Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2359

Development of a Visualization Tool for Kinematics and Dynamics for hyper redundant robots

Manuel Bischof

Course of Study: Computer Science

Examiner: Prof. Dr. rer. nat. habil. Paul Levi

Supervisor: Dipl.-Ing. Eugen Meister

Commenced: 16.06.2011

Completed: 16.12.2011

CR-Classification: D.2.13, H.5.2, I.2.9, I.2.11

Abstract

Swarm robotic is an difficult, but promissing approach in modern robotics. Multi-robot organisms can achieve tasks, an individual robot could not accomplish alone.

But first you need kinematics and dynamics to model them. Becaus of the huge amount of degrees of freedom, many classical approaches fail. Through its geometrical representation, screw theory could be the solution. There already existed a mathematical framework, but no visualization for it.

So in this thesis I created a Graphical User Interface, that supports the creation of different user-defined topologies and visualizes its results.

In the end, this GUI should assist developer with their work, but hopefully also students to understand this subject more easily.

Contents

List of Figures V

1 Introduction 3
- 1.1 Motivation 4
- 1.2 Goal 5
- 1.3 Organisazion of this thesis 5

2 Basics 6
- 2.1 Screw Theory 6
  - 2.1.1 Basic principles 6
- 2.2 Matrices used 9
  - 2.2.1 AIM 9
  - 2.2.2 CM 9
- 2.3 GUI basics in Matlab 10
  - 2.3.1 GUIDE 10
  - 2.3.2 Handles 10
  - 2.3.3 Tabs 11
  - 2.3.4 Embedding Latex 11

3 State of the art 12
- 3.1 3D Puma Robot Demo 12
- 3.2 Robotics 13
- 3.3 3_Bar Robot Kinematics and Kinetics Analysis and Simulation 13
- 3.4 Cylindrical Robot Simulator 15

4 Realization 18
- 4.1 Elements of the GUI 18
  - 4.1.1 General overview 18
  - 4.1.2 Kinematic and Dynamic tab 18
  - 4.1.3 Topology Graph 22
  - 4.1.4 Kinematic tab only 27
  - 4.1.5 Dynamic tab only 31
  - 4.1.6 Result tab 32
- 4.2 User Guide 34
  - 4.2.1 Customizations 34

4.2.2 Quick Start 34
4.2.3 Unused functions 35

5 Conclusion **36**

Literatur **A**

List of Figures

1.1 A swarm of robots . . . 3
1.2 Multi-Robot-Organism climbing an obstacle . . . 4

2.1 A screw motion [Source: Wikipedia:Screw Theory] . . . 6
2.2 A prismatic joint [Source: Wikipedia] . . . 7
2.3 A revolute joint [Source: Wikipedia] . . . 7

3.1 6DOF Puma Robot . . . 12
3.2 Robot with one prismatic and two revolute joints . . . 13
3.3 Simulation options . . . 14
3.4 Screenshot of the animation . . . 15
3.5 Resultating plots . . . 16
3.6 Resultating plots . . . 17

4.1 General overview of the Kinematic tab . . . 19
4.2 General overview of the Dynamic tab . . . 20
4.3 General overview of the Result tab . . . 20
4.4 Example of the Topology Matrix . . . 21
4.5 Context menu to change the robot type . . . 22
4.6 PushButton 1_1 as example for the topology matrix buttons . . . 23
4.7 The two pre-defined robot types: Scout and Kbot . . . 24
4.8 Topology graph drawn by biograph . . . 24
4.9 Creation of the biograph in function run_Callback . . . 25
4.10 Generation of AIM and CM in function run_Callback . . . 26
4.11 Code of reset_Callback . . . 28
4.12 How each simulation is saved in store_Callback . . . 29
4.13 Example Trajectory . . . 30
4.14 Toggle pan and zoom mode for axis . . . 31
4.15 Scale an axis using a slider . . . 31
4.16 Example Dynamics . . . 33
4.17 Scale an axis using a slider . . . 34

1 Introduction

It is commonly known, that robots are a great help in our life. A world without them can hardly be imagined, as they are used in nearly every factory. Yet scientist are already several steps ahead and are researching on continually shrinking robots. Although there will be different obstacles, like a small step, which would be not mentionable for a bigger robot, this is a promising approach. Espacially if you don't only use one robot, but a lot of them, they can help each other and accomplish task, that a single robot couldn't. This feature is called *emergence*.

The European Research Community has focused upon swarm robotics with their projects SYMBRION and REPLICATOR [Com11]. Their goal is to develop adapta-

Figure 1.1: A swarm of robots

tional and evolutional approaches for multi-robot organisms, that can share their energy, help each other out and even dock together to build one bigger organisms. This should be intelligent to make good use of its new features. For instance it can come over obstacles, a single robot would not stand a chance. This symbiotic behavior is adopted from nature itsself. Multiple robots sharing a common energy source and telling each other about it can be compared to foraging of ants. They spread out, looking for something to eat. Once they found something they go back

Figure 1.2: Multi-Robot-Organism climbing an obstacle

to their nest, leaving pheromones behind. Other ants can follow them to the source.

Symbrion is an abbreviation for "Symbiotic Evolutionary Robot Organisms". This project focuses on artificial evolution in huge number of small and light robots.

Replicator (Robotic Evolutionary Self-Programming and Self-Assembling Organisms) on the other hand concentrates on the adaptability of a smaller number of larger robots in open-end environments.

1.1 Motivation

Swarm robots have the advantage, that you can reach great achievments with comparatively cheap robots, as a single one is built quiet simple. But before they can be deployed, there has to be done some work and research. One part is to find out about their kinematics and dynamics.

Matlab Central [TMb] offers a *File Exchange* [TMc] Platform, where you can find several robotic toolboxes and simulators. But nearly all of them are based on classical mechanics and mostly for just one fixed 3 or 6 DOF (=Degree of Freedom) robot type with a predefined topology.

Still, there is none, that allows the implementation for hyper redundant systems.

Another problem is, that traditional approaches often run into difficulties when dealing with too many degrees of freedom, as would be the case with swarm robots. Screw theory is based on a more geometrical level and is therefore auspicious. There already exists such a tool, but only as a mathematical framework.

So this study thesis will provide a Graphical User Interface for further work in the fields of swarm and collective robotics. Especially it will be suitable for the use with screw theory.

1.2 Goal

Goal of this work is to establish a GUI to support and visualize the construction of a vast variety of topologies using the two robot types *Scout* and *Kbot* 4.1.2. This will not only help the improvement of kinematic and dynamic models, but also other students to understand this topic.

1.3 Organisazion of this thesis

Chapter 1 explained, what this thesis is about and why this work is important. Next you will get some basic knowledge about both, screw theory and GUI creation in Matlab. In chapter 3 I will briefly introduce some currently existing frameworks and simulators in this field. Chapter 4 will guide you through the realization of this project and how to make adjustments to it, followed by a conclusion in chapter 5.

2 Basics

2.1 Screw Theory

Screw theory is a alternative approach to kinematics and dynamics in contrary to classical mechanics also dealing with rigid body motions. This chapter will basicly rely on Murray [RMM94] as it provides not only a good introduction to this topic, but also further explanations and approaches to manipulations of robots and multifingered robot hands.

2.1.1 Basic principles

A *rigid body* can be multiple particles forming one unit, where the distance between each pair of them is constant. This is a useful compound, as you can handle for instance one robot as just one object and therefore just need one transformation for it.

The movement of a rigid body can be described as a rotation and translation about one axis, maintaining the prerequisite named above. This is beeing refered to as a *screw motion*. Image 2.1 illustrates this. The rotation around the axis is called

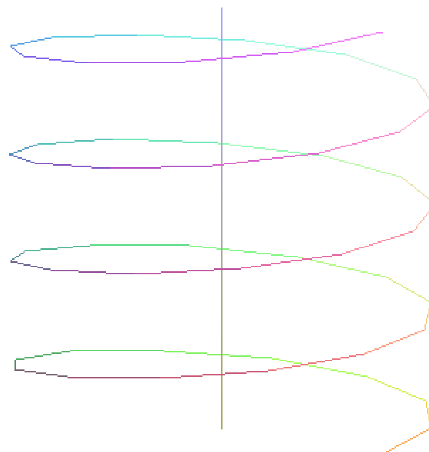

Figure 2.1: A screw motion [Source: Wikipedia:Screw Theory]

the *pitch*. To define a screw you need the axis along which is translated and rotated,

the pitch and a magnitude which declares the distance of the transformation. It is easy to see two extreme cases: First, if the translation is zero, there is only the rotation. The other way round this can represent a pure translation by choosing an infinite rotation.

If the screw motion is infinitesimal it is called a *twist* and describes the velocities both, along the axis and angular around it. Twist coordinates are defined by $\xi = (v, \omega)$, where $v = -\omega \times q$, $\omega \in \mathbb{R}^3$ the axis of rotation and finally $q \in \mathbb{R}^3$ a point on the axis.

With this knowledge we can describe a rigid body transformation by exponentiating twists: $g = \exp(\hat{\xi}\theta)$ with $\hat{\xi} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}$.

In the special cases of pure rotation (revolute joint, see 2.3) or pure translation (prismatic joint2.2) we can write the twists as $\xi = \begin{bmatrix} \omega \times q \\ \omega \end{bmatrix}$ for the first case and $\xi = \begin{bmatrix} v \\ 0 \end{bmatrix}$ for the latter.

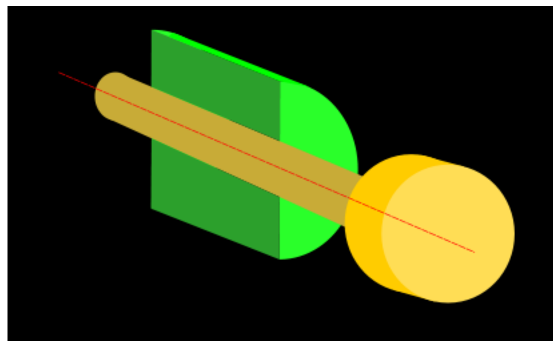

Figure 2.2: A prismatic joint [Source: Wikipedia]

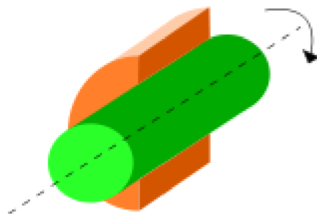

Figure 2.3: A revolute joint [Source: Wikipedia]

Another concept of screw theory are *wrenches*, which describe forces, that act on

rigid bodies and are composed of a force and a torque, again along the same axis. This can be written as a vector of $\mathbb{R}^6$: $F = \begin{bmatrix} f \\ \tau \end{bmatrix}$, where $f \in \mathbb{R}^3$ is the linear component (force) and $\tau \in \mathbb{R}^3$ the rotational (moment).

2.2 Matrices used

2.2.1 AIM

AIM is the abbreviation for Assembly Incidence Matrix. This matrix shows us how robots are assembled in multi-robot-systems. Here we assume, that each robot has four docking points - one on each side. They are numbered from 1 to 4, 1 the viewing direction and the others clockwise. Each coloumn in the matrix represents one edge connecting two robots. The rows stand for robots. You can conclude, that each coloumn has exactly two non-zero entries.

As an example look at picture 4.5 on page 22. The corresponding AIM would be:

	$e1$	$e2$	$e3$	$e4$	$e5$
$v0$	1	2	0	0	0
$v1$	1	0	0	0	0
$v2$	0	1	3	0	0
$v3$	0	0	3	4	1
$v4$	0	0	0	1	0
$v5$	0	0	0	0	1

Note, that it is not necessary, that the matrix is square. This is just the case, if it has the same number of nodes and links.

2.2.2 CM

CM is a simple connection matrix, also adjacency matrix. It can be derived from the AIM, but not the other way round, because you are missing the information of the orientation about the robots. In contrast to the AIM this is a square matrix an both coloumns and rows represent vertices of the graph. A value of 0 means, that two nodes have no common link, any other (usually just 1) says, that there is one. For the same example the CM would look like this:

	$v0$	$v1$	$v2$	$v3$	$v4$	$v5$
$v0$	0	1	1	0	0	0
$v1$	1	0	0	0	0	0
$v2$	1	0	0	1	0	0
$v3$	0	0	1	0	1	1
$v4$	0	0	0	1	0	0
$v5$	0	0	0	1	0	0

An undirected graph results like in this case in a symmetric matrix.

2.3 GUI basics in Matlab

MATLAB is a mathematical program, based on matrices, that not only supports numerical computing, but also plotting data. Matlab offers a command line for simple calculations but also the support for larger algorithms, using .m files. It is even possible to create Graphical User Interfaces.

If that is not enough, you can find a vast variety of toolboxes and community-writen projects to extent its abilities even more.

2.3.1 GUIDE

To create the GUI of this thesis, I used the comfortable, integrated environment of Matlab: GUIDE (= *G*raphical *U*ser *I*nterface *D*evelopment *E*nvironment). GUIDE is an GUI itsself, that allows the user to simply drag and drop interface controls and set their properties via context menus. Additionally, GUIDE is able to create a .m-file, where, in the easiest case, you just have to add some code for callbacks, like buttons and other controls. Of course you can read or write properties of objects during runtime. *set* and *get* are the commands for this purpose.

```
set(handles.button1, 'String', 'This is a button');
```

would set the caption of button1 to "This is a button". To read the same just use

```
get(handles.button1, 'String');
```

where the arguments are:

- handle of the desired object (for further information about handles see next paragraph)
- (optional for get) Property to read or write
- (set only) Value

If *get* is called with only one argument (the handle), it displays all settings you could also see in the ObjectInspector in GUIDE.

2.3.2 Handles

GUIDE also creates a structure named *handles* that is automatically passed to all functions created by GUIDE as an argument. Therein Matlab stores an unique number for all objects, so that you can use it as an pointer, if you want to read or write properties of that object. It is possible to expand this structure for your own purposes, like storing information that is needed in other functions than the one it is created in. It is important that you don't forget to save changes made to the handles-structure at the end of the function. To do so, just write:

```
guidata(handles, hObject)
```

2.3.3 Tabs

Unfortunately GUIDE does not provide native support for tabs. There are several possibilities to realize them:

- Use the `uitabgroup` or `tabdlg`, which are still undocumented functions (See [Alt10]). Therefor they could change or even vanish in other Matlab versions.
- Matlab Central's File Exchange [TMc] offers several solutions to implement tabs
- Create panels and just switch their visibility

I decided to take the last alternative. It is an easy and straightforward possibility without the need to rely on other implementations.

I just created an UIPanel for each Tab I wanted to have. As callbacks for the tabbuttons I simply moved the other tabs on top of the first one, then made all but one Panel invisible. Simplified it looks something like this:

```
% --- Executes on button press in tab_button1. -> Kinematics
function tab_button1_Callback(hObject, eventdata, handles)
% hObject handle to tab_button1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
set(handles.tab1,'Visible','On'); % main tab for kinematics and dynamics
set(handles.tab3,'Visible','Off'); % Result tab
```

2.3.4 Embedding Latex

It is possible to embed latex code for instance in captions of axes. This improves the labels considerably, as it allows you to write $\dot{q}$ and $\ddot{q}$ instead of q_dot. Here is the code for that part:

```
set (handles.figure1, 'CurrentAxes', handles.axes_velocity);
ylabel('$\rm\dot{q}$', 'fontsize', 14, 'interpreter', 'latex', 'rotation', 0);
```

The first line just sets the current axes. The second sets the label, where the first argument is the string itself. Code between the two $ is handled in math-mode. \rm just says the interpreter, that the text should be upright and not bold; \dot{q} is the code for $\dot{q}$. After that follow name-value-pairs, which are rather self-explanatory. Important is that you specify latex as interpreter.

3 State of the art

Now I want to briefly introduce some robotic related projects. All of them can be found on MatlabCentrals File Exchange [TMc].

3.1 3D Puma Robot Demo

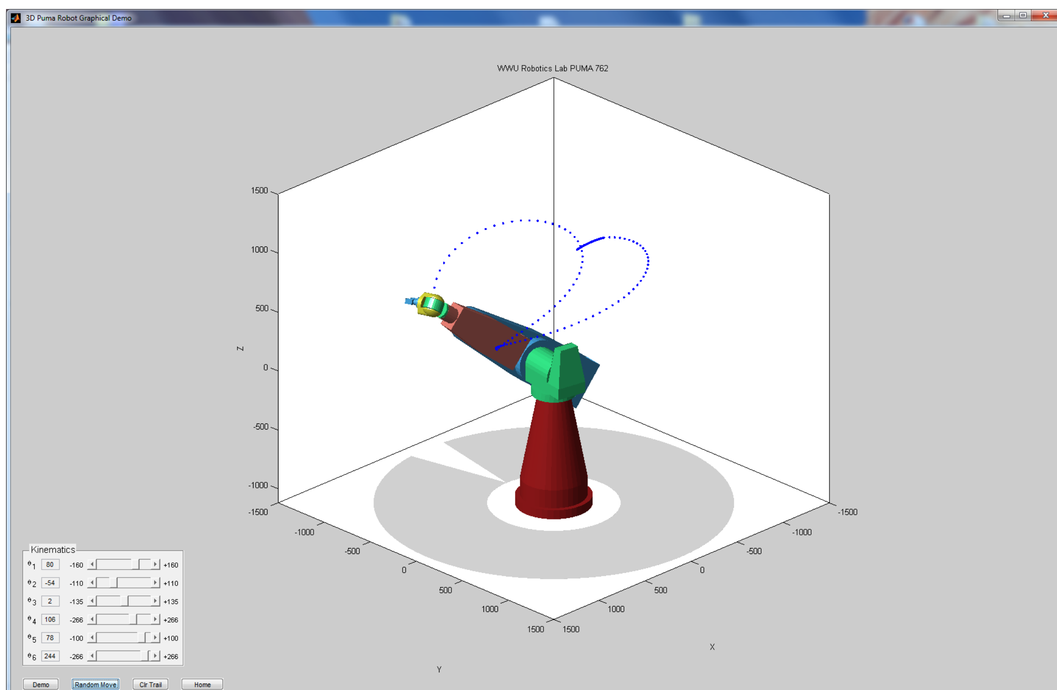

Figure 3.1: 6DOF Puma Robot

This is a small GUI designed by Don Riley[Ril07], that depicts a standard 6 DOF PUMA (=Programmable Universal Machine for Assembly). PUMA is an industrial robot arm, that is quite common. It consists only of six revolute joints.

Most of the space is an axis, containing the illustration of the robot. Other than that, there are six edit boxes and sliders; each for one of the joints.

The user can either enter values in a certain range for the revolute joints. The robot will then be led to its destined position. The trajectory of the manipulator

will be displayed by blue dots. Or he can watch a little demo, that will draw some circles and then return to its home position.

At last you can reset the position and clear the trajectory.

3.2 Robotics

This [Hig09] simulation is similar to the first one. You only have three joints, but these can be either revolute or prismatic joints. There are also two modes for the end effector: drilling or welding. Again, you can just let the program create a random motion or tune the parameter to your comfort. One example can be seen in picture 3.2.

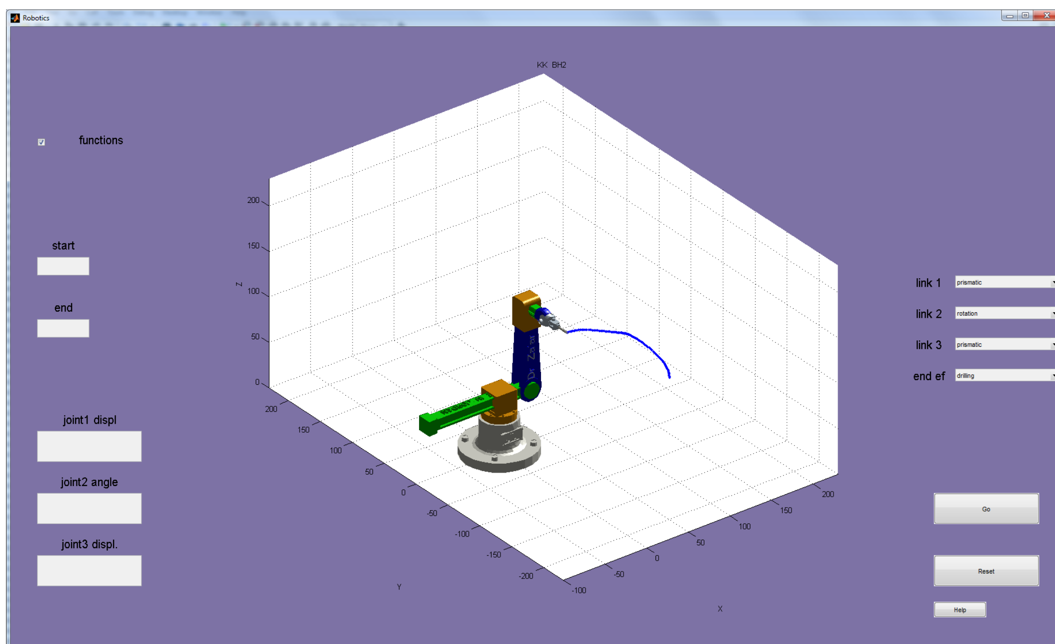

Figure 3.2: Robot with one prismatic and two revolute joints

3.3 3_Bar Robot Kinematics and Kinetics Analysis and Simulation

The next project is a lot more complex. *3_Bar Robot Kinematics and Kinetics Analysis and Simulation*[Kol07] offers a lot of editboxes to change parameter, as seen in 3.3.

Object of simulation is a robot with 3 bars. The options let you edit torques for the actuators and the end effector as well as the initial configuration such as position,

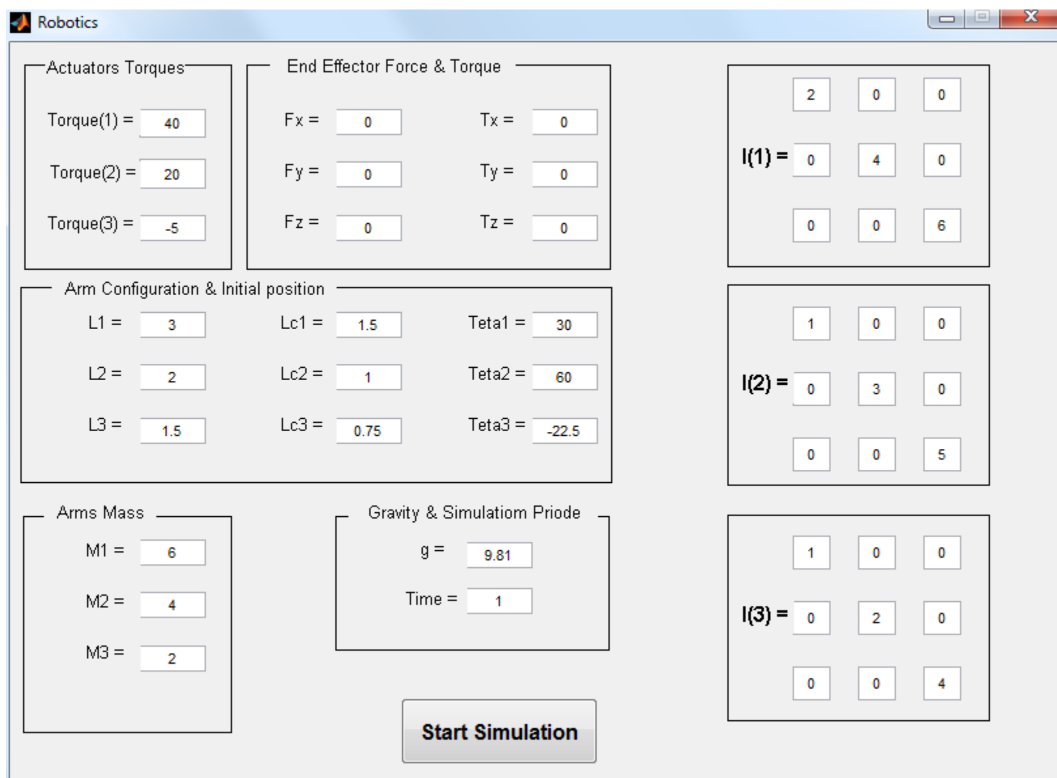

Figure 3.3: Simulation options

length and mass of the arms. Gravity and the simulation period are also to be set.

After you press “Start Simulation” you need some patience, while the simulation is in operation and Matlabs VR toolbox will be started.

Result is an animation of the preset robot arm and how it is affected by the forces and torques 3.4. After the simulation is closed there are some plots of important

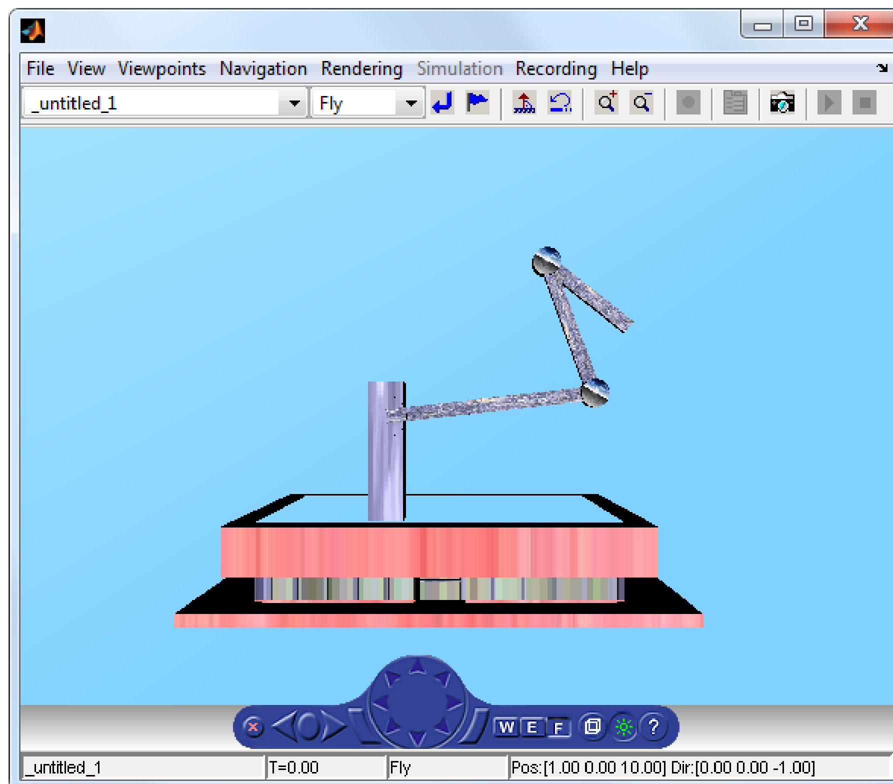

Figure 3.4: Screenshot of the animation

variables, see 3.5

3.4 Cylindrical Robot Simulator

Unfortunately this tool [And07] is in Portuguese apart from the title. A screenshot is shown in figure 3.6

Again, we can see a simplified robot arm. Although I didn't test it, you should be able to steer it with an joystick or a gamepad.

The robot arm has got four revolute joints and two prismatic joints. The application can handle direct or inverse kinematics, so you have got six parameter to set. If you chose direct kinematics these are the angles for the revolute joints and

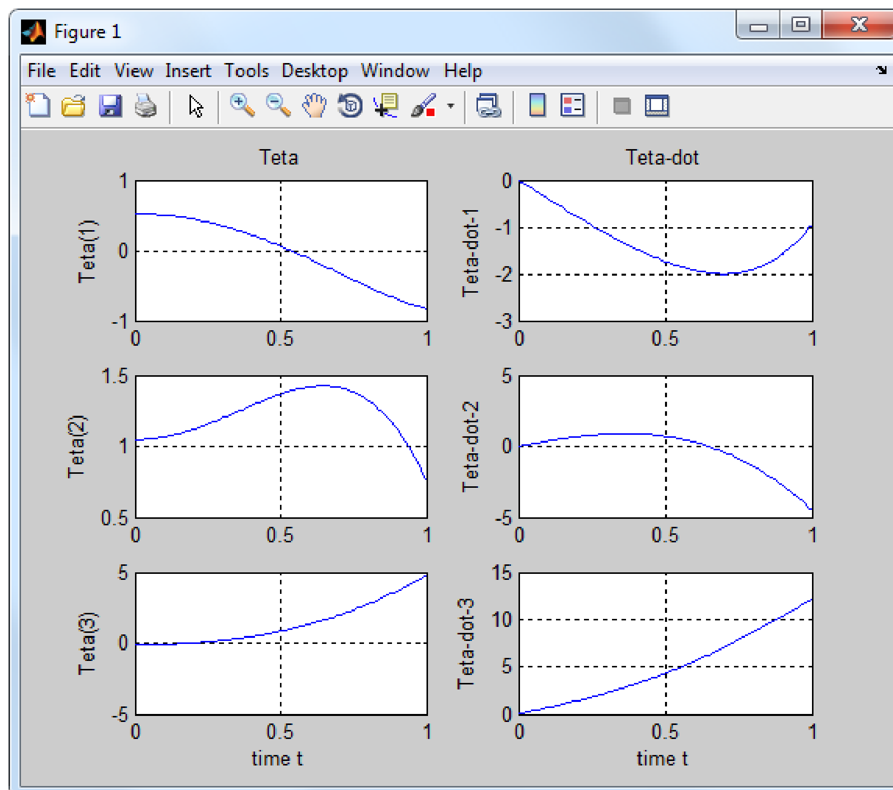

Figure 3.5: Resultating plots

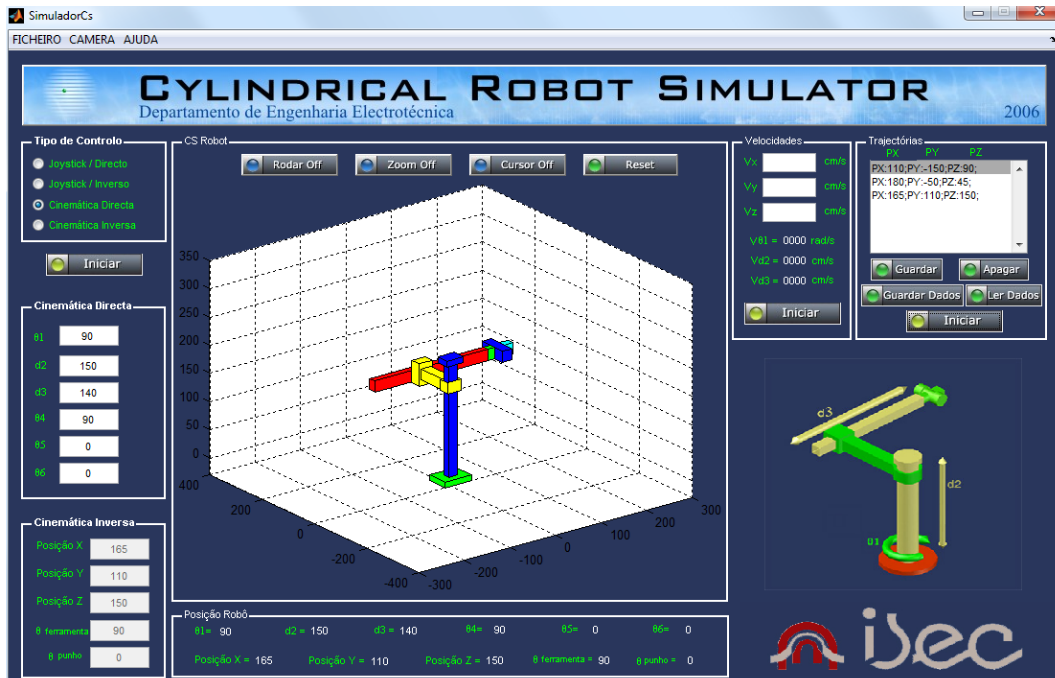

Figure 3.6: Resultating plots

translations for the prismatic joints, otherwise (inverse kinematics) the position and orientation of the end effector.

Configurations of the joints can be saved by pressing “Guardar” on the right. With this help, you can create your own trajectories step-by-step. Selecting the first position of your trajectory and hitting “Iniciar” will animate the robot arm with your given configurations.

4 Realization

4.1 Elements of the GUI

4.1.1 General overview

The GUI is divided in several sections. First, it contains a menu to create new projects and to open or save existing ones. Second, you can see three tab buttons, which enable different parts of the GUI: Kinematics, Dynamics or the results. In the pictures 4.1, 4.2 and 4.3, you see a first general overview over these three tabs. Next, there are several UIPanels, not all of which are visible at the same time.

I will now explain the contents of each tab.

4.1.2 Kinematic and Dynamic tab

These two tabs share some common components, as they are used in both modes. Independent of the selection, you can see your current topology both as matrix and as graph, the two pre-defined robot types Scout and Kbot, the parameter tab to adjust settings and a radio-button, that allows you to switch between forward and inverse kinematic or dynamic respectively.

I will now move on to some more detailed explanations of the components.

Forward / Inverse radio buttons

If you change the selection of this radio group it swaps the 'Enabled' option of the components in the Parameter tab, so you can edit only the preferences, which shall be editable, and the others are just read-only. In addition, this control should be used to determine the current mode, if an kinematic or dynamic algorithm is run.

Topology Matrix

With this field of 7×7 buttons you can easily configure an topology with two different types of pre-defined robots as is shown in picture 4.4. When the GUI is started or reset just the central button is active. By left-clicking, you change the status of the buttons respectivly robots in the following order:

- Up
- Right

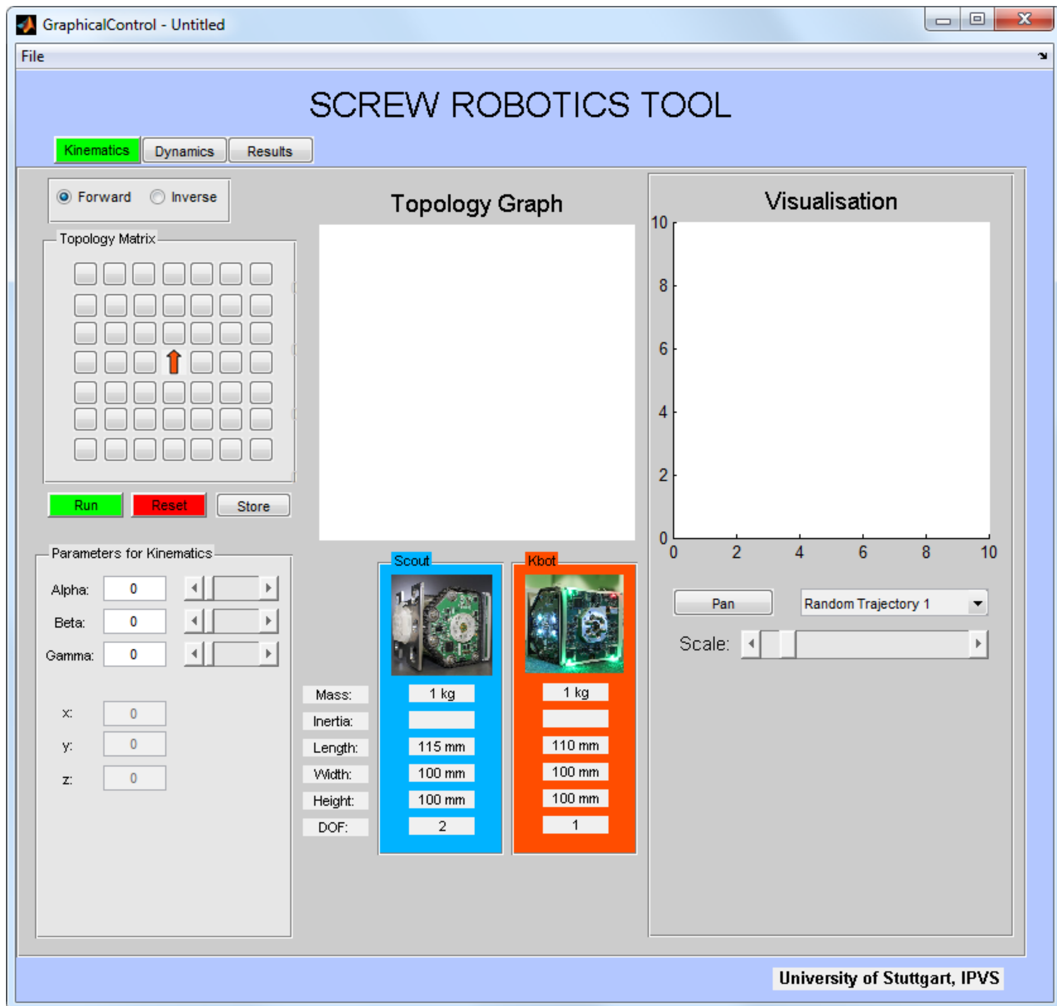

Figure 4.1: General overview of the Kinematic tab

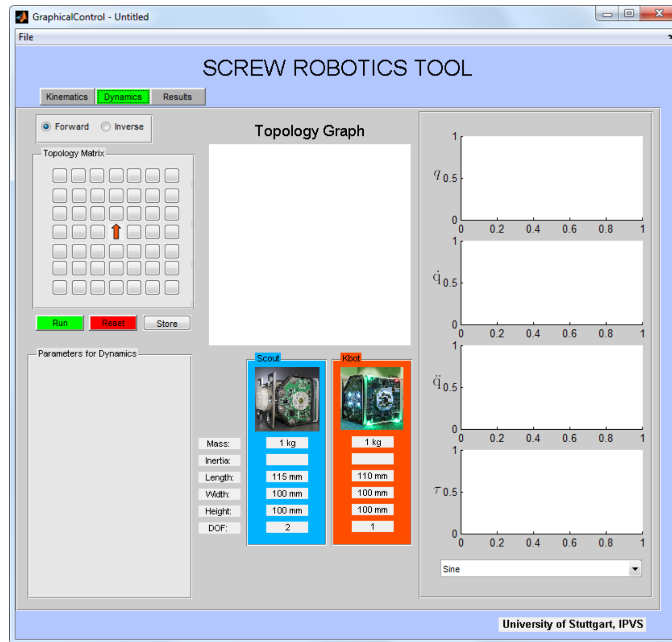

Figure 4.2: General overview of the Dynamic tab

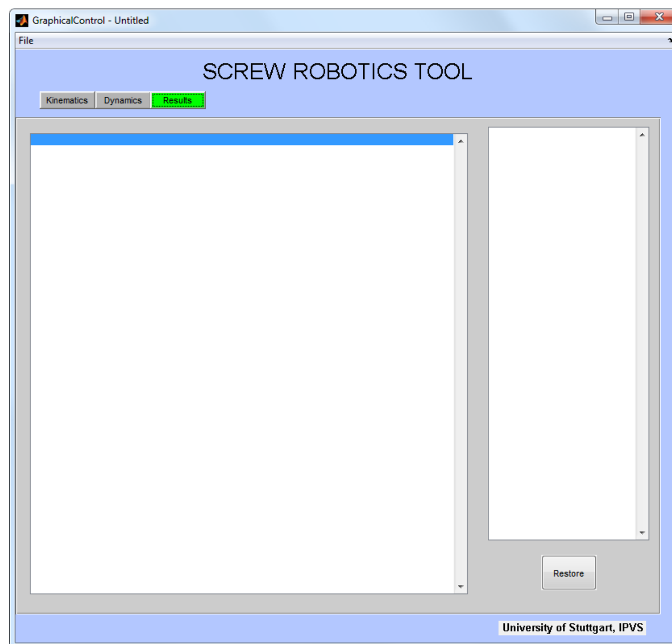

Figure 4.3: General overview of the Result tab

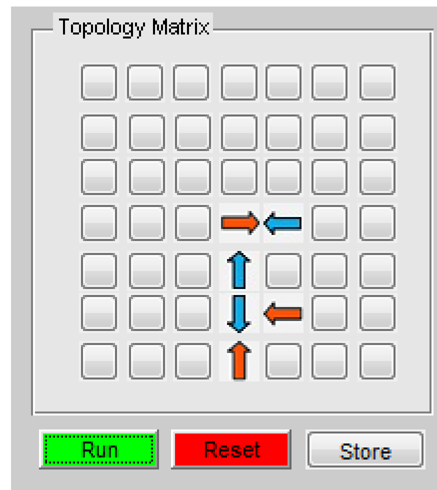

Figure 4.4: Example of the Topology Matrix

- Down
- Left
- Deactive

That means, you can turn the viewing direction of the robot. In addition to that, you can choose between two different pre-defined robot-types by right-clicking a robot and choosing the desired type in the context menu as seen in picture 4.5. More information about the robot types see in the following paragraph.

The tags of these buttons contain coloumn and row in the matrix, so that the code is easy to understand. A shortened example can be seen in figure 4.6. The code is only executed, if there exists at least one active robot on any side, else a messagebox with an error is raised. During execution, an internal variable is increased, that indicates the viewing direction or that the robot is deactivated. Then, the image is updated, dependend on the current state and the chosen robot type. At last, the ω-vector, meaning the rotation axis, is readjusted.

Robot information

This framework is designed to work with two pre-defined robot types: *Scout* and *Kbot*. You can find some information about these two types in the corresponding UIPanels (Picture 4.7). The Panels have a color-coded background for fast recognition. The same colors are reflected in the topology matrix and its context menu (pictures 4.4 and 4.5). Blue depicts the *Scout* and red *Kbot*. Both weigh about 1 kg and have a length and width of 100mm. Scout just got a little arm, which makes it with 115mm vs. 100mm a bit longer, but grants a second degree of freedom.

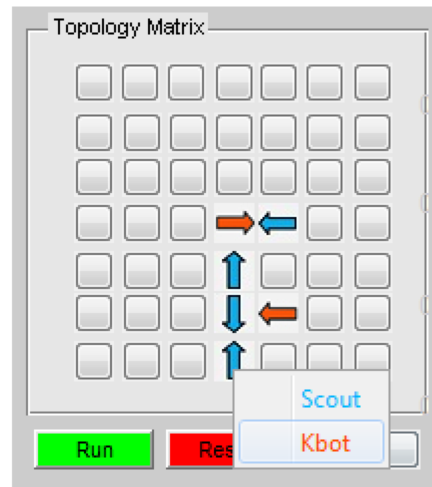

Figure 4.5: Context menu to change the robot type

4.1.3 Topology Graph

In image 4.8 you see the associated topology graph to the matrix in 4.4. It is a simple, directed graph where the nodes depict the active robots and the edges the links between them. Just the base robot is marked red and labled as '0', all others are yellow and consecutively numbered.

To draw the graph, I used the *biograph* from the *Bioinformatics Toolbox* [TMa]. Its input is a connection matrix (short: CM), which is calculated after hitting the run button. Optional is the cell of labels. Unfortunately I found no way to draw the graph directly to an axes, as the toolbox opens its own window. So I had to save the graph as an image file and load it to the axes. This is the reason why you can see a window poping up on slow computers, when the run button is activted. The corresponding excerpt is displayed in figure 4.9.

Run button

The `function run_Callback` of this button contains most of the important code to calculate the *AIM* (=Assembly Incidence Matrix, see 2.2.1) and *CM* (=Connection Matrix, an adjacency matrix), generates the output in the listbox (result tab), draws the topology graph and is finally the place to put your own code for kinematic and dynamic calculations.

At first a nodelist is created, that keeps track of all active robots in the matrix as preparation. In the next step AIM and CM are created, going through the active nodes in the nodelist and marking them, when dealt with. For each entry it is checked if they have an active, adjacent robot and how they are orientated. Next,

```
function pb1_1_Callback(hObject, eventdata, handles)
% hObject handle to pb1_1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

if hasActiveNeighbour(1, 1)
  load ActiveMatrix;
  load RobotTypeMatrix;
  load c11;
  c11=c11+1;
  if(c11==1)
    if (RobotTypeMatrix (1, 1) == 2)
      pic = importdata('UP_red.jpg');
      set(handles.pb1_1,'CData',pic);
    else
      pic = importdata('UP.jpg');
      set(handles.pb1_1,'CData',pic);
    end
    save c11 c11;
    ActiveMatrix(1,1)=1;
    save ActiveMatrix ActiveMatrix;
    omega11 = [0 1 0];
    save omega11 omega11;

  ...
  ...
  ...

  elseif(c11==5)
    Blank2 = importdata('blank.jpg');
    set(handles.pb1_1,'CData',Blank2);
    c11=0;
    save c11 c11;
    ActiveMatrix(1,1)=0;
    save ActiveMatrix ActiveMatrix;
    omega11 = [0 0 0];
    save omega11 omega11;
  end
else
  msgbox('Error,␣only␣nodes␣with␣active␣neighbours␣permitted.', 'Error');
end
```

Figure 4.6: PushButton 1_1 as example for the topology matrix buttons

	Scout	Kbot
Mass:	1 kg	1 kg
Inertia:		
Length:	115 mm	110 mm
Width:	100 mm	100 mm
Height:	100 mm	100 mm
DOF:	2	1

Figure 4.7: The two pre-defined robot types: Scout and Kbot

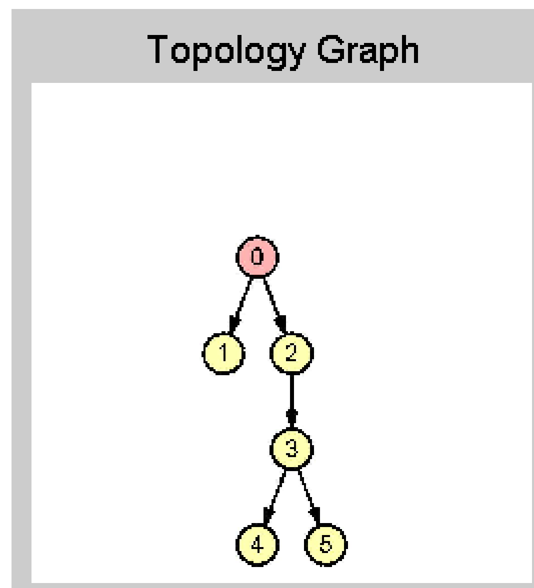

Figure 4.8: Topology graph drawn by biograph

```
% create cell of IDs for biograph
ids = {'0'};
for i = 2:nnz(ActiveMatrix)
   ids{i} = int2str(i-1);
end

% create Biograph
kingraph = biograph(CM, ids);

%Modification of nodes
set(kingraph.nodes,'Shape','circle')
set(kingraph.nodes,'Size',[5 5]);
set(kingraph.nodes,'LineWidth',3);
set(kingraph.nodes,'LineColor',[0 0 0]);
set(kingraph.nodes,'FontSize',21);
Base = getancestors(kingraph.nodes(1),0);
set(Base,'Color',[1 .7 .7]);
set(Base,'Size',[5 5]);

% %Modification of edges
set(kingraph.edges,'LineWidth',3)
set(kingraph.edges,'LineColor',[0 0 0]) %black

set(kingraph, 'ArrowSize', 12);

% Insert to Axes
g = biograph.bggui(kingraph);
f = figure;
copyobj(g.biograph.hgAxes,f);
saveas(gcf, 'AIMGraph','jpg');
close(f);
close(g.hgFigure);
set (handles.figure1, 'CurrentAxes', handles.Plot2);
axis equal;
GraphImage = importdata('AIMGraph.jpg');
% cut a square out of it. Original is always 1201x901
cutImage = GraphImage(1:900, 1:900, :);
image(cutImage);
axis off;
```

Figure 4.9: Creation of the biograph in function run_Callback

```
nodelist = [4 4 0];
% connection matrix is a square matrix, size is the number of active
    robots
CM = zeros(nnz(ActiveMatrix));
 for i = 1:7
   for j = 1:7
     if ActiveMatrix(i, j) ~= 0 && (i ~= 4 | j ~= 4)
       % store this robot in the list of nodes to calculate the AIM
       % matrix later on
       % First column = x-value, second column = y value, third column =
       % 0, if node was not yet dealt with, so initially all
       % one row for each entry, beginning with the base node followed
       % by the others in read order (left to right and up to down)
       nodelist = [nodelist; i j 0];
     end
   end
 end
disp(nodelist);

% create minimum AIM size
AIM = zeros (size(nodelist,1), size(nodelist,1)-1);
numberOfEdges = 0;

% check all neighbours and set 'done flag' (last column)
for x = 1:size(nodelist, 1)
  a = nodelist (x, 1); b = nodelist (x, 2);

  % check if left neighbour exists and is Active
  if ((b > 1) && (ActiveMatrix(a, b-1) ~= 0))
     neighbour = InNodelist (nodelist, a, b-1);
     % if left neighbour is in the nodelist (should be) and was not yet
     % checked...
     if ((neighbour ~= 0) && (nodelist(neighbour, 3) == 0))
       % ...add to AIM
       numberOfEdges = numberOfEdges + 1;
       CM (x, neighbour) = 1;
       AIM (x, numberOfEdges) = getOrientation (nodelist(x, 1), nodelist(
           x, 2), 'left');
       AIM (neighbour, numberOfEdges) = getOrientation (nodelist(
           neighbour, 1), nodelist(neighbour, 2), 'right');
     end
  end
  ...
  ...
    nodelist(x, 3) = 1;
end
save CM CM; save AIM AIM;
```

Figure 4.10: Generation of AIM and CM in function run_Callback

the biograph is set up, so that the there exist labels for each node and that size and shape of the nodes and edges look nice. Then the graph itsself is drawn.

After this, you should put you own code, as explained later on in chapter 4.2.1.

This whole procedure is encapsulated by Matlabs built-in functions `tic` and `toc`. This makes it possible to determine the time needed to calculate the code. It is included in the last part of this function, where the current system time, the elapsed time since calling the function and, by default, the AIM and CM are printed to the output_box in the result tab.

Reset button

The only visible result from this button is, that the topology graph and matrix are cleared. You can use it for instance, if you activated a lot of robots and now want to go on with a rather small topology to save you the trouble to deactivate all manually. As shown in figure 4.11, there are some variables reinitialized to their default value in the background, such as the ActiveMatrix and the variables for the state of the topology matrix. The `eval` function is of great help here, because you can handle the buttons in a loop, and don't have to deal with each one seperately.

Store button

The store button should not be confound with *Save Project* in the File Menu.

Store should be called after you run the simulation and are content with its result so that it is worthy to have a second look at it. This callback 4.12 gathers currently relevant data for the chose topology such as AIM, CM, simulation time, ActiveMatrix and of course the topology itself. These information are stored in a consecutively numbered file in a subfolder, named like the project. So you should make sure, you created a new or loaded an existing project at first. At last an entry is created in the corresponding listbox in the result tab. It is also there, where you find the *restore* button to load all graph, variables and the matrices.

Parameter

Actually there are two different UIpanels just in the same position, each for one of kinematic and dynamic. Yet there is only an example in the kinamtic parameter panel to show how it goes along with the Forward/Inverse-RadioButton. The contents of these panels are dependent on the algorithms used and thus have to be implemented when using this framework.

4.1.4 Kinematic tab only

As already mentioned, the parameter panel is a different one than in the dynamic tab. Yet I described them in their common section, as it is about the same.

```
load ActiveMatrix;
% calls initiallisation, like at startup
Init();

% Set blank image for all except the base button in the topology matrix
    and
% Values for each button to 1, except baseButton, which goes for 2 (Robot
% type)
Blank = importdata('blank.jpg');
for i = 1:7
  for j = 1:7
    if (i == 4 && j == 4)
      up_red = importdata('UP_red.jpg');
      set (handles.buttonBase, 'Value', 2);
      set (handles.buttonBase, 'CData', up_red);
      ActiveMatrix (4, 4) = 1;
    else
      eval (['set␣(handles.pb' num2str(i) '_' num2str(j) ',␣''Value'',␣1)']);
      eval (['set(handles.pb' int2str(i) '_' int2str(j) ',''CData'',Blank);']);
      eval (['ActiveMatrix(' int2str(i) ',' int2str(j) ')␣=␣0;']);
    end
  end
end
save ActiveMatrix ActiveMatrix;

% clear graph and set axis invisible
set (handles.figure1,'CurrentAxes', handles.Plot2);
cla (gca, 'reset');
set (gca, 'XColor', get(get(gca,'Parent'), 'BackgroundColor'));
set (gca, 'YColor', get(get(gca,'Parent'), 'BackgroundColor'));
set (gca, 'TickLength', [0 0]);
```

Figure 4.11: Code of reset_Callback

```
% get all relevant data and store in a consecutively numbered file
load ActiveMatrix;
load RobotTypeMatrix;
load CM;
load AIM;
load eTime;
handles = guidata(hObject);
counter = increment_counter(hObject);
filedir = fullfile(handles.projectname, [int2str(counter) '.mat']);
file = [int2str(counter), '.mat'];
newentry = sprintf('%3i   %-25s %5i', counter, filedir, nnz(ActiveMatrix));
set (handles.listbox1, 'String', [get(handles.listbox1, 'String'); newentry]);
if ~isdir(handles.projectname)
  mkdir (handles.projectname);
end
filetosave = fullfile(handles.projectname, file);

% store Matrices and then append other variables and information for
% all 49 buttons of the topology matrix
save (filetosave, 'ActiveMatrix');
save (filetosave, 'RobotTypeMatrix', '-append');
save (filetosave, 'CM', '-append');
save (filetosave, 'AIM', '-append');
save (filetosave, 'eTime', '-append');
for i = 1:7
  for j = 1:7
    if ~(i == 4 && j == 4)
      eval (['load c' int2str(i) int2str(j) ';']);
      eval (['save (filetosave, ''c' int2str(i) int2str(j) ''', ''-append'');']);
      eval (['button' int2str(i) int2str(j) ' = get (handles.pb' int2str(i) '_'
          int2str(j) ',''CData'');']);
      eval (['save (filetosave, ''button' int2str(i) int2str(j) ''', ''-append'');']);
    else
      load c44;
      save (filetosave, 'c44', '-append');
      button44 = get (handles.buttonBase, 'CData');
      save (filetosave, 'button44', '-append');
    end
  end
end
```

Figure 4.12: How each simulation is saved in store_Callback

Visualization axes and trajectory pulldown menu

The visualization tab on the right-hand side is the main distinction of kinematic and dynamic tab as far as the layout is concerned. The axes if for drawing trajectories. Once there are algorithms implemented you can use the pulldown menu to select different trajectory solvers. For now, this is a dummy function displaying a random trajectory, created with the `animframetraj` function of the *kinematics toolbox* by Brad Kratochvil[Kra09]. As you can see in picture 4.13, it already draws an good-looking, three-colored trajectory of a manipulator. I commented out the line containing `nice3D();`, as it resulted in an ongoing scaling of the axes, which is not appropriate for a GUI.

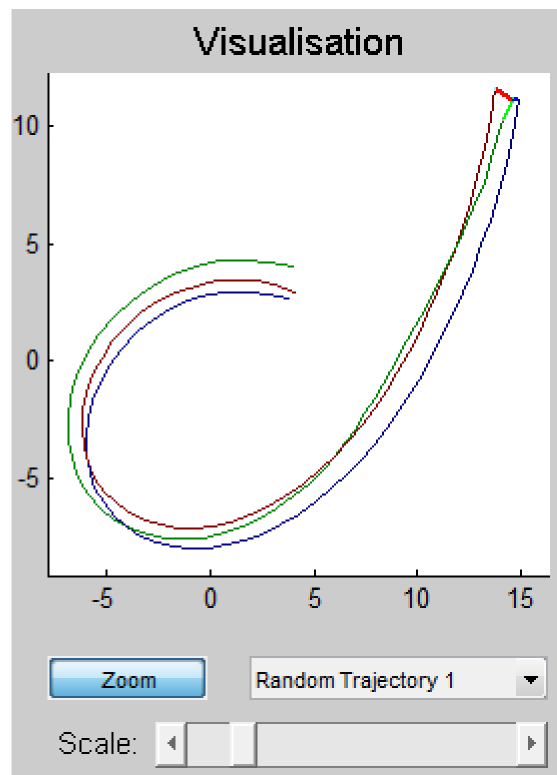

Figure 4.13: Example Trajectory

Pan-Zoom-Button

As the name implies, this button toggles the visualization axes between pan and zoom mode, 4.14. They behave just like the accustomed Matlab build-in tools.

In pan-mode you can simply drag the canvas. If you are double-clicking on the

```
% --- Executes on button press in panZoomButton.
% This button toggles the Visualisation axes between zooming and panning
function panZoomButton_Callback(hObject, eventdata, handles)
% hObject handle to panZoomButton (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of panZoomButton

set (handles.figure1, 'CurrentAxes', handles.axes1);
if get(hObject, 'Value')
  set (hObject, 'String', 'Zoom');
  zoom on;
else
  set (hObject, 'String', 'Pan');
  pan on;
end
```

Figure 4.14: Toggle pan and zoom mode for axis

canvas, you reset to the default view.

In zoom-mode you can either left-click to enlarge at that position or you can drag a rectangle to zoom to that specific image section. Again, double-clicking resets to default zoom. Shift+click will zoom out.

Scale slider

This simple slider scales the whole image, while focusing on the origin. It can be helpful to get a rough scaling done with this slider and using the zoom-function afterwards. The two lines of code can be seen in 4.15

```
set (handles.figure1, 'CurrentAxes', handles.axes1);
set(gca,'xlim',get(gcbo,'value')*[-1 1], 'ylim',get(gcbo,'value')*[-1 1]);
```

Figure 4.15: Scale an axis using a slider

4.1.5 Dynamic tab only

Visualization

Again, like in the kinematic tab, there is a visualization panel on the right. This time, there are four plots and one pulldown-menu. The latter is yet used to draw

examples like in image 4.16, but is determined to let the user choose between different integrators. The four axes illustrate important variables:

- q, position of a robot
- $\dot{q}$, its velocity, derivative of q
- $\ddot{q}$, the acceleration, derivative of $\dot{q}$
- τ, a force

4.1.6 Result tab

The structure of this tab is quiet simple: It just contains two listboxes and one button. The bigger listbox on the left is the output_box. It is used to display important information about the simulations, like AIM and CM. It is either filled during the simulation or when an former one is restored. The first possibility is shown in figure 4.17 with the tic-toc structure surrounding most of the code.

The right listbox contains entry with three coloumns for each stored simulation. At first is a sequence number, followed by the full filename. The last coloumn display the number of active robots, so that you have an indication to relocate older simulations.

Beneath this listbox is the restorebutton, which is already explained at its counterpart (see 4.1.3) and sets the selected, saved topology along with the correct variables.

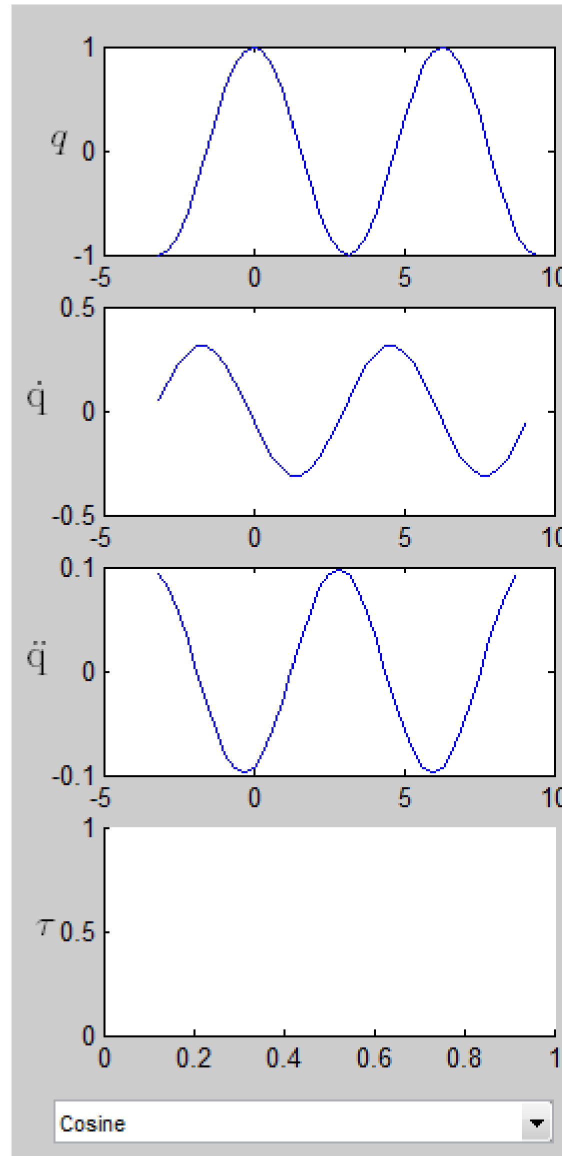

Figure 4.16: Example Dynamics

```
tic
% put code to be timed here. Simulations, calculation of matrices...

eTime = num2str(toc);
save eTime eTime;
% display nodelist and AIM in output_box in the result tab
disp ('nodelist:␣'); disp(nodelist);
disp ('AIM:␣'); disp(AIM);
c = fix(clock);
hours = num2str(c(4));
if (c(5) < 10)
  minutes = ['0' num2str(c(5))]
else
  minutes = num2str(c(5));
end
set(handles.output_box, 'String', strvcat(get(handles.output_box, 'String'),
    ...
  ['Time:␣' hours ':' minutes '.␣Time␣needed␣for␣calculation:␣' eTime '␣seconds'], ...
  'AIM:␣', num2str(AIM), ...
  'Adjacency␣Matrix:␣', num2str(CM), '␣'));
```

Figure 4.17: Scale an axis using a slider

4.2 User Guide

4.2.1 Customizations

I already mentioned several locations, where you should add or customize code to your suit. In this section I want to point them out again, more specificly.

Most of your own code should go to the run_Callback. At the end, but just before the output to the listbox you have to include several things: Most important the implementations for kinematics and dynamics. Then plots for q, $\dot{q}$, $\ddot{q}$ and τ when using dynamics.

The parameter panel for both tabs can contain different possibilities for settings, to configure your simulation individually.

The callback of the pulldown menus in both, kinematics and dynamics, also need adjustments. Here you can include different trajectory solvers (kinematics) and integrators (dynamics).

At last I want to mention, that you can easily change the pictures in the topology matrix. There are a total of nine: a blank one and four for each robot type (“Up.jpg”, “Up_red.jpg”, “Right.jpg”, “Right_red.jpg” and so on). Simply replace these icons with your own.

4.2.2 Quick Start

Here is a brief guide, how you should start your first own simulation, after the framework was adapted to your purpose.

1. Choose *New Project* from the file menu and give it a distinctive name, best with date
2. Decide whether to use kinematics or dynamics
3. Choose forward or inverse
4. Create a topology to run the simulations on (For further information see 4.1.3 on page 22).
5. Set Parameter
6. Choose a solver or iterator
7. Click run to start the simulation
8. View the results, don't forget to have a look at the output_box in the result tab; if they please you, store them
9. Repeat steps 4 - 8 as long as you like
10. You can restore simulations from the result tab to review or modify them
11. Save your project (File Menu) if you are finished or want to take a break

4.2.3 Unused functions

At the end of the .m file you find some commented code, that I originally wrote but did not use in the end or just some small snippets, like create_functions, that merely set the backgroundcolor of specific components (like edit boxes).

At first I didn't use the biograph and built my own by drawing arrows and ellipses (using a patch to color the latter), using the equally named functions found at Matlabs File Excange [TMc].

5 Conclusion

Hopefully this GUI will help both, students and developers, to understand and work with kinematic and dynamic models in hyper redundant robot systems. As the kinematic and dynamic concepts are not included, it would be possible to use this GUI to compare different approuces like classical mechanics and screw theory.

I hope this thesis will be part of the advancement in this topic, as I think, there could be some very promising products in the future.

Just think of swarms of robots, that can enter burning houses to search and possibly even rescue people by forming different topologies to navigate through different obstacles. Or secure areas, that are too dangerous for humans to enter. If we think even more ahead, nano robots, that are injected in our bodies could identify and combat medical conditions, that can be hardly treated by now, like cancer. I don't think, we can grasp the whole scope of this by now, do you?

Bibliography

[Alt10] Yair Altman. Undocumented Matlab. http://undocumentedmatlab.com/blog/tab-panels-uitab-and-relatives/, 2010.

[And07] Victor Andrade. Cylindrical Robot Simulator. http://www.mathworks.com/matlabcentral/fileexchange/24361-robotics, 2007. [Updated in April 2009].

[Com11] European Research Community. Symbrion and Replicator. http://www.symbrion.eu/, 2011. [also accessible via http://www.replicators.eu/].

[Hig09] Badr Higab. Robotics. http://www.mathworks.com/matlabcentral/fileexchange/24361-robotics, 2009.

[Kol07] Ahmad Kolahi. 3_bar robot kinematics and kinetics analysis and simulation. http://www.mathworks.com/matlabcentral/fileexchange/14031-3bar-robot-kinematics-and-kinetics-analysis-and-simulation, 2007.

[Kra09] Brad Kratochvil. Kinematics Toolbox. http://www.mathworks.com/matlabcentral/fileexchange/24589-kinematics-toolbox, 2009.

[Ril07] Don Riley. 3D Puma Robot Demo. http://www.mathworks.com/matlabcentral/fileexchange/14932-3d-puma-robot-demo, 2007.

[RMM94] S. Shankar Sastry Richard M. Murray, Zexiang Li. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[TMa] The MathWorks. Bioinformatics Toolbox. http://www.mathworks.de/products/bioinfo/?s_cid=HP_FP_ML_bioinfo.

[TMb] The MathWorks. MatlabCentral. http://www.mathworks.de/matlabcentral/.

[TMc] The MathWorks. MatlabCentral File Exchange. www.mathworks.com/matlabcentral/fileexchange/.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Manuel Bischof)