The template based design of nodes' dynamics allows great flexibility for the creation of complex interconnected dynamical systems with the possibility of implementing clusters/layers. MTIDS is an open-source project under the GNU GPL v2 license. This document presents a general description of MTIDS and instructions for its use.

Zusammenfassung

Ziel dieses Projektes war es, eine Testvorrichtung für grosse, gekoppelte dynamische Systeme zu entwickeln. Das Ergebnis ist MTIDS oder Matlab Toolbox for Interconnected Dynamical Systems. Diese Toolbox verbindet Toolboxen für die Graphenanalyse mit Toolboxen für die Simulation von dynamischen Systemen. MTIDS ermöglicht die Definition und Analyse von Graphen, bei denen jeder Knoten eine spezifische dynamische Anweisung ausführt. Das Template basierte Design der Knotendynamik erlaubt eine große Flexibilität für die Erstellung von komplexen, gekoppelten dynamischen Systemen. Es besteht beispielsweise die Möglichkeit Cluster bzw. Layer zu implementieren. MIDS ist ein Open-Source-Projekt unter der GNU GPL v2 Lizenz. Im Rahmen des nachfolgenden Berichts wird eine generelle Beschreibung von MTIDS sowie deren Benutzung gegeben.

Contents

Chapter 1

Introduction

In this first Chapter the motivation behind the MTIDS project is explained and the project's goal and framework is presented.

1.1 Motivation

Large-scale interconnected dynamical system are everywhere: biological systems, power and water distribution systems, the brain, social interaction networks, economic markets, etc. In a canonical form all of this systems can be thought as a bunch of nodes with local dynamics that interact with each other, e.g. a graph. Different topologies of the graph, may lead to different behavior. An example of various large scale interconnected systems can be seen in Figure 1.1.
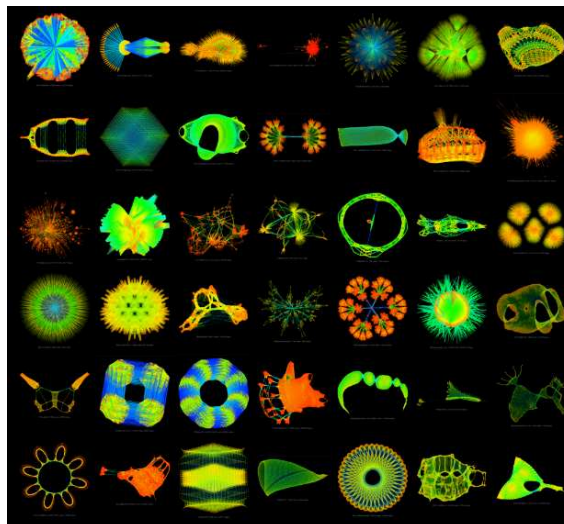

Figure 1.1: Visualization of various large scale systems using the sfdp algorithm © Dr. Yifan Hu of AT&T Labs

There are many tools available for the analysis of interconnected dynamical systems,

for example, in power systems you have PSSE and Power Factory. However, this simulation programs are normally very system specific and in most cases it takes a long time to learn how to use them correctly. The difficulties are specially noticed while testing control concepts, where small changes on the topology of the grid or control concept could lead to a painful redesign of your simulation set up. You may actually end up spending the most of your time in the implementation of a simulation. A more general and easy to use solution for the simulation of interconnected dynamical systems is needed.

1.2   Idea and Goal

MTIDS (Matlab Toolbox for Interconnected Dynamical Systems) is a project that aims to design an easy to use and flexible toolbox to make the simulation of large scale dynamical systems easier for students and researchers. The **goal** is to produce a mash-up that wraps different toolboxes used for graph analysis and dynamic systems simulation together into a framework.

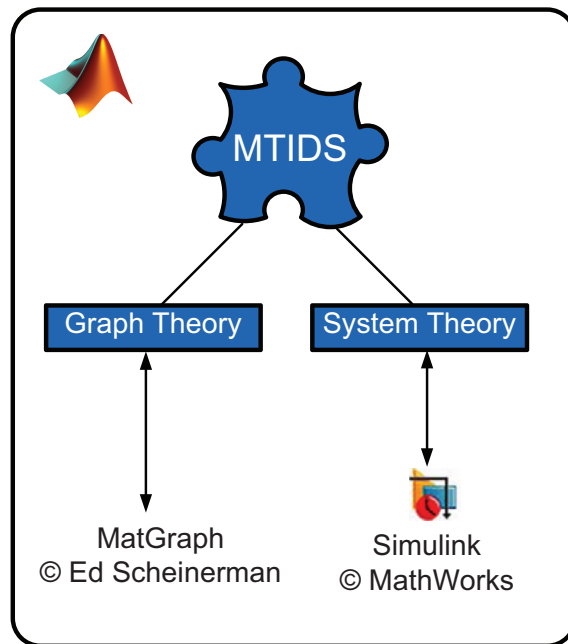Figure 1.2: MTIDS: Matlab Toolbox for Interconnected Dynamical Systems

As we can see in Figure 1.2 MTIDS runs in the MATLAB environment and is basically a GUI that allows the interaction of tools used in graph theory and control theory. For graph theory we use Matgraph a toolbox design by Prof. Scheinerman of the John Hopkins University [Sch06] and for dynamical simulations we use Simulink[Mat11].

1.3 Framework

The current framework of MTID is made out of three basic components. A GUI
(**mtids.m**) an export to simulink function (**exportSimulink.m**) and an import
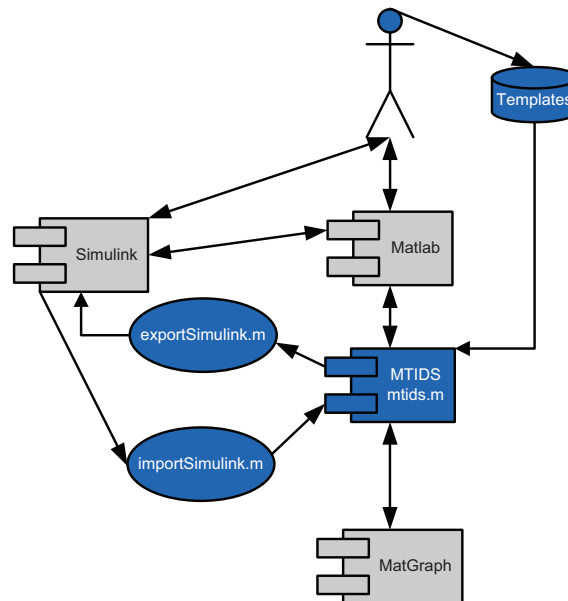from Simulink function (**importSimulink.m**).

Figure 1.3: MTIDS: Components diagram

In Figure 1.3 we can see that the most important component is the user, specially
its head. The better you are at producing templates and interacting with Matlab
and Simulation the more functional MTIDS is going to be for you. In a nutshell
MTIDS works as follows:

- GUI (mtids.m) runs inside Matlab.

- GUI interacts with Matgraph: create, modify and visualize graphs.

- System Inteconnector (SI): exportSimulink.m and importSimulink.m called
  from GUI to interact with Simulink.

- Templates done by User in Matlab/Simulink.

- Simulations done in Simulink.

Chapter 2

Graph Theory

2.1 Algebraic Graph Theory

Graphs are an easy way of representing networked structures. A graph consists of:

1. **Vertices:** Represent nodes, dynamical subsystems and/or agents.

2. **Edges:** Represent connections, links or couplings between vertices.

In this project, we consider only simple, undirected and unweighted graphs.

Algebraic graph theory describes graph properties using numbers (like invariants) and equations. It is important first, to convert the graphical representation of a graph into a mathematically sound description like matrices. Once a the corresponding matrices (like the Laplacian or adjacency matrices) have been derived, we can calculate it's properties [HE11]
The use of linear algebraic tools like rank, eigenvalue and eigenvector extraction of graph matrices is called *Spectral Graph Theory*

Degree of a vertex and degree vector

The degree of a vertex is the number of edges that are connected to it. The vector whose i-th component is the degree of the i-th vertex is the degree vector $\vec{d}$. The degree vector can analysed using basic statistical tools for for mean, variance, min, max, span, etc.

Graph heterogenity

Graph heterogenity is the quotient between the standard deviation of the degree vector $\vec{d}$ and the mean of the degree vector.

$$H = \frac{\sqrt{var(\vec{d})}}{\bar{\vec{d}}} \tag{2.1}$$

It describes the heterogenity between the connections to the nodes.

Maximal number of edges and graph density

A complete graph is a graph where every vertex is connected to every other vertex. The number of vertices inside a complete graph is:

$$N_{e_{MAX}} = \frac{1}{2} N_v (N_v - 1) \tag{2.2}$$

Alternatively:

$$N_{e_{MAX}} = \frac{1}{2} \left(N_v^2 - N_v \right) \tag{2.3}$$

Graph density is a number which describes the *completeness* of a graph. It is defined as the quotient between the actual number of edges N_e and number of edges of a complete graph $N_{e_{MAX}}$ with the same number of vertices:

$$D = \frac{N_e}{N_{e_{MAX}}} \tag{2.4}$$

$$D = 2 \frac{N_e}{N_v^2 - N_v} \tag{2.5}$$

2.1.1   Graph matrices

Adjacency Matrix

The adjacency matrix is defined as a N_v x N_v matrix whose $(A)_{ij}$ component is equal to one if there is a connection between vertex i and vertex j or zero if not.

Degree Matrix

A degree matrix is a diagonal matrix whose diagonal $(D)_{ii}$ components are equal to the degree of the i-th vertex. All other components for $(D)_{ij}$ $i \neq j$ are zero.

$$(D)_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ N_v(i) & \text{if } i = j \end{cases} \tag{2.6}$$

Laplacian Matrix

The Laplacian matrix is defined as the degree matrix D minus the adjacency matrix A:

$$L = D - A \tag{2.7}$$

The Laplacian matrix is used to characterize the algebraic connectivity of a graph.

2.1.2   Algebraic connectivity and Fiedler vector

Algebraic connectivity

For sorted the sorted eigenvalues of the Laplacian matrix $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{N_v}$ we define the algebraic connectivity as the first nonzero eigenvalue. It's corresponding eigenvector is the Fiedler vector. The algebraic connectivity is a meassure of how connected a system is. For example, in the consensus problem, the algebraic connectivity describes the rate of convergence of the system to it's final state.

Number of connected subgraphs

The multiplicity of the zero eigenvalue ot the Laplacian matrix is the number of connected subgraphs inside the system. This is also the rank of the nullspace of the Laplacian matrix $Rank(Null(L))$. From the rank-nullity theorem of linear algebra we know that the rank of a matrix and the rank of it's nullspace add up to the number of collumns. Therefore for a square matrix, the rank of the nullspace can be calculated using:

$$Rank(Null(L)) = Rank(L) - N_v \tag{2.8}$$

2.2   Creating and visualizing systems in MTIDS
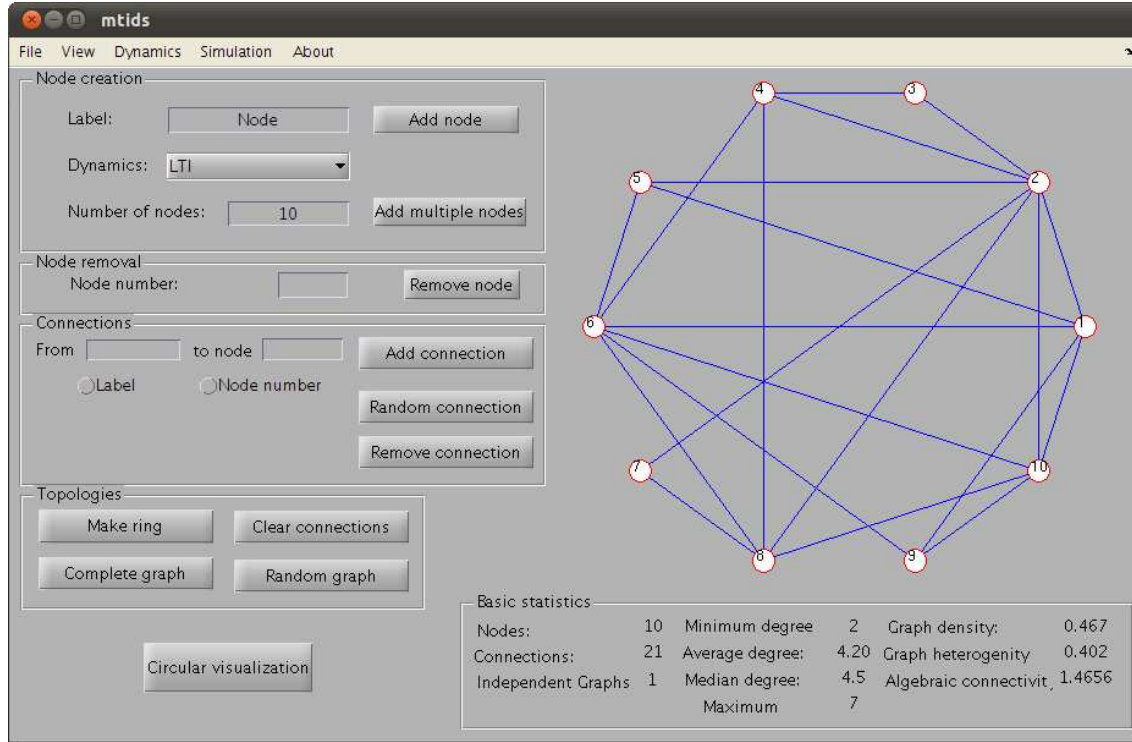
2.2.1   MTIDS GUI Overview

Figure 2.1: The MTIDS Graphical User Interface

The MTIDS GUI is divided in two parts: The left side contains the controls that allow for modification of the system and the right side contains the graph and statistics visualization.

2.2.2   Building an interconnected system in MTIDS

Adding nodes

When adding a node, the User has to assign a unique node label and node dynamics at the "node creation" panel Nodes can be added with the "add node" button or by (`<Shift>-<Left-click>`) with the mouse. Nodes added by using the mouse are given the last selected node dynamics and node label.

Adding multiple nodes

Adding multiple nodes at the same time is possible by introducing the number of nodes to be added and by clicking the "Add multiple nodes" button. These nodes start all with the same name (eg. node1, node2,...) and share the same dynamics.

Creating and removing connections

Connections can be added or removed using:

- **Mouse:** By `<Ctrl>-<Left click>` on both nodes. If no connection exists, it will create one, otherwise it will delete this connection.

- **Connection subwindow:** Connections to be added or removed can be specified by using node label or node number.

- **"Edit node" dialog:** See Subsection 2.2.3.

Connections in MTIDS are bidirectional.
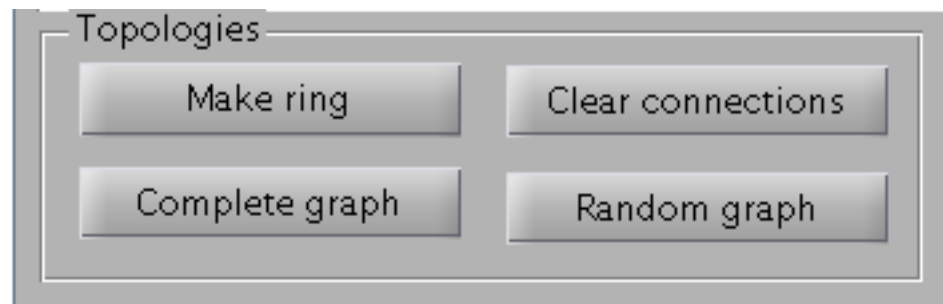
Default topologies

Figure 2.2: The default topologies

- **Complete graph:** Makes the graph into a complete graph.

- **Remove connections:** Removes all edges inside the graph.

- **Make ring:** Connects all nodes into a ring.

- **Random graph:** Uses the random graph function from Matgraph. It adds random connections between nodes up to a density of ca. 0.5.

2.2.3 Node properties

Node properties can be edited by `<double-clicking>` a node in the graph window. Following node properties can be changed: Node label, node dynamics (from list) and the connection list vector in Matlab format. The selected node can also be removed using this window.
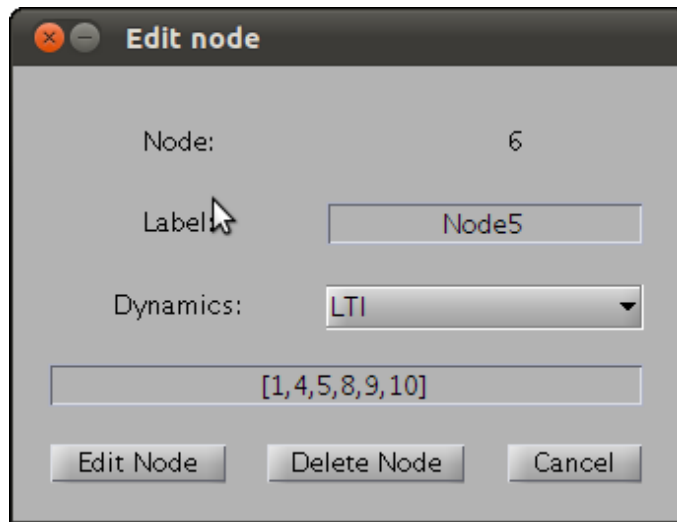
Figure 2.3: The node properties window

2.2.4   Load and Save

A system can be saved by selecting **File → Save As...**. This saves the graph proper-
ties (adjacency matrix), node labels, assigned node dynamics and template list into
a binary .mat file. Loading an already saved system for modification is possible by
selecting **File → Load...**. This loads the selected .mat file which contains the system
properties.
Remember that a saved system file contains the template names previously im-
ported so reimporting dynamics is not necesary. If a template .mdl file is missing or
was deleted, the exporting to Simulink functionality will not work correctly.

2.2.5   Exporting matrices to Matlab

Select the corresponding workplace inside Matlab, variable name and matrix type
to export. Supported matrices are Laplacian matrix, adjacency matrix and edge
list.

2.2.6   Importing graph matrices into MTIDS

Graph matrices like Laplacian matrix, adjacency matrix or edge list. The matrix
type is automatically detected and replaces the current graph. Node dynamics are
assigned based on the last selected template.

Chapter 3

System Theory

In this Chapter we explain the design and simulation capabilities that MTIDS offers for interconnected dynamical systems.

3.1 MTIDS Concepts for Simulink Models

MTIDS has two different concepts for building Simulink models. The difference relies on the amount of inputs that each node has.

In the first concept, **concept 1**, each node has the same amount of inputs as there are nodes on the graph, even if the input is not connected. This concept is very efficient for matrix based description of system dynamics, where you describe the whole system at once, like with LTI modeling.

In **concept 2**, each node has only an input for the nodes that are connected to it. This is very efficient for agent or subsystem based descriptions, where the dynamics of each node may not allow a matrix based description of the whole system. This concept is used for the examples of consensus and fireflies synchronization.

3.1.1 Nodes and connections

Node:

Each node in MTIDS is a subsystem block in Simulink. Each node must have a unique label. In **concept 1** each node has an in-port for every node in the graph and a single out-port. In **concept 2** each node has an in-port only for nodes connected to it and a single out-port.

Connections:

Any connection made inside MTIDS is bidirectional and unweighted. As Simulink connections are directed, each MTIDS bidirectional connection is represented with

two directed connections between nodes. To implement weighted graphs, weights on branches have to be realized in the node's dynamic model, e.g inside the subsystem. Another option is to define junction nodes.

3.2   Export to Simulink

In order to export the model created in the MTIDS GUI to simulink: **Simulation** → **Export to Simulink 1...** for **concept 1** or **Simulation** → **Export to Simulink 2...** for **concept 2** (see Figure 3.2).
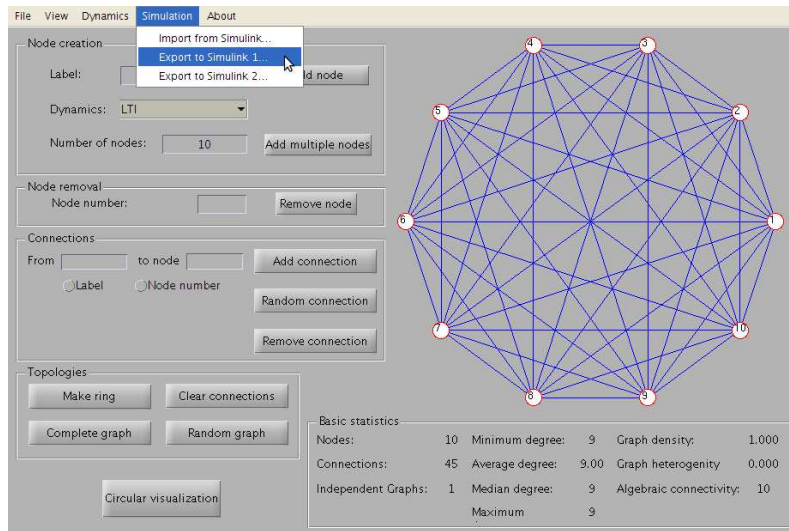

Figure 3.1: MTIDS: export model to Simulink in the MTIDS format 1. Example is a complete graph with 10 LTI nodes.

The MTIDS GUI then calls the function exportSimulink.m, which builds a Simulink model with the following information: model name, list of nodes' dynamics templates, list of templates that are available in mtids, adjacency matrix, position of the nodes and nodes' names. The result is a Simulink model in the MTIDS format.

To adjust the size of the model to the Simulink window size: **View** → **Fit System To View**

In Figure 3.2, the nodes are ordered in a circle, the topology of the system defined in MTIDS remains. The circle arrangement is a design decision, made, in order to allow a better access to the nodes. Each one of the nodes is a subsystem block. The dynamic of the nodes is defined inside the subsystem block. Moreover, depending on the MTIDS export concept choosen, the number of inputs of a node
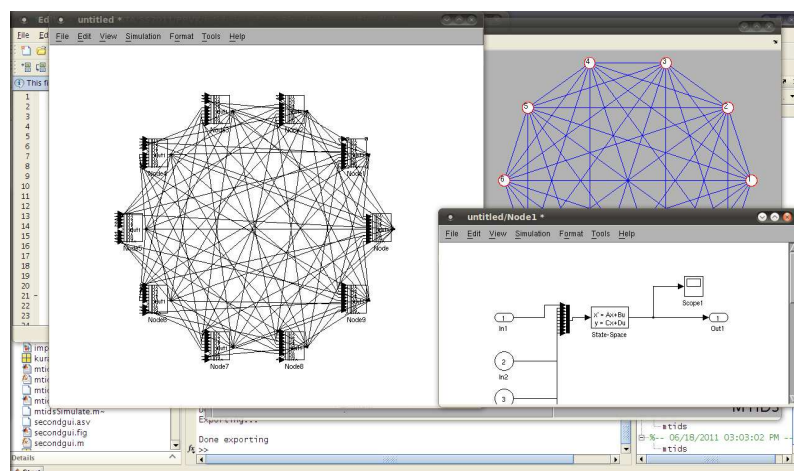
Figure 3.2: Interconnected system Simulink Model. Example is a complete graph with 10 LTI nodes.

will be different, compare with a zoom of the first node for our example in Figure 3.3.

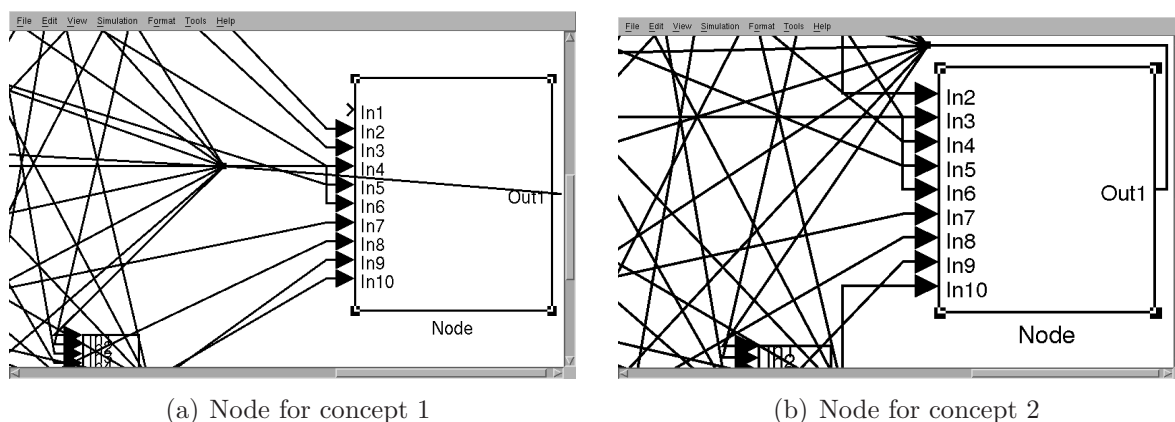(a) Node for concept 1   (b) Node for concept 2

Figure 3.3: Node inputs and output: (a) Each node has an in-port for every node in the model and a single out-port. (b) Each node has an in-port for nodes connected to it and a single out-port. Example is the first node of a complete graph with 10 LTI nodes.

Notice that for concept 1 the port corresponding to the node's number should always be free, e.g. the first node has an unconnected in-port 1, the second node an unconnected in-port 2, etc. For concept 2 all in-port are always connected. In case that a loop of a node with itself is required, we recommend doing this inside the subsystem/node. Another important aspect to point out is that when simulating the system, Simulink will issue a warning for unconnected in-ports and set their input to the subsystem to 0, this means that an **unconnected port enters the**

subsystem (local dynamics of the node) with a value of zero during simulations. This can be exploited to make the parametrization of the nodes' dynamics easier.

3.3   Nodes' Dynamics

As mentioned in the past section 3.2 each node defined in MTIDS is exported to Simulink as a subsystem. The exportSimulink.m function uses predefined dynamic templates, which are modified according to the, in MTIDS, defined topology. Next, we explain how to define your own templates and show how to use 2 preloaded templates: the LTI template and the kuramoto template. This templates can be found in the **/templates** directory.

3.3.1   Build your own Template

The real power of MTIDS depends on your ability to build templates.

Figure 3.4: Scratch template for node's dynamics (**interfaceTemplate.mdl**).

In Figure 3.4 we can see the components that each template should have: an in-port connected to a mux and an out-port. Between the mux and the out-port you can design your own custom dynamics. The input to the system comes from the mux block as vector, which is composed by the values entering the node/subsystem. The output of your system should also be aggregated to a vector and routed to the out-port. The separation of the different inputs and outputs is left to the system's designer. With a well thought architecture complex systems are very easy to achieve, please refer to the LTI and kuramoto template examples.

The dynamic of a node can be as simple as a junction that only reroutes the incoming signals or more complicated to include controllers and systems inside of it. It is this feature that allows the implementation of clusters or layered systems, see subsection 3.4.

3.3.2 The LTI template

The LTI template is an example that defines a linear time invariant dynamic for nodes. The mathematical model of a simple interconnected LTI system as explained in [Sil78] can be written as:

$$
\begin{pmatrix} \dot{x_1} \\ \vdots \\ \dot{x_N} \end{pmatrix} = \begin{pmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & \ddots & \vdots \\ A_{N1} & \cdots & A_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} + \begin{pmatrix} B_1 & \cdots & B_N \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix} \tag{3.1}
$$

or

$$
\dot{x} = Ax + Bu \tag{3.2}
$$

The local view of a single subsystem/node (here for the first node) is:

$$
\dot{x_1} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1N} & B_1 & \cdots & B_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ u_1 \\ \vdots \\ u_N \end{pmatrix} \tag{3.3}
$$

this can be reformulated as

$$
\dot{x_1} = A_{11}x_1 + \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1N} & B_1 & \cdots & B_N \end{pmatrix} \begin{pmatrix} 0 \\ x_2 \\ \vdots \\ x_N \\ u_1 \\ \vdots \\ u_N \end{pmatrix} \tag{3.4}
$$

In order to keep things simple, we define the local output of each node as

$$
y_i = x_i, \text{ for } i = 1, \ldots, N. \tag{3.5}
$$

To implement this dynamic as a template we make use of the State-Space block. The LTI template shown in Figure 3.5 can be used in MTIDS to create interconnected LTI systems with different topologies. One way to avoid the work of parameterizing the A,B,C and D matrix of every node separately, is to define the same matrix for all subsystems. Using formula (3.4) and exporting with **concept 1** we can define an A, B, C, and D matrix for every node. An example can be

$$
\begin{aligned}
A &= 1 \\
B &= [-1 \cdots -1]^{1 \times N} \\
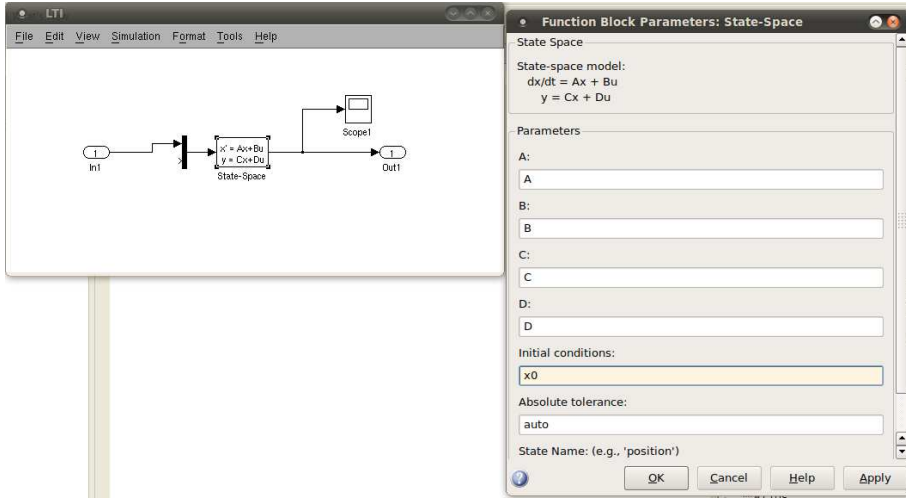C &= 1 \\
D &= [0 \cdots 0]^{1 \times N}
\end{aligned}
$$

Figure 3.5: LTI template, **/templates/LTI.mdl**

As the in-port corresponding to the node itself is automatically set to zero by Simulink during the simulation, we can define the same matrices for all nodes without getting any errors.

For more complicated LTI models we recomend creating many different LTI templates.

3.3.3 The Kuramoto Template

The kuramoto template is an example of a non-linear dynamic for a node. This kuramoto model is used to model the behavior of coupled oscillators. One of the interesting behaviors that may occur in this type of systems is synchronization. The mathematical equation of N coupled oscillators is written for each oscillator i as

$$\dot{\theta}_i = \omega_i + \frac{K}{N}\sum_{j=1}^{N}\sin(\theta_j - \theta_i). \tag{3.6}$$

In Figure 3.6 we see the MTIDS's kuramoto template after formula (3.6). The goal was to model fireflies synchronization, thus the model has an embedded function that changes the color of the parent block for a defined threshold, this mimics the flashing of a firefly. With the right parameters one can build a synchronizing firefly colony using MTIDS. Remember that the kuramoto.mdl template needs to be added in MTIDS: Dynamics → Add mdl template. You also need to set up the right parameters for synchronization, an example can be seen in Figure 3.7. In order to see the blinking in the correct time scale, you need to set up the simulation with a fixed step size in Simulink, more on this in section 3.5. An **example** can be found under **demos/firefliesTemplate.mdl** with the parameters **firefliesTemplate.mat**.
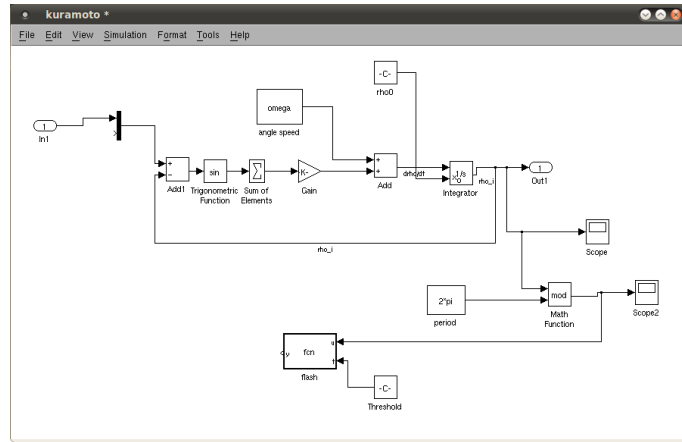
Figure 3.6: Kuramoto template for fireflies synchronization, **/templates/kuramoto.mdl**
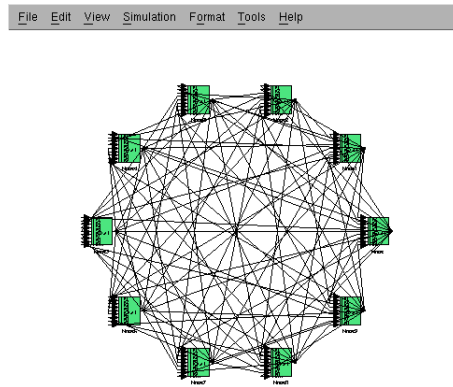

Figure 3.7: Simulation of 10 fireflies synchronizing, exported using concept 2. Parameters: K=0.1, N=10, $\omega = 0.3$, $\theta_0 = 10$, var=5 (variation of starting point), threshold= $3/4 * 2\pi$

3.3.4 The Consensus Problem Templates

The consensus or rendevous problem is a classic formulation where we see how a distributed hostless algorithm with simple local interactions can achieve a well understood and studied global interaction.

A graph built in MTIDS using consensus nodes should be exported using the **Export to Simulink mode 2**

Follower node:

The consensus follower template (/templates/consensus.mdl) contains a simple local dynamical model using standard Simulink blocks (see Figure 3.8). This model is

the realization of the local (decoupled) model is (as in [HE11]):

$$\dot{x}_i = -\gamma \sum_{j=1}^{N_v(i)} (x_i - x_j) \tag{3.7}$$

Every internal integrator state is initialized on random before the simulation. As
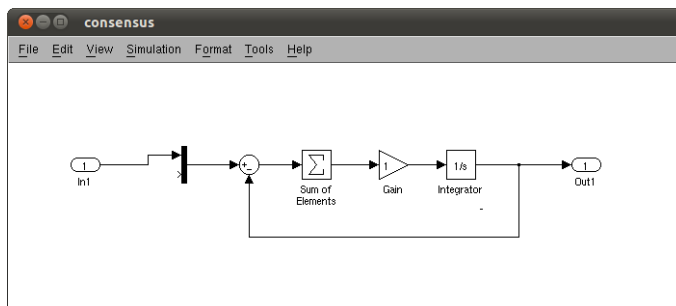
Figure 3.8: Consensus template, **/templates/consensus.mdl**

an example we built in MTIDS a graph containing 6 consensus follower nodes as shown on Figure 3.9. The results of the simulation are shown on Figure 3.10. It is easily recognizable that all nodes tend to a common state.
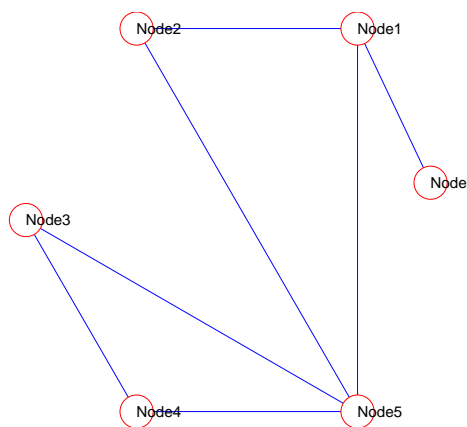

Figure 3.9: Consensus Graph Example
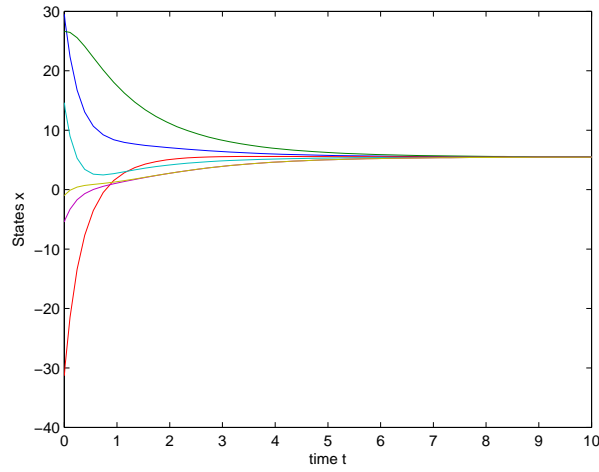
Figure 3.10: Consensus State Plot

Static Leader:

A static leader node is a node which stays on a specific point in state-space. It has following dynamics:

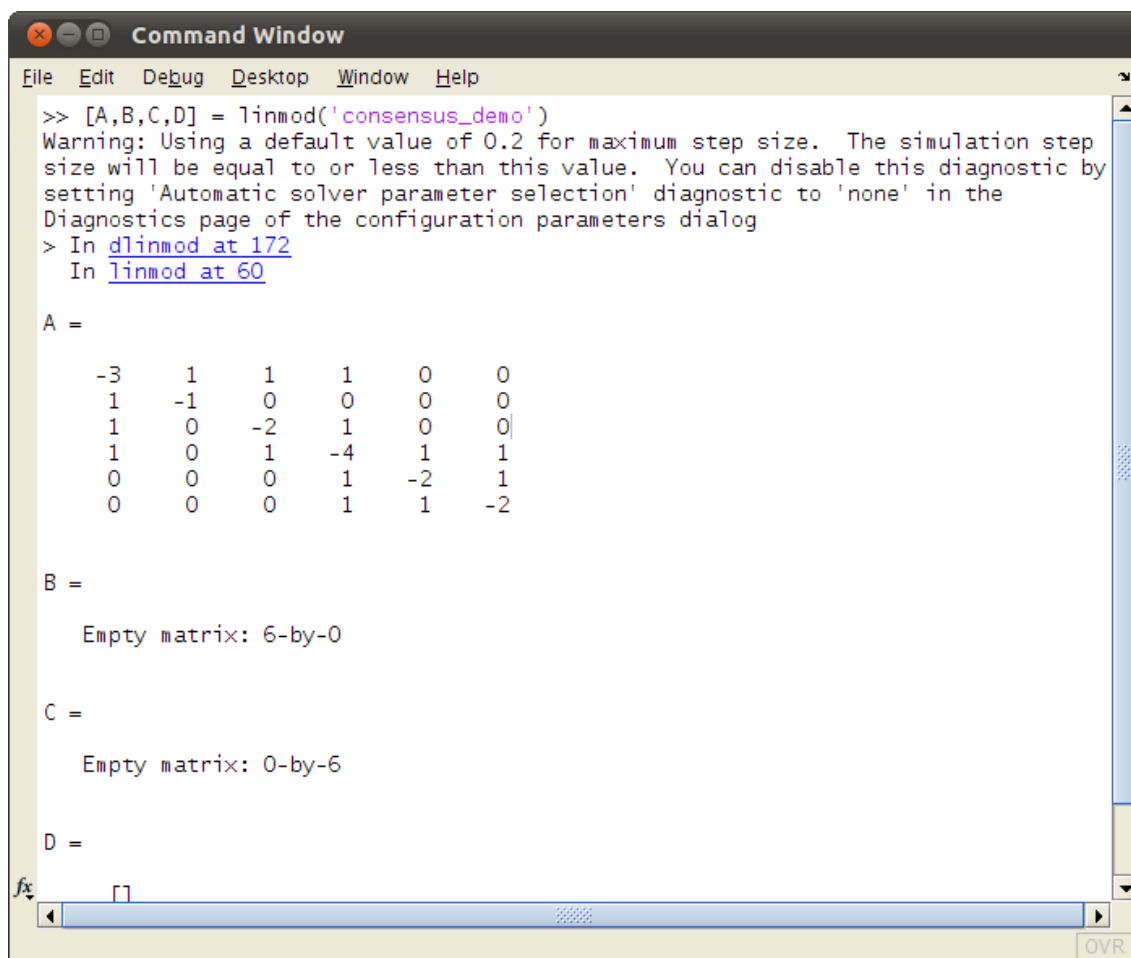$$\dot{x}_i = 0 \tag{3.8}$$

$$x_i(t) = x_0 \tag{3.9}$$
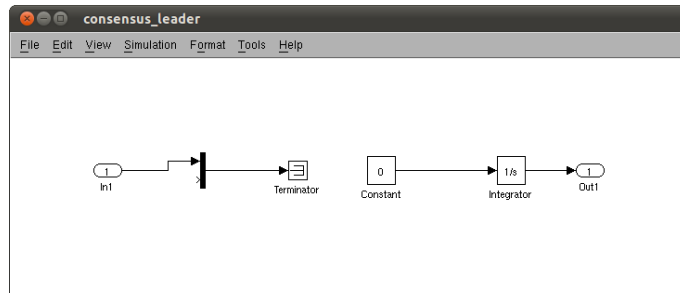
Figure 3.11: Consensus leader template, **/templates/consensus leader.mdl**

linmod() Example with the Consensus Problem:

Matlab function linmod() extracts a continuos linearized state-space model of a system around the operating point. Every block inside the main system is linearized individually. The outputs of this function are the standard A,B,C,D matrices of an LTI-model. Usage of linmod: `[A,B,C,D] = linmod(sys)`.

We see that the result shown in Figure 3.12 correspond to the negative of the Laplacian matrix of the graph shown in Figure 3.9. The output of linmod() can be used as a LTI-system which to be analysed using the Matlab Control Toolbox for stability, controller synthesis, etc.

Figure 3.12: linmod() example output

3.4 Layering/Clustering Nodes

The **key** to implement layering and clustering in MTIDS is the **interface node**. On clustered/layered interconnected systems there is always an interface that regulates the data communication of one layer/cluster to other layers/clusters. The interface has access to all nodes in the current cluster/layer and on most cases does some information processing (compression, principal component extraction, etc). This processed information is used to interact with other clusters/layers. Inside the cluster/layer the interface node acts like a normal node.

3.4.1 Create a layer/cluster

In oder to create a layer/cluster in MTIDS you need to export your system as a layer: **File → Export as a layer 1** for concept 1 or **File → Export as a layer 2** for concept 2.

MTIDS will ask you to select a template for the interface node. You may select an interface node created by you or select the MTIDS provided interface node : **layerInterface.mdl**.
The result is a Simulink model that differs from the one that MTIDS normally produces, as it includes an extra node, the interface node. If we look more closely in Figure 3.13, we notice that this model also fulfills the requirements needed to build a standard node template, see subsection 3.3.1. By saving this model and using it as a node template in MTIDS we can build many of this interconnected systems, which interact with each other through their interface nodes, see Figure 3.14. An **example** can be found under **demos/clusterDemo.mdl** with the parameters **clusterDemo.mat**.
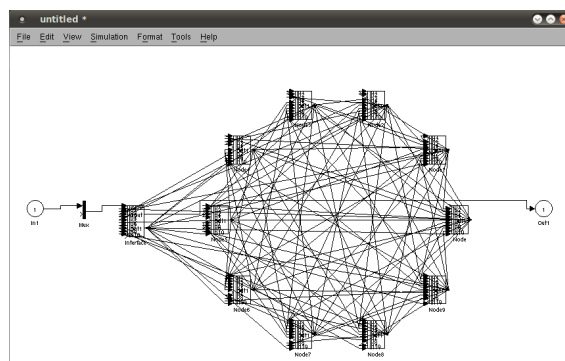

Figure 3.13: Resulting layer/cluster when exporting in MTIDS as layer. Example is a complete graph with 10 LTI nodes, build using concept 1.
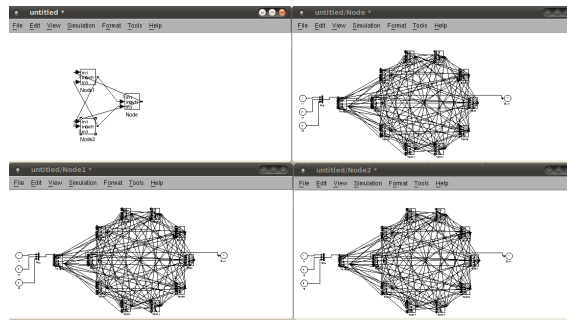
Figure 3.14: Resulting model of layered/clustered interconnected systems. Example uses a complete graph of 3 nodes where each node is an interface/cluster of a complete graph with 10 LTI nodes plus 1 interface node.

3.4.2   Build your own Interface Node

The user may define your own interface node template using the interface node scratch template (**interfaceTemplate.mdl**).
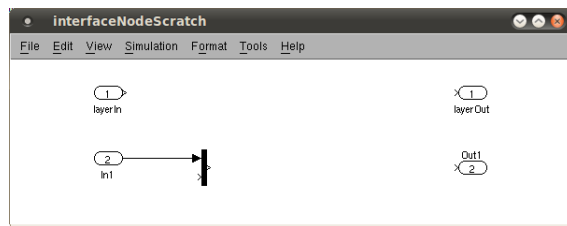

Figure 3.15: Interface node, scratch template model : **interfaceTemplate.mdl**.

As we see in Figure 3.15 the interface node requires a layer in and out port numbered as 1, which interacts with other layers/clusters. Moreover it requires the basic components of a node template (an in-port connected to a mux and an outport) in order to interact with the nodes in the cluster/layer. MTIDS also comes preloaded with a simple aggregating interface template called **layerInterface.mdl** found under **/templates**.

3.5   Working in Simulink

Here we review a couple of basic concepts which you will need to run and analyze the simulation of the interconnected dynamic systems produced by MTIDS. This is only a quick overview, for more detail please go to : Mathworks Online Simulink Tutorial or [ABRW09].

3.5.1 Simulation Parameters

The parameters of your model can be defined in Simulink by hand. However, a better way is to define the parameters in the Simulink model as variables and give a value to this variables in the Matlab workspace. This also allows you to save this parameters by saving the workspace as a mat file with the command: `>> save name.mat`

3.5.2 Solver

Before running a simulation you need to define the simulation runtime and the numerical solver that Simulink will use to simulate your model. In Simulink: Simulation → Configure Parameters, and then going to the solver tab, you may define different types of solver, see Figure 3.16.
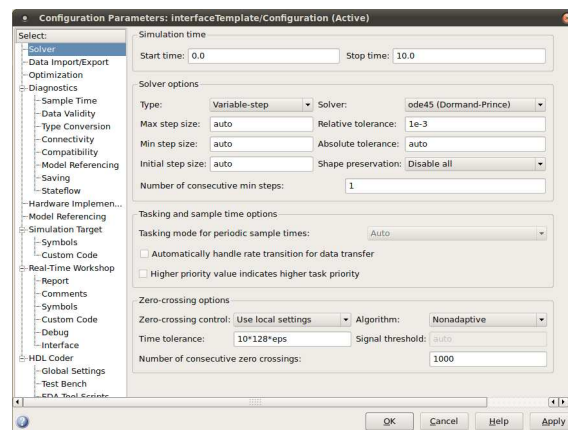

Figure 3.16: Simulink define solver

It is important to notice that if you want a correct time scale during simulation runtime, you need to define a fixed step size. This is used for the demo of fireflies synchronization to see the blinking of fireflies in the correct time scale.

3.5.3 Visualize Simulation Data

The easiest way to visualize data during simulations in Simulink is to use scope blocks. The data can then be seen in real-time during simulation. Another way to do this in real-time, is to use the 'To File' block or the 'To Workspace' block, which allow a direct interaction with Matlab. Plotting functions can be defined in Matlab to interact with the simulation to show the data in real-time.

The other option is to visualize the results after the simulation. For this, you can define in Simulink which data should be send to the workspace under: Simulation → Configure Parameters in the tab of Import/Export, see Figure 3.17. Here, it

is useful to export the states, which Simulink defines as the values coming out of integrator blocks, but it is also possible to define them by hand.
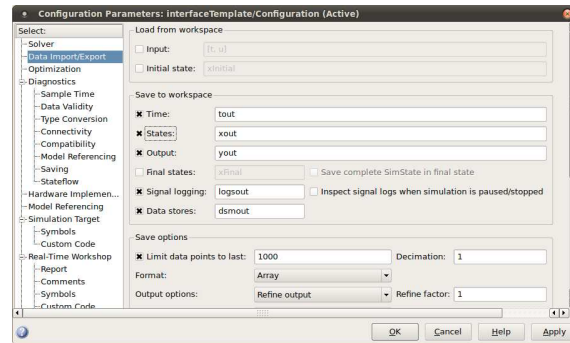

Figure 3.17: Simulink define import/export data

3.5.4   Working with a closed Model

Simulations can also be run from Matlab without the need to open the Simulink model. This is particularly useful for large Simulink models, like models of over 100 nodes that you may produce with MTIDS. You can write a Matlab script to parametrize and run the simulation. **Examples** can be found on the **help for the sim command**.

3.6   Import from Simulink

MTIDS also offers the possibility of importing Simulink models which were produced by MTIDS or are constructed following the MTIDS format, see section 3.1.

Notice that the dynamic of the imported system nodes will be the one that you have selected in the MTIDS GUI.

Node labels, position and topology of the subsytems are imported from Simulink to MTIDS. To do this in MTIDS: **Simulation → Import from Simulink** , see Figure 3.18. Finally, select the model you wish to import.
Sometimes the nodes are imported in a different order, to correct this click on the **Circular Visualization** button.
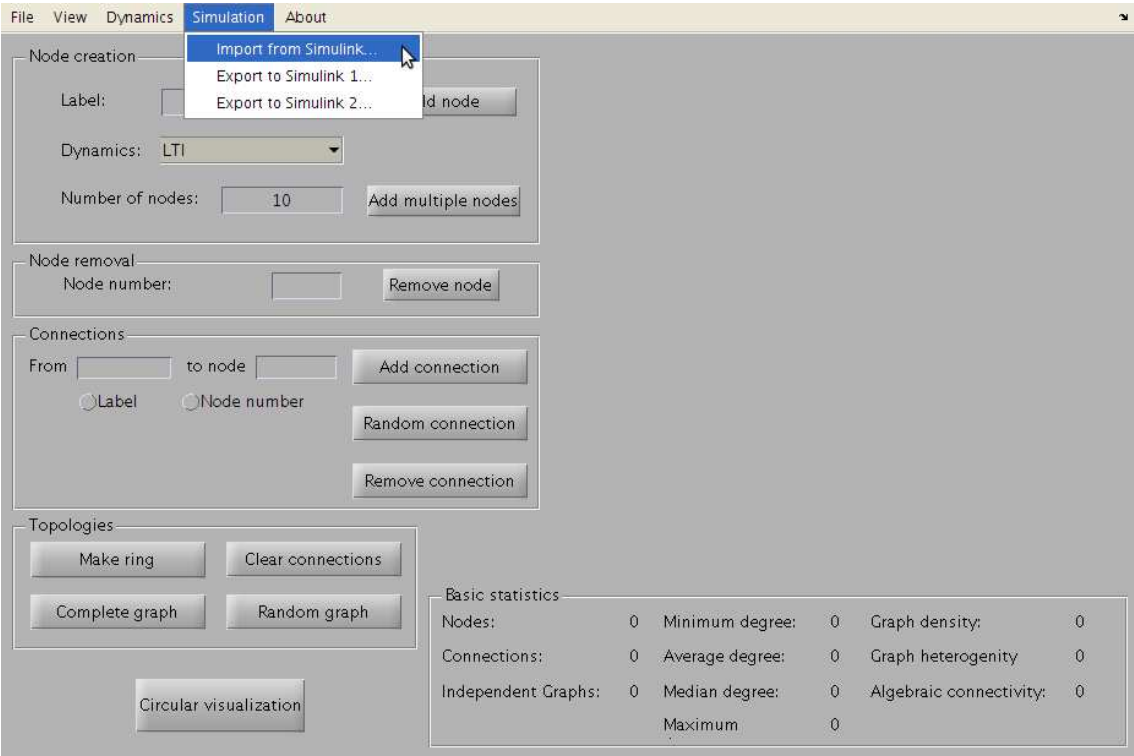
Figure 3.18: MTIDS import from Simulink

Chapter 4

Future Development and Conclusion

4.1 Future Development

MTIDS is a toolbox aimed for students and researchers working in the field of large-scale and interconnected dynamical systems. We aim to make this a free and open source tool which can be accessed and improved by the open-source software community in order to push research on the field of networked control system forward.

There are still some issues and functions which will improve or be implemented in MTIDS:

- Support for full import from Simulink.

- Support for switching and random interconnections.

- Export to Simulink "wizzard".

- Support for directed graphs.

- GUI for interaction with the Control Toolbox.

4.2   Conclusion

In this project we developed a fully functioning toolbox which allows for visual creation and simulation of inteconnected dynamical systems. MTIDS (Matlab Toolbox of Interconnected Dynamical Systems) is a wrapper that combines tools used for graph theory analysis (Matgraph) and for the simulation of interconnected systems (Simulink) to create a flexible and extensible framework.

As MTIDS is based on standard tools used by academia and industry, the User doesn't have to leave the familiar and well documented enviroment of Matlab. MTIDS capabilities allow for fast setups which makes it an excelent tool for demonstrations (for example in courses on networked control systems), rapid prototyping for   networked   and   distributed   controller   design   and   many   more   uses.

MTIDS features summed up:

- Full integration with Matlab and Simulink for simulation and analysis.

- Easy Drag and Drop visual system building and connection.

- Visualization of key graph properties.

- Nonlinear and switching behavior possible at node level.

- Highly customizable thanks to templates.

- Layering and clustering capabilities.

- Extensible.

MTIDS is available at:

http://code.google.com/p/mtids/

List of Figures

Bibliography

[ABRW09] Anne Angermann, Michael Beuschel, Martin Rau, and Ulrich Wohl-
          farth. *Matlab-Simulink-Stateflow*. Oldenbourg Wissenchaftsverlag, Mu-
          nich, Germany, 6th edtition edition, 2009.

[HE11]   Sandra Hirsche and Sebastian Erhart. *Lecture Notes for Networked
          Control Systems*. Lehrstuhl für Steuerungs- und Regelungstechnik TU-
          München, Munich, Germany, 2011.

[Mat11]   Mathworks. *Matlab Documentation*. The Mathworks Inc., Natick, USA,
          2011.

[Sch06]   Ed Scheinerman. *Matgraph by Example*. Johns Hopkins University,
          Baltimore, USA, 2006.

[Sil78]   Dragoslac D. Siljak. *Large-Scale Dynamic Systems*. North-Holland, New
          York, USA, 1978.