Extensions to Numerical Test Rig for Large-Scale and Interconnected Dynamical Systems (MTIDS)

submitted
Project Laboratory
Networked and Cooperative Control
by

cand. M.Sc. Ferdinand Trommsdorff

born on September 11, 1979
resident:
Rotdornstr. 10c
85764 Oberschleissheim
Tel.: 089 44 23 77 34

Institute of
Automatic Control Engineering
Technische Universitat München

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

Supervisor: F. Deroo, D. Xue, H. Mangesius
Start: 18.10.2011
Submitted: 20.01.2012

Abstract

The Matlab Toolbox for Interconnected Dynamical Systems (MTIDS) provides methods for a quick simulation of connected subsystems with different local dynamics. To improve the capabilites of the toolbox, some extension are needed, for example, the support of directed graphs or the assistance in visualizing the local state behaviour. MTIDS allows the definition and analysis of graphs, where each node has a specific dynamic assigned to it. MTIDS is an open-source project under the GNU GPL v2 license. This document describes the new features and changes of MTIDS.

Zusammenfassung

Die Matlab Toolbox für gekoppelte dynamische Systeme (MTIDS) stellt Methoden für eine schnelle Simulation von miteinander verbundenen Agenten mit unterschiedlichen lokalen Dynamiken zur Verfügung. Um die Fähigkeiten dieser Toolbox zu verbessern, sind einige Erweiterungen nötig, wie zum Beispiel die Unterstützung von gerichteten Graphen oder eine Erleichterung zur Erstellung von Zeitdiagrammen, die den Verlauf der lokalen Zustände zeigen. Damit ermöglicht MTIDS die Definition und Analyse von Graphen, bei denen jeder Knoten eine spezifische dynamische Anweisung ausführt. MIDS ist ein Open-Source-Projekt unter der GNU GPL v2 Lizenz. Im Rahmen dieses Dokuments werden neue Features und Änderungen von MTIDS beschrieben.

Contents

Chapter 1

Introduction

In this chapter, the field of application, in which MTIDS can be used, is discussed. A selection of different improvements is presented and the new features which have been implemented, are described.

1.1 Where MTIDS is helpful

There are many examples about systems which are interconnected and which do exchange informations in between the local systems. Technical systems are for example the world wide web and all kinds of supply networks like electric power, water or gas. The cohesions at some economic markets like the Stock Exchange and many biological systems can also be described by a general model of local subsystems with a specific dynamical behaviour which are often called agents. Another example is the human brain, where neurons are coupled via synapses, or the behaviour of flocks of birds or shoals. The amount of vertices and edges in such kind of networks ranges from hundreds and thousands up to a magnitude of 10^{11} of neurons in a human brain. Visualizing of large scale networks is a detached area of research, see [Hu10]. Todays' real world control problems are often leading to these kinds of large scale dynamical systems which cannot be tackled by classical control and system theory. In this case, a numerical simulation is a heuristical method to show the dynamical behaviour of a system. For most technical sytems, very specific software exists which are often proprietary, complex to learn and time-consuming to setup simulations. For these reasons, the Matlab Toolbox for Interconnected Dynamical Systems (MTIDS) was developed as an open source project. It is a mash-up of various toolboxes of Mathworks' *Matlab* [Mat11] and a redistributed version of Ed Scheinermans' open source toolbox *Matgraph* [Sch06]. The structural scheme for MTIDS can be seen in figure (1.1). With it, a quick design of graphs and agents with different dynamical behaviors is possible, including an immediate simulation.

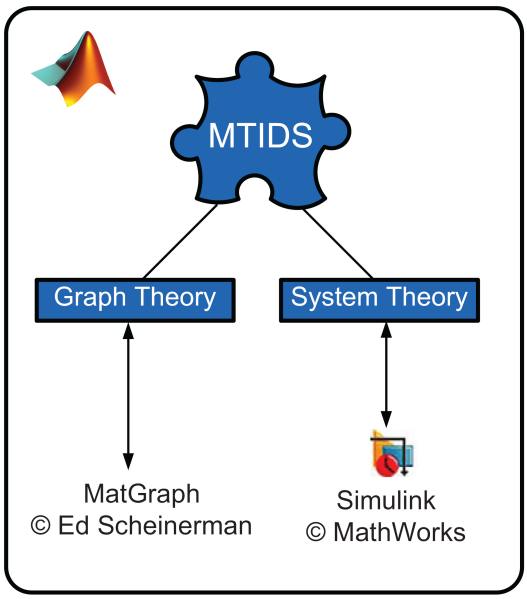

Figure 1.1: The framework of MTIDS: Matlab Toolbox for Interconnected Dynamical Systems

1.2 Selection of possible improvements

MTIDS was created during the Project Laboratory of Networked and Cooperative Control at the TU München. Due to the character of a laboratory, many topics cannot be handled promptly. Many possible improvements were collected during the laboratory and afterwards. Some of them are more easy to implement, some of them require more work than one student can bring up in a one semesters' laboratory. In the following, the features which are suitable to enhance the capabilities of MTIDS are summarized. A discussion of some of these improvements is given in section (1.3).

- Analysis: structural controllability / observability, stability
- Interaction with control toolbox
- Visualization: adequate function for usability of MTIDS and visualization of results
- Functionality: directed graphs
- Distributed controller design
- Creation of graphs based on statistical measures like the degree distribution
- Support for switching and random interconnections

- Export to Simulink "wizard"

1.3 Realized implementations

In the actual project laboratory, the following topics are mainly treated: support for the design and simulation of directed graphs, and functions for visualizing of simulation results. The need for a use of directed graphs is based on real world information exchange which can happen in one direction only. As one can see in figures (1.2) and (1.3), different methods used for communication lead to directed respectively undirected information exchange. For example, a direct network graph arises with sensors based on video signals. On the other hand, an undirected network graph occurs if a wireless sensor network is used, where a bidirectional information exchange is possible.
The second main aspect is about the visualization of the simulation results. After a graph is designed in MTIDS and the system with the specified local dynamics is exported to simulink, a kind of assistance would relieve the presentation of the simulation results. An additional condition is that if the local dynamic of an agent exhibits more than one state, then it can be chosen which of these states should be visualized.

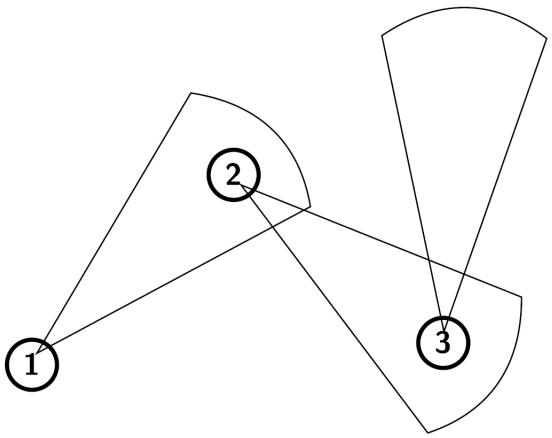

Figure 1.2: Directed sensing, e.g. through visual communication. Figure taken from lecture slides *Networked Control Systems* by Prof. Hirche, [HE11]

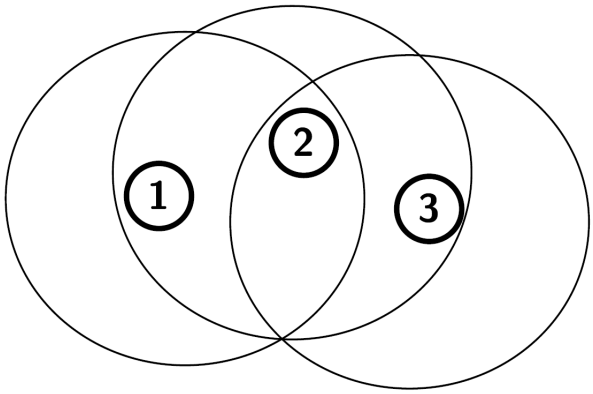

Figure 1.3: Undirected sensing, e.g. through wireless network. Figure taken from lecture slides *Networked Control Systems* by Prof. Hirche, [HE11].

Chapter 2

Selected aspects of graph theory

This chapter treats some parts of the algebraic graph theory. It outlines the main differences between directed and undirected graphs and some definitions are given. Afterwards, some attributes of directed graphs are described like *connectivity* and the aspect about *indegree and outdegree*.

2.1 Difference between directed and undirected graphs

Graphs are an intuitive way to describe relations between objects which are pairwise connected. The objects are called nodes, vertices or points, the connections are called edges, arcs or lines. A more detailed introduction into graph theory can be found in various textbooks, a few examples are [JNC11] and [BJG07]. Mathematically spoken, a graph is an ordered tupel with a set of vertices and edges: $G = (V, E)$. Every edge in E is related to two vertices $\{v_i, v_j\}$. If a graph is directed, then v_i denotes the starting point or tail of the edge, v_j denotes the endpoint or head of the edge. The undirected graph in figure (2.1) is an example for a simple and unweighted graph. A simple graph has no loops or cycles and only one edge between any two different nodes. Unweighted means that the edges do not have specified values which stands for the costs to go there or use this edge.

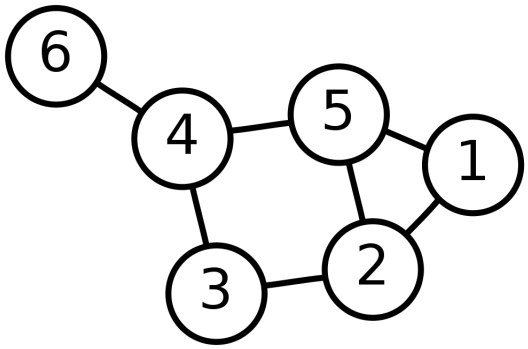

Figure 2.1: Example of an undirected graph. Picture is public domain.

In opposite, directed graphs or digraphs are able to possess loops or cylces. In the depicted example of figure (2.2), the graph has weighted edges and some cylces are observable, for example there is a path $\pi = uvu$ and a path $\pi = uxvu$. A path is a sequence of vertices and edges, with one vertex more than edges contained in it. If there is only one edge between any two vertices, it is obvious every time which edge is taken from vertex v_i to v_j, thus the entry of the edge can be left out. A cycle is a path, where all vertices occur only once, except for the starting node and the end node. Thus directed graphs can be classifed into acyclic ones and cyclic ones.

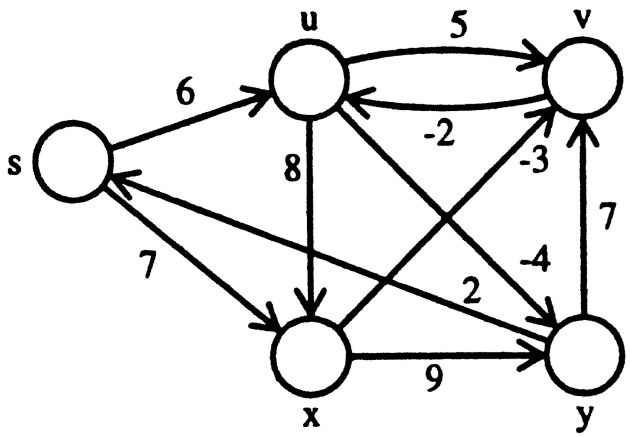

Figure 2.2: Example of an undirected graph. Picture is public domain.

2.2 Connectivity of a digraph

For undirected graphs, a statement about connected subgraphs can be given for example with help of the *Laplacian* L. The number of eigenvalues of L which are identical to 0, exhibit the number of connected subgraphs. Due to the fact that L can be computed in different ways for a digraph, another classification is needed. By that we mean the notions about strong and weak connectivity of a digraph.

2.2.1 Strong connected components

Strongly connected components are a class of subgraphs of digraphs. Two vertices are members of a strong connected component, if there is a way from one to another and back. Precisely, if there exists a way from vertex u to vertex v, it can be written as $u \to v$. Vice versa, if it holds that $v \to u$, then the condition for a strong connection is fulfilled which can be denoted as $u \leftrightarrow v$ which is equivalent to $v \leftrightarrow u$. Strictly mathematically spoken, strong connectivity is an equivalence relation.

- It is reflexive: $u \leftrightarrow u$, with a path length of 0,
- it is symmetric: $u \leftrightarrow v \Leftrightarrow v \leftrightarrow u$,
- and it is transitive, because: $u \leftrightarrow w$ and $w \leftrightarrow x \Rightarrow u \leftrightarrow x$.

The algorithm which can be used to implement the search for strongly connected components, is described by [Weg08]. An example of a graph with strongly connected components is given in figure (2.3).

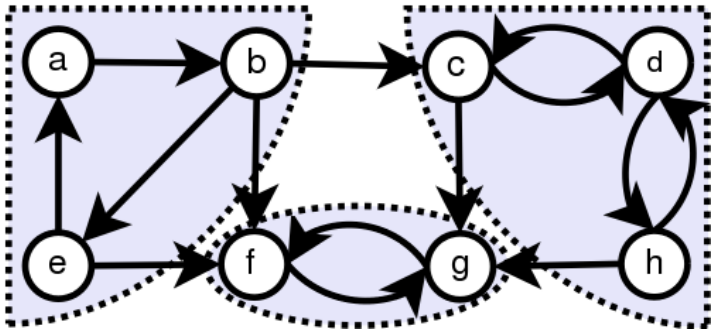

Figure 2.3: Directed graph with 3 strongly connected components which are marked. Picture is published under GNU Free Documentation License.

2.2.2 Weak connected components

Some other classification for digraphs is weak connectivity. Vaguely spoken, this can be compared with the attribute of connectivity in undirected graphs. Weakly connected components (WCC) are also subgraphs of G. One can obtain the amount of WCC by replacing all directed edges with undirected edges and then check how many connected subgraphs are there. This is called the *undirected underlying graph*. If the amount of WCC is 1, a digraph is called weakly connected. In contrast to undirected graphs, it is not possible to compute a measurement for algebraic connectivity like the *fiedler value*. If it is needed to make a statement about the robustness of the weak connectivity of a digraph, one can use the concept of bridges. The simplest case of a bridge occurs, if one specific edge of a digraph will be cut and the graph is not weakly connected anymore. In general, when there is a weak connected digraph, the edge cut of G is a group of edges whose complete removals results in

a disconnected graph. In graph theory this is referred to *k-edge-connectivity*. A graph is k-edge-connected, if it remains connected for the case when fewer than k edges are removed. A heavy handicap of this concept is the computability. There exist algorithms to determine the largest k of a graph, but with polynomial time to compute.

2.3 Indegree and Outdegree

Another important difference between undirected and directed graphs is the degree of a vertex. In undirected graphs, the degree of a node consists of the number of edges which are connected to him. In directed graphs, for each node an indegree and an outdegree exists. The indegree denotes the number of edges which are pointing into that vertex. Vice versa, the outdegree denotes the amount of edges which are point out of this vertex. The indegree is written as $\deg^-(v)$ and the outdegree as $\deg^+(v)$. It is possible to use indegree and outdegree to define some attributes of vertices which are familiar to electrical engineers: a vertex, for which it holds $\deg^-(v) = 0$, is called a *source*, a vertex, for which it holds $\deg^+(v) = 0$, is a sink. These attributes are useful in information theory just as well in physics. At this point, only unweighted edges are considered. The sum of indegrees and outdegrees of all nodes must be the same which can be formulated as:

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |E|, \tag{2.1}$$

where $|E|$ is the amount of edges in a graph G and V is the collection of vertices in that graph. If it holds for every vertex $v \in V$ that $\deg^-(v) = \deg^+(v)$, then the graph is called *balanced*.

Chapter 3

Implementation and documentation

This part of the documentation explains how to work with the new features of MTIDS. For a general use of the toolbox, see [LR11]. First, the design of directed graphs and their export to simulink is described. Next, the assistance functions to visualize the simulation results are shown. At last, there are some minor changes to MTIDS which will be discussed.

3.1 Directed version of MTIDS

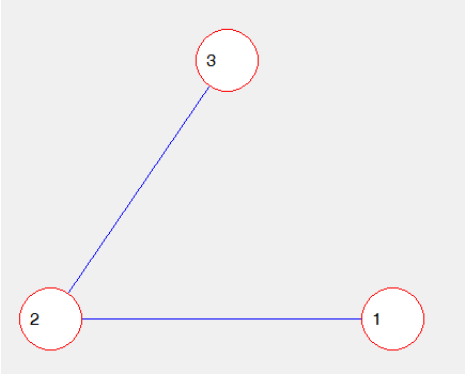

Figure 3.1: Arrows used in modus 'undirected'.

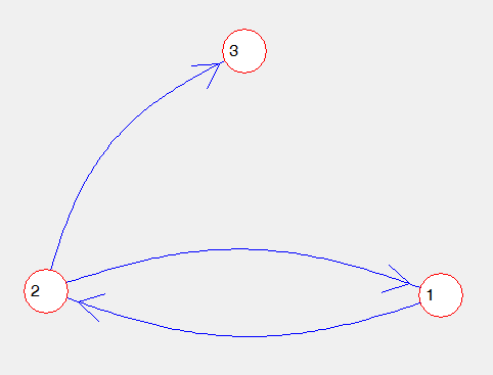

Figure 3.2: Arrows used in modus 'directed'.

One important new feature for MTIDS is the capability to work with directed graphs, or short digraphs. Because MTIDS uses Matgraph, an open source toolbox for undirected graphs, [Sch06], there is need of rewriting some scripts of Matgraph. Principally, most of the edited Matgraph functions can now be used for undirected and directed graphs also, by passing a boolean argument at the call of the function:

```
nrOfVertices = 10;
g = graph(nrOfVertices);
isDirected = 1;
add(g,1,2,isDirected);
```

The example creates a graph with 10 vertices and adds a directed edge from vertex 1 to vertex 2. This approach is implemented in most functions of Matgraph which are used in MTIDS.
A visual difference in the design of undirected and directed graphs are the connecting arrows. As be seen in figure (3.1), the draw-function of Matgraph paints straight lines between the vertices. In contrast to the figure on the right, where the directed edges must be curved that the direction is visible. The curve is realised as a straight line with a sinusoidal deviation. If an edge should be drawn with use of the mouse, hold 'Ctrl' and first click at the starting node, then at the end node.
The choice, if it should either be worked with undirected or directed graphs, is done in the main window of MTIDS, see figure (3.1). All other functions on the main window are working in both modi. If a new mode is chosen, the current graph will be deleted without a notice.

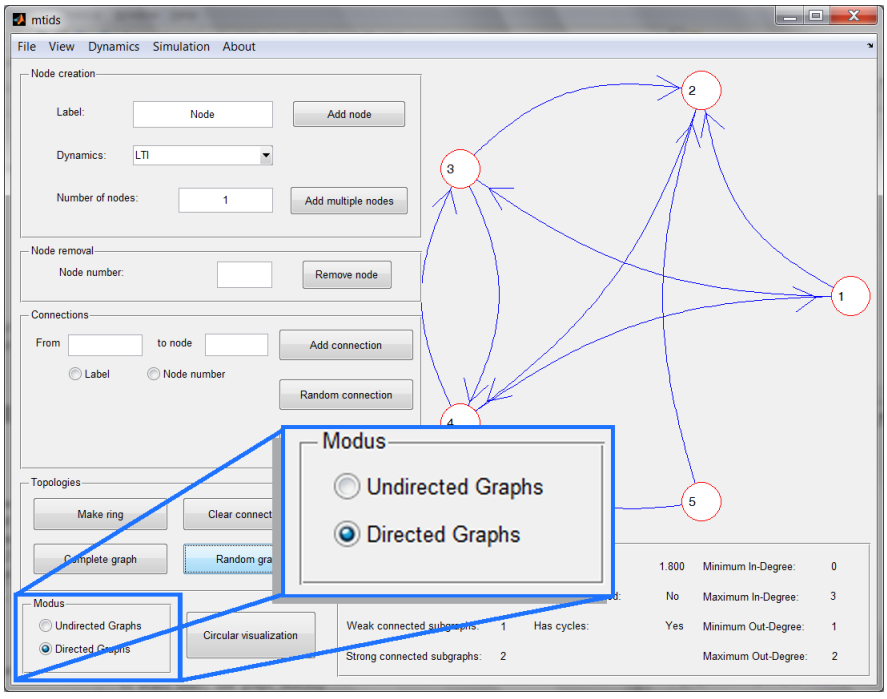

Figure 3.3: Main window of MTIDS, with an increase in size of the buttons, where the mode can be chosen.

There are different statistical values about the current graph which is shown in the right bottom of the main window. Among the statistics for directed graphs are the

amount of weak and strong components, if the graph is balanced and if the graph has cycles or not. Compare figure (3.1).

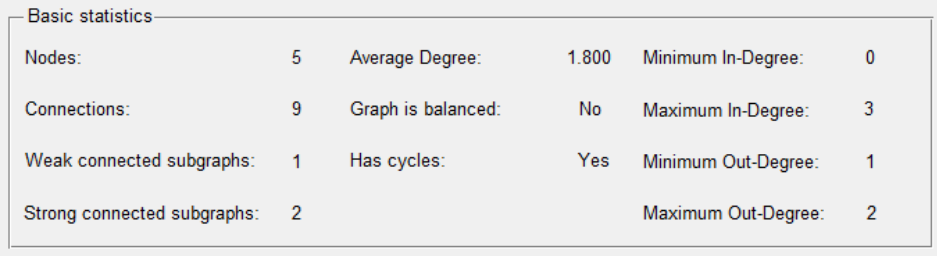

Figure 3.4: Increase in size of the basic statistics for directed graphs at the main windows of MTIDS.

There is also no difference in the export function to simulink which works identically for both modi. Use 'Export to Simulink2' at the slide 'Simulation' to guarantee a clean building in simulink. In figure (3.1), there is an example of how an exported system in Simulink looks like.

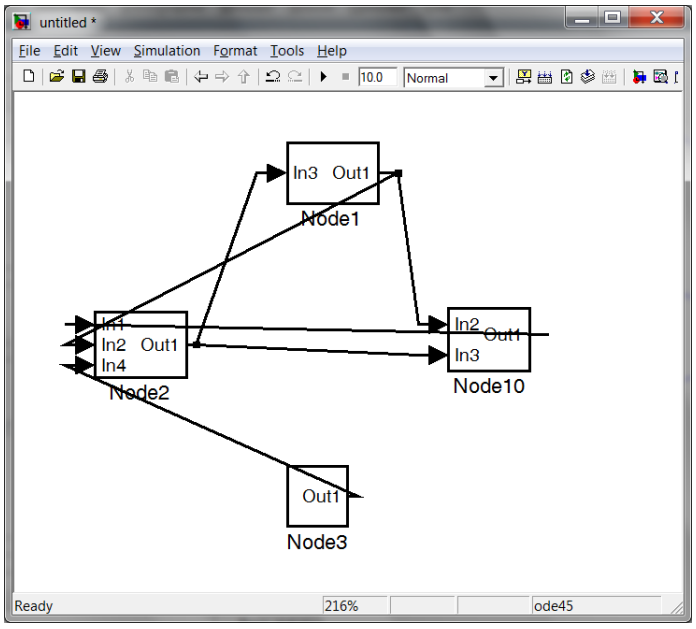

Figure 3.5: Example of an exported directed system to Simulink, with 4 nodes and 6 edges. Notice that Node 3 has no input.

3.2 Assistance to visualize the local states

The aim of this task is to support the visualization of the simulation results in Simulink. For this reason, it must be defined which node outputs and states should

be visualized. This must be done before the export to Simulink. It is possible to generate plots for every node output and internal state. To define which outputs and states should be plotted, double-click a node. A new figure will open, see (3.2). The checkbox 'Plot node output' is set initially. If internal states should be plotted, activate the checkbox 'Plot selected states'. On the right hand side, the numbers of the internal states must be entered, separated by a blank. Since at this design step, there may not exist a concrete dynamic, it is necessary to declare how many local internal states this node has. For the actual version 1.1 of MTIDS, just the dynamic template 'LTI' has more than one state, beside of individually composed dynamic templates. That means, if another dynamic than 'LTI' was chosen, it makes no difference, wheter output or internal state should be plotted, they are identical.

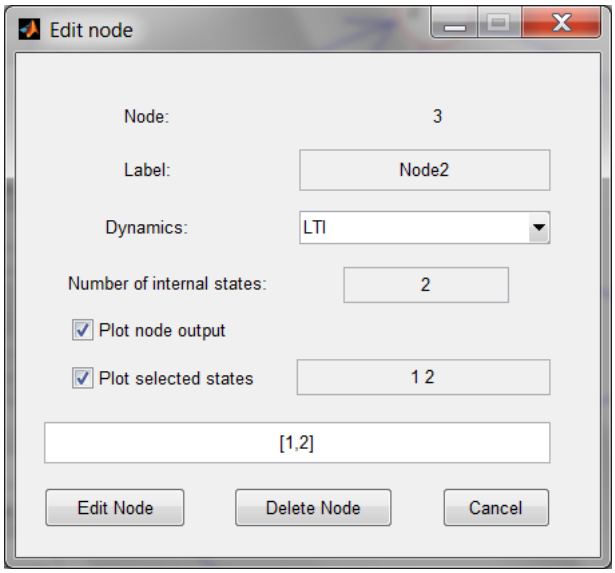

Figure 3.6: The edit-node window of MTIDS appears by double-clicking on a node.

After all visualization settings are made, the next step is an export to Simulink (use 'Export to Simulink2') and the parametrization of the system. This step must be done manually. The parameters for the system must be available in the Matlab workspace. If the system is ready for a simulation, choose 'Run Simulation' under the slider 'Simulation', see figure (3.2).

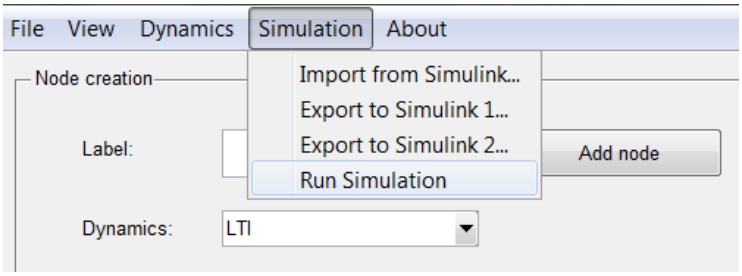

Figure 3.7: The function 'Run Simulation' can be chosen after an export to Simulink and if all parameters for the system are existing at the workspace of Matlab.

After a simulation has ended and if at least one signal was chosen to visualize, the plots are generated. For each node one figure will appear. If internal states should be plotted, the output and the internal states are plotted in one figure. See figure (3.2) for an example plot. The attribute 'colorSpec' of the graphs of the internal states are generated randomly.

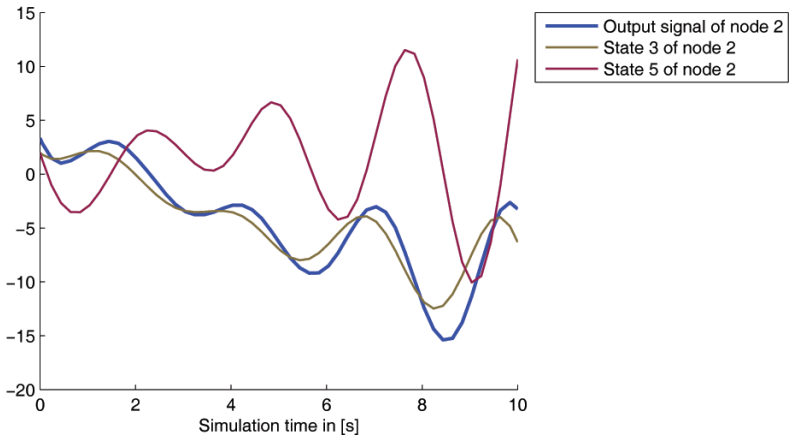

Figure 3.8: Example plot of the simulation results. As it is recognizable, the internal states 3 and 5 were chosen to be plotted.

3.3 Collection of minor changes

In this section, some small changes to MTDIS are described. Either they do not have effects on the functionality, nor they are not visible to the user at once.

Node numbering

In MTIDS version 1.0, the automated node numbering logic worked as follows: if a name is prompted in the field 'Label' and there exists a node with the same label, it

simply concatenated the number 1 to the new vertex. This led to the case that the first vertex has the name 'Node' and the second one has the name 'Node1'. This problem is fixed. The logic works now like this: if a node name, prompted in the field 'Label' is new to the system, it will be taken as it is. If the case happens that there still exists an identical name, it will rename it, concatenating a 1. The new node will be concatenated with a 2.

Identification of system parameters

A major easing of MTIDS would be a kind of parametrization wizard. This feature should for example automatically check, how many system paramters exist and prompt it before exporting to Simulink. With it, it would be much easier, to provide all parameters at the Matlab workspace. Some preparations to this feature are done yet, in comparison to MTIDS version 1.0. In the first version, during the export to Simulink, there was no adaption of the templates' parameters. That means, if a template was modified for the visualization in Simulink, the parameter name were always the same. That leads to the case, if a template is used for a system more than once, the paramter name is also used more than once. This could lead to side effects during the parametrization of the system. The problem is fixed, editing the interconnector component of MTIDS which is responsible for the export to Simulink. Now every parameter will be concatenated during the export with the number of its parent node. Thus an easier differentiation of the parameters is possible.

Chapter 4

Conclusion and future development

At this point a summary is given about the contents of this documentation and about the changes in MTIDS 1.1 compared to version 1.0.

4.1 Conclusion

At first there was a short review of graph theory. Though undirected and directed graphs were discriminated. Some characteristics of digraphs has been shown, for example strong and weak connectivity. It can also be distinguished between indegree and outdegree of nodes in digraphs. The next chapter treats the implemented changes of MTIDS 1.1. Digraphs could be designed switching to the corresponding modus. It is shown how the functions of Matgraph are edited to obtain a framework for digraphs. An assistance to visualize the Simulink simulation results was created. By double clicking a node, alignments can be done to choose if output or states should be plotted. To generate the plots, the Simulink simulation has to be started by MTIDS. At last, some minor changes are described. The way how new nodes are labeled and named is explained. A step towards a smart parametrization wizard is done by distinguishing the subsystem paramaters during the export to Simulink.

4.2 Future development

Beside the new capabilites of MTIDS, there a still many things left to do:

- Tools for analyzing the system / subsystems in relation to controllability, observability and stability.
- Interaction with control toolbox of Matlab.
- At the visualization of the simulation results, the specifications of the chart should be editable.

- Actually, there is no functionality to design (distributed) controllers.
- Graphs should be automatically designed based on statistical measures like degree distribution.
- A support for switching interconnections is needed.
- At the stage of parametrizing the system, a kind of assistance or parametrization wizard would be helpful.
- The designed graphs should be exportable as figures, either directly in Matlab or via a tex based painting tool like PGF/TikZ [Tan07].
- Due to the rising complexity of the program, some bugs are still not fixed. An incomplete list of known problems is contained in the distribution of MTIDS.

List of Figures

Bibliography

[BJG07] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2007.

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

[Hu10] Yifan Hu. *A Gallery of Large Graphs*. AT and T Labs Research, Florham Park, New Jersey, USA, 2010.

[JNC11] David Joyner, Minh Van Nguyen, and Nathann Cohen. *Graphbook: A book on algorithmic graph theory*. code.google.com/p/graph-theory-algorithms-book/, 2011.

[LR11] Francisco Llobet and Jose Rivera. *Numerical Test Rig for Large-Scale and Interconnected Dynamical 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.

[Tan07] Till Tantau. *The TikZ and PGF Packages: Manual for version 2.10*. Institut für Theoretische Informatik Universität zu Lübeck, Lübeck, Germany, 2007.

[Weg08] Ingo Wegener. *Effiziente Algorithmen und Komplexitätstheorie*. Lehrstuhl für Effiziente Algorithmen und Komplexitätstheorie, TU Dortmund, Dortmund, Germany, 2008.