Oliver Ruf

Dynamic Modeling of Robots with Kinematic Loops

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 1.8.2014

Thesis supervisors:

Prof. em. D. Sc. (Tech.) Aarne Halme
Aalto University, Helsinki

Prof. Thomas Gustafsson
Luleå University of Technology, Kiruna

Thesis advisors:

Roberto Lampariello
DLR German Aerospace Center, Oberpfaffenhofen

Martin Stelzer
DLR German Aerospace Center, Oberpfaffenhofen
Closed kinematic loops occur in many different robotic systems. Simply grasping an object with two or more arms, generates a closed kinematic loop. Recently, ESA commissioned a feasibility study for the deorbiting of Europe’s largest earth observing satellite – Envisat. It is expected that multiple arms are needed in order to grasp this satellite, which would generate a closed kinematic loop. The German Aerospace Center DLR would like to contribute to this feasibility study and simulate the deorbiting of Envisat. The SpaceDynamics Library, a simulation environment of the DLR, is however incapable to simulate robots with closed kinematic loops. The thesis discusses the dynamic modeling of robots with closed kinematic loops and shows the implementation and validation of an algorithm, in order to enable the dynamic modeling of closed loops with the SpaceDynamics Library.

Keywords: Dynamic Modeling, Kinematic Loops, Closed Loops, Space Robots
## Contents

*Abstract* | [III]  
---|---  
*Contents* | [IV]  
*List of Symbols and Abbreviations* | [VI]  
*List of Figures* | [VIII]  
*List of Tables* | [IX]  

### 1. Introduction

1.1.1. Kinematic Modeling | [2]  
1.1.2. Dynamic Modeling | [2]  
1.2. Kinematic Loops and Kinematic Trees | [3]  
1.2.1. Kinematic Loops | [3]  
1.2.2. Kinematic Trees | [3]  
1.4. Motivation | [4]  
1.5. Outline | [5]  

### 2. Examples and Applications of Kinematic Loops

2.2. Humanoid Robots | [7]  
2.3. Space Robots with Multiple Manipulators | [8]  
2.4. Parallel Robots | [9]  

### 3. Dynamic Modeling of Kinematic Trees

3.1. Equations of Motion | [10]  
3.2. Inverse Dynamics - Recursive Newton-Euler Method | [11]  
3.2.1. Recurrence Relations | [11]  
3.2.2. The System Model | [11]  
3.2.3. The Recursive Newton-Euler Method | [12]  
3.2.4. Equations in Link Coordinates | [13]  

### 4. Dynamic Modeling of Kinematic Loops

4.2. Dynamic Modeling with Generalized Coordinates | [16]  
4.2.1. Generalized Coordinates of Kinematic Loops | [16]  

---
List of Symbols and Abbreviations

General Symbols

\( a_i \)  acceleration of link \( i \) in absolute coordinates
\( C \)  matrix of Coriolis and centrifugal terms
\( f^*_i \)  force that acts on link \( i \), depending on spatial inertia
\( f^e_i \)  external force that acts on link \( i \)
\( f_i \)  total force that is transmitted through joint \( i \) (from link \( i - 1 \) to link \( i \))
\( g \)  gravity vector
\( H \)  joint space inertia matrix
\( I_i \)  spatial Inertia of link \( i \) in absolute coordinates
\( I'_i \)  spatial Inertia of link \( i \) in link coordinate system \( i \)
\( n \)  number of links and joints in the system
\( q_i \)  joint position or displacement of joint \( i \)
\( q \)  vector of joint positions
\( \dot{q}_i \)  joint velocity of joint \( i \)
\( \dot{q} \)  vector of joint velocities
\( \ddot{q}_i \)  joint acceleration of joint \( i \)
\( \ddot{q} \)  vector of joint accelerations
\( s_i \)  vector of link \( i \) in absolute coordinates
\( s'_i \)  vector of link \( i \) in link coordinate system \( i \)
\( \tau_i \)  force of joint \( i \)
\( \tau \)  vector of joint forces
\( v_i \)  velocity of link \( i \) in absolute coordinates
\( _iX_j \)  homogeneous transformation between coordinate system \( i \) and \( j \)

Note: Bold symbols represent vectors, whereas normal symbols represent scalars.
Additional Symbols of Chapter 4

\( J_C \) constraint matrix of all closed Loops
\( J_{Cm} \) linearly independent rows of \( J_C \)
\( J_G \) linearly dependent columns of \( J_{Cm} \)
\( J_{Li} \) velocity constraint matrix of Link \( L \)
\( J_S \) linearly independent columns of \( J_{Cm} \)
\( m \) number of independent constraints
\( N_A \) number of actuated joints
\( N_F \) number of degrees of freedom
\( N_J \) number of all joints in the closed loop system
\( N_O \) number of joints in the open loop system
\( S \) Jacobian matrix
\( \tau_A \) actuator forces in the closed loop system
\( \tau_G \) generalized forces
\( \tau_O \) joint forces in the open loop system
\( \theta_A \) joint angles of actuated joints
\( \theta_G \) generalized coordinates (independent joints)
\( \theta_J \) all joint angles
\( \theta_O \) joint angles of the open loop system
\( \theta_S \) joint angles of the dependent joints
\( W \) Jacobian matrix

Abbreviations

DEOS  Deutsche Orbitale Servicing Mission
DLR  German Aerospace Center
EVA  Extra-vehicular activity
ESA  European Space Agency
GEO  Geostationary orbit (altitude 35786km)
IADC  Inter-Agency Space Debris Coordination Committee
LEO  low Earth orbit (altitude 160km - 2000km)
RMC  Robotic and Mechatronic Center
SDL  SpaceDynamics Library
TORO  Torque controlled humanoid Robot of the DLR
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Humanoid Justin holding a basketball, DLR</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>Comparison between Kinematic Loops and Kinematic Trees</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>Piano Mover’s Problem</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>Humanoid TORO, DLR</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>Space Robots</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td>Hexapod Platform</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Closed Loop with varying Degrees of Freedom</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Kinematic Loop and Kinematic Tree System</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>Closed Loop</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>Forming of W and S</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>Closed Loop with indeterminate Forces</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Class Diagram of the SpaceDynamics Library – 1</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>Class Diagram of the SpaceDynamics Library – 2</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Simpack</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Test Procedures</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Forward Kinematics Test 1</td>
<td></td>
</tr>
<tr>
<td>6.4</td>
<td>Forward Kinematics Test 2</td>
<td></td>
</tr>
<tr>
<td>6.5</td>
<td>Dynamics Test for Kinematic Tree with Fixed Base</td>
<td></td>
</tr>
<tr>
<td>6.6</td>
<td>SpaceDynamics Library</td>
<td></td>
</tr>
<tr>
<td>6.7</td>
<td>Dynamics Test for Kinematic Tree with Free-Floating Base</td>
<td></td>
</tr>
<tr>
<td>6.8</td>
<td>Results of Forward Kinematics Test</td>
<td></td>
</tr>
<tr>
<td>6.9</td>
<td>Dynamics Test for Closed Loop with Fixed Base</td>
<td></td>
</tr>
<tr>
<td>6.10</td>
<td>Results of Forward Kinematics Test</td>
<td></td>
</tr>
<tr>
<td>6.11</td>
<td>Inverse Dynamics Test for Closed Loop</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>Realistic Test Model</td>
<td></td>
</tr>
<tr>
<td>7.2</td>
<td>DEOS Simulator</td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

5.1. Important Variables of the Class Model ........................................ 29
5.2. Important Functions of the Class spd .......................................... 30
5.3. Important Functions of the Class matrix ....................................... 31
5.4. Important Functions of the Class rot ........................................... 32
5.5. Parameters of the model function .............................................. 36
5.6. Parameters of the f_{kin,e} function ........................................... 37
5.7. Parameters of the i_{dyn,fix} function .......................................... 37
5.8. Parameters of the f_{dyn,fix} function .......................................... 38
5.9. Parameters of the i_{dyn,fix} function .......................................... 38
5.10. Parameters of the f_{dyn,fix} function ......................................... 39

6.1. Results of Forward Kinematics Test 1 .......................................... 42
6.2. Results of Forward Kinematics Test 2 .......................................... 45
6.3. Model Configuration of Dynamic Test .......................................... 45
6.4. Results of the Forward Dynamics Test ......................................... 46
6.5. Results of the Inverse Dynamics Test .......................................... 46
6.6. Results of the Forward Dynamics Test ......................................... 47
6.7. Results of the Inverse Dynamics Test .......................................... 47
6.8. Model Configuration of Dynamic Test .......................................... 49
6.9. Results of the Forward Dynamics Test ......................................... 50
6.10. Results of the Inverse Dynamics Test ......................................... 50
6.11. Parameters of the H_{CL,fix} function .......................................... 51
6.12. Parameters of the i_{dyn,CL,fix} function ..................................... 51
6.13. Parameters of the f_{dyn,CL,fix} function ..................................... 52
6.14. Model Configuration of Dynamic Test .......................................... 52
6.15. Results of Forward Kinematics Test .......................................... 53
6.16. Results of the Forward Dynamics Test ......................................... 54
6.17. Results of the Inverse Dynamics Test ......................................... 54
6.18. Model Configuration of Dynamic Test .......................................... 55
6.19. Results of Forward Kinematics Test .......................................... 55
6.20. Results of the Inverse Dynamics Test ......................................... 57
Listings

5.1. spacedyn/tests/minimal_model_CL.def ....................................... 33
5.2. Example of Model Loading ...................................................... 36

6.1. spacedyn/src/f_dyn_fix.cpp .................................................... 48

A.1. Forward Kinematics Test .......................................................... 61
A.2. Dynamics for Fixed Base Kinematic Tree System .......................... 62
A.3. Inverse Dynamics for Fixed Base Kinematic Tree System .............. 63
A.4. Forward Dynamics for Fixed Base Kinematic Tree System .............. 64
A.5. spacedyn/tests/LBR3_JtoJ_light.def ......................................... 65
A.6. spacedyn/tests/LBR3_JtoJ_light_CL.def ...................................... 67
A.7. spacedyn/tests/LBR3_JtoJ_light_CL2.def ...................................... 69
1. Introduction

Artificial objects exist in the space environment of Earth since the launch of the first satellite Sputnik-1 in 1957. Once in earth's orbit, artificial objects remain there for decades. In the low-earth-orbit (LEO), atmospheric drag and solar pressure effects lead to altitude loss of the objects until they finally burn up in the Earth’s atmosphere. According to [1], only 6% of artificial objects in space were active satellites in 2002.

All man-made objects orbiting the earth, which are not functional, are called space debris [2] and make up the biggest part of the space object population. Space debris consists of inactive satellites, parts of launch vehicles and products of on-orbit collisions and explosions. The latter is the largest contributor to space debris and therefore the most dangerous. The debris population in operationally important orbits is growing due to new space missions. A chinese anti-satellite test in 2007 and the collision of the Iridium 33 and the Cosmos 2251 satellite in 2009 led to a large growing of debris in recent years [3]. Every artificial object can collide with other objects in space and produce even more debris, which is known as the cascading effect. Space debris represents a huge hazard to active and future satellites because of possible collisions. To attack the task of reducing space debris, the Inter-Agency Space Debris Coordination Committee (IADC) was formed in 1993. In 2002 the members agreed on the guidelines to decrease space debris.

One possible solution to reduce space debris is the servicing of satellites. Servicing includes life extension, repair and removal of satellites [4]. Especially in the geostationary orbit (GEO) and in polar orbits the removal of satellites is needed. In these orbits almost no atmospheric drag effects exist, that would lead to a self regulated removal of the debris. In addition, these orbits are operationally important for Earth observation and communication. One example of on-orbit servicing is DEOS (Deutsche Orbitale Servicing Mission) [5]. The goal of the DEOS mission is to grasp a tumbling target satellite with the manipulator of a service satellite and to eventually deorbit the target satellite.

The largest earth-observing satellite of the European Space Agency (ESA) is Envisat. It is an Environmental satellite with a launch mass of 8221 kg which is orbiting the Earth on a sun-synchronous polar orbit at an altitude of 790 km. After 10 years of operation, the contact to Envisat was lost on April 12th, 2012. Since then, Envisat is not controllable anymore and is therefore considered as space debris. At the 63rd International Astronautical Congress, Martha Mejia-Kaiser described Envisat as a large danger for other satellites [6]. Due to little drag effects at the current altitude, Envisat will remain in the orbit of Earth for around 150 years. With the cross-section of 26 meters the probability of a collision with another satellite is
1. Introduction

very high. Against the guidelines of the IADC, Envisat was operated until too little fuel was left for moving it to a lower orbit where a shorter lifetime would result. For this reason ESA could be held liable for occurring collisions. A feasibility study for the deorbiting of Envisat was now commissioned by ESA. Considering the size and mass, it is expected that multiple arms are necessary to grasp Envisat for deorbiting. By grasping Envisat with multiple arms, a closed kinematic loop would occur. This thesis explains now why the *Dynamic Modeling of Robots with Kinematic Loops* is needed for the simulation of the deorbiting of Envisat and how the modeling can be done.

1.1. Kinematic and Dynamic Modeling of Robots

1.1.1. Kinematic Modeling

The kinematic modeling of robots sets the position of the end effector of a robot in relation to the positions and values of the robot's joints. This is done without considering the forces or torques that are responsible for the motion [7]. The kinematic modeling plays an important role in the design and analysis of a robot and is the foundation of the dynamic modeling.

Forward kinematics is the computation of the pose (position and orientation) of the end effector of a robot for given joint positions and values and the geometric structure of the robot. If the velocity of all joints is added to the given values, the resulting velocity of the end effector can be calculated. This is called forward instantaneous kinematics.

Inverse kinematics is exactly the opposite: It is the computation of the joint position and values required for a given pose of the end effector. Similar to the forward kinematics, the computation of the joint velocities for a given end effector velocity is called inverse instantaneous kinematics.

1.1.2. Dynamic Modeling

In contrast to the kinematic modeling, the dynamic modeling of robots sets the forces of actuators and forces due to contact with external objects in relation to the acceleration and motion of the robot [8]. Therefore the dynamic modeling, especially the *inverse* and the *forward dynamics* are an important part in simulating or controlling robotic systems.

The inverse dynamics is the computation of the required actuator torques and forces for a given trajectory the robot should perform. The trajectory of the robot is described by the joint angles, the joint velocities and the joint accelerations. To control a robotic system, the actuator torques and forces that are required for the robot to perform a certain motion have to be calculated. Therefore the inverse dynamics is the computation that is needed for controlling a robotic system.
1. Introduction

In contrast to the inverse dynamics, the forward dynamics is the computation of the joint accelerations for given actuator torques and forces. In this case the trajectory performed by the robot is calculated for certain torques and forces that are applied to the actuators. This computation is needed for simulating the motion of the robotic mechanism.

1.2. Kinematic Loops and Kinematic Trees

![Humanoid Justin holding a basketball, DLR](image)

Figure 1.1.: Humanoid Justin holding a basketball, DLR

1.2.1. Kinematic Loops

Closed loops or kinematic loops occur in many different robot mechanisms. According to Featherstone [9, p. 155] a “kinematic loop exists in a mechanism if it is possible to trace a circuit from some link back to itself without traversing any joint more than once”. This definition holds true even if there is an external object involved, as for example a robot carrying a rigid object with two robotic arms. In Figure 1.1 the humanoid Justin from the German Aerospace Center (DLR) forms a closed loop with its two arms by holding a basketball. More examples of robotic systems with kinematic loops are (space-) robots with cooperative manipulators, humanoid robots or parallel robots and will be described in chapter 2.

1.2.2. Kinematic Trees

A mechanism that contains however no kinematic loops is called tree-structure open kinematic chain, open loop system or kinematic tree. According to Featherstones definition of kinematic loops, in a kinematic tree it is only possible to trace a circuit from a link back to itself by traversing some joints more than once. The difference between kinematic loops and kinematic trees is shown in Figure 1.2. The two
1. Introduction

kinematic loops in Figure 1.2 (a) are cut at the red marked joints to form a kinematic tree structure shown in Figure 1.2 (b).

![Figure 1.2.: Comparison between Kinematic Loops and Kinematic Trees](image)

1.3. Fixed Base, Free Flying and Free Floating Systems

Ongoing research regarding kinematic loops includes the dynamic modeling of robotic systems in space, where special dynamic effects occur [5]. These are for example closed loops in free flying and free floating robotic systems which represent particularities in space robotics. In fixed base robotic systems, that mostly occur on Earth, the movement of the robotic arms does not affect the pose of the base of the robot. It is assumed to be fixed. In contrast to fixed base robots, the base of a free flying or free floating robot changes its position and orientation according to the movement of the robots arm [10]. This is due to the conservation of momentum and is mostly the case for space robots. The difference between free flying and free floating base is that a free floating base is not actively controlled whereas the pose of a free flying robot is controlled by thrusters and reaction wheels. So far there has not been a free flying or free floating base robot with closed kinematic loops in space.

1.4. Motivation

The SpaceDynamics Library (SDL) [11] is an existing simulation environment at the Robotic and Mechatronic Center (RMC) of the DLR. It is used for calculating the kinematics and dynamics for all kind of robotic applications, especially free-floating orbital robots. The SpaceDyn library however lacks the functionality of modeling robotic systems with closed kinematic loops. The dynamics computation of a robotic system with closed kinematic loops is needed at the RMC in order to simulate a free-flying orbital robot with multiple arms that forms a kinematic loop while grasping an object. This simulation could be used for the feasibility study commissioned by ESA for the deorbiting of Envisat.
1. Introduction

Different solutions to calculate the kinematics and dynamics of closed kinematic loops exist already in the literature, for example [9, 12–14]. One of the existing approaches should be selected and implemented in the SpaceDynamics Library to enable the modeling of closed kinematic loops. The chosen implementation should involve a state-of-the-art method in terms of computation time. The question that is answered with this thesis is: How to compute the inverse and forward dynamics for a fixed base, free-flying or free-floating robotic system including kinematic and actuation redundancy with $N$ arms and $L$ closed loops.

The goal of the research is the simulation of the kinematics and dynamics of a closed loop robotic system with the SpaceDynamics Library. For successful completion of the thesis, first one existing solution for the kinematic and dynamic modeling of robots with closed kinematic loops has to be chosen. Second, the chosen algorithm has to be implemented in the SpaceDynamics Library. Finally its functionality has to be validated with SIMPACK, that is a commercial multi-body simulation software, in order to allow the simulation of a use case scenario with the SpaceDynamics Library.

A possible use case scenario for the SpaceDynamics Library is the Envisat case, where the movement of a satellite that is grasped with multiple arms should be simulated. The Robonaut case (see section 2.3) is a possible scenario as well. In this case, on-orbit manipulation and assembly tasks with multiple arms are performed. The SpaceDynamics Library is however not restricted to space robot modeling, therefore another use case scenario could be the biped walking case. For this case, the simulation of a humanoid robot forming a closed loop with two legs on the floor is needed.

A challenging issue is to understand the theory, which is quite complex. Since the software should be able to control a robotic system, the dynamics computation has to be done in real time. Therefore the implementation of the software is also challenging. Furthermore the validation of the algorithms has to be done in a very accurate way, to confirm the functionality and validity of the computed dynamics.

1.5. Outline

After this introduction, chapter 2 shows now some examples and applications of robot mechanisms including closed kinematic loops in different environments. After that, the general dynamic modeling of robots is explained in chapter 3. Chapter 4 extends the general dynamic modeling by describing methods for systems with kinematic loops. Within this scope, the report presents two different approaches of Featherstone [9] and Nakamura [12] to compute the inverse and the forward dynamics of closed loops. Chapter 5 describes how the the SpaceDynamics Library works and gives detailed information about important functions. In chapter 6 the implementation of the algorithms to calculate the dynamics of closed loop systems is shown. Furthermore, the tests and validation of the implemented algorithms is presented. Chapter 7 shows the results of this work and gives a summary of the report. Finally, an outlook on further work is given.
2. Examples and Applications of Kinematic Loops

This chapter gives a summary of the most important examples and applications of robots with kinematic loops. Among them are cooperative manipulators, humanoid robots, space robots and parallel robots. These systems are described in detail and an overview of existing literature for further reading is given.

2.1. Cooperative Manipulators

Closed loops occur especially in cooperative manipulators and humanoid robots. Cooperative manipulation is defined as the manipulation of one object with multiple robotic arms [15]. Cooperative manipulation is used for certain tasks, that are not possible to perform with only one arm, for example unscrew the cap of a bottle. For this and other applications a second arm is needed. Furthermore, the ability to manipulate heavier objects with multiple arms than possible with only one arm is a goal of cooperative manipulators.

Figure 2.1.: Piano Mover’s Problem, [16]

Figure 2.1 shows a piano that is moved by three cooperative mobile manipulators, where three closed kinematic loops occur. The motion planning for systems with kinematic loops is however more difficult than for kinematic tree systems. Therefore, Cortés [16] describes computational efficient motion planning algorithms for systems with closed loops.
2.2. Humanoid Robots

Humanoid robots are designed to perform the same manipulation tasks as humans. Simply holding an object with two or more arms forms a kinematic loop. A humanoid robot with two legs, standing on the floor is also creating a closed loop through the ground. Here the dynamic modeling is very important for the robot in order to not fall down due to different weight distribution on the legs. Figure 2.2 shows TORO, the Torque controlled humanoid Robot of the DLR, that is able to walk and to compensate hits against the legs. TORO is forming a closed loop with both legs on the floor.

Much literature exists about the dynamic modeling of humanoid robots. presents a general dynamic modeling method that is intended for the modeling of humanoids. Despite that, it describes the modeling of free flying robots with closed loops and is therefore very interesting. Due to the general approach this method is presented in chapter.
2. Examples and Applications of Kinematic Loops

2.3. Space Robots with Multiple Manipulators

For the simulation of the deorbiting of Envisat, the dynamic modeling of robots with kinematic loops is needed. Thus, space robots are the main applications intended for the dynamics computation of this research. Although much literature exists to space robots with multiple manipulators, no free-floating or free-flying robot performing closed loop manipulation existed in space so far.

A generalized formulation of the dynamic equations for a space robot with open and closed chain configurations is described by [13]. Further the control and the kinematic and dynamic modeling of space robots with kinematic loops is described in [14,18–20].

Future orbital robotic concepts include the control of autonomous space robots for construction [21] and on-orbit maneuvering of large space structures [22]. Future exploration missions will also make use of in-space assembly [23], where closed loops occur as well.

![Figure 2.3.: Space Robots](image)

(a) Robonaut, [10]  
(b) Construction of Large Space Structures, [21]

Another future concept of space robotics is the Robonaut [10] developed by NASA. It is a humanoid space robot intended to perform extra-vehicular activity (EVA) tasks and planetary exploration. The controlling is done by Telepresence, which is similar to a remote control. The astronauts are able to steer the Robonaut by joysticks. The Robonaut is able to perform closed-loop manipulation, but it is considered to be always connected to a spacecraft and is therefore a fixed-base system. Of course the movement of the Robonaut adds a dynamic momentum on the pose of the spacecraft, which is a free flying base system. In case of the ISS being the spacecraft, this force is very small due to the mass differences and can therefore be neglected.
2.4. Parallel Robots

In parallel robots, closed-loops exist by definition: A parallel manipulator is a closed loop mechanism with one end effector that is connected to the base by multiple independent kinematic chains [24].

Figure 2.4 shows a Gough platform which is a well known example for a parallel robot. In that sense the term Hexapod describes a parallel robot with six independent kinematic chains (legs). The major advantage of parallel robots is the ability to handle big loads compared to serial manipulators. Another advantage is the high accuracy of the positioning of the end effector. Since the end effectors position is dependent on multiple kinematic chains, the workspace of a parallel robot is rather small.

![Hexapod Platform](http://en.wikipedia.org/wiki/File:Hexapod0a.png) (March 10, 2014)

The main applications for parallel robots are flight and driving simulators, high speed and high accuracy manufacturing and mirror alignment system in telescopes. An example of high speed and high accuracy manufacturing is PCB assembly. In February 1999, a fixed base parallel robot with 8 legs was used in space during the Space Shuttle mission STS-63, to prevent the payload from vibration [24].

The dynamic modeling of parallel robots is described in [25], where Rose presents a method to compute the inverse dynamics in real time for controlling the parallel robot TRIGLIDE. However the dynamic modeling was assumed to not fit to the general approach of this research, therefore this method is not described in detail.
3. Dynamic Modeling of Kinematic Trees

As already described in the introduction, the dynamic modeling represents the relationship between the motion of a robot and the forces that act on a robot. These forces are the forces and torques that are applied to the actuators of the robot as well as the external forces that result due to the contact of the robot with other objects. If the actuator forces $\tau$ and the contact forces $f$ are known, the resulting joint accelerations $\ddot{q}$ are calculated by the forward dynamics. This calculation is needed for simulating robotic systems.

The motion of the robot is described by the joint angles $q$, the joint velocities $\dot{q}$ and the joint accelerations $\ddot{q}$. If the motion of a robot is known, the required actuator forces $\tau$ in order to perform this motion are calculated by the inverse dynamics. Therefore, the inverse dynamics is needed for the controlling of a robotic system.

This chapter explains now the basic equations for the dynamic modeling of kinematic tree systems and shows the most efficient way to perform the inverse dynamics calculation, which is needed for the dynamic modeling of robots with closed loops.

3.1. Equations of Motion

The equations of motion describe the relationship between the actuator forces and the motion of a robot. Therefore the equations of motion are the fundamental equations of the dynamic modeling. The joint-space formulation of the equations of motion according to [8] is

$$H(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau$$

(3.1.1)

The vectors $q, \dot{q}, \ddot{q}, g, \tau \in \mathbb{R}^{N_F}$ represent the joint position $q$, the joint velocity $\dot{q}$, the joint acceleration $\ddot{q}$, the gravitational terms $g$ and the forces $\tau$, with the degrees of freedom of the system $N_F$. $H \in \mathbb{R}^{N_F \times N_F}$ is the generalized inertia matrix of the system. From $C \in \mathbb{R}^{N_F \times N_F}$, the Coriolis and centrifugal terms are represented by $C\dot{q}$. If a force $f$ is applied to the end-effector of the robot, the term $J^T f$ has to be added to the right side of (3.1.1) where $J$ is the Jacobian matrix of the end-effector. The dependencies are shown by the brackets: The inertia matrix $H$ depends on $q, g$ depends on $q$ and $C$ depends on $q$ and $\dot{q}$. 
The equations of motion (3.1.1) can be used to calculate the inverse dynamics of the system by calculating $\tau$ directly. By solving the equation of motion for $\ddot{q}$, it can be used to calculate the forward dynamics:

$$\ddot{q} = H(q)^{-1} \left[ \tau - C(q, \dot{q}) \dot{q} - g(q) \right]$$  

(3.1.2)

However, the computational complexity of computing the inverse and forward dynamics directly from these equations is typically $O(n^4)$ [9] and therefore very high. This is mostly due to the calculation of $H$ with complexity $O(n^2)$ and $C$ with complexity $O(n^3)$. The next section introduces the recursive Newton-Euler algorithm, that reduces the computational complexity of the inverse dynamics to $O(n)$.

### 3.2. Inverse Dynamics - Recursive Newton-Euler Method

Inverse dynamics is the computation of required actuator torques $\tau$ of a robot in order to perform a given trajectory. The actuator torques are needed to control a robotic system. Therefore, the inverse dynamics are part of the control loop. To allow real-time control, the calculation of the inverse dynamics must be executed very fast.

The most efficient way to calculate the inverse dynamics is the recursive Newton-Euler Method as described in [9]. It was first presented by Luh, Walker and Paul [26] and is based on recurrence relations, which reduce the computational complexity from $O(n^4)$ to $O(n)$.

#### 3.2.1. Recurrence Relations

In general, a recurrence relation is a recursive sequence, that defines each term as a function of the preceding terms. For example the Fibonacci numbers are defined by the following recurrence relation: $F_n = F_{n-1} + F_{n-2}$ with the starting values $F_0 = 0$ and $F_1 = 1$. A recurrence relation in dynamic modeling means for example, that the acceleration of link $i$ is a function of the acceleration of link $i - 1$. With the use of partial results that recurrence relations offer, unnecessary calculations are reduced. Therefore, algorithms that make use of recurrence relations are faster than algorithms without recursion. A more detailed description how to increase the efficiency of algorithms with the help of recurrence relations is given in [9, pp. 68]

#### 3.2.2. The System Model

The use of recurrence relations requires a certain system model of the robot. Therefore a robot should be defined by $n$ links, that are connected by $n$ joints, each with one degree of freedom. Joint $i$ connects link $i - 1$ and link $i$, where link 0 is the base of the
robot. One coordinate system is attached to each link using the Denavit-Hartenberg scheme \[27\], so that coordinate system \(i\) is attached to link \(i\). The advantage of the Denavit-Hartenberg scheme is that instead of six, only four parameters are needed to describe the transformation from one link to another. The four parameters are the link length, link twist, joint offset and joint angle. The spatial inertia \(I'_i\) of link \(i\) is constant in the coordinate system \(i\). The possible motion of link \(i\) through joint \(i\) is described by \(s'_i\) in the coordinate system \(i\). Both, \(I'_i\) and \(s'_i\) are expressed in \(i\)-coordinates.

\(0 X_{i-1}\) is the coordinate transformation between coordinate system \(i\) and \(i-1\). \(0 X_i\) describes the transformation between the link coordinate system \(i\) and the absolute coordinates. It is calculated by the recurrence relation

\[0 X_i = 0 X_{i-1} i_{-1} X_i.\]

With this transformation all quantities expressed in link coordinates can be transformed into absolute coordinates, as for example:

\[I_i = 0 X_i I'_i\] and \[s_i = 0 X_i s'_i,\]

where \(I_i\) and \(s_i\) is the spatial Inertia and motion of link \(i\) expressed in absolute coordinates.

To complete the system model, the system variables are defined. They describe the joint displacement \(q_i\), the joint velocity \(\dot{q}_i\), the joint acceleration \(\ddot{q}_i\) and the joint force \(\tau_i\), each for joint \(i\). If joint \(i\) is a rotational joint, \(q_i\) describes the joint angle, \(\dot{q}_i\) the angular velocity, \(\ddot{q}_i\) the angular acceleration, and \(\tau_i\) the joint torque. All system variables for each link are merged in \(q, \dot{q}, \ddot{q}, \tau \in \mathbb{R}^n\). For all applications, \(q\) and \(\dot{q}\) are considered to be known. If the joint accelerations \(\ddot{q}\) are known, then the joint forces \(\tau\) can be calculated by the inverse dynamics. In contrast to that, if the joint forces \(\tau\) are known, the joint accelerations \(\ddot{q}\) can be calculated by the forward dynamics.

### 3.2.3. The Recursive Newton-Euler Method

The Recursive Newton-Euler method calculates the inverse dynamics for a robot in three steps:

1. The velocity \(v_i\) and the acceleration \(a_i\) of each link is calculated.

2. The force \(f^*_i\), that acts on each link due to the motion and inertia of the link is calculated.

3. The joint forces \(\tau_i\) that generate the forces \(f^*_i\) in step 2 are calculated.

#### 1. Velocity and Acceleration

The absolute velocity \(v_i\) of link \(i\) is depending on the velocity of the links’ predecessor and the velocity across joint \(i\). It is calculated by the recurrence relation

\[v_i = v_{i-1} + s_i \dot{q}_i\]  

(3.2.1)
with the velocity of the base $v_0 = 0$ for a fixed base robot. Differentiating Equation 3.2.1 gives the absolute acceleration $a_i$ of link $i$:

$$a_i = a_{i-1} + v_i \times s_i \dot{q}_i + s_i \ddot{q}_i$$  \hspace{1cm} (3.2.2)

with the base acceleration for a robot on earth $a_0 = -g$, where $g$ is the gravitational vector.

2. Link Force

The force that acts on link $i$ is depending on the spatial inertia of the link:

$$f_i^* = \frac{d}{dt}(I_i v_i) = I_i a_i + v_i \times I_i v_i$$  \hspace{1cm} (3.2.3)

The link velocities, accelerations and forces are calculated according to Equations 3.2.1 - 3.2.3 for each link, starting at the base ($i = 0$).

3. Joint Force

The spatial joint force that is transmitted from link $i-1$ to link $i$ through joint $i$ is calculated by the recurrence relation

$$f_i = f_{i+1} + f_i^* - f_i^x$$  \hspace{1cm} (3.2.4)

with the link force $f_i^*$ calculated in step 2 and the external force $f_i^x$, which is acting on link $i$. This recurrence relation computes $f_i$ for each link, starting at the end-effector ($i = n$) with $f_n = f_n^*$. The components of the spatial joint force, that act in the same direction as the joint motion form the joint forces $\tau_i$ and are calculated for each joint by

$$\tau_i = s_i^T f_i$$  \hspace{1cm} (3.2.5)

This is the complete Recursive Newton-Euler Method for the inverse dynamics in absolute coordinates and has a computational complexity of $O(n)$.

3.2.4. Equations in Link Coordinates

For a more efficient algorithm, the calculations for each link $i$ should be done in the link associated coordinate system $i$, since the transformation of vectors $v, a$ and $f$ from one coordinate system to another, is faster than the transformation of $s'$ and $I'$. The resulting equations for the link velocity, acceleration and force are then

$$v_i = X_{i-1} v_{i-1} + s'_i \dot{q}_i$$  \hspace{1cm} (3.2.6)

$$a_i = X_{i-1} a_{i-1} + v_i \times s'_i \dot{q}_i + s'_i \ddot{q}_i$$  \hspace{1cm} (3.2.7)

$$f_i^* = I'_i a_i + v_i \times I'_i v_i$$  \hspace{1cm} (3.2.8)
3. Dynamic Modeling of Kinematic Trees

Again the calculation of these equations is performed for each link, starting at the base \((i = 0)\) with the starting values for the base velocity \(v_0 = 0\) for a fixed base robot and the base acceleration \(a_0 = -g\) for a robot on earth.

The joint forces \(\tau\) are then calculated by

\[
f_i = X_{i+1} f_{i+1} + f_i^* - f_i^x
\]

\[
\tau_i = s_i^T f_i
\]

beginning at the end-effector \((i = n)\) with starting value \(f_n = f_n^*\)
This chapter first explains the differences between the dynamic modeling for robots with and without kinematic loops. Then, two different approaches to the dynamic modeling of kinematic loop systems are described in detail. These approaches are based on the general dynamic modeling, which is explained in chapter 3.

**4.1. Difficulties of the Dynamic Modeling**

Closed kinematic loops occur in many robotic systems as described in chapter 2. To control and simulate those systems, the dynamic modeling of closed kinematic loops is needed. Unfortunately, the dynamic modeling of kinematic loops is much more difficult than the modeling of kinematic tree systems. Since every kinematic loop introduces constraints on the system, kinematic loops comprise more complicated dynamics than kinematic trees. For example there is no correspondence between the degrees of freedom of the system and the joint variables as it is for kinematic trees according to 9. Furthermore the degrees of motion freedom of mechanisms with kinematic loops can vary, while they stay the same for kinematic tree structures.

Figure 4.1.: Closed Loop with varying Degrees of Freedom, 8

Figure 4.1 shows a closed loop system, with varying degrees of freedom. If \( \theta \neq 0 \), then the system has only one degree of freedom, since the movement of \( A \) and \( B \) is
4. Dynamic Modeling of Kinematic Loops

depending on the movement of $\theta$. If $\theta = 0$ then $A$ and $B$ can move independently and the system has two degrees of freedom.

Other challenges of closed loops are the kinematic redundancy [28] and the actuation redundancy [29] that has to be dealt with. Furthermore, in kinematic tree structures it is possible to compute all forces whereas not all forces can be computed in closed loops according to [8].

This chapter presents now two different approaches to the computation of forward and inverse dynamics for closed-loop systems.

The first approach was introduced by Nakamura and Yamane [12] as a solution to the dynamics computation of structure-varying mechanisms without switching among algorithms. The method makes use of the generalized coordinates of a kinematic loop. This means that the minimal number of coordinates is used for the computation and results therefore in a highly efficient algorithm. The method aims at the dynamics computation of human figures, nevertheless it is a general approach for closed-loop systems and is therefore described here in detail.

The second approach is a motion simulation method for closed-loop mechanisms described by Featherstone [9]. This forward dynamics computation method is using unknown reaction forces to replace the loop-closing joints. The motion equations for the resulting open-loop system are then computed. Eventually, the acceleration constraints imposed by the closed-loops are added to calculate the dynamics of the whole closed-loop system. After the description of the first approach, this method is described in detail as well.

Similar for both approaches is that the closed loops are first virtually cut for computing the dynamics of the resulting tree structure. Then the constraints equations imposed by the closed loops are added. The differences are that the first approach makes use of generalized coordinates and the second approach is only aiming at the forward dynamics computation.

4.2. Dynamic Modeling with Generalized Coordinates

This section describes the dynamic computation of kinematic loops according to Nakamura [12].

4.2.1. Generalized Coordinates of Kinematic Loops

In a closed-loop system as shown in Figure 4.2 (a) the total number of joints is $N_J$ and the joint angles are represented by $\theta_J \in \mathbb{R}^{N_J}$. The number of the actuated joints in the closed loop is $N_A$, accordingly the actuated joint angles are $\theta_A \in \mathbb{R}^{N_A}$ and the actuator torques are $\tau_A \in \mathbb{R}^{N_A}$.

Now the closed-loop system is virtually cut at joints in such a way that there are no kinematic loops in the system anymore as shown in Figure 4.2 (b). The resulting
4. Dynamic Modeling of Kinematic Loops

(a) Kinematic Loop System

(b) Kinematic Tree System

Figure 4.2.: Kinematic Loop and Kinematic Tree System

open-loop system (open kinematic tree-structure) has the number of joints \( N_o \). The joint angles of the tree-structure are \( \theta_o \in \mathbb{R}^{N_o} \). All joints are now assumed to be actuated and therefore the joint torques are \( \tau_o \in \mathbb{R}^{N_o} \).

The kinematic tree-structure is assumed to perform the same movement as the original closed loop system, however without interaction of the cut joints. With the recursive inverse dynamics algorithm for kinematic trees as described in section 3.2.3 all joint torques \( \tau_o \) required for this motion are calculated.

The closed-loop system has \( N_f \) (see 4.2.13) degrees of freedom. The generalized coordinates \( \theta_g \in \mathbb{R}^{N_f} \) are the independent variables that represent the mobility of the kinematic loop. They describe the motion of the whole closed-loop system with \( N_f \) joints. These joints are selected from \( \theta_J \) as shown later. Accordingly, the generalized forces \( \tau_g \in \mathbb{R}^{N_f} \) are the torques and forces that act on the generalized coordinates.

The movement of the whole closed-loop system is determined by the generalized coordinates \( \theta_g \), therefore \( \theta_o \) and \( \theta_A \) are dependent on \( \theta_g \):

\[
\theta_o = \theta_o(\theta_g) \quad (4.2.1)
\]
\[
\theta_A = \theta_A(\theta_g) \quad (4.2.2)
\]

With the d’Alembert’s principle and the principle of virtual work, the following equation is derived from (4.2.1) as described in [29]:

\[
\tau_g^T \delta \theta_g = \tau_o^T \delta \theta_o = \tau_o^T W \delta \theta_g \quad (4.2.3)
\]

with

\[
W \triangleq \frac{\delta \theta_o}{\delta \theta_g} \in \mathbb{R}^{N_o \times N_f}. \quad (4.2.4)
\]
4. Dynamic Modeling of Kinematic Loops

The following equation is derived from (4.2.2) analogously:

\[ \tau_G^T \delta \theta_G = \tau_A^T \delta \theta_A = \tau_A^T S \delta \theta_G \]  \hspace{1cm} (4.2.5)

with

\[ S \triangleq \frac{\delta \theta_A}{\delta \theta_G} \in \mathbb{R}^{N_A \times N_F}. \]  \hspace{1cm} (4.2.6)

Eliminating \( \delta \theta_G \) in (4.2.3) and in (4.2.5) leads to the following equations:

\[ \tau_G = W^T \tau_O \]  \hspace{1cm} (4.2.7)

\[ \tau_G = S^T \tau_A. \]  \hspace{1cm} (4.2.8)

With the equations (4.2.7) and (4.2.8), the actuator torques \( \tau_A \) for the kinematic loop can be calculated from \( W, S \) and the actuator torques of the virtually cut kinematic loop \( \tau_O \).

4.2.2. Computation of W and S

In the closed loop in Figure 4.3, the linear and angular velocities of link \( L \) are dependent on the linear and angular velocities \( \dot{\theta}_A \) and \( \dot{\theta}_B \). \( J_A \) and \( J_B \) are the Jacobian matrices of the position and the orientation of Link \( L \) with respect to \( \theta_A \) and \( \theta_B \). The velocities of Link \( L \) are computed by multiplying \( \dot{\theta}_A \) or \( \dot{\theta}_B \) with the corresponding Jacobian matrix \( J_A \) or \( J_B \). Due to the closed loop, the resulting velocities of Link \( L \) computed from \( \dot{\theta}_A \) and \( \dot{\theta}_B \) have to be equal:

\[ (J_A - J_B) \begin{pmatrix} \dot{\theta}_A \\ \dot{\theta}_B \end{pmatrix} = 0. \]  \hspace{1cm} (4.2.9)

This is the velocity constraint enforced by the kinematic loop. In general, the velocity constraint imposed by the \( i \)th kinematic loop is

\[ J_{Li} \dot{\theta}_j = 0 \]  \hspace{1cm} (4.2.10)
4. Dynamic Modeling of Kinematic Loops

with \( J_{Li} \in \mathbb{R}^{6 \times N_J} \). \( J_{Li} \) is the velocity constraint matrix of Link \( L \). It is made up of the columns of the Jacobian matrices of link \( L \) with respect to the joint angles. The Jacobian matrices are calculated the same way as those for open-loop systems as described in \([30]\).

\( N_L \) is the number of independent kinematic loops in the mechanism and therefore \( N_L \) constraint matrices exist. \( J_C \in \mathbb{R}^{6N_L \times N_J} \) is formed of those constraint matrices as follows:

\[
J_C \overset{\Delta}{=} \begin{pmatrix}
J_{L1} \\
J_{L2} \\
\vdots \\
J_{LN_L}
\end{pmatrix},
\]

so all kinematic constraints are comprised in \( J_C \). If not all rows of \( J_C \) are independent, then \( J_C \) is not full rank and the constraints are dependent. All linearly independent rows of \( J_C \) are then selected to form \( J_{Cm} \in \mathbb{R}^{m \times N_J} \) with \( m = \text{rank}(J_C) \). The \( m \) independent constraints introduced by the \( N_L \) kinematic loops are represented by

\[
J_{Cm} \dot{\theta}_J = 0 \tag{4.2.12}
\]

which is derived from (4.2.10). From the \( N_J \) joints and \( m \) independent constraints, the degrees of freedom of the closed kinematic loop structure \( N_F \) are calculated:

\[
N_F = N_J - m. \tag{4.2.13}
\]

\( J_{Cm} \) is now divided in \( J_S \) and \( J_G \) in such a way, that \( J_S \in \mathbb{R}^{m \times m} \) contains \( m \) linearly independent columns from \( J_{Cm} \), while \( J_G \in \mathbb{R}^{m \times N_F} \) is built from the remaining ones. Accordingly, \( \theta_J \in \mathbb{R}^{N_J} \) is divided in \( \theta_S \in \mathbb{R}^m \) and the generalized coordinates \( \theta_G \in \mathbb{R}^{N_F} \). For example if the first column of \( J_{Cm} \) is independent, it is inserted in \( J_S \). Since the first column of \( J_{Cm} \) corresponds to the first line of \( \theta_J \), the first line of \( \theta_J \) is inserted in \( \theta_S \). The division of \( J_{Cm} \) and \( \theta_J \) results in:

\[
J_{Cm} \dot{\theta}_J = \begin{pmatrix} J_S & J_G \end{pmatrix} \begin{pmatrix} \dot{\theta}_S \\ \dot{\theta}_G \end{pmatrix} = 0. \tag{4.2.14}
\]

Rearranging the terms leads to

\[
J_S \dot{\theta}_S = -J_G \dot{\theta}_G. \tag{4.2.15}
\]

\( J_S \) is always invertible, therefore \( \dot{\theta}_S \) can be determined by

\[
\dot{\theta}_S = -J_S^{-1}J_G \dot{\theta}_G \tag{4.2.16}
\]

with

\[
H \overset{\Delta}{=} \frac{\delta \theta_S}{\delta \theta_G} = -J_S^{-1}J_G. \tag{4.2.17}
\]
4. Dynamic Modeling of Kinematic Loops

From \( H \) the Jacobian matrices \( W \) and \( S \) are built according to the following description:

\( W \): If the \( i \)th joint of \( \theta_G \) is not in \( \theta_G \) but corresponds with the \( j \)th joint of \( \theta_S \), then the \( j \)th row of \( H \) is included as the \( i \)th row of \( W \). If the \( i \)th joint of \( \theta_G \) is in \( \theta_G \) and corresponds with the \( j \)th joint of \( \theta_G \), then a vector with the \( j \)th element = 1 and all other elements = 0 is included as \( i \)th row of \( W \).

\( S \) is formed similarly to \( W \): If the \( i \)th joint of \( \theta_A \) is not in \( \theta_G \) but corresponds with the \( j \)th joint of \( \theta_S \), then the \( j \)th row of \( H \) is included as the \( i \)th row of \( S \). If the \( i \)th joint of \( \theta_A \) is in \( \theta_G \) and corresponds with the \( j \)th joint of \( \theta_G \), then a vector with the \( j \)th element = 1 and all other elements = 0 is included as \( i \)th row of \( S \).

The forming of \( W \) and \( S \) is visualized in Figure 4.4.

![Figure 4.4: Forming of W and S](image)

(a) Forming \( W \)  
(b) Forming \( S \)

4.2.3. Relationship of Accelerations

The acceleration of the dependent joints \( \ddot{\theta}_S \) is needed for the forward dynamics computation. It can be calculated from the generalized acceleration \( \ddot{\theta}_G \) by differentiating \( \dot{\theta}_S = H\dot{\theta}_G \) (4.2.16):

\[
\ddot{\theta}_S = H\ddot{\theta}_G + \dot{H}\dot{\theta}_G
\]  

(4.2.18)

By differentiating \( H = -J_S^{-1}J_G \) (4.2.17), the second part of this equation is calculated:

\[
H\ddot{\theta}_G = -\left( \frac{d}{dt}(J_S^{-1})J_G + J_S^{-1}\dot{J}_G \right)\dot{\theta}_G.
\]  

(4.2.19)

Since \( J_S^{-1}J_S = I \),

\[
\frac{d}{dt}(J_S^{-1})J_S + J_S^{-1}\dot{J}_S = 0.
\]  

(4.2.20)
Expanding (4.2.19) leads to:
\[
\dot{H}_G = -\frac{d}{dt}(J_S^{-1})J_G\dot{\theta}_G - J_S^{-1}\dot{J}_G\dot{\theta}_G.
\]
Inserting (4.2.15) results in:
\[
\dot{H}_G = \frac{d}{dt}(J_S^{-1})J_S\dot{\theta}_S - J_S^{-1}\dot{J}_S\dot{\theta}_G.
\]
Rearranging (4.2.20) to
\[
\frac{d}{dt}(J_S^{-1})J_S = -\frac{d}{dt}(J_S^{-1})J_G\dot{\theta}_G.
\]
and inserting leaves:
\[
\dot{H}_G = -J_S^{-1}J_S\dot{\theta}_S - J_S^{-1}\dot{J}_G\dot{\theta}_G.
\]
With use of (4.2.14), equation (4.2.19) becomes:
\[
\dot{H}_G = -J_S^{-1}(J_S\dot{\theta}_S + J_G\dot{\theta}_G) = -J_S^{-1}J_{Cm}\dot{\theta}_J.
\] (4.2.21)

\(\dot{J}_C\dot{\theta}_J\) can be calculated by the same algorithm as for open-loop systems \[31\]. By extracting linearly independent rows from \(J_C\dot{\theta}_J\) similar to the extraction of \(J_{Cm}\) as described before, \(J_{Cm}\dot{\theta}_J\) is built.

### 4.2.4. Inverse Dynamics

The computation of the required actuator torques \(\tau_A\) for a given trajectory of the mechanism \((\theta_J, \dot{\theta}_J, \ddot{\theta}_J)\) is called inverse dynamics. With the equations described in the preceding sections, the inverse dynamics of closed-loop systems is calculated as follows:

1. The Jacobian matrices \(W\) and \(S\) are computed according to (4.2.2).
2. The closed loops are virtually cut and \(\tau_O\) for kinematic tree structures is calculated with the recursive Newton-Euler inverse dynamics algorithm as described in section 3.2.3.
3. The generalized force \(\tau_G\) is computed with \(W\) and \(\tau_O\) according to (4.2.7).
4. The actuator torques \(\tau_A\) are computed with \(\tau_G\) by solving (4.2.8):
\[
\tau_A = S^{-T}\tau_G
\] (4.2.22)

If the degrees of freedom \(N_F\) equals the number of actuators \(N_A\) then the system has no actuation redundancy and \(S \in \mathbb{R}^{N_A \times N_F}\) becomes a square matrix. \(\tau_A\) is then calculated with the inverse of \(S\).
If \(N_F \neq N_A\), \(S\) is not a square matrix and therefore \(\tau_A\) can not be calculated without optimization methods. An optimization method that can be applied to solve \(\tau_A\) is described by \[32\].
4. Dynamic Modeling of Kinematic Loops

4.2.5. Forward Dynamics

Contrary to inverse dynamics, forward dynamics is the computation of the joint accelerations \( \ddot{\theta}_J \) for given actuator torques \( \tau_A \). The unit vector approach described by [33] is a forward dynamics algorithm intended for kinematic tree structures. This algorithm is extended by the inverse dynamics algorithm described in 4.2.1 for the application to closed-loops systems.

\[
\tau_G = A(\theta_G)\dot{\theta}_G + b(\theta_G, \dot{\theta}_G) \tag{4.2.23}
\]

is the equation of motion of closed-loop systems. \( A \in \mathbb{R}^{N_F \times N_F} \) is the symmetric inertia matrix. The gravitational, centrifugal and Coriolis effects are specified by \( b \in \mathbb{R}^{N_F} \). The computation of the forward dynamics for kinematic loops including the computation of \( A \) and \( b \) is done as follows:

1. The given actuator torques \( \tau_A \) are transformed to the generalized forces \( \tau_G \) according to (4.2.8).

2. The inverse dynamics for the generalized acceleration \( \ddot{\theta}_G = 0 \) is computed and the resulting generalized force is set to \( b \). Now the dependent joint accelerations \( \ddot{\theta}_S \) are calculated from (4.2.18) with \( \dot{\theta}_G \) being zero: \( \ddot{\theta}_S = \dot{H} \theta_G \)

3. For each degree of freedom \((i = 1, 2, \ldots, N_F)\):
   a) The inverse dynamics for \( \ddot{\theta}_G = e_i \) with \( e_i \in \mathbb{R}^{N_F} \) is computed. \( e_i \) is a unit vector with the \( i \)-th element being 1 and all other elements being 0. The dependent joint accelerations \( \ddot{\theta}_S \) are computed according to (4.2.18). Set the resulting generalized force to \( f_i \).
   b) \( a_i = f_i - b \) is calculated.
   c) \( a_i \) is set as the \( i \)-th column of \( A \).

4. With \( \tau_G \), \( A \) and \( b \), the generalized acceleration \( \ddot{\theta}_G \) is computed by rearranging (4.2.23) to:

\[
\ddot{\theta}_G = A^{-1}(\tau_G - b) \tag{4.2.24}
\]

5. The dependent joint accelerations \( \ddot{\theta}_S \) are computed from (4.2.18) with \( \dot{H} \theta_G \) already computed in step 2.

With the dependent joint accelerations \( \ddot{\theta}_S \) and the generalized joint accelerations \( \ddot{\theta}_G \) the accelerations of all joints is known by \( \ddot{\theta}_J = (\ddot{\theta}_S, \ddot{\theta}_G) \).

4.3. Dynamic Modeling with Loop Joints

The dynamic modeling of kinematic loops with loop joints according to Featherstone [9] is described in this section.
4. Dynamic Modeling of Kinematic Loops

4.3.1. The Effect of Kinematic Loops on Mechanism Dynamics

According to Featherstone the dynamic model of a closed-loop system is made up of a tree structure system and additional joints, that close the loops. So in a system with $N$ links and $L$ independent kinematic loops there exist $N + L$ joints. The $N$ joints are called tree joints whereas the $L$ joints are called loop joints. As example, in the closed loop in Figure [4.2] there are $N = 6$ tree joints (the one joint connecting 3 links has actually to be modelled as 2 joints) and $L = 2$ loop joints, which are marked red. The system has $n$ degrees of freedom.

A loop joint affects the dynamics of a tree structure system by imposing a constraint on the generalized velocity of the tree structure. For example loop joint $k$ closes the kinematic loop $k$ of an open-loop system by connecting its predecessor link $p_k$ and its successor link $s_k$. The velocity of $s_k$ is then

$$v_{s_k} = v_{p_k} + S_k \dot{q}_k$$

with the motion space $S_k$ of joint $k$ and its velocity $\dot{q}_k$. Another expression of the velocity constraint is

$$R_k^T (v_{s_k} - v_{p_k}) = 0$$

with the reaction force space $R_k = S_k^\perp$ of joint $k$. The dimension of $R_k$ equals the number of velocity constraints imposed by the closed loop.

The force that joint $k$ applies on link $s_k$ is

$$\hat{f}_k = f_k^a + R_k f_k$$

with the actuation force $f_k^a$ and the unknown reaction force coefficients $f_k$. Similarly the force that joint $k$ applies on link $p_k$ is $-\hat{f}_k$. Each loop joint introduces $r$ motion constraints and equals the number of unknown reaction forces. The system has $n - r$ degrees of motion freedom if the constraints are independent.

4.3.2. The Equations of Motion for Robots with Kinematic Loops

$$H \ddot{q} = Q - C$$

is the equation of motion of a general tree structure mechanism with the joint-space inertia matrix $H$, the vector of generalized forces for velocity-product effects like gravity $C$, the vector of generalized forces $Q$ and the accelerations $\ddot{q}$. The motion equations for a system with kinematic loops result from the following method:

1. An acceleration constraint of the kinematic tree is obtained for each loop and expressed in terms of the generalized acceleration.

2. The loop-closure forces are expressed in terms of the generalized forces.
4. Dynamic Modeling of Kinematic Loops

3. The loop-closure forces are added to the motion equation of the kinematic tree structure. Combining them with the acceleration constraints leads to the motion of the closed loop system.

1. Acceleration constraints

By differentiating the velocity constraint equation (4.3.2), the constraint on the acceleration results:

\[ R_k^T (a_s - a_p) + \dot{R}_k^T (v_s - v_p) = 0 \]  

(4.3.5)

where \( a_s \) and \( a_p \) are the accelerations of link \( p_k \) and \( s_k \). Now the acceleration constraint has to be expressed in terms of the generalized acceleration of the kinematic tree \( \ddot{q} \).

The velocity of link \( i \) in a kinematic tree in terms of the kinematic tree structure velocities is:

\[ v_i = \sum_{j=1}^{N} e_{ij} S_j \dot{q}_j \]  

(4.3.6)

with \( S_j \) being the motion space of joint \( j \) and its velocity \( \dot{q}_j \). The scalar \( e_{ij} = 1 \) if joint \( j \) is between link \( i \) and the base, otherwise \( e_{ij} = 0 \). A matrix expression of the velocity of link \( i \) is:

\[ v_i = J_i \dot{q} \]  

(4.3.7)

with

\[ J_i = [e_{i1} S_1 \ldots \ e_{iN} S_N] \]  

(4.3.8)

and the generalized velocity of the kinematic tree \( \dot{q} \). The expression for the acceleration is given by differentiating (4.3.7):

\[ a_i = J_i \ddot{q} + \dot{J}_i \dot{q} \]  

(4.3.9)

The acceleration of link \( i \) due to velocity-product effects is \( \dot{J}_i \dot{q} \). This is already calculated for each link to compute \( C \) [9, chapter 5] and therefore a known quantity. Now the velocity-product acceleration of link \( i \) is called \( a_{i\text{vp}} = J_i \dot{q} \) and therefore the acceleration of link \( i \) is:

\[ a_i = J_i \ddot{q} + a_{i\text{vp}} \]  

(4.3.10)

Inserting (4.3.10) in (4.3.5) leads to

\[ R_k^T (J_i - J_p) \ddot{q} = -R_k^T (a_{s\text{vp}} - a_{p\text{vp}}) - \dot{R}_k^T (v_s - v_p) \]  

(4.3.11)

By defining the velocity across joint \( k \): \( v_k = v_s - v_p \), the velocity-product acceleration across joint \( k \): \( a_{s\text{vp}} = a_s^{\text{vp}} - a_p^{\text{vp}} \) and the Jacobian of loop \( k \): \( J_k = J_s - J_p \) results

\[ R_k^T J_k \ddot{q} = -R_k^T a_{s\text{vp}} - \dot{R}_k^T v_k \]  

(4.3.12)

This equation expresses the acceleration constraints of the kinematic tree structure introduced by loop \( k \). The dimension of \( R_k \) is the same as the number of constraints imposed by the closed loops.
4. Dynamic Modeling of Kinematic Loops

2. Loop-closure forces

To express the loop-closure forces in terms of the equivalent generalized forces it is supposed that a generalized force \( Q \) is applied to the tree. \( Q \) is applying a force \( f \) to link \( i \). For balancing, the force \(-f\) must be applied to link \( i \) from its predecessors. Therefore each joint between the base and link \( i \) applies the force \(-f\) on the successor. The generalized force for joint \( j \) is then

\[
Q_j = -S_j^T f.
\]

is the generalized force of the whole kinematic tree structure. Using the Jacobian definition (4.3.8) results in

\[
Q = -J^T f. \tag{4.3.14}
\]

If the loop closure force is \( f \) on link \( s_k \) and \(-f\) on link \( p_k \) the loop closure forces expressed in equivalent generalized forces is:

\[
Q_k^L = J_{s_k}^T f_k - J_{p_k}^T f_k = J_k^T f_k \tag{4.3.15}
\]

The dynamics of the system with closed loop \( k \) is modeled by adding \( Q_k^L \) to the generalized force. Closing all loops implies the addition of \( \sum_{k=1}^{L} Q_k^L \) to the generalized force.

3. Equations of motion

The overall equations of motion of the whole system containing kinematic loops is:

\[
H \ddot{q} = Q - C + \sum_{k=1}^{L} Q_k^L \tag{4.3.16}
\]

Adding the loop acceleration constraints (4.3.5) and using (4.3.15) and (4.3.3) leads to

\[
\begin{bmatrix}
H & J_1 R_1 & \cdots & J_L R_L \\
R_1^T J_1 & 0 & \cdots & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
R_L^T J_L & \vdots & \ddots & 0
\end{bmatrix}
\begin{bmatrix}
\ddot{q} \\
-f_1 \\
\vdots \\
-f_L
\end{bmatrix}
= \begin{bmatrix}
Q - C + \sum_{k=1}^{L} Q_k^L \\
-R_1^T a_{1e} - R_1^T v_1 \\
\vdots \\
-R_L^T a_{Le} - R_L^T v_L
\end{bmatrix} \tag{4.3.17}
\]

where the coefficient matrix (on the left) \( \in \mathbb{R}^{n+r \times n+r} \) with \( n \) is the degrees of freedom of the kinematic tree and \( r \) the number of constraints imposed by the closed loops and the submatrix \( H \in \mathbb{R}^{n \times n} \). Solving the equation for \( \ddot{q} \) and the unknown loop closure forces \(-f_1 \ldots -f_L\) gives the forward dynamics. This is only possible if the coefficient matrix is not singular, otherwise one or more loop closure forces can not be determined. By removing the indeterminate forces, the remaining forces can be calculated. For more detailed explanation see the following example:
The closed loop system in Figure 4.5 has one degree of freedom. If the loop is cut at the red marked joint, the resulting kinematic tree system has three degrees of freedom (one prismatic and two rotational). The red marked loop joint however adds five constraints to the closed loop system. One constraint for each translational movement of the joint in \(x, y\) and \(z\) direction and one for the rotation around its \(x\) and \(y\) axis. Only the rotation around its \(z\) axis, which points into the picture plane, is possible. Therefore the red marked joint has one degree of freedom and imposes five constraints on the system.

Three of the five constraints are linearly dependent on constraints imposed by the other joints of the system, which allow only planar motion for the system. These three linearly dependent constraints have therefore no effect on the system, since it is already constrained to have planar motion. The two other constraints are independent and therefore affecting the system by closing the loop. Subtracting the linearly independent constraints \(r = 2\) from the degrees of freedom of the kinematic tree system \(n = 3\) leads to the number of degrees of freedom of the closed loop system, which is 1.

The forces of the three linearly independent constraints can not be calculated and lead the coefficient matrix in 4.3.17 to be singular, therefore they should be removed from the system. To do this, a four degree of freedom joint is used instead of the red marked one degree of freedom joint. This could be modeled by a sphere-in-cylinder joint, that has four degrees of freedom and adds only the two linearly independent constraints. Replacing the rotational joint by the 4 degree of freedom joint, wouldn’t change the behaviour of the system and would lead to a nonsingular coefficient matrix, which enables the forward dynamics computation.
5. SpaceDynamics Library

The algorithms described in the preceding chapter have to be implemented in the environment of the SpaceDynamics Library in order to enable the dynamic modeling of robots with closed loops. Since the algorithms make use of the existing functions, this chapter describes now the structure and the different functions of the SpaceDynamics Library.

Figure 5.1.: Class Diagram of the SpaceDynamics Library – 1
5. SpaceDynamics Library

<table>
<thead>
<tr>
<th>rot</th>
<th>spn</th>
</tr>
</thead>
<tbody>
<tr>
<td>* aw(double*, double*) : void</td>
<td>* cross(double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* cx(double*, double*) : void</td>
<td>* crossM(double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* cy(double*, double*) : void</td>
<td>* spl(double*, double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* cz(double*, double*) : void</td>
<td>* xrotB(double*, double*) : void</td>
</tr>
<tr>
<td>* rpy2dz(double*, double*) : void</td>
<td>* xrotA(double*, double*, double*) : void</td>
</tr>
<tr>
<td>* dc2py(double*, double*) : void</td>
<td>* Xtrans(double*, double*) : void</td>
</tr>
<tr>
<td>* dcz(qn(double*, double*) : void</td>
<td>* Xrots(double*, double*) : void</td>
</tr>
<tr>
<td>* qnt2dz(double*, double*) : void</td>
<td>* Xrots(double*, double*) : void</td>
</tr>
<tr>
<td>* w2dqh(double*, double*, double*) : void</td>
<td>* Xrots(double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* r2py(double*, double*) : void</td>
<td>* Xrotsz(double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* R2qnt(double*, double*) : void</td>
<td>* i_tilde(double*, double*, double*) : void</td>
</tr>
<tr>
<td>* r2qnt(double*, double*) : void</td>
<td></td>
</tr>
<tr>
<td>* rpy2qz(double*, double*) : void</td>
<td></td>
</tr>
<tr>
<td>* tilde(int, double*, double*) : void</td>
<td></td>
</tr>
<tr>
<td>* pi_conv(char) : double</td>
<td></td>
</tr>
<tr>
<td>* pi2deg(double) : double</td>
<td></td>
</tr>
<tr>
<td>* deg2pi(double) : double</td>
<td></td>
</tr>
<tr>
<td>* QMmultiply(double*, double*, double*) : void</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>spd</th>
</tr>
</thead>
<tbody>
<tr>
<td>* model(string, MODEL&amp; int, int) : void</td>
</tr>
<tr>
<td>* model_test(string) : void</td>
</tr>
<tr>
<td>* model_CL(string, MODEL&amp; int) : void</td>
</tr>
<tr>
<td>* model_new(string, MODEL&amp; ) : void</td>
</tr>
<tr>
<td>* model_init(MODEL, int) : void</td>
</tr>
<tr>
<td>* model_converter(string, string) : void</td>
</tr>
<tr>
<td>* calc_SPN(MODEL&amp; ) : void</td>
</tr>
<tr>
<td>* ch_conf(MODEL&amp; int, MODEL&amp;) : void</td>
</tr>
<tr>
<td>* dat_conn_fromy(MODEL, int, MODEL&amp; int) : void</td>
</tr>
<tr>
<td>* calc(SP(MODEL&amp; ) : void</td>
</tr>
<tr>
<td>* calc_SP_dot(MODEL&amp; ) : void</td>
</tr>
<tr>
<td>* calc_hh(MODEL, double*) : void</td>
</tr>
<tr>
<td>* calc_Rp(MODEL, double*) : void</td>
</tr>
<tr>
<td>* calc_1d(MODEL, double**) : void</td>
</tr>
<tr>
<td>* calc_1e(MODEL, int, double*) : void</td>
</tr>
<tr>
<td>* calc_1e_dot(MODEL, int, double*) : void</td>
</tr>
<tr>
<td>* calc_Gjb(MODEL, double*) : void</td>
</tr>
<tr>
<td>* calc_Gje(MODEL, int, double*) : void</td>
</tr>
<tr>
<td>* calc_Gje_r(MODEL, int, double*) : void</td>
</tr>
<tr>
<td>* calc_Gb(MODEL, int, double*) : void</td>
</tr>
<tr>
<td>* calc_1g(MODEL, double*) : void</td>
</tr>
<tr>
<td>* calc_fe_CL_fx(MODEL, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* f_kin_e(MODEL, int) : void</td>
</tr>
<tr>
<td>* f_kin_e(MODEL&amp;) : void</td>
</tr>
<tr>
<td>* f_dyn(MODEL, double*, double*) : void</td>
</tr>
<tr>
<td>* f_dyn_fkx(MODEL, double*, double*) : void</td>
</tr>
<tr>
<td>* f_dyn_fy(MODEL, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* f_dyn_fx(MODEL, double*, double*) : void</td>
</tr>
<tr>
<td>* f_dyn_fy(MODEL, double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* f_dyn_fz(MODEL, double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* f_dyn_fz(MODEL, double*, double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* f_dyn_rk(MODEL, double*, double*) : void</td>
</tr>
<tr>
<td>* calc_Xup_t(MODEL&amp; ) : void</td>
</tr>
<tr>
<td>* calc_Xup_t(MODEL) : void</td>
</tr>
<tr>
<td>* calc_Xup_1(MODEL, double*) : void</td>
</tr>
<tr>
<td>* calc_SS(MODEL, double*) : void</td>
</tr>
<tr>
<td>* i_num(MODEL, int, int, int) : void</td>
</tr>
<tr>
<td>* aw(double*, double, double*) : void</td>
</tr>
<tr>
<td>* m0(int, int, double*, double*, double) : void</td>
</tr>
<tr>
<td>* aw(double*, double, double*) : void</td>
</tr>
<tr>
<td>* calc_CLC2(MODEL&amp; ) : void</td>
</tr>
<tr>
<td>* i_dyn_CL(MODEL, double*, double*) : void</td>
</tr>
<tr>
<td>* i_dyn_CL_f(MODEL, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* f_dyn_CL_f(MODEL, double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* f_dyn_CL_f(MODEL, double*, double*, double*, double*, double*) : void</td>
</tr>
<tr>
<td>* H_CL_f(MODEL, int, int, unsigned int, unsigned int*, unsigned int**) : double *</td>
</tr>
<tr>
<td>* dh_CL_fi(MODEL, int, int, double) : double *</td>
</tr>
<tr>
<td>* w2dQn(double*, double*, double*) : void</td>
</tr>
</tbody>
</table>

Figure 5.2.: Class Diagram of the SpaceDynamics Library – 2
5. SpaceDynamics Library

5.1. Class Diagram

The SpaceDynamics Library is developed at the DLR since 2007 and mainly used for space robot applications. It offers functions that allow the simulation and control of a robot. For example there are functions to calculate the kinematics and dynamics of a robotic system. All functions of the SpaceDynamics Library are grouped in the six different classes Model, spd, matrix, vector, rot and spn. Figures 5.1 and 5.2 show the class diagram of the library with the associated functions. The library is still under development and some functions are not yet working, or not working for all possible robot configurations. The different classes and some important functions are now explained in detail.

Model

The class Model contains all variables to describe a robotic system. In order to use the SpaceDynamics Library functions, the robot has to be represented as an object of this class first. The number of links for example is stored in Linknum. The joint configuration of the robot is represented by the joint angles $q (= \mathbf{q})$, the joint velocity $qd (= \dot{\mathbf{q}})$ and the joint acceleration $qdd (= \ddot{\mathbf{q}})$. In general, $d$ in a variable name means derivative. As for example the linear velocity of the robots base is represented by $v0$, whereas its derivative, the linear base acceleration, is stored in $vd0$. A robot can be modeled easily by loading all parameters from a def-file into the SpaceDynamics Library. Table 5.1 gives an overview of the most important variables of the class Model.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linknum</td>
<td>number of links in the system</td>
</tr>
<tr>
<td>E_Num</td>
<td>number of end effectors</td>
</tr>
<tr>
<td>J_type</td>
<td>joint type</td>
</tr>
<tr>
<td>JtoC</td>
<td>vector from joint to center of link</td>
</tr>
<tr>
<td>CtoJ</td>
<td>vector from center of link to joint</td>
</tr>
<tr>
<td>CtoE</td>
<td>vector from center of link to end effector</td>
</tr>
<tr>
<td>v0</td>
<td>linear velocity of base $v_0$</td>
</tr>
<tr>
<td>vd0</td>
<td>linear acceleration of base $\dot{v}_0$</td>
</tr>
<tr>
<td>w0</td>
<td>angular velocity of base $\omega_0$</td>
</tr>
<tr>
<td>wd0</td>
<td>angular acceleration of base $\dot{\omega}_0$</td>
</tr>
<tr>
<td>q</td>
<td>joint position $\mathbf{q}$</td>
</tr>
<tr>
<td>qd</td>
<td>joint velocity $\dot{\mathbf{q}}$</td>
</tr>
<tr>
<td>qdd</td>
<td>joint acceleration $\ddot{\mathbf{q}}$</td>
</tr>
<tr>
<td>tau</td>
<td>joint force (torque) $\mathbf{\tau}$</td>
</tr>
</tbody>
</table>
5. *SpaceDynamics Library*

**spd**

The class *spd* offers the main functions of the SpaceDynamics Library, for example *i_dyn_CL_fix()* calculates the inverse dynamics for a fixed base robot model. In general, a *CL* in the function name means, that the function is used for a closed loop robot. In contrast to that, a function without a *CL* in the function name is intended for a kinematic tree system. A *fix* in the function name means, that the function is calculating the solution for a fixed base robot. Functions without *fix* in the function name are intended for free floating base robots.

The differences between functions for fixed base and functions for free floating base are the algorithm and the dimension of the result. For fixed base functions, the robots base has no velocity or acceleration and the movement of the robots links does not affect the pose of the robots base. Therefore the dimension of the output for inverse or forward dynamics is the number of degrees of freedom \( N_F \). For free floating base functions, the movement of the robots links affects the pose of the robots base due to conservation of momentum. Therefore the dimension of

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>loads parameters for a robot from a def-file into the model</td>
</tr>
<tr>
<td>calc_SP</td>
<td>calculates the spatial notation of the robot</td>
</tr>
<tr>
<td>calc_hh</td>
<td>calculates the inertia matrix ( H )</td>
</tr>
<tr>
<td>calc_Je</td>
<td>calculates the end effector Jacobian</td>
</tr>
<tr>
<td>f_kin_e</td>
<td>calculates the pose of the end effector</td>
</tr>
<tr>
<td>f_kin_j</td>
<td>calculates the pose of all joints</td>
</tr>
<tr>
<td>i_dyn</td>
<td>calculates the inverse dynamics ( (\tau) ) for an open loop robot with free floating base</td>
</tr>
<tr>
<td>i_dyn_fix</td>
<td>calculates the inverse dynamics ( (\tau) ) for an open loop robot with fixed base</td>
</tr>
<tr>
<td>f_dyn</td>
<td>calculates the forward dynamics ( (\dot{q}) ) for an open loop robot with free floating base</td>
</tr>
<tr>
<td>f_dyn_fix</td>
<td>calculates the forward dynamics ( (\dot{q}) ) for an open loop robot with fixed base</td>
</tr>
<tr>
<td>i_dyn_CL</td>
<td>calculates the inverse dynamics ( (\tau) ) for a closed loop robot with free floating base</td>
</tr>
<tr>
<td>i_dyn_CL_fix</td>
<td>calculates the inverse dynamics ( (\tau) ) for a closed loop robot with fixed base</td>
</tr>
<tr>
<td>f_dyn_CL_fix</td>
<td>calculates the forward dynamics ( (\dot{q}) ) for a closed loop robot with fixed base</td>
</tr>
<tr>
<td>H_CL_fix</td>
<td>calculates the inertia matrix ( H ) for a closed loop robot with fixed base</td>
</tr>
<tr>
<td>dH_CL_fix</td>
<td>calculates matrix ( H ) for a closed loop robot with fixed base</td>
</tr>
</tbody>
</table>
5. SpaceDynamics Library

the output sums up to $N_F + 6$, because the robots base has six degrees of freedom –
three translational degrees of freedom in x, y and z direction and three rotational
degrees of freedom around those axes.

The functions require the model as input, on which the calculations are performed.
The solution of the function is stored in a variable, which is usually received by the
function as well. An overview of the most important functions of the class spd is
given in Table 5.2 These functions are described in detail in section 5.3.

matrix

Functions for basic matrix operations like multiplication of matrices are implemented
in the class matrix. The functions of this class use functions of the GNU Scientific
Library (GSL). These matrix functions are the basis of all implemented algorithms
in the SpaceDynamics Library. An overview of the most important functions that
are offered by this class is given in Table 5.3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix_get</td>
<td>get a new matrix (memory allocation)</td>
</tr>
<tr>
<td>matrix_cpy</td>
<td>copy a matrix</td>
</tr>
<tr>
<td>matrix_I</td>
<td>set an identity matrix</td>
</tr>
<tr>
<td>matrix_Z</td>
<td>set a zero matrix</td>
</tr>
<tr>
<td>matrix_print</td>
<td>print the matrix in matrix form</td>
</tr>
<tr>
<td>matrix_mult</td>
<td>multiply two matrices</td>
</tr>
<tr>
<td>matrix_add</td>
<td>add two matrices</td>
</tr>
<tr>
<td>matrix_sub</td>
<td>subtract a matrix from another</td>
</tr>
<tr>
<td>matrix_trans</td>
<td>transpose matrix</td>
</tr>
<tr>
<td>matrix_scale</td>
<td>multiply matrix with scalar</td>
</tr>
<tr>
<td>matrix_inv</td>
<td>calculate inverse of matrix</td>
</tr>
<tr>
<td>matrix_pinv</td>
<td>calculate pseudoinverse of matrix</td>
</tr>
<tr>
<td>matrix_det</td>
<td>return determinant of matrix</td>
</tr>
<tr>
<td>matrix_svd</td>
<td>perform a singular value decomposition</td>
</tr>
<tr>
<td>matrix_ext_row</td>
<td>extract a row vector from a matrix</td>
</tr>
<tr>
<td>matrix_ext_col</td>
<td>extract a column vector from a matrix</td>
</tr>
<tr>
<td>matrix_cpy_row</td>
<td>copy a row vector into a matrix</td>
</tr>
<tr>
<td>matrix_cpy_col</td>
<td>copy a column vector into a matrix</td>
</tr>
<tr>
<td>matrix_ext_sub</td>
<td>extract a submatrix from a matrix</td>
</tr>
<tr>
<td>matrix_cpy_sub</td>
<td>copy a submatrix into a matrix</td>
</tr>
</tbody>
</table>

Table 5.3.: Important Functions of the Class matrix
vector

The `vector` class offers similar functions as the `matrix` class. Vectors could also be modeled as one dimensional matrices, however the functions offered by the `vector` class are faster than the `matrix` class functions. Therefore `vector` objects are usually used where it is possible. With the explanation of the functions of the `matrix` class given above, it is easy to understand what a function of the `vector` class is doing. For example function `vector_get` returns a new vector and `vector_cross3` calculates the cross product of two $3 \times 1$ matrices. All functions of this class can be found in Figure 5.1.

rot

There are different possibilities to illustrate rotations, for example they can be represented by roll-pitch-yaw angles, direction cosines, rotation matrices and quaternions. The class `rot` offers functions to convert between these different rotation representations. An overview of those functions is given in Table 5.4.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rpy2dc</td>
<td>convert roll-pitch-yaw angle to direction cosine</td>
</tr>
<tr>
<td>dc2rpy</td>
<td>convert direction cosine to roll-pitch-yaw angle</td>
</tr>
<tr>
<td>dc2qtn</td>
<td>convert direction cosine to quaternion</td>
</tr>
<tr>
<td>qtn2dc</td>
<td>convert quaternion to direction cosine</td>
</tr>
<tr>
<td>rpy2R</td>
<td>convert roll-pitch-yaw angle to rotation matrix</td>
</tr>
<tr>
<td>R2rpy</td>
<td>convert rotation matrix to roll-pitch-yaw angle</td>
</tr>
<tr>
<td>deg2pi</td>
<td>convert degrees to radians</td>
</tr>
<tr>
<td>pi2deg</td>
<td>convert radians to degrees</td>
</tr>
</tbody>
</table>

spn

The class `spn` offers functions for the spatial notation of the robot model. A spatial vector in general is a 6-dimensional vector, that represents the translational and rotational components of a robotic system. With this representation, less equations are needed and it is easier to understand them. The functions calculate for example the coordinate transformation matrix for a rotation around the $x,y$ or $z$ axis with the functions $Xrotx()$, $Xroty()$ and $Xrotz()$. These functions are however not used directly by the new algorithms, therefore they are not explained in detail.
5.2. Model Representation

As described before, a model is represented by the variables offered by the `Model` class. A model is defined in an external def-file, which is then loaded into the SpaceDynamics Library. Listing 5.1 shows a minimal model that illustrates the structure of a def-file. The order of the configuration options is fixed and cannot be changed. First, parameters of the robot structure are described, then the kinematic parameters are set and finally the dynamics of the system is fixed. After the example def-file, the keywords and parameters are explained.

Listing 5.1: spacedyn/tests/minimal_model_CL.def

```plaintext
# # #Parameters
for MODEL (DO NOT CHANGE THIS ORDER!!)

# # # LINK NUMBER
3

# # # LINK CONNECTIVITY
BB [ 0 1 ]

# # # Joint Type
[ 0 = rotational, 1 = prismatic ]

# # # EE
EE [ 0 1 ]

# # # Link flag
CL [ -1 0 ]

# # # Relative Coordinate For Link: Roll, Pitch, Yaw Angle of Each Bodies [ x, y, z ] [ deg ]

rpy1 [ 0 0 0 ]
rpy2 [ 0 0 0 ]

# # # Vector Of Link Length, JtoC_0_0_is_always [ 0 0 0 ]

JtoC_0_1 [ 10 0 0 ]
CtoJ_1_2 [ 2 0 0 ]

# # # Vector To End-Effector
CtoE [ 5 0 0 ]

# # # Relative Coordinate For End-Effector
rpyE_1 [ 0 0 0 ]

# # # Mass Parameter
mass [ 1 2 3 ]

# # # Inertia Matrix
[
[ 0.5, 0.2, 0.3 ],
[ 0.2, 1.0, 0.4 ],
[ 0.3, 0.4, 2.0 ]
]

# # # Link0
I11= 1
I22= 1
I33= 1

# # # Link1
I11= 1
I22= 1
I33= 1
```

33
5. SpaceDynamics Library

<table>
<thead>
<tr>
<th>55</th>
<th>I12= 0</th>
<th>I13= 0</th>
<th>I23= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>###Link2</td>
<td>I11= 1</td>
<td>I22= 1</td>
</tr>
<tr>
<td>63</td>
<td>I12= 0</td>
<td>I13= 0</td>
<td>I23= 0</td>
</tr>
<tr>
<td>67</td>
<td>###EOF 777</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LINK_NUMBER** defines the number of links in the robot's structure. The value includes the base link. In the example of Listing 5.1, there are 3 links.

**BB** defines the structure of the robot. Elements of this vector define the parent link of link \( i \). The first value defines the parent link of link 1. Listing 5.1 shows a serial kinematic chain. The parent of link 1 is the base link (link 0) and the parent of link 2 is link 1.

**J_type** describes the type of each joint, starting at joint 1. There are 4 different joint types:

- \( 0 = \) rotational joint around and the z-axis
- \( 1 = \) translational joint along z-axis
- \( 2 = \) rotational joint around custom axis
- \( 3 = \) translational joint along custom axis

For example in Listing 5.1, all joints are rotational.

**EE** defines which end effector is attached to which link, starting at link 1. 0 means that no end effector is attached to this link. A number \( e \) indicates that the end effector \( e \) is attached to this link. Thus, only one end effector is possible per link. For example EE[0 0 1 0 2] shows that end effector 1 is connected to link 3, and end effector 2 is connected to link 5. In Listing 5.1, there is only one end effector that is connected to link 2.

**CL** is the closed loop flag and defines on which link a loop is closed. The value -1 means that this link is not closing a loop. A value \( l \) at position \( i \) describes that link \( i \) closes a loop, by being connected to link \( l \). In Listing 5.1, link 2 is closing the loop by being connected to link 0, the base link.
5. **SpaceDynamics Library**

**rpy** defines how the link associated coordinate system is rotated around the roll, pitch and yaw angles for each link. The roll, pitch and yaw angles describe the rotation around the x, y and z-axis, respectively. The values are given in degree. For example $\text{rpy3} \ [90 \ 0 \ 0]$ indicates that the transformation of coordinate system of link 2 to coordinate system of link 3 is a rotation by 90 degrees around the x-axis. To define a custom joint axis, one can use $\text{rpy}[r \ p \ y \ x \ y \ z]$, were x, y and z define the unit vector of the joint. All joints must have a rpy definition. In the example of Listing 5.1 all link coordinate systems have the same orientation.

**CtoJ** defines the translational vector from the center of mass of the link to the next joint. JtoJ can be used instead, which defines the translational vector from the previous joint to the following. The values are given in meters. In the example of Listing 5.1 $\text{CtoJ}_{0,1} \ [10 \ 0 \ 0]$ means that joint 1 is located 10 meters in x-direction from the center of mass of the base link (link 0).

**JtoC** defines the translational vector from the joint to the center of mass of the link with values in meters. The CtoJ, JtoJ and JtoC vectors must be in a special order, which is used in Listing 5.1. All joints must have a CtoJ or JtoJ and JtoC vector.

**CtoE** is the translational vector from the center of mass of the link to the end effector. Similar to CtoJ, all end effectors must have a CtoE definition. In the example of Listing 5.1 the end effector is located 5 meters in x-direction from the center of mass of link 2.

**rpyE** describes similar to rpy how the end effector associated coordinate system is rotated around the roll, pitch and yaw angles for each end effector with angles in degrees. In Listing 5.1 the end effector is oriented in the same direction as link 2, to which it is connected.

**mass** defines the mass of each link starting with the base link (link 0). The values are in kg. In the example of Listing 5.1 link 0 has a mass of 1 kg, link 1 has 2 kg and link 2 has 3 kg.

**Inertias.** The inertia tensor is needed for each link. The parameters are given in kg m$^2$. The inertia values are listed as described in the minimal model in Listing 5.1. The inertia tensor matrix is given by

$$
I = \begin{bmatrix}
I_{11} & I_{12} & I_{13} \\
I_{21} & I_{22} & I_{23} \\
I_{31} & I_{32} & I_{33}
\end{bmatrix}
$$

With all these parameters, a robot can be described and used for calculations with the SpaceDynamics library. The def-file is loaded into a model of the SpaceDynamics Library with the function **model**, which is described in the next section.
5.3. Functions

This section presents now some important functions of the SpaceDynamics Library and describes how they are used.

5.3.1. model

```c
void model(string filename, MODEL &m, bool output, bool closed_loop)
```

To load a def-file into the SpaceDynamics Library, the function `model` is used. This function loads the kinematic and dynamic parameters from the def-file into the model and calculates the corresponding spatial notation with `calc_SPN`.

**Parameters.** The function expects four input parameters:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>string</td>
<td>the path to the def-file</td>
</tr>
<tr>
<td>m</td>
<td>MODEL</td>
<td>the pointer to the model object to which the parameters are loaded</td>
</tr>
<tr>
<td>output</td>
<td>boolean</td>
<td>if true then debug output is printed</td>
</tr>
<tr>
<td>closed_loop</td>
<td>boolean</td>
<td>if true then the model contains closed loops</td>
</tr>
</tbody>
</table>

**Example.** An example how to load the parameters from a def-file into a model is given in the Listing 5.2.

Listing 5.2: Example of Model Loading

```c
void load_model(void){
    /* file name */
    string def_file = "LBR3_JtoJ_light.def";
    /* create a new model */
    MODEL m;
    /* load parameters from def-file into model m */
    model(def_file.c_str(), m, false, false);
}
```

5.3.2. f_kin_e

```c
void f_kin_e(MODEL &m, int e_num)
```

The function calculates the forward dynamics of the selected end effector for the current robot configuration. The forward dynamics is the calculation of the pose – that is the position and the orientation of the end effector.
5. SpaceDynamics Library

**Parameters.** The function expects the model and the number of the end effector as input. The orientation and position is then calculated and stored in the model variables `m.POS_e` and `m.ORI_e`.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>MODEL</td>
<td>the pointer to the model object</td>
</tr>
<tr>
<td>e_num</td>
<td>int</td>
<td>the number of the end effector</td>
</tr>
<tr>
<td>m.POS_e[e_num]</td>
<td>Vector3 (3 x 1)</td>
<td>the position of the end effector e_num</td>
</tr>
<tr>
<td>m.ORI_e[e_num]</td>
<td>Matrix3 (3 x 3)</td>
<td>the rotation matrix of the end effector e_num</td>
</tr>
</tbody>
</table>

**Example.** An example of this function is given in Listing [A.1](#).

5.3.3. `i_dyn_fix`

```c
void i_dyn_fix (MODEL &m, double *Gravity, double *tau)
```

The function `i_dyn_fix` computes the inverse dynamics for a fixed base multi-body system. This means it calculates the required forces and torques to generate a given motion for a fixed base system. The algorithm implements the recursive Newton-Euler Formulation which is described in section [3.2.3](#). It has a computational complexity of O(n).

**Parameters.** The motion of the system is given by the model variables for joint position `m.q`, velocity `m.qd` and acceleration `m.qdd`. The function requires furthermore three parameters: The model `m` and the gravity are needed as input, `tau` is the variable where the result is stored.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>MODEL</td>
<td>the pointer to the model object</td>
</tr>
<tr>
<td>Gravity</td>
<td>double (3 x 1)</td>
<td>the gravity vector</td>
</tr>
<tr>
<td>tau</td>
<td>double (N_j x 1)</td>
<td>the calculated joint forces and torques</td>
</tr>
</tbody>
</table>

**Example.** An example of this function is given in Listing [A.2](#).
5. SpaceDynamics Library

5.3.4. f_dyn_fix

```c
void f_dyn_fix (MODEL &m, double *Gravity, double *qdd)
```

The function `f_dyn_fix` computes the forward dynamics for a fixed base multi-body system. It calculates the joint accelerations $\ddot{q}$ for a given set of joint forces $\tau$.

**Parameters.** The input of this function is represented by the model variable for the joint torques m.tau. The function requires furthermore three parameters: The model m and the gravity are needed as input, qdd is the output variable where the result is stored.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>MODEL</td>
<td>the pointer to the model object</td>
</tr>
<tr>
<td>Gravity</td>
<td>double ($3 \times 1$)</td>
<td>the gravity vector</td>
</tr>
<tr>
<td>qdd</td>
<td>double ($N_J \times 1$)</td>
<td>the calculated joint accelerations</td>
</tr>
</tbody>
</table>

**Example.** An example of this function is given in Listing A.2

5.3.5. i_dyn

```c
void i_dyn (MODEL &m, double *Gravity, double *Force)
```

The function `i_dyn` computes the inverse dynamics for a multi-body system with free floating base, which are usually space robots. In contrast to a fixed base system, the base of a free floating robot can be accelerated. This is done either passively by momentum conservation or actively by applying a torque, for example by using a thruster. The function calculates the required forces of the robots joints and base, that generate a given motion for the system. Here Force[0]- Force[5] represent the forces and torques that are acting on the base, Force[6] - Force[$N_J + 6$] represent the joint forces and torques.

**Parameters.** The motion of the system is given by the model variables for joint position m.q, velocity m.qd and acceleration m.qdd. The base position, velocity

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>MODEL</td>
<td>the pointer to the model object</td>
</tr>
<tr>
<td>Gravity</td>
<td>double ($3 \times 1$)</td>
<td>the gravity vector</td>
</tr>
<tr>
<td>Force</td>
<td>double ($N_J + 6 \times 1$)</td>
<td>the calculated base and joint forces</td>
</tr>
</tbody>
</table>
5. SpaceDynamics Library

and acceleration is set by the 3 dimensional vectors \( \text{m.POS0} \), \( \text{m.v0} \) and \( \text{m.vd0} \), respectively. If the base has a angular velocity or acceleration, this is set by \( \text{m.w0} \) and \( \text{m.wd0} \). The function requires three parameters: The model \( \text{m} \) in order to access the model variables and the gravity are needed as input, Force is the variable where the result is stored.

### 5.3.6. f_dyn

```c
void f_dyn(MODEL &m, double *Gravity, double *vd0, double *wd0, double *qdd)
```

The function \( f_{\text{dyn}} \) computes the forward dynamics for a free floating multi-body system. It calculates the joint accelerations and the linear and angular base accelerations for a given set of joint forces, base forces and base torques.

**Parameters.** The input of this function is represented by the model variables for the joint torques \( \text{m.tau} \), the base forces \( \text{m.F0} \) and the base torques \( \text{m.T0} \). Furthermore, forces \( \text{(m.Fe)} \) and torques \( \text{(m.Te)} \) on the end effectors are taken into consideration. The function requires five parameters: The model \( \text{m} \) and the gravity are needed as input, \( \text{vd0, wd0} \) and \( \text{qdd} \) are the output variables where the result is stored.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m )</td>
<td>MODEL</td>
<td>the pointer to the model object</td>
</tr>
<tr>
<td>Gravity</td>
<td>double ((3 \times 1))</td>
<td>the gravity vector</td>
</tr>
<tr>
<td>( \text{vd0} )</td>
<td>double ((3 \times 1))</td>
<td>the calculated linear base acceleration</td>
</tr>
<tr>
<td>( \text{wd0} )</td>
<td>double ((3 \times 1))</td>
<td>the calculated angular base acceleration</td>
</tr>
<tr>
<td>( \text{qdd} )</td>
<td>double ((N_J \times 1))</td>
<td>the calculated joint accelerations</td>
</tr>
</tbody>
</table>
6. Implementation, Tests and Simulations

This chapter illustrates the implementation and testing of the algorithms to calculate the dynamics of robots with closed loops, which are described in chapter 4. In order to implement the new functions, the existing functions have first to be validated. Then the algorithms for dynamic modeling of robots with closed kinematic loops are implemented and tested. This chapter first shows how the commercial software SIMPACK is used in order to validate the functions of the SpaceDynamics Library. In this scope, the approach to test these functions is described. After that, the implementation, tests and results of the dynamic functions for kinematic trees and for kinematic loops are presented.

6.1. Simpack

SIMPACK is a commercial multi-body simulation software that used for mechanical system design. It is able to simulate the dynamics of different systems for example in aerospace, automotive and rail applications. SIMPACK is used at the DLR for the simulation and demonstration of robotic systems.

Figure 6.1 shows the main window of SIMPACK. One model, that is used for the validation of the SpaceDynamics Library is already displayed on the 3D page. On the right side there is the model tree, which shows the different bodies, joints and other model specific preferences. SIMPACK offers a wide range of options to model a system in a very accurate way.

6.1.1. Test Procedures

Since SIMPACK is a simulation software, it is only capable to compute the forward dynamics – that is the calculation of the joint accelerations $\dot{\mathbf{q}}$ for given joint torques $\mathbf{\tau}$. However the inverse dynamics calculation is not possible. Nevertheless both, the inverse and the forward dynamics functions have to be tested for validation.

To test the dynamic functions, the same model must be build in SIMPACK and build as a def-file for the SpaceDynamics Library first. Now, some forces $\mathbf{\tau}_{in}$ are applied to the joints in SIMPACK. After the forces simulation, SIMPACK shows the joint accelerations $\dot{\mathbf{q}}_{SP}$ that result due to the applied forces. The same forces $\mathbf{\tau}_{in}$ are now applied to the SpaceDynamics Library model. The joint accelerations $\dot{\mathbf{q}}_{SDL}$
6. Implementation, Tests and Simulations

are now calculated with the forward dynamics function. If the joint accelerations calculated by SIMPACK \( \ddot{q}_{SP} \) correspond with those calculated by the SpaceDynamics Library \( \ddot{q}_{SDL} \), the implemented forward dynamics function is valid.

In order to test the inverse dynamics, the joint accelerations \( \ddot{q}_{SDL} \) calculated by the forward dynamics are applied to the SpaceDynamics Library model. The joint torques \( \tau_{out} \) that are needed to generate those accelerations are calculated with the inverse dynamics. If those calculated joint torques \( \tau_{out} \) correspond with the originally applied joint torques \( \tau_{in} \), the implemented inverse dynamics function is valid.

Figure 6.2.: Test Procedures
6. Implementation, Tests and Simulations

Figure 6.2 illustrates the test procedure. If $\ddot{q}_{SP} = \ddot{q}_{SDL}$, then the forward dynamics function of the SpaceDynamics Library is valid. If $\tau_{in} = \tau_{out}$, then the implemented inverse dynamics function of the SpaceDynamics Library is valid.

6.2. Functions for Kinematic Trees

6.2.1. Test of Forward Kinematics

Before implementing the new algorithms, the existing functions of the SpaceDynamics Library had to be tested. A forward kinematics test was implemented in order to test if the def-file model corresponds with the model built in SIMPACK. As described in the introduction, forward kinematics is the calculation of the pose of the end effector for given joint positions. In the appendix, the implemented test function is shown in Listing A.1.

Figures 6.3 and 6.4 show the tests for different configurations. In the Forward Kinematics Test 1, the model is in its initial setup. All joint angles are 0. With the measure tool, the pose of the end effector is measured in SIMPACK. The SpaceDynamics Library uses the forward kinematics function $f_{kin,e}$ (section 5.3.3) in order to determine the pose of the end effector. The result of the Forward Kinematic Test 1 are shown in the Figure 6.3 and Table 6.1 and are identical.

<table>
<thead>
<tr>
<th>Variable</th>
<th>SIMPACK</th>
<th>SDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance x</td>
<td>-0.885 m</td>
<td>-0.885 m</td>
</tr>
<tr>
<td>Distance y</td>
<td>0.012 m</td>
<td>0.012 m</td>
</tr>
<tr>
<td>Distance z</td>
<td>2.106 m</td>
<td>2.106 m</td>
</tr>
<tr>
<td>Angle 1</td>
<td>0 deg</td>
<td>0 deg</td>
</tr>
<tr>
<td>Angle 2</td>
<td>0 deg</td>
<td>0 deg</td>
</tr>
<tr>
<td>Angle 3</td>
<td>0 deg</td>
<td>0 deg</td>
</tr>
</tbody>
</table>

To be sure that the model representation in SIMPACK is corresponding with the SpaceDynamics Library, a second forward kinematics test is performed. This time, the joint angles of joint 2, 3 and 4 are each set to 90 deg. The results are shown in Figure 6.4 and Table 6.2. As for the first test, the results correspond with each other. Therefore the forward kinematics method is valid. The model representation in both, SIMPACK and the SpaceDynamics Library correspond with each other. For this reason, the model can be used to perform the following tests. The def-file of the model is displayed in Listing A.5.
6. Implementation, Tests and Simulations

(a) SIMPACK

(b) SpaceDynamics Library

Figure 6.3.: Forward Kinematics Test 1
6. Implementation, Tests and Simulations

(a) SIMPACK

(b) SpaceDynamics Library

Figure 6.4.: Forward Kinematics Test 2
6. Implementation, Tests and Simulations

Table 6.2.: Results of Forward Kinematics Test 2

<table>
<thead>
<tr>
<th>Variable</th>
<th>SIMPACK</th>
<th>SDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance x</td>
<td>-1.285 m</td>
<td>-1.285 m</td>
</tr>
<tr>
<td>Distance y</td>
<td>0.514 m</td>
<td>0.514 m</td>
</tr>
<tr>
<td>Distance z</td>
<td>1.18 m</td>
<td>1.18 m</td>
</tr>
<tr>
<td>Angle 1</td>
<td>-90 deg</td>
<td>-90 deg</td>
</tr>
<tr>
<td>Angle 2</td>
<td>7.016e-15 deg</td>
<td>7.016e-15 deg</td>
</tr>
<tr>
<td>Angle 3</td>
<td>7.016e-15 deg</td>
<td>7.016e-15 deg</td>
</tr>
</tbody>
</table>

6.2.2. Test of Dynamics for Fixed Base

After the successful forward kinematics test, the forward and inverse dynamic functions for a kinematic tree system with fixed base were tested. These methods especially the function \textit{i\_dyn\_fix} (see 5.3.3) is needed for the dynamics calculation of closed loops.

For the test, the same robot configuration as in the Forward Kinematics Test 2 was used. Additionally, an initial velocity was set to joint 2 and different torques were set to joint 2 and 4. The parameters of the robot configuration are shown in Table 6.3 and the implemented test is printed in Listing A.2.

Table 6.3.: Model Configuration of Dynamic Test

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>LBR3_JtoJ_light.def</td>
<td>Model</td>
</tr>
<tr>
<td>m.q[2]</td>
<td>90 deg</td>
<td>joint 2 angle</td>
</tr>
<tr>
<td>m.q[3]</td>
<td>90 deg</td>
<td>joint 3 angle</td>
</tr>
<tr>
<td>m.q[4]</td>
<td>90 deg</td>
<td>joint 4 angle</td>
</tr>
<tr>
<td>m.qd[2]</td>
<td>$-\frac{1}{m}$</td>
<td>joint 2 velocity</td>
</tr>
<tr>
<td>m.tau[2]</td>
<td>1 Nm</td>
<td>joint 2 torque</td>
</tr>
<tr>
<td>m.tau[4]</td>
<td>-1 Nm</td>
<td>joint 4 torque</td>
</tr>
</tbody>
</table>

By analyzing the results of the SpaceDynamics Library which are displayed in Figure 6.5 (b) and Table 6.5, one can easily identify an error. The input of the forward dynamics function are the joint torques, as shown in Table 6.3. With these torques, the forward dynamics calculates the resulting joint accelerations. These joint accelerations are now taken as the input of the inverse dynamics. Now, the inverse dynamics calculates the joint torques that generate those accelerations. Unfortunately, the input joint torques $\tau_{in}$ do not correspond with the the calculated joint torques $\tau_{out}$. Therefore an error is in the code.

Now, the results of the SIMPACK simulation and the SpaceDynamics library forward dynamic function are compared. They are shown in Figure 6.5 and in Table 6.4. The acceleration of joint 1 and joint 2 corresponds with both functions. The calculated accelerations of the other joints do however not correspond with each
Table 6.4.: Results of the Forward Dynamics Test

<table>
<thead>
<tr>
<th>Joint</th>
<th>$\dot{q}_{SP}$ [rad/s]</th>
<th>$\dot{q}_{SDL}$ [rad/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-0.751662</td>
<td>-0.751662</td>
</tr>
<tr>
<td>2</td>
<td>31.115875</td>
<td>31.1313156434</td>
</tr>
<tr>
<td>3</td>
<td>-7.537469</td>
<td>28.2272513543</td>
</tr>
<tr>
<td>4</td>
<td>-4.664762</td>
<td>-3.6110301003</td>
</tr>
<tr>
<td>5</td>
<td>28.435888</td>
<td>1.5123947484</td>
</tr>
<tr>
<td>6</td>
<td>-8.734599</td>
<td>1.7151791601</td>
</tr>
<tr>
<td>7</td>
<td>2.679986</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.5.: Results of the Inverse Dynamics Test

<table>
<thead>
<tr>
<th>Joint</th>
<th>$\tau_{in}$ [Nm]</th>
<th>$\tau_{out}$ [Nm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-0.1548350031</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>19.9850212193</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>16.3392891096</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-0.4487079260</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>-0.1811652399</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0.0284925225</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>-0.0035631562</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6.5.: Dynamics Test for Kinematic Tree with Fixed Base

other. At this point, it is clear that at least one error exists in the forward dynamics function.
6. Implementation, Tests and Simulations

6.2.3. Matrix Based Dynamics for Fixed Base

In order to find the error of the implemented forward dynamics algorithm, the new functions \texttt{i\_dyn\_fix\_matrix\_based} and \texttt{f\_dyn\_fix\_matrix\_based} were implemented.

```c
void i\_dyn\_fix\_matrix\_based (MODEL &m, double *Gravity, double *tau)
void f\_dyn\_fix\_matrix\_based (MODEL &m, double *Gravity, double *qdd)
```

The functions \texttt{i\_dyn\_fix\_matrix\_based} and \texttt{f\_dyn\_fix\_matrix\_based} compute the inverse and forward dynamics similar to the function \texttt{i\_dyn\_fix} (see 5.3.1) and \texttt{f\_dyn\_fix}. Instead of using the recursive Newton-Euler Formulation, these algorithms implement equation 3.1.1 directly and have therefore a computational complexity of \(O(n^4)\). The parameters of \texttt{i\_dyn\_fix\_matrix\_based} are exactly the same as described for the function \texttt{i\_dyn\_fix}. The implemented functions are shown in Listings A.3 and A.4.

With these functions it is possible to test the dynamic model representation, for example if the inertia matrix \(H\) is calculated appropriately. The implementation of the test equals the one of the dynamics for a fixed base system (see Listing A.2). Except the functions \texttt{i\_dyn\_fix} and \texttt{f\_dyn\_fix} were replaced by \texttt{i\_dyn\_fix\_matrix\_based}

### Table 6.6.: Results of the Forward Dynamics Test

<table>
<thead>
<tr>
<th>Joint</th>
<th>(\ddot{q}_{SP} [m/s^2])</th>
<th>(\ddot{q}_{SDL} [m/s^2])</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.751662</td>
<td>-0.7516621817</td>
</tr>
<tr>
<td>2</td>
<td>31.115875</td>
<td>31.1158748271</td>
</tr>
<tr>
<td>3</td>
<td>-7.537469</td>
<td>-7.5374685603</td>
</tr>
<tr>
<td>4</td>
<td>-4.664762</td>
<td>-4.6647625308</td>
</tr>
<tr>
<td>5</td>
<td>28.435888</td>
<td>28.4358887822</td>
</tr>
<tr>
<td>6</td>
<td>-8.734599</td>
<td>-8.7345991083</td>
</tr>
<tr>
<td>7</td>
<td>2.679986</td>
<td>2.6799860449</td>
</tr>
</tbody>
</table>

### Table 6.7.: Results of the Inverse Dynamics Test

<table>
<thead>
<tr>
<th>Joint</th>
<th>(\tau_{in} [Nm])</th>
<th>(\tau_{out} [Nm])</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 6.6.: SpaceDynamics Library
and \texttt{f\_dyn\_fix\_matrix\_based}, respectively.

For the test, the same parameters as described in Table 6.3 are used. The results of the matrix based calculations are shown in Figure 6.6. The results of the SIMPACK simulation (see Figure 6.5) are compared to the results of the forward dynamics matrix based test in Table 6.7. The results of the inverse dynamics is shown in Table 6.7. Since \( \ddot{q}_{SP} = \ddot{q}_{SDL} \) and \( \tau_{in} = \tau_{out} \), the both implemented functions, \texttt{i\_dyn\_fix\_matrix\_based} and \texttt{f\_dyn\_fix\_matrix\_based} are valid.

The results also prove the dynamic model representation of the SpaceDynamics Library to be valid. Therefore, the error of the forward dynamics function \texttt{f\_dyn\_fix} is not because of a wrong calculation of the inertia matrix \( H \) for example, but rather because of an error in the code.

### 6.2.4. Dynamics for Fixed Base – Error Correction

The error in the forward dynamics function for fixed base robots was found after debugging the code step by step. In Listing 6.1 one can detect the error.

```
Listing 6.1: spacedyn/src/f\_dyn\_fix.cpp

250 qdd[i] = ( u[i] - (\*tmp3) + (\*tmp4) ) \times (d\_recip);
252 matrix\_scale( 6, 1, m.qdd[i], m.S[i], tmp );
```

In line 250 the variable \( qdd \) is set which represents the output. In the next step, \( qdd \) should be multiplied with a scalar value and stored in the variable \( tmp \). However, not \( qdd \) but \( m.qdd \), which is not yet set, is multiplied. This leads to a wrong value of \( tmp \), which has a big effect on the further calculation. By replacing \( m.qdd \) with \( qdd \), the error is removed. Now the forward dynamics function \texttt{f\_dyn\_fix} obtains the very same results as the function \texttt{f\_dyn\_fix\_matrix\_based}, which are displayed in Figure 6.6 and Tables 6.6 and 6.7. Since \( \ddot{q}_{SP} = \ddot{q}_{SDL} \) and \( \tau_{in} = \tau_{out} \), the functions \texttt{f\_dyn\_fix} and \texttt{i\_dyn\_fix} are now valid as well.

Of course this was not the only error that occurred during the testing and the implementation of the new algorithms. The description of the error, the implementation of other functions as temporary solutions and the error correction shows exemplary how errors have been detected and corrected in general.

### 6.2.5. Dynamics for Free Floating Base

After the error in the function \texttt{f\_dyn\_fix} was detected and corrected, the next functions to test are the forward and inverse dynamics for free floating base robots – \texttt{f\_dyn} and \texttt{i\_dyn} (see sections 5.3.5 and 5.3.6). The tests are performed similar to the tests of the dynamics functions for fixed base robots. This is the same implementation as in Listing 6.2, except the functions \texttt{f\_dyn\_fix} and \texttt{i\_dyn\_fix} are replaced by the functions \texttt{f\_dyn} and \texttt{i\_dyn}, respectively. The robot configuration for this test is shown in Table 6.8.
6. Implementation, Tests and Simulations

Table 6.8.: Model Configuration of Dynamic Test

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>LBR3_JtoJ_light.def</td>
<td>Model</td>
</tr>
<tr>
<td>m.q[2]</td>
<td>90 deg</td>
<td>joint 2 angle</td>
</tr>
<tr>
<td>m.q[3]</td>
<td>90 deg</td>
<td>joint 3 angle</td>
</tr>
<tr>
<td>m.q[4]</td>
<td>90 deg</td>
<td>joint 4 angle</td>
</tr>
<tr>
<td>m.qd[2]</td>
<td>$-1 m/s$</td>
<td>joint 2 velocity</td>
</tr>
<tr>
<td>m.T0[2]</td>
<td>100 Nm</td>
<td>z-axis base torque</td>
</tr>
<tr>
<td>m.tau[2]</td>
<td>1 Nm</td>
<td>joint 2 torque</td>
</tr>
<tr>
<td>m.tau[4]</td>
<td>-1 Nm</td>
<td>joint 4 torque</td>
</tr>
</tbody>
</table>

Forward Dynamics for Open-Loop Robot with Free-Floating Base

![Diagram](image)

Figure 6.7.: Dynamics Test for Kinematic Tree with Free-Floating Base

The same error that was in the function \( f_{dyn\_fix} \), also occurred in function \( f_{dyn} \). Furthermore the gravity was defined in the wrong direction. After removing these errors, the output of the test is shown in Figure 6.7 and Tables 6.9 and 6.10. The base linear acceleration is represented by \( \text{vd0} \) whereas the base angular acceleration is represented by \( \text{wd0} \). The force that acts on the base is represented by \( \text{F0} \), the torque by \( \text{T0} \).
6. Implementation, Tests and Simulations

Table 6.9.: Results of the Forward Dynamics Test

<table>
<thead>
<tr>
<th>Joint</th>
<th>$\dot{q}_{SP}$ [m/s²]</th>
<th>$\dot{q}_{SDL}$ [m/s²]</th>
</tr>
</thead>
<tbody>
<tr>
<td>vd0[x]</td>
<td>0.0013803</td>
<td>0.0013802981</td>
</tr>
<tr>
<td>vd0[y]</td>
<td>-0.00260104</td>
<td>-0.0026010391</td>
</tr>
<tr>
<td>vd0[z]</td>
<td>0.00411488</td>
<td>0.0041148829</td>
</tr>
<tr>
<td>wd0[x]</td>
<td>-0.0105823</td>
<td>-0.0105822554</td>
</tr>
<tr>
<td>wd0[y]</td>
<td>0.0137185</td>
<td>0.0137184927</td>
</tr>
<tr>
<td>wd0[z]</td>
<td>0.242617</td>
<td>0.2426168710</td>
</tr>
<tr>
<td>1</td>
<td>-1.61052</td>
<td>-1.6105159758</td>
</tr>
<tr>
<td>2</td>
<td>1.67362</td>
<td>1.6736159316</td>
</tr>
<tr>
<td>3</td>
<td>-1.82478</td>
<td>-1.8247797613</td>
</tr>
<tr>
<td>4</td>
<td>-5.06547</td>
<td>-5.0564718393</td>
</tr>
<tr>
<td>5</td>
<td>-0.702101</td>
<td>-0.7021009102</td>
</tr>
<tr>
<td>6</td>
<td>-8.04763</td>
<td>-8.0476330328</td>
</tr>
<tr>
<td>7</td>
<td>2.362</td>
<td>2.3620009491</td>
</tr>
</tbody>
</table>

Table 6.10.: Results of the Inverse Dynamics Test

<table>
<thead>
<tr>
<th>Joint</th>
<th>$\tau_{in}$ [Nm]</th>
<th>$\tau_{out}$ [Nm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0[x ]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F0[y ]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F0[z ]</td>
<td>0</td>
<td>-0</td>
</tr>
<tr>
<td>T0[x ]</td>
<td>0</td>
<td>-0</td>
</tr>
<tr>
<td>T0[y ]</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T0[z ]</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>-0</td>
</tr>
</tbody>
</table>

All results obtained by the SpaceDynamics Library correspond with the results obtained by the SIMPACK simulation. Therefore the functions $f_{dyn}$ and $i_{dyn}$ are valid.

6.3. Functions for Kinematic Loops

After successful testing of the dynamic functions for kinematic trees, the functions for kinematic loops were implemented. First, one algorithm of those presented in chapter 4 had to be chosen. The dynamic modeling with generalized coordinates by Nakamura [12] (see section 4.2.1) offers a forward and inverse dynamics method, whereas the dynamic modeling with loop joints by Featherstone [9] (see section 4.3) offers only a method to compute the forward dynamics. Nakamuras approach uses the generalized coordinates of a kinematic loop, which is the minimal number of coordinates that is needed for the computation. The method is therefore computational efficient. Although the method aims at the application to human figures, it is a general approach to the dynamic modeling of kinematic loops. Furthermore the algorithm offers the dynamic modeling for robots with free floating base and varying structure. For these reasons the algorithm of Nakamura was chosen to be implemented in the SpaceDynamics Library. The dynamic modeling method with generalized coordinates is distributed over several functions, which are explained now in detail.

6.3.1. H_CL_fix

```c
double *H_CL_fix( MODEL *m, int *index_m, int *index_n, unsigned int *Nf, unsigned int **q, unsigned int **q_G){
```

50
The function $H_{\text{CL\_fix}}$ calculates the Jacobian matrices $J_C$, $J_{Cm}$, $J_S$, $J_G$ and $H$ according to the algorithm presented in section 4.2.1 for a fixed base robotic system. This function is used by the forward dynamics and inverse dynamics computation. Equations 4.2.11 - 4.2.17 are implemented.

**Parameters.** The function expects six parameters. The only input parameter is the model, the other parameters are output variables. Table 6.11 gives an overview of the parameters for this function. The function returns the pointer to matrix $H$.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>MODEL</td>
<td>the pointer to the model object</td>
</tr>
<tr>
<td>index_m</td>
<td>int</td>
<td>the number of lines of $H$</td>
</tr>
<tr>
<td>index_n</td>
<td>int</td>
<td>the number of columns of $H$</td>
</tr>
<tr>
<td>N_f</td>
<td>int</td>
<td>the degrees of freedom $N_F$</td>
</tr>
<tr>
<td>$q_S$</td>
<td>int</td>
<td>(index_m $\times$ 1) the index of the dependent joint angles $\theta_S$</td>
</tr>
<tr>
<td>$q_G$</td>
<td>int</td>
<td>(index_n $\times$ 1) the index of the generalized coordinates $\theta_G$</td>
</tr>
</tbody>
</table>

### 6.3.2. i_dyn_CL_fix

```c
void i_dyn_CL_fix( MODEL &m, double *Gravity, double *Torque_CL, double *p_taudist )
```

The function $i_{\text{dyn\_CL\_fix}}$ computes the inverse dynamics for a kinematic loop system with fixed base. It calculates the required actuator torques in order to perform a certain motion. Similar to the function $i_{\text{dyn\_fix}}$ for kinematic tree systems, the motion is given by the joint positions, velocities and accelerations. The method implements the inverse dynamics of the algorithm as described in section 4.2.4 and uses $i_{\text{dyn\_fix}}$.

**Parameters.** The function expects four parameters. The Model, the gravity vector and the torque distribution vector are the inputs, Torque_CL is output variables containing the result. Table 6.12 gives an overview of the parameters for this function.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>MODEL</td>
<td>the pointer to the model object</td>
</tr>
<tr>
<td>Gravity</td>
<td>double (3 $\times$ 1)</td>
<td>the gravity vector</td>
</tr>
<tr>
<td>p_taudist</td>
<td>double ($N_A - N_F \times 1$)</td>
<td>the torque distribution vector</td>
</tr>
<tr>
<td>Torque_CL</td>
<td>double ($N_A \times 1$)</td>
<td>the calculated actuator torques</td>
</tr>
</tbody>
</table>
6. Implementation, Tests and Simulations

6.3.3. f_dyn_CL_fix

```c
void f_dyn_CL_fix( MODEL &m, double *Gravity, double *qdd )
```

The function `f_dyn_CL_fix` computes the forward dynamics for a kinematic loop system with fixed base. It calculates the joint accelerations for a given set of actuator torques. The method implements the inverse dynamics of the algorithm as described in section 4.2.5 and uses `i_dyn_CL_fix`.

**Parameters.** The function expects three parameters. The Model and the gravity vector are the inputs, qdd is the output variable, that is containing the result. Table 6.13 gives an overview of the parameters for this function.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>MODEL</td>
<td>the pointer to the model object</td>
</tr>
<tr>
<td>Gravity</td>
<td>double (3 × 1)</td>
<td>the gravity vector</td>
</tr>
<tr>
<td>qdd</td>
<td>double (N_J × 1)</td>
<td>the calculated joint accelerations</td>
</tr>
</tbody>
</table>

6.3.4. Test of Dynamics for Fixed Base – 1

The implemented dynamic functions for closed loop robots are now tested. To create a closed loop, the same system is used as for the open loop tests, but the end effector is set to a fixed position. The motion of the end effector is constrained in 6 dimensions. So no linear movement along and no rotation around the x, y and z-axis is possible. In the def-file, this is done by adding a line with the closed loop flag. The def-file of the used model is displayed in Listing A.6. The loop is closed on Link 7. The robot configuration is shown in Table 6.14.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>LBR3_JtoJ_light_CL.def</td>
<td>Model</td>
</tr>
<tr>
<td>m.q[2]</td>
<td>45 deg</td>
<td>joint 2 angle</td>
</tr>
<tr>
<td>m.q[4]</td>
<td>90 deg</td>
<td>joint 4 angle</td>
</tr>
<tr>
<td>m.q[6]</td>
<td>45 deg</td>
<td>joint 6 angle</td>
</tr>
<tr>
<td>m.tau[1]</td>
<td>1 Nm</td>
<td>joint 1 torque</td>
</tr>
</tbody>
</table>

First, a forward kinematics test is performed, in order to test the model representation in SIMPACK and the SpaceDynamics library. The results of the tests are displayed in Figure 6.8 and Table 6.15.

The results obtained by SIMPACK, correspond with the results from the SpaceDynamics Library function, therefore the model representation is valid and can be used for further testing.
6. Implementation, Tests and Simulations

(a) SIMPACK

(b) SpaceDynamics Library

Figure 6.8.: Results of Forward Kinematics Test

Table 6.15.: Results of Forward Kinematics Test

<table>
<thead>
<tr>
<th>Variable</th>
<th>SIMPACK</th>
<th>SDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance x</td>
<td>-0.8920711 m</td>
<td>-0.892071 m</td>
</tr>
<tr>
<td>Distance y</td>
<td>-0.012 m</td>
<td>-0.012 m</td>
</tr>
<tr>
<td>Distance z</td>
<td>1.874614 m</td>
<td>1.87461 m</td>
</tr>
<tr>
<td>Angle 1</td>
<td>7.06225e-31 deg</td>
<td>7.06225e-31 deg</td>
</tr>
<tr>
<td>Angle 2</td>
<td>1.272222e-14 deg</td>
<td>1.1441e-14 deg</td>
</tr>
</tbody>
</table>
Now, the forward and inverse dynamics of the system are tested. The test is implemented similar to the dynamic tests for the kinematic tree system, which is shown by Listing A.2. However, the model configuration is not the same and the functions $\mathbf{f}_{\text{dyn\_fix}}$ and $\mathbf{i}_{\text{dyn\_fix}}$ are replaced by $\mathbf{f}_{\text{dyn\_CL\_fix}}$ and $\mathbf{i}_{\text{dyn\_CL\_fix}}$. The results are shown in Figure 6.9 and in Tables 6.16 and 6.17. The results of the forward dynamics differ by only 0.228%, they are therefore corresponding with each other. The inverse dynamics function calculates the very same result as the SIMPACK simulation. The functions are therefore considered to be valid.

### Table 6.16.: Results of the Forward Dynamics Test

<table>
<thead>
<tr>
<th>Joint</th>
<th>$\ddot{q}_{SP}$ [m/s²]</th>
<th>$\ddot{q}_{SDL}$ [m/s²]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3.070515</td>
<td>3.0635261271</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>-3.149247</td>
<td>-3.1420780791</td>
</tr>
</tbody>
</table>

### Table 6.17.: Results of the Inverse Dynamics Test

<table>
<thead>
<tr>
<th>Joint</th>
<th>$\tau_{in}$ [Nm]</th>
<th>$\tau_{out}$ [Nm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
6.3.5. Test of Dynamics for Fixed Base – 2

Now the same functions \( f_{\text{dyn\_CL\_fix}} \) and \( i_{\text{dyn\_CL\_fix}} \) are tested with a different robot model, in order to validate the functions. The new model is built from the model used before. A second arm, similar to the already existing is added. Then, both end effectors are connected to perform a closed loop motion. The def-file of this model is displayed in Listing A.7. The model configuration is shown in Table 6.18.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>LBR3_JtoJ_light_CL2.def</td>
<td>Model</td>
</tr>
<tr>
<td>m.q[1]</td>
<td>0.7768443341 deg</td>
<td>joint 1 angle</td>
</tr>
<tr>
<td>m.q[2]</td>
<td>-63.8578805898 deg</td>
<td>joint 2 angle</td>
</tr>
<tr>
<td>m.q[4]</td>
<td>26.1951589399 deg</td>
<td>joint 4 angle</td>
</tr>
<tr>
<td>m.q[8]</td>
<td>-3.050640798 deg</td>
<td>joint 8 angle</td>
</tr>
<tr>
<td>m.q[9]</td>
<td>64.0131671931 deg</td>
<td>joint 9 angle</td>
</tr>
<tr>
<td>m.q[10]</td>
<td>8.6962449759 deg</td>
<td>joint 10 angle</td>
</tr>
<tr>
<td>m.tau[2]</td>
<td>1 Nm</td>
<td>joint 1 torque</td>
</tr>
</tbody>
</table>

First the forward kinematics test is performed in order to validate the model representation. The results are shown in Figure 6.10 and Table 6.19.

<table>
<thead>
<tr>
<th>Variable</th>
<th>SIMPACK</th>
<th>SDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance x</td>
<td>-7.9696e-09 m</td>
<td>-7.96957e-09 m</td>
</tr>
<tr>
<td>Distance y</td>
<td>1.2529e-10 m</td>
<td>1.25294e-10 m</td>
</tr>
<tr>
<td>Distance z</td>
<td>1.3558 m</td>
<td>1.35575 m</td>
</tr>
<tr>
<td>Angle 1</td>
<td>-180 deg</td>
<td>-180 deg</td>
</tr>
<tr>
<td>Angle 2</td>
<td>89.94696 deg</td>
<td>89.947 deg</td>
</tr>
<tr>
<td>Angle 3</td>
<td>-179.2232 deg</td>
<td>-179.223 deg</td>
</tr>
</tbody>
</table>

After the successful forward kinematics test with the new model, the tests of the dynamic functions are performed. For the beginning, only joint 2 is actuated by applying a torque. Unfortunately the forward dynamics function for fixed base robots with kinematic loops \( f_{\text{dyn\_CL\_fix}} \) throws an error because of occurring singularities. Nevertheless, the inverse dynamics function is tested by using the joint accelerations obtained with SIMPACK as the input of \( i_{\text{dyn\_CL\_fix}} \). The result is displayed in Figure 6.11 and Table 6.20. The calculated torque is very accurate and corresponds with the input value.

At the point of writing the forward dynamics function was still not working for every robotic model. The error is due to singularities by inverting matrices, that are no square matrices. A function for building the pseudo-inverse is implemented in the
6. Implementation, Tests and Simulations

(a) SIMPACK

(b) SpaceDynamics Library

Figure 6.10.: Results of Forward Kinematics Test

SpaceDynamics Library and used for example if $S$ is not a square matrix but it is not working for all systems.
6. Implementation, Tests and Simulations

Forward Dynamics for Closed-Loop Robot with Fixed Base

Diagram

<table>
<thead>
<tr>
<th>Joint</th>
<th>$\tau_{in}$ [Nm]</th>
<th>$\tau_{out}$ [Nm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-0.9999972663</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-0.0000000998</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.20.: Results of the Inverse Dynamics Test

Figure 6.11.: Inverse Dynamics Test for Closed Loop
7. Results, Conclusion and Future Work

7.1. Results

With the results obtained by the tests described in the preceding chapter, the appropriate operation of the SpaceDynamics Library for kinematic trees is proved. The results calculated by the functions of the SpaceDynamics Library correspond with the results obtained by SIMPACK. The kinematic tree dynamic functions are therefore valid functions and can be used to simulate fixed base, free floating or free flying robotic systems. The functions for the dynamic modeling of robots with kinematic loops are however not working for all robotic systems, as shown in the preceding chapter. One main problem occurs when there are more actuators than degrees of freedom in the system. Then the system is over-determined and it’s not always possible to calculate the results. The problem can be solved by defining the joints as actuated and passive joints, however, this is normally not known beforehand.

Investing more time into this research could solve the problem and would lead to an algorithm that is able to handle all kind of systems. However there were some time consuming issues during this research, which made the goal unachievable in the given period of time. The first issue was the theory because it is very complex. It took a lot of time, in order to understand the individual basics and put them together to solve other problems. In order to implement these algorithms, the understanding of all functions and the operation of the SpaceDynamics Library was necessary. In the SpaceDynamics Library almost no comments existed in the code, so this task was even more time consuming. Furthermore the array representations are inconsistent, because some arrays start at index 0, others at index 1. Probably the most time consuming part was to identify and remove errors of different functions, that were implemented by others. It is very hard to understand what a function is doing if there are no comments or documentation, and it is even harder to find errors in those functions. On the other hand, the smallest issue of the research was to learn working with the simulation environment SIMPACK. It was rather easy to build and simulate models, however it takes a certain amount of time to get to know the program.

Nevertheless, the thesis reaches almost all goals. With the knowledge of the theory and the understanding of the SpaceDynamics Library functions, only more time is needed to debug the code in order to validate the functions for robots with kinematic loops.
7. Results, Conclusion and Future Work

7.2. Conclusion

This report first described the motivation for the dynamic modeling of robots with kinematic loops. The dynamic modeling of robots with kinematic loops is for example needed for the simulation of the deorbiting of Envisat. Envisat is the largest Earth observing satellite of ESA and is not controllable anymore. Therefore, the deorbiting of Envisat would be an important step in actively reducing space debris with a service satellite. The report explained the terminology and definitions and gave a fundamental insight into the topic. Examples and applications of robotic systems with kinematic loops were given. Among the various applications, cooperative manipulators, humanoid robots, space robots and parallel robots were found to be the most important examples of robots with kinematic loops. The theory and equations of the dynamic modeling for robots with and without kinematic loops were explained. The theory and the equations also answer the research question of the thesis: How to compute the inverse and forward dynamics for a fixed base, free-flying or free-floating robotic system including kinematic and actuation redundancy with $N$ arms and $L$ closed loops. Two different methods of modeling kinematic loops were described in detail and compared with each other. The SpaceDyn library, which is used at the DLR German Aerospace Center was presented. It is a library of functions that calculates and simulates the kinematics and dynamics of a robot. In this scope the structure and the most important functions of the SpaceDynamics Library were described. After that, the implementation of the new algorithms for calculating the dynamics of robots with kinematic loops was shown. The tests and the results of the existing and newly implemented algorithms were presented and proved the SpaceDynamics Library to be valid to a certain extent. The goal of the research, to simulate the dynamics of a closed loop robotic system with the SpaceDynamics Library was reached for some robot models.

7.3. Future Work

Further work of this research includes solving the the forward dynamics problem for a robot with kinematic loops. Then the algorithms should be applied to a more realistic model, which is illustrated in Figure 7.1. This model is very close to the Envisat Case, where an object is grasped by two arms. It was already implemented in Listing A.7. As soon as the dynamics functions for kinematic loops are validated with this model, the next step is to control a similar robotic system. For the DLR it would be interesting to model and control the DEOS simulator with these algorithms. The DEOS simulator is simulating the grasping of a tumbling satellite with a manipulator of a service satellite for maintenance or deorbiting. In the simulator, a closed loop occurs over the simulator robots and the ground, as soon as the manipulator grasps the target satellite. Figure 7.2 shows the DEOS Simulator. On the left side is the tumbling target satellite, on the right side is the service satellite with the orange arm, which is pointing towards the target satellite. DEOS is the first mission that aims on reducing space debris actively.
7. Results, Conclusion and Future Work

Figure 7.1.: Realistic Test Model

Figure 7.2.: DEOS Simulator
A. Listings

A.1. Tests

The following tests are implemented in spacedyn/tests/spacedyn_test.cpp

Listing A.1: Forward Kinematics Test

```cpp
/**
 * Tests the forward kinematics for a model without closed loops
 * Computes the pose of the end-effector
 * @param output verbose output
 */
void test_forward_kinematics(bool output){
  if(output) std::cout << "Start forward kinematics test" << endl;
  // file
  string path("././.");
  string filename("LBR3_JtoJ_light.def");
  string modelfile = path + "/tests/" + filename;
  if(output) std::cout << "Model-File: " << modelfile << endl;
  // model
  MODEL m;
  // load model without output and without closed loops
  model(modelfile.c_str(), m, false, false);
  model_init(m);
  // joint angles
  m.q[2] = deg2pi(90.0);
  m.q[3] = deg2pi(90.0);
  m.q[4] = deg2pi(90.0);
  // calculate spatial notation
  calc_SP(m);
  // calculate position (m.Pos.e) and orientation (m.ORI.e) of the end-effector
  f_kine(e(m,1));
  // convert orientation rotation matrix to roll, pitch and yaw angles
  Vector3 RPY_EE;
  R2rpy(m.ORI.e[1], RPY_EE);
  // convert rad joint angles to deg
  double* angles = new double[m.LINKNUM];
  matrix_cpy(m.LINKNUM,1,m.q,angles);
  for (int i=0;i<m.LINKNUM;i++){
    angles[i]=pi2deg(angles[i]);
  }
  // print results
  if(output) {
    std::cout << "Joint-Angles:" << endl;
    matrix_print(m.LINKNUM-1,1,angles);
    std::cout << "Pose of end-effector:" << endl;
    std::cout << "Distance x: " << m.POS.e[1][0] << endl;
  }
}
```
A. Listings

Listing A.2: Dynamics for Fixed Base Kinematic Tree System

```cpp
/**
 * @param output verbose output
 */
void test_open_loop_fix(bool output) {
  if (output) std::cout << "Start open loop test for fixed Base" << endl;

  // file
  string path("././");
  string filename("LBR2_3JtoJ_light.def");
  string modelfile = path + "/tests/" + filename;
  if (output) std::cout << "Model-File: " << modelfile << endl;

  // model
  MODEL m;
  // load model without output and without closed loops
  model(modelfile.c_str(), m, false, false);
  model_init(m);

  // Gravity
  double *gravity;
  gravity = vector_get(6);
  vector_z(6, gravity);
  if (output) std::cout << "Open Loop Test: Forward Dynamics" << endl;

  // input torques
  m.tau[2] = 1;
  m.tau[4] = -1;
  m.q[2] = deg2pi(90.0);
  m.q[3] = deg2pi(90.0);
  m.q[4] = deg2pi(90.0);
  m.qd[2] = -1;

  // output variable
  double* qdd;
  qdd = vector_get(m.LINKNUM);
  vector_z(m.LINKNUM, qdd);

  // calculate spatial notation
  calc_SP(m);

  // calculate joint accelerations qdd
  f_dyn_fix(m.gravity, qdd);

  // print results
  if (output) {
    std::cout << "[input] Joint-Torques tau:" << endl;
    matrix_print(m.LINKNUM, 1, m.tau);
    std::cout << "[output] Joint-Accelerations qdd:" << endl;
  }
}
```
A. Listings

```cpp
matrix_print(m.LINKNUM,1,qdd);
}
if(output) std::cout << "Open Loop Test: Inverse Dynamics" << endl;
// input accelerations
m.qdd = qdd;
// output variable
double* tau;
tau = vector_get(m.LINKNUM);
vector_z(m.LINKNUM,tau);
// calculate joint torques tau
i_dyn_fix(m.gravity,tau);
// print results
if(output) {
  std::cout << "[input] Joint−Accelerations qdd:" << endl;
  matrix_print(m.LINKNUM,1,m.qdd);
  std::cout << "[output] Joint−Torques tau:" << endl;
  matrix_print(m.LINKNUM,1,tau);
}
m.destruct_ee();
m.destructor();
}

A.2. Functions

Listing A.3: Inverse Dynamics for Fixed Base Kinematic Tree System

```cpp
/**
 * Inverse Dynamics for a fixed base multi−body system
 * Calculates the required forces/torques to generate a given motion for a fixed base system.
 * The algorithm implements the following equation directly and is therefore rather slow. It has a computational complexity of O(n).
 * Equation:
 * tau = Hm * m.qdd + C
 * calc_SP must be called prior to this function.
 * The motion is given by
 * − MODEL::qd
 * − MODEL::qdd
 * and the kinematic situation:
 * − MODEL::Xup (thus indirectly MODEL::q)
 * Additional inputs are the given forces on the end−effectors:
 * − MODEL::Fe
 * − MODEL::Te
 * @param [in] m MODEL class
 * @param [in] Gravity vector (6 x 1)
 * @param [out] tau joint_torques (LINKNUM−1 x 1)
 * @author { Oliver Ruf}
 * @date 2014
 */

void i_dyn_fix_matrix_based(MODEL &m, double *Gravity, double *tau) {
  // Variable initialization
  double *HH, *Hm, *CC, *Cm,*tmp;
  HH = matrix_get( 6+m.LINKNUM–1, 6+m.LINKNUM–1 );
  Hm = matrix_get( m.LINKNUM–1, m.LINKNUM–1 );
```

63
A. Listings

```cpp
CC = vector.get(6+m.LINKNUM−1);
Cm = vector.get(m.LINKNUM−1);
tmp = vector.get(m.LINKNUM−1);

// calculate the inertia matrix
calc_hh(m,HH);
// extract the manipulator inertia matrix
matrix_ext_sub( 6+m.LINKNUM−1, 6+m.LINKNUM−1, 7, 6+m.LINKNUM−1, 7, 6+m.LINKNUM
−1, HH, Hm );

// calculate non-linear velocity dependent term
calc_C(m, Gravity ,CC);
// extract the manipulator term
matrix_ext_sub(6+m.LINKNUM−1,1,7,6+m.LINKNUM−1,1,1, CC,Cm );

// calculate inverse dynamics
matrix_mult(m.LINKNUM−1,m.LINKNUM−1,1,Hm,m. qdd , tmp);
matrix_add(m.LINKNUM−1,1,tmp ,Cm, tau );

delete [] HH;
delete [] Hm;
delete [] CC;
delete [] Cm;
delete [] tmp ;
}
```

Listing A.4: Forward Dynamics for Fixed Base Kinematic Tree System

```cpp
/**
 * Forward Dynamics for a fixed base multi-body system
 * Calculates the acceleration qdd for given forces/torques tau for a fixed base
 * system.
 * The algorithm implements the following equation directly and is therefore rather
 * slow. It has a computational complexity of O(n )
 * Equation:
 * qdd = Hm_inv (m.tau-C)
 * calc_SP must be called prior to this function.
 * The motion is given by
 * - MODEL::tau
 * and the kinematic situation:
 * - MODEL::Xup (thus indirectly MODEL::q)
 * Additional inputs are the given forces on the end-effectors:
 * - MODEL::Fe
 * - MODEL::Te
 * @param [in] m MODEL class
 * @param [in] Gravity vector (6 x 1)
 * @param [out] qdd joint accelerations (LINKNUM−1 x 1)
 * @author { Oliver Ruf }
 * @date 2014
 */

void f_dyn_fix_matrix_based(MODEL &m, double *Gravity, double *qdd) {
  // Variable initialization
  double *HH, *Hm, *Hm_inv, *CC, *Cm,*tmp;
  HH = matrix_get( 6+m.LINKNUM−1, 6+m.LINKNUM−1 );
  Hm = matrix_get( m.LINKNUM−1, m.LINKNUM−1 );
  Hm_inv = matrix_get( 6+m.LINKNUM−1, 6+m.LINKNUM−1 );
  CC = vector_get(6+m.LINKNUM−1);
  Cm = vector_get(m.LINKNUM−1);
tmp = vector_get(m.LINKNUM−1);

  // calculate the inertia matrix
calc_hh(m,HH);
  // extract the manipulator inertia matrix
```
A. Listings

```c
matrix_ext_sub( 6+m.LINKNUM−1, 6+m.LINKNUM−1, 7, 6+m.LINKNUM−1, 7, 6+m.LINKNUM−1, HH, Hm );
// invert Hm
matrix_inv(m.LINKNUM−1,Hm, Hm_inv);
// calculate non-linear velocity dependent term
calc_C(m,Gravity ,CC);
// extract the manipulator term
matrix_ext_sub(6+m.LINKNUM−1,7,6+m.LINKNUM−1,1, CC,Cm );
// calculate forward dynamics
matrix_sub(m.LINKNUM−1,1,m. tau ,Cm, tmp);
matrix_mult(m.LINKNUM−1,m.LINKNUM−1,1,Hm_inv , tmp , qdd );
```

```c
delete[] HH;
delete[] Hm;
delete[] Hm_inv;
delete[] CC;
delete[] Cm;
delete[] tmp;
```

A.3. Def-Files

Listing A.5: spacedyn/tests/LBR3_JtoJ_light.def

```plaintext
###Parameters_for_MODEL(_.DO_NOT_CHANGE_THIS_ORDER!!_.)
###LINK_NUMBER 8
###LINK_CONNECTIVITY
BB[ 0 1 2 3 4 5 6 ]
###Joint_Type_:_0=rotational_:_1=prismatic
J_type[ 0 0 0 0 0 0 0 ]
###EE
EE[ 0 0 0 0 0 0 1 ]
###Relative_Coordinate_For_Link:Roll,Pitch,Yaw,Angle_of_Each_Bodies{x−y−z}_[deg]
rpy1[ 0 0 0 ]
rpy2[ 90 0 0 ]
rpy3[ −90 0 0 ]
rpy4[ −90 0 0 ]
rpy5[ 90 0 0 ]
rpy6[ 90 0 0 ]
rpy7[ −90 0 0 ]
###Vector_Of_Link_Length_JtoC_0_0_is_always_[_0_0_0_]
CtoJ_0_1[ 0.885 −0.012 0.98 ]
JtoC_1_1[ 0.01698 0.14087 ]
CtoJ_1_2[ −0.01698 0.05913 ]
2.7082 2.71 2.5374 2.5053 1.3028 1.5686 0.1943
JtoC_2_2[ 0.1109 0.0141 ]
CtoJ_2_3[ 0.0891 −0.0141 ]
JtoC_3_3[ −0.01628 0.13379 ]
CtoJ_3_4[ 0.01628 0.06621 ]
JtoC_4_4[ −0.10538 0.01525 ]
CtoJ_4_5[ −0.09462 −0.01525 ]
JtoC_5_5[ 0.01566 0.06489 ]
CtoJ_5_6[ −0.01566 0.12511 ]
```
A. Listings

\[
\begin{align*}
\text{JtoC}_6\_6 & \begin{bmatrix} 0 & 0.00283 & -0.00228 \end{bmatrix} \\
\text{CtoJ}_6\_7 & \begin{bmatrix} 0 & -0.00283 & 0.00228 \end{bmatrix} \\
\text{JtoC}_7\_7 & \begin{bmatrix} 0 & 0 & 0.06031 \end{bmatrix} \\
\end{align*}
\]

### Vector To End-Effector
\[
\text{CtoE}_1 \begin{bmatrix} 0.0 & 0.0 & 0.07569 \end{bmatrix}
\]

### Relative Coordinate For End-Effector
\[
\text{rpyE}_1 \begin{bmatrix} 0.0 & 0.0 & 0.0 \end{bmatrix}
\]

### Mass_Parameter
\[
\text{mass} \begin{bmatrix} 724.10 & 2.7082 & 2.71 & 2.5374 & 2.5053 & 1.3028 & 1.5686 & 0.1943 \end{bmatrix}
\]

### Inertia Matrix
\[
\begin{bmatrix}
\text{I}_{11} & \text{I}_{12} & \text{I}_{13} \\
\text{I}_{12} & \text{I}_{22} & \text{I}_{23} \\
\text{I}_{13} & \text{I}_{23} & \text{I}_{33}
\end{bmatrix}
\]

#### Link0
\[
\begin{align*}
\text{I}_{11} &= 277.21 \\
\text{I}_{22} &= 410.07 \\
\text{I}_{33} &= 420.26 \\
\text{I}_{12} &= -11.33 \\
\text{I}_{13} &= 21.85 \\
\text{I}_{23} &= -3.8
\end{align*}
\]

#### Link1
\[
\begin{align*}
\text{I}_{11} &= -0.022632 \\
\text{I}_{22} &= -0.022793 \\
\text{I}_{33} &= 0.0049639 \\
\text{I}_{12} &= 0.0 \\
\text{I}_{13} &= 0.0 \\
\text{I}_{23} &= 0.0
\end{align*}
\]

#### Link2
\[
\begin{align*}
\text{I}_{11} &= 0.024444 \\
\text{I}_{22} &= 0.0052508 \\
\text{I}_{33} &= 0.023995 \\
\text{I}_{12} &= 0.0 \\
\text{I}_{13} &= 0.0 \\
\text{I}_{23} &= 0.0
\end{align*}
\]

#### Link3
\[
\begin{align*}
\text{I}_{11} &= -0.012993 \\
\text{I}_{22} &= -0.01326 \\
\text{I}_{33} &= 0.004697 \\
\text{I}_{12} &= 0.0 \\
\text{I}_{13} &= 0.0 \\
\text{I}_{23} &= 0.0
\end{align*}
\]

#### Link4
\[
\begin{align*}
\text{I}_{11} &= 0.023167 \\
\text{I}_{22} &= 0.0048331 \\
\text{I}_{33} &= 0.022751 \\
\text{I}_{12} &= 0.0 \\
\text{I}_{13} &= 0.0 \\
\text{I}_{23} &= 0.0
\end{align*}
\]

#### Link5
\[
\begin{align*}
\text{I}_{11} &= 0.023045 \\
\text{I}_{22} &= 0.022408 \\
\text{I}_{33} &= 0.0030151 \\
\text{I}_{12} &= 0.0 \\
\text{I}_{13} &= 0.0 \\
\text{I}_{23} &= 0.0
\end{align*}
\]

#### Link6
Listing A.6: spacedyn/tests/LBR3_JtoJ_light_CL.def

###Parameters_for_MODEL(__DO_NOT_CHANGE_THIS_ORDER!! __)
###LINK_NUMBER 8
###LINK_CONNECTIVITY
BB[ 0 1 2 3 4 5 6 ]
###Joint_Type_[0=rotational_1=prismatic]
J_type[ 0 0 0 0 0 0 0 ]
###EE
EE[ 0 0 0 0 0 1 ]
###L.flag
CL[ -1 -1 -1 -1 -1 0 ]
###Relative_Coordinate_For_Link: Roll_Pitch_Yaw_Angle_of_Each_Bodies {x-y-z}_[deg]
rpy1[ 0 0 0 ]
rpy2[ 90 0 0 ]
rpy3[ -90 0 0 ]
rpy4[ -90 0 0 ]
rpy5[ 90 0 0 ]
rpy6[ 90 0 0 ]
rpy7[ -90 0 0 ]
###Vector_Of_Link_Length_JtoC_0.0_is_always_[0.0 0.0]
CtoJ_0.1[ -0.885 -0.012 0.98 ]
JtoC_1.1[ 0 0.01698 0.14087 ]
CtoJ_1.2[ 0 -0.01698 0.05913 ]
JtoC_2.2[ 0.1109 0.0141 ]
CtoJ_2.3[ 0.0891 -0.0141 ]
JtoC_3.3[ 0 -0.01628 0.13379 ]
CtoJ_3.4[ 0.01628 0.06621 ]
JtoC_4.4[ 0 -0.10538 0.01525 ]
CtoJ_4.5[ 0 -0.09462 -0.01525 ]
JtoC_5.5[ 0 0.01566 0.06489 ]
CtoJ_5.6[ 0 -0.01566 0.12511 ]
JtoC_6.6[ 0 0.00283 -0.00228 ]
CtoJ_6.7[ 0 -0.00283 0.00228 ]
JtoC_7.7[ 0 0 0.06031 ]
###Vector_To_End–Effector
CtoE_1[ 0.0 0.0 0.07569 ]
### Relative Coordinate For End-Effector

```
rypE_1 [  0.0  0.0  0.0 ]
```

### Mass Parameter

```
mass [  724.10  2.7082  2.71  2.5374  2.5053  1.3028  1.5686  0.1943 ]
```

### Inertia Matrix

```
I11 = 277.21  I22 = 410.07  I33 = 420.26
I12 = −11.33  I13 = 21.85  I23 = −3.8
```

### Link 0

```
I11 = 277.21  I22 = 410.07  I33 = 420.26
I12 = −11.33  I13 = 21.85  I23 = −3.8
```

### Link 1

```
I11 = −0.020632  I22 = −0.022793  I33 = 0.0049639
I12 = 0.0  I13 = 0.0  I23 = 0.0
```

### Link 2

```
I11 = 0.024444  I22 = 0.0052508  I33 = 0.023995
I12 = 0.0  I13 = 0.0  I23 = 0.0
```

### Link 3

```
I11 = −0.012993  I22 = −0.01326  I33 = 0.004697
I12 = 0.0  I13 = 0.0  I23 = 0.0
```

### Link 4

```
I11 = 0.023167  I22 = 0.004231  I33 = 0.022751
I12 = 0.0  I13 = 0.0  I23 = 0.0
```

### Link 5

```
I11 = 0.023045  I22 = 0.022408  I33 = 0.0030151
I12 = 0.0  I13 = 0.0  I23 = 0.0
```

### Link 6

```
I11 = 0.0033636  I22 = 0.0029876  I33 = 0.0029705
I12 = 0.0  I13 = 0.0  I23 = 0.0
```

### Link 7

```
I11 = 7.93e−05
```
A. Listings

```
A22 = 7.83e-05
A33 = 0.0001203
A12 = 0.0
A13 = 0.0
A23 = 0.0

###EOF 777

Listing A.7: spacedyn/tests/LBR3_JtoJ_light_CL2.def

```###Parameters_for_MODEL (._DO_NOT_CHANGE_THIS_ORDER!!._)

```###LINK_NUMBER 15

```###LINK_CONNECTIVITY

BB[ 0 1 2 3 4 5 6 8 9 10 11 12 13]

```###Joint_Type. [ _0=rotational, _1=prismatic ]

J_type[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]

```###EE

EE[ 0 0 0 0 0 1 0 0 0 0 0 0 2 ]

```###Lflag

CL[ -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 7 ]

```###Relative_Coordinate For Link: Roll_Pitch_Yaw_Angle_of_Each_Bodies{ x-y-z } [deg]


```
  rpy1[ 0 0 0 ]
  rpy2[ 90 0 0 ]
  rpy3[ -90 0 0 ]
  rpy4[ -90 0 0 ]
  rpy5[ 90 0 0 ]
  rpy6[ 90 0 0 ]
  rpy7[ -90 0 0 ]
  rpy8[ 0 0 0 ]
  rpy9[ 90 0 0 ]
  rpy10[ -90 0 0 ]
  rpy11[ -90 0 0 ]
  rpy12[ 90 0 0 ]
  rpy13[ 90 0 0 ]
  rpy14[ -90 0 0 ]

```###Vector Of Link Length JtoC 0.0 is always [ _0_0_0 ].

```
CtoJ_0_1[ -0.885 -0.012 0.98 ]
  JtoC_1_1[ 0 0.01698 0.14087 ]
  CtoJ_1_2[ -0.01698 0.05913 ]
  JtoC_2_2[ 0.1109 0.0141 ]
  CtoJ_2_3[ 0.0891 -0.0141 ]
  JtoC_3_3[ -0.01628 0.13379 ]
  CtoJ_3_4[ 0.01628 0.06621 ]
  JtoC_4_4[ -0.10538 0.01525 ]
  CtoJ_4_5[ -0.09462 -0.01525 ]
  JtoC_5_5[ 0.01566 0.06489 ]
  CtoJ_5_6[ -0.01566 0.12511 ]
  JtoC_6_6[ 0.00283 -0.00228 ]
  CtoJ_6_7[ -0.00283 0.00228 ]
  JtoC_7_7[ 0 0 0.06031 ]
  CtoJ_0_8[ 0.885 -0.012 0.98 ]
  JtoC_8_8[ 0 0.01698 0.14087 ]
  CtoJ_8_9[ -0.01698 0.05913 ]

69
A. Listings

```plaintext

[[JtoC_9_9[ 0 0.1109 0.0141 ]
CtoJ_9_10[ 0 0.0891 -0.0141 ]
JtoC_10_10[ 0 -0.01628 0.13379 ]
CtoJ_10_11[ 0 0.01628 0.06621 ]
JtoC_11_11[ 0 -0.10538 0.01525 ]
CtoJ_11_12[ 0 -0.09462 -0.01525 ]
JtoC_12_12[ 0 0.01566 0.06489 ]
CtoJ_12_13[ 0 -0.01566 0.12511 ]
JtoC_13_13[ 0 0.00283 -0.00228 ]
CtoJ_13_14[ 0 -0.00283 0.00228 ]
JtoC_14_14[ 0 0 0.06031 ]

###Vector_To_End-Effector
CtoE_1[ 0.0 0.0 0.07569 ]
CtoE_2[ 0.0 0.0 0.07569 ]

###Relative_Coordinate_For_End-Effector
rpyE_1[ 0.0 0.0 0.0 ]
rpyE_2[ 0.0 -180.0 0.0 ]

###Mass_Parameter
mass[ 724.10 2.7082 2.71 2.5374 2.5053 1.3028 1.5686 0.1943 2.7082 2.71 2.5374 2.5053 1.3028 1.5686 0.1943 ]

###Inertia_Matrix
###\[
I_{11} = 277.21 
I_{12} = 410.07 
I_{13} = 420.26 
I_{12} = -11.33 
I_{13} = 21.85 
I_{23} = -3.8 
###\]

###Link0
I_{11} = -0.022632 
I_{12} = -0.022793 
I_{13} = 0.0049639 
I_{12} = 0.0 
I_{13} = 0.0 
I_{23} = 0.0 

###Link1
I_{11} = 0.024444 
I_{12} = 0.0052508 
I_{13} = 0.023995 
I_{12} = 0.0 
I_{13} = 0.0 
I_{23} = 0.0 

###Link2
I_{11} = -0.012993 
I_{12} = -0.01326 
I_{13} = 0.004697 
I_{12} = 0.0 
I_{13} = 0.0 
I_{23} = 0.0 

###Link3
I_{11} = -0.012993 
I_{12} = -0.01326 
I_{13} = 0.004697 
I_{12} = 0.0 
I_{13} = 0.0 
I_{23} = 0.0 

###Link4
I_{11} = 0.023167 
I_{12} = 0.0048331 
I_{13} = 0.022751 
```

70
A. Listings

<table>
<thead>
<tr>
<th>A. Listings</th>
</tr>
</thead>
<tbody>
<tr>
<td>112 = 0.0</td>
</tr>
<tr>
<td>113 = 0.0</td>
</tr>
<tr>
<td>123 = 0.0</td>
</tr>
<tr>
<td># # #Link5</td>
</tr>
<tr>
<td>I11 = 0.023045</td>
</tr>
<tr>
<td>I22 = 0.022408</td>
</tr>
<tr>
<td>I33 = 0.0030151</td>
</tr>
<tr>
<td>I12 = 0.0</td>
</tr>
<tr>
<td>I13 = 0.0</td>
</tr>
<tr>
<td>I23 = 0.0</td>
</tr>
<tr>
<td># # #Link6</td>
</tr>
<tr>
<td>I11 = 0.0033636</td>
</tr>
<tr>
<td>I22 = 0.0029876</td>
</tr>
<tr>
<td>I33 = 0.0029705</td>
</tr>
<tr>
<td>I12 = 0.0</td>
</tr>
<tr>
<td>I13 = 0.0</td>
</tr>
<tr>
<td>I23 = 0.0</td>
</tr>
<tr>
<td># # #Link7</td>
</tr>
<tr>
<td>I11 = 7.93e-05</td>
</tr>
<tr>
<td>I22 = 7.83e-05</td>
</tr>
<tr>
<td>I33 = 0.0001203</td>
</tr>
<tr>
<td>I12 = 0.0</td>
</tr>
<tr>
<td>I13 = 0.0</td>
</tr>
<tr>
<td>I23 = 0.0</td>
</tr>
<tr>
<td># # #Link8</td>
</tr>
<tr>
<td>I11 = -0.022632</td>
</tr>
<tr>
<td>I22 = -0.022793</td>
</tr>
<tr>
<td>I33 = 0.0049639</td>
</tr>
<tr>
<td>I12 = 0.0</td>
</tr>
<tr>
<td>I13 = 0.0</td>
</tr>
<tr>
<td>I23 = 0.0</td>
</tr>
<tr>
<td># # #Link9</td>
</tr>
<tr>
<td>I11 = 0.024444</td>
</tr>
<tr>
<td>I22 = 0.0052508</td>
</tr>
<tr>
<td>I33 = 0.023995</td>
</tr>
<tr>
<td>I12 = 0.0</td>
</tr>
<tr>
<td>I13 = 0.0</td>
</tr>
<tr>
<td>I23 = 0.0</td>
</tr>
<tr>
<td># # #Link10</td>
</tr>
<tr>
<td>I11 = -0.012993</td>
</tr>
<tr>
<td>I22 = -0.01326</td>
</tr>
<tr>
<td>I33 = 0.004697</td>
</tr>
<tr>
<td>I12 = 0.0</td>
</tr>
<tr>
<td>I13 = 0.0</td>
</tr>
<tr>
<td>I23 = 0.0</td>
</tr>
<tr>
<td># # #Link11</td>
</tr>
<tr>
<td>I11 = 0.023167</td>
</tr>
<tr>
<td>I22 = 0.0048331</td>
</tr>
<tr>
<td>I33 = 0.022751</td>
</tr>
<tr>
<td>I12 = 0.0</td>
</tr>
<tr>
<td>I13 = 0.0</td>
</tr>
<tr>
<td>I23 = 0.0</td>
</tr>
<tr>
<td># # #Link12</td>
</tr>
<tr>
<td>I11 = 0.023045</td>
</tr>
<tr>
<td>I22 = 0.022408</td>
</tr>
<tr>
<td>I33 = 0.0030151</td>
</tr>
<tr>
<td>I12 = 0.0</td>
</tr>
<tr>
<td>I13 = 0.0</td>
</tr>
<tr>
<td>I23 = 0.0</td>
</tr>
<tr>
<td># # #Link13</td>
</tr>
<tr>
<td>I11 = 0.0033636</td>
</tr>
</tbody>
</table>
### A. Listings

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>$I_{22}$</td>
<td>$0.0029876$</td>
<td>$I_{13}$</td>
<td>$0.0029705$</td>
<td>$I_{12}$</td>
</tr>
<tr>
<td>202</td>
<td>$I_{23}$</td>
<td>$0.0$</td>
<td>$I_{12}$</td>
<td>$0.0$</td>
<td>$I_{13}$</td>
</tr>
<tr>
<td>208</td>
<td>$I_{22}$</td>
<td>$7.83 \times 10^{-5}$</td>
<td>$I_{33}$</td>
<td>$0.0001203$</td>
<td>$I_{12}$</td>
</tr>
<tr>
<td>214</td>
<td>$I_{11}$</td>
<td>$7.83 \times 10^{-5}$</td>
<td>$I_{33}$</td>
<td>$0.0001203$</td>
<td>$I_{12}$</td>
</tr>
<tr>
<td>214</td>
<td>$I_{11}$</td>
<td>$7.83 \times 10^{-5}$</td>
<td>$I_{33}$</td>
<td>$0.0001203$</td>
<td>$I_{12}$</td>
</tr>
<tr>
<td>214</td>
<td>$I_{11}$</td>
<td>$7.83 \times 10^{-5}$</td>
<td>$I_{33}$</td>
<td>$0.0001203$</td>
<td>$I_{12}$</td>
</tr>
<tr>
<td>214</td>
<td>$I_{11}$</td>
<td>$7.83 \times 10^{-5}$</td>
<td>$I_{33}$</td>
<td>$0.0001203$</td>
<td>$I_{12}$</td>
</tr>
<tr>
<td>214</td>
<td>$I_{11}$</td>
<td>$7.83 \times 10^{-5}$</td>
<td>$I_{33}$</td>
<td>$0.0001203$</td>
<td>$I_{12}$</td>
</tr>
<tr>
<td>214</td>
<td>$I_{11}$</td>
<td>$7.83 \times 10^{-5}$</td>
<td>$I_{33}$</td>
<td>$0.0001203$</td>
<td>$I_{12}$</td>
</tr>
<tr>
<td>214</td>
<td>$I_{11}$</td>
<td>$7.83 \times 10^{-5}$</td>
<td>$I_{33}$</td>
<td>$0.0001203$</td>
<td>$I_{12}$</td>
</tr>
<tr>
<td>214</td>
<td>$I_{11}$</td>
<td>$7.83 \times 10^{-5}$</td>
<td>$I_{33}$</td>
<td>$0.0001203$</td>
<td>$I_{12}$</td>
</tr>
</tbody>
</table>
Bibliography


