

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

Bachelor Thesis Nr. 2378

Modelling and Analysis of Nonlinear Multibody Dynamic Systems

Alexander Gutenkunst

Course of Study:	Engineering Cybernetics
Examiner:	Prof. Dr. rer. nat. habil. Paul Levi
Supervisor:	Dipl.-Ing. Eugen Meister
Commenced:	10.12.2011
Completed:	09.05.2012
CR-Classification:	I.2.2, I.2.9, I.2.11

Acknowledgements

I would like to thank Prof. Dr. rer. nat. habil. Paul Levi for making the studies on modular robotics, including this thesis, at this university possible.

I also would like to express my gratitude to Dipl.-Ing. Eugen Meister for his guidance and encouragement as supervisor.

Abstract

Modular self-reconfigurable robots are characterised by a high versatility. While most robots are designed for a special purpose these robots are designed to be multi-talented and adaptive.

This thesis uses a modern framework for the automatic model generation of these modular self-reconfigurable robots. Based on a two-step Newton-Euler approach in combination with elements of screw-theory, this framework provides an elegant way to automatically calculate the equations of motion in a closed form.

To simulate the kinematics and dynamics the framework has been implemented in MATLAB[®].

To verify both the implementation and modelling, the equations of motion of two robot structures have been calculated using the framework and compared to the equations obtained by a Lagrangian approach.

Finally it is shown that the linearisation and decoupling of the systems concerned in this thesis can be done easily using feedback linearisation.

For completion two ways to control these linearised systems are demonstrated.

Abstract - deutsch

Modulare selbstrekonfigurierende Robotersysteme zeichnen sich durch ihre hohe Vielseitigkeit aus. Während die meisten Robotersysteme für einen bestimmten Zweck entwickelt werden, sind diese Roboter multitalentiert und anpassungsfähig.

In dieser Arbeit wird ein modernes Framework zur automatische Modellerstellung von modularen selbstrekonfigurierende Robotersysteme genutzt. Basierend auf einem Zwei-Schritt Newton-Euler Ansatz in Kombination mit Elementen der Schraubentheorie, bietet dieses Framework einen eleganten Weg die Bewegungsgleichungen in einer geschlossenen Form zu bestimmen.

Um die Kinematik und Dynamik der berechneten System zu bestimmen wurde das Framework in MATLAB[®] implementiert. Diese Implementierung wird in dieser Arbeit überprüft.

Schlussendlich wird Feedback Linearisierung und lineare Regler zur Regelung des Systems eingesetzt.

Contents

List of Figures	II
List of Tables	IV
1. Introduction	1
1.1. Motivation	2
1.2. State of the art	2
1.3. Structure of this thesis	4
2. Theoretical Background	5
2.1. The robot module	5
2.2. Link assemblies	7
2.2.1. Joint modules	7
2.2.2. Link assemblies and dyads	8
2.3. Rigid body motion	9
2.3.1. The skew symmetric matrix representation	9
2.3.2. Homogeneous coordinates	9
2.3.3. The two Lie groups $SO(3)$ and $SE(3)$	10
2.3.4. Rigid body motion using exponential coordinates	11
2.3.5. Twist	13
2.3.6. Wrench	14
2.3.7. The operators Ad_T and adv_V	14
3. Geometric model generation	16
3.1. The Assembly Incidence Matrix(AIM)	16
3.2. The Accessibility Matrix	17
3.3. Forward Kinematics	18
3.3.1. Recursive forward kinematics	19
3.3.2. Calculating the initial transformations	19
3.3.3. Recursive forward kinematics using lists	20
3.4. Dynamics	24
3.4.1. Newton-Euler equations of a link assembly	24
3.4.2. The two-step approach	26

4. Implementation	30
4.1. Code example	30
4.2. Comparison of the numerical and the symbolical calculation	34
4.2.1. Symbolic calculation	34
4.2.2. Numeric calculation	34
4.2.3. Comparison	35
5. Verification	36
5.1. Verification using the double pendulum	36
5.1.1. Modelling	36
5.1.2. Simulation	39
5.1.3. Analytic verification	39
5.2. Verification using the crane model	43
5.2.1. Modelling	43
5.2.2. Simulation	45
5.3. Verification of unbounded robot structures using Newton's first law	47
6. Controller design	49
6.1. Feedback linearisation	49
6.1.1. Theory	49
6.1.2. The zero dynamics	53
6.2. PID-controller	54
6.3. Pole placement	56
6.3.1. Example	57
7. Conclusion	59
A. Code	60
A.1. CreatePendulumAIM.m	60
A.2. GetTijFromPair.m	61
A.3. GetRijFromPair.m	62
A.4. CreateTCalcWrapper.m	63
A.5. CreateTCalcOrder.m	64
Literatur	A

List of Figures

1.1. Different robot projects	1
1.2. Different modular robot projects	3
2.1. Degree of freedom of the robot modules	6
2.2. Sides of the robot modules	6
2.3. Drawing of a backbone module	6
2.4. Two revolute joints	7
2.5. Prismatic joint	8
2.6. Link assembly accordingly to [4]	8
2.7. Revolute and prismatic joint motion	11
3.1. Tree-Structured robot and the corresponding graph	17
3.2. AIM	17
3.3. The <code>TreeRobotKinematics</code> algorithm	19
3.4. Initial translation	20
3.5. Direct and indirect transformation list	21
3.6. Traversing algorithm	22
3.7. Link assembly and applied wrench	24
3.8. Two step approach	26
4.1. MATLAB-ODE	34
4.2. Comparison of the symbolic and numeric approach	35
5.1. Double pendulum	36
5.2. Double pendulum structure	37
5.3. Coordinate frames used to model the double pendulum	37
5.4. Verification plots	39
5.5. The different angle representations	39
5.6. Crane model	43
5.7. Coordinate frames of the crane model	44
5.8. Simulation of the crane model	46
5.9. Moving unbounded robot structure	48
6.1. Feedback-Linearisation	53
6.2. SIMULINK-Diagram	54
6.3. The subsystems	55

List of Figures

6.4. Step response of the PID-controlled system	55
6.5. Control via pole placement	58
6.6. Step response of the system controlled with pole placement	58

List of Tables

3.1. Initial rotations	20
----------------------------------	----

1. Introduction

The ongoing development of tools, electronics and software in the last decades enabled the construction of more and more sophisticated robots.

While robots today are mostly used in the industry, providing assistance and manpower, we can expect the world of the future to be characterised by robots and autonomous systems. Researchers all around the globe are currently working on the development of this robots.

Some of todays robots are constructed especially for an industrial purpose while others are inspired by the human body, or are shaped like animals such as snake or spider as shown in Fig.1.1.

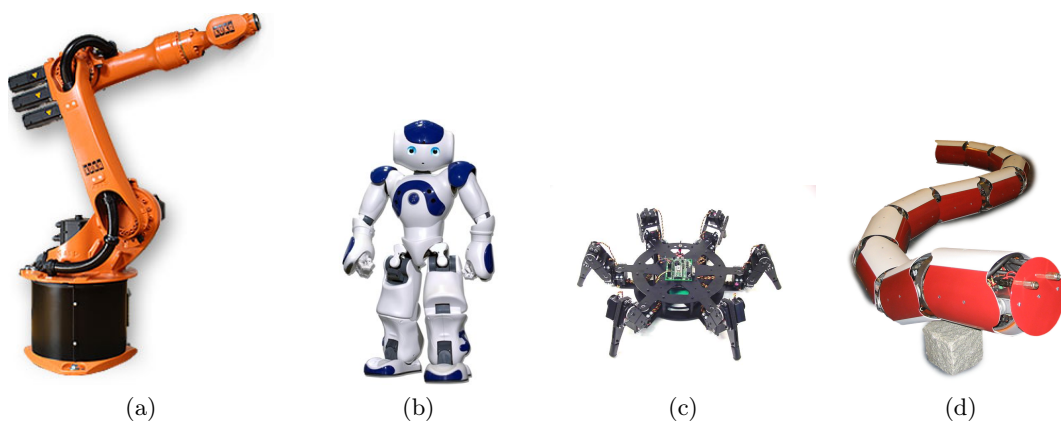


Figure 1.1.: Different robot projects: (a) Kuka KR 16 S [9]; (b) Aldebaran Nao [1]; (c) Lynxmotion BH3-R hexapod[10]; (d) Anna Konda [23]

What all these different robot types have in common is that they are all designed for a special purpose and environment. In our daily life we often use specialised tools such as a screwdriver, but if we want to be prepared for different unpredictable situations we carry a swiss-tool or leatherman respectively. This concept of flexibility found its way into robotics in form of modular robots.

While most of todays robots a rather constant and maintain their shape and function, these multi-talented robots are equipped for many situations.

1.1. Motivation

Consisting of multiple modules which can connect to each other, these robots gain high flexibility and adaptability.

While most robot systems deployed by the industry are designed to perform one task such as sorting or welding with high efficiency, self-reconfigurable modular robots are equipped with the ability to adapt their structure to a wide variety of tasks and environments. The key to this structural flexibility lies in the modularity of these robots. With every structural change not only the shape itself, but also the motion pattern changes.

Imagine a self-reconfigurable modular robot on a search and rescue mission in a highly inhomogeneous environment. Being able to change its structure the robot would be able to move as fast as possible even in a changing environment. In a plain field the robot could change its configuration to a loop to move fast. In debris field the robot could reconfigure itself into the shape of a spider to easily crawl over obstacles. In a snake-like structure the robot could easily enter cavities through small holes in order to search for buried persons.

Versatility is not the only advantage of self-reconfigurable robots. Since the modules are mostly homogenous or only exist in slightly differing types, malfunctioning or damaged modules within the robot structure can be replaced easily. This robustness is a big advantage in environments where spare parts or maintenance can not be supplied(e.g. extraterrestrial or deep-sea environments).

Because the structure of the robot is not determined, the kinematics and dynamics of self-reconfigurable robots are much more complex than the ones of robots with a fixed structure. The kinematics and dynamics of a robot with a fixed structure only have to be calculated once, what can be actually done 'by hand' in advance.

In the contrary to a modular self-reconfigurable robot the kinematics and dynamics have to be calculated with every structural change.

Since it is unpredictable what structures the robot system is going to adapt, all characteristics of the system especially the equations of motion, have to be calculated online without human interference.

The indeterminacy of the robot structure also is a great challenge for the dynamic calculation of control algorithms. As with the structure the system itself varies the control algorithms have to be adapted as well.

The development of this advanced control mechanisms is and will be a great challenge of the future.

1.2. State of the art

In the ongoing scientific process various projects concerning self-reconfigurable robots have been established since around the early 90's.

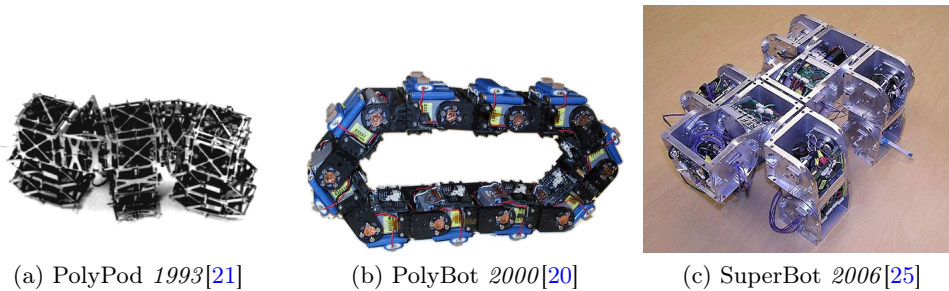


Figure 1.2.: Different modular robot projects

One of the first modular robot projects was the PolyPod, shown in Fig.1.2a, developed by Mark Yim in 1993. Despite the lack of self-reconfiguration Yim showed that the use of multiple connected modules enables the robot to move in multiple stable locomotion modes.

In 2000 the PolyBot, shown in Fig.1.2b, a descendent of the PolyPod with the first attempts towards self-reconfiguration was introduced. This project among others showed that one key problem of modular robots is the actual connecting process. The establishment of a connection between two modules, wether mechanically or using magnets, is still one of the main problems in modular robotics and part of the ongoing research.

In 2006 Shen et al. presented SuperBot, shown in Fig.1.2c, a modular self-reconfigurable robot designed for NASA space exploration programs. This robot was especially designed to meet the demands of a space mission. The interested reader can find a more detailed description and history of modular robots in [24].

The robot modules concerned in this thesis are part of both the projects “SYMBRION” and “REPLICATOR” funded since 2008 by the European Commission[26]. The main goal of these projects is to develop new methods for the handling of symbiotic multi-robot organisms. Key aspects of this projects are self-programming and self-assembling as well as the transfer of evolutionary paradigms onto robot systems.

1.3. Structure of this thesis

This thesis is structured as follows.

Chapter 1 is a brief introduction into general robotics and describes the characteristics of modular self-reconfigurable robots. It describes the motivation for this thesis and gives a brief history on modular robot projects.

Chapter 2 describes the theoretical background, including a short introduction into rigid body motion and screw theory.

Chapter 3 concerns the framework introduced by Chen et. al in [4] for the dynamic model creation of modular robots.

Chapter 4 is about the implementation of Chen's framework in MATLAB[®]. It includes a comparison of numerical and symbolical implementation of the framework regarding calculation time.

Chapter 5 shows a way to verify the correct application and implementation by simulating two robot examples.

Chapter 6 finally shows an elegant way to control robots created with this framework.

2. Theoretical Background

The two essential pillars of the theoretical background of this thesis is linear algebra and the theory of screws. The combination of these two methods forms a powerful instrument for the handling of multi-body-systems.

The theory of Louis Poinot and Michel Chasles developed in the early 1800s was that a rigid body can be moved from one position to another by rotating about one line in combination with a translation parallel to the same line. Such movement is called a *screw motion* inspired by the thread of a screw.

Based on the theory of Poinot and Chasles, Robert S. Ball further developed this theory. 1900 he published *A Treatise on the Theory of Screws*, see [2], providing the full theory of screws. Ball introduced the so called *twist*, the infinitesimal version of a screw motion describing the velocity, which will be precised in chapter 2.3.5. He also presented, based on the theorem of Poinot, the *wrench* reducing all external forces and torque to a a single tuple consisting of a force and a torque.

2.1. The robot module

The modules considered in this thesis originate from the projects SYMBRION and REPLICATOR founded by the European Commission. In Fig.2.1 both modules and their degree of freedom(DOF) are shown. Both modules are able to ‘bend’ from within their center with a 180° limit. The modules are cubical-shaped and the sides are numbered according to the numbers on a gaming dice as shown in Fig.2.2. According to the movement no connection on the sides 1 and 6 is possible. In this thesis often drawings as shown in Fig.2.3 are used to illustrated the robot modules in certain robot structure.

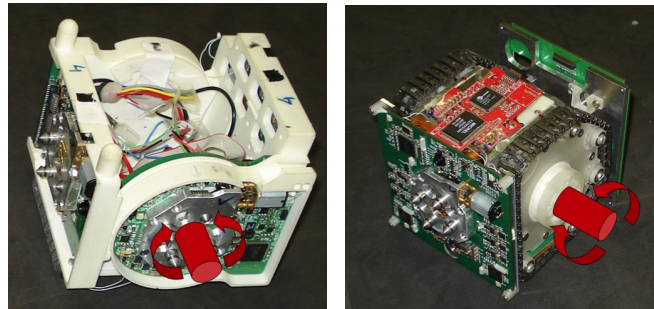


Figure 2.1.: Degree of freedom of the robot modules

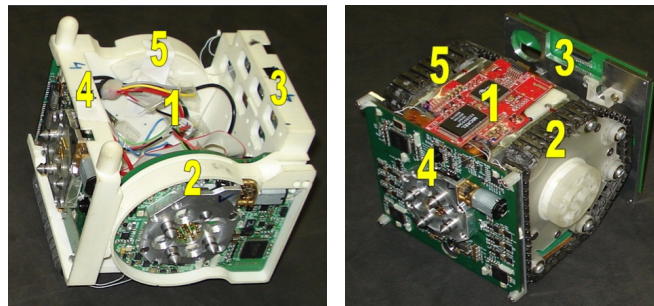


Figure 2.2.: Sides of the robot modules

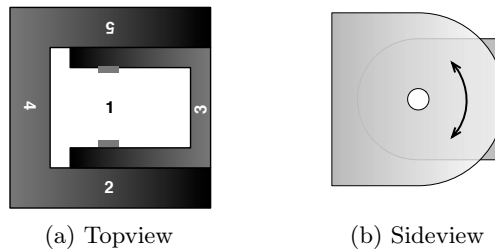


Figure 2.3.: Drawing of a backbone module

2.2. Link assemblies

To describe the robot structure Chen et al. [4] introduced a way to represent most robot structures. Chen differentiates between so called *link modules* and *joint modules*. Link modules have multiple ports that can connect to a other link module through a joint module. These joint modules enable rotational or prismatic motion between the connected link modules. At first glance this way seems not to be directly adaptable to the robot modules backbone and scout used in this thesis since the motion actually happen ‘within’ to robot module itself and not between two modules. Though in this thesis it will be shown that the framework develop by Chen et al. [4] can be used directly, and that there is no need for adjustments in any manner.

2.2.1. Joint modules

As described before a joint is a connection between two modules allowing both to move with respect to each other. Mostly joints provide only one degree of freedom as will the joints used in this thesis. Joints can be divided into 3 groups: revolute, prismatic and virtual joints.

Revolute joints

A revolute joint connecting two modules allows a rotary movement as shown in Fig. 2.4. In general the axis of rotation is arbitrary.

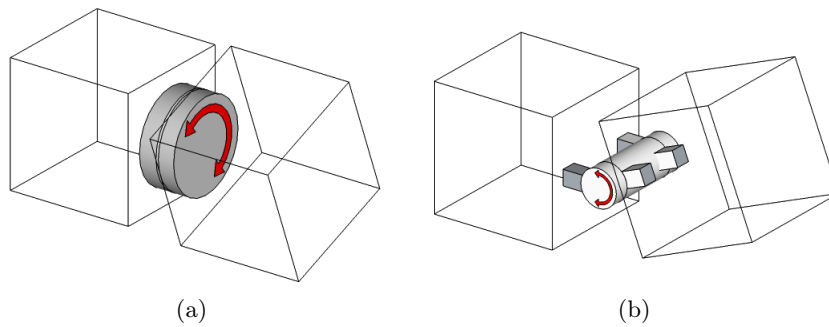


Figure 2.4.: Two revolute joints

Prismatic Joints

Modules connected with prismatic joints as shown in Fig.2.5 allow transitional movement. These joints are not part of this thesis and neither do exist in the modules of SYMBRION or REPLICATOR.

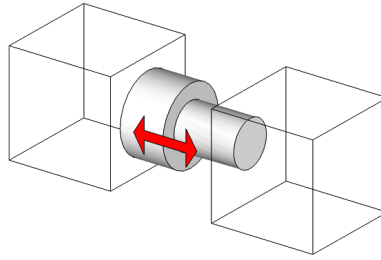


Figure 2.5.: Prismatic joint

Virtual Joints

Virtual joints give the ability to model the kinematics of robots that are not bound to any fixed module. While these joints are mostly assumed to be massless there are cases where this does not hold. Such as case is shown in chapter 5.2. Note that these definition for virtual joints differs from the definition used in [5]. The DOF obtained by a virtual joint can be rotary or translational.

2.2.2. Link assemblies and dyads

A link assembly as defined in [4] is a link module connected to a joint. In Fig.2.6 the link assembly j , consisting of the link module v_j and the joint e_j is shown. There are two essential frames defined with a link assembly. One is the module frame j located at the gravity centre of the link module j . The second frame is the mass frame j^* located at the mass centre of the link assembly.

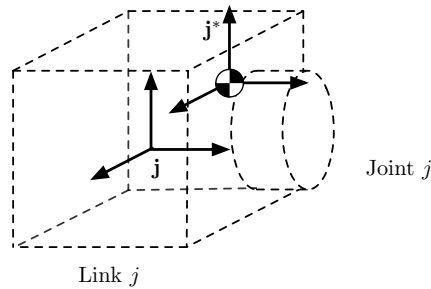


Figure 2.6.: Link assembly accordingly to [4]

Along with this representation the transformation

$$T_{j^*j} = \begin{bmatrix} R_{j^*j} & p_{j^*j} \\ 0 & 1 \end{bmatrix} \quad (2.1)$$

where $R_{j^*j} \in SO(3)$ and $p_{j^*j} \in \mathbb{R}^3$ are the rotation and translation of the frame j^* with respect to frame j .

2.3. Rigid body motion

A robot motion is always a motion of multiple connected rigid bodies with respect to each other. Here the theory of rigid body motion as described in [16] is presented.

A motion is called a rigid body motion if:

1. The distance between all points is constant.
2. All vectors between points of the body preserve the cross product.

The first property results directly from the rigidity. Thus at first glance the first property seems to be sufficient one has to be aware that the first property would also allow the mirroring of objects. Because this is physically impossible the second property has to hold.

2.3.1. The skew symmetric matrix representation

In general the cross product between two vectors $a, b \in \mathbb{R}^3$ is defined as

$$a \times b = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix}. \quad (2.2)$$

By defining the so called *skew symmetric representation* of a as

$$\hat{a} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \quad (2.3)$$

the cross product can be rewritten as

$$a \times b = \hat{a}b \quad (2.4)$$

2.3.2. Homogeneous coordinates

A point $q = [q_1 \ q_2 \ q_3]^T$ is transformed by a rigid body motion $g = (p, R) \in SE(3)$ using the rotation matrix $R \in SO(3)$ and a translation vector $p \in \mathbb{R}^3$ by

$$q^* = Rq + p \quad (2.5)$$

The equation (2.5) can be rewritten using a 4×4 matrix

$$\bar{q}^* = \begin{bmatrix} q^* \\ 1 \end{bmatrix} = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \begin{bmatrix} q \\ 1 \end{bmatrix} = \bar{g} \cdot \bar{q} \quad (2.6)$$

By using the so called *homogeneous representation* \bar{g} of the transformation g

$$\bar{g} = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \quad (2.7)$$

and the *homogeneous coordinates*

$$\bar{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ 1 \end{bmatrix} \quad (2.8)$$

which can be easily obtained from $q \in \mathbb{R}^3$ by appending a 1 thus increasing the dimension to $\bar{q} \in \mathbb{R}^4$. The origin in homogeneous coordinates is

$$\bar{O} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad (2.9)$$

Since vectors are the difference between two points the additional 1s are subtracted and the homogeneous representation of a vector is

$$\bar{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix} \quad (2.10)$$

While the last row of \bar{g} seems to be a useless overhead it can be used to perform scaling and perspective projections. Though this is not used in this thesis, it is just mentioned for the sake of completeness.

2.3.3. The two Lie groups $SO(3)$ and $SE(3)$

Both $SE(3)$ and $SO(3)$ are two groups based on Lie-Algebra that are of high importance for this concept of calculating robot kinematics.

The special orthogonal group $SO(3)$

Rotational matrices in \mathbb{R}^3 are constructed from three orthonormal columns which are arranged in right hand order. Matrices with this properties form the special orthogonal group $SO(3)$ and are defined by the space of rotation matrices in $\mathbb{R}^{3 \times 3}$ with

$$SO(3) = \{R \in \mathbb{R}^{3 \times 3} : RR^T = I, \det R = +1\}$$

A more detailed explanation can be found in [16, p.23f]

The special Euclidean group $SE(3)$

Elements of the $SE(3)$ group are rigid transformations in \mathbb{R}^3 . The mapping $g : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ with $g(x) = Rx + p$ and $R \in SO(3)$, $p \in \mathbb{R}^3$ defines the group $SE(3)$ as:

$$SE(3) = \{(p, R) : p \in \mathbb{R}^3, R \in SO(3)\} \quad (2.11)$$

2.3.4. Rigid body motion using exponential coordinates

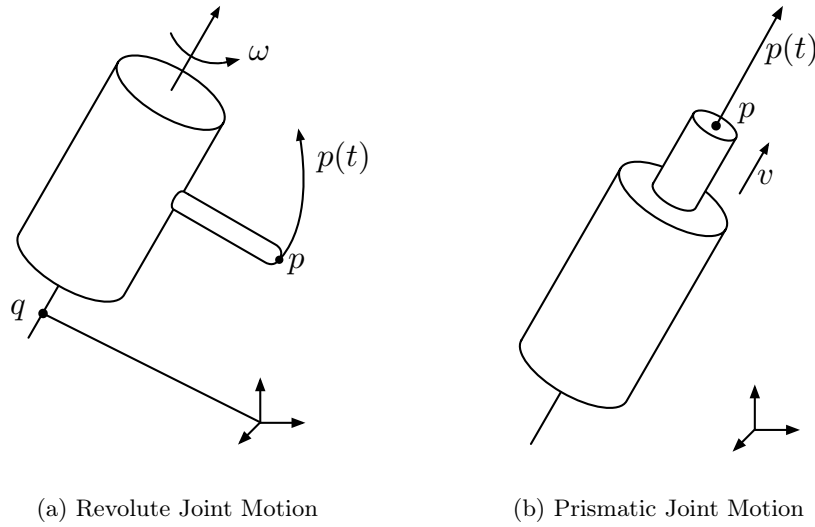


Figure 2.7.: Joint motions accordingly to [16, Fig.2.5]

In Fig.2.7a one link of a robot is shown. The axis $\omega \in \mathbb{R}^3$ is the axis of rotation and $q \in \mathbb{R}^3$ is some arbitrary point on the same axis. The point p may be the point at the end of the tip.

Both ω and $v = -\omega \times q$ form the group of $se(3)$ defined as

$$se(3) = \{(v, \hat{\omega}) : v \in \mathbb{R}^3, \hat{\omega} \in so(3)\} \quad (2.12)$$

The velocity of the point at the end of the tip can be described as

$$\dot{p}(t) = \omega \times (p(t) - q) \quad (2.13)$$

with $v = -\omega \times q$:

$$\dot{p}(t) = \omega \times p(t) + v \quad (2.14)$$

By defining $\hat{\xi} \in se(3)$ as

$$\hat{\xi} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} \quad (2.15)$$

2. Theoretical Background

the homogenous coordinate form of (2.14) is obtained:

$$\begin{bmatrix} \dot{p} \\ 0 \end{bmatrix} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ 1 \end{bmatrix} = \hat{\xi} \begin{bmatrix} p \\ 1 \end{bmatrix} \quad (2.16)$$

With $\bar{p} = [p \ 1]^T$ the differential equation is given by

$$\dot{\bar{p}} = \hat{\xi} \bar{p} \quad (2.17)$$

The solution to this equation is

$$\bar{p}(t) = e^{\hat{\xi}t} \bar{p}(0) \quad (2.18)$$

with $p(0)$ being the initial position.

The exponential $\hat{\xi}t$ is a 4×4 matrix und thus defined by

$$e^{\hat{\xi}t} = I + \hat{\xi}t + \frac{(\hat{\xi}t)^2}{2!} + \frac{(\hat{\xi}t)^3}{3!} + \dots \quad (2.19)$$

by assuming that $|\omega| = 1$ (2.19) can be rewritten to:

$$e^{\hat{\xi}q_r} = I + \hat{\xi}q_r + \frac{(\hat{\xi}q_r)^2}{2!} + \frac{(\hat{\xi}q_r)^3}{3!} + \dots \quad (2.20)$$

where q_r is the angle of rotation.

The motion of a prismatic joint, as shown in Fig. 2.7b, is defined analog by

$$\dot{p}(t) = v \quad (2.21)$$

with

$$\hat{\xi} = \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix} \quad (2.22)$$

(2.21) can be rewritten to

$$\begin{bmatrix} \dot{p} \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ 1 \end{bmatrix} = \hat{\xi} \begin{bmatrix} p \\ 1 \end{bmatrix} \quad (2.23)$$

Using homogenous coordinates it holds that

$$\dot{\bar{p}} = \hat{\xi} \bar{p} \quad (2.24)$$

with $\bar{p} = [p \ 1]^T$.

The solution to this differential equation is given by

$$\bar{p}(t) = e^{\hat{\xi}t} \bar{p}(0) \quad (2.25)$$

where $p(0)$ is the initial position.

Accordingly to the definition of a matrix exponential it holds that

$$e^{\hat{\xi}t} = I + \hat{\xi}t + \frac{(\hat{\xi}t)^2}{2!} + \frac{(\hat{\xi}t)^3}{3!} + \dots \quad (2.26)$$

Assuming that $\|v\| = 1$ (2.26) can be rewritten to:

$$e^{\hat{\xi}q_t} = I + \hat{\xi}q_t + \frac{(\hat{\xi}q_t)^2}{2!} + \frac{(\hat{\xi}q_t)^3}{3!} + \dots \quad (2.27)$$

where q_t is the translated distance.

Rodrigues' formula

Both (2.20) and (2.27) can be transformed from the series representation into a closed form by using the Rodrigue' formula defined by

$$e^{\hat{\xi}q} = I + \hat{\xi} \sin q + \hat{\xi}^2(1 - \cos q) \quad (2.28)$$

2.3.5. Twist

As mentioned before, elements of the $se(3)$ group are called twists. A twists consists of linear and angular velocities and is the infinitesimal version of a screw motion.

$$s_i = \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (2.29)$$

By using revolute joints which do not allow translational movement the twist coordinates are

$$s_i = \begin{bmatrix} \omega_i \\ 0 \end{bmatrix} \quad (2.30)$$

Accordingly a prismatic joint is defined by a twist of the form

$$s_i = \begin{bmatrix} 0 \\ v_i \end{bmatrix} \quad (2.31)$$

Along with the twists two operators are defined.

The \vee (vee) operator which transforms homogeneous coordinates of a twist (see eq.(2.15)) into a vector in \mathbb{R}^6 . Such vectors with $\xi := (v, w)$ are called the twist coordinates of $\hat{\xi}$.

$$\begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}^\vee = \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (2.32)$$

The \wedge (wedge) operator As being defined as the inverse operator the \wedge (wedge) performs the reverse calculation by transforming the twist coordinates into a matrix of $se(3)$.

$$\begin{bmatrix} v \\ \omega \end{bmatrix}^\wedge = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} \quad (2.33)$$

2.3.6. Wrench

The theorem of poinsot states that all forces and torques applied on a rigid body can be reduced to a single force along one line in combination with a single torque. Such tuples are called *wrenches* and are represented in vectors of \mathbb{R}^6 .

$$F = \begin{bmatrix} f \\ \tau \end{bmatrix} \quad (2.34)$$

These wrenches consists of the linear force f and a rotational torque τ .

2.3.7. The operators Ad_T and ad_V

The adjoint representation Ad_T is the adjoint mapping $Ad_T : SE(3) \rightarrow SE(3)$ with

$$T = \begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix} \in SE(3) \quad (2.35)$$

with $R \in SO(3)$ and $p \in \mathbb{R}^3$ defined in [Murray1994] as

$$Ad_T = \begin{bmatrix} R & \hat{p}R \\ 0 & R \end{bmatrix} \in SE(3) \quad (2.36)$$

It holds that the transpose Ad_T^T of Ad_T is given by

$$Ad_T^T = (Ad_T)^T = \begin{bmatrix} R & \hat{p}R \\ 0 & R \end{bmatrix}^T = \begin{bmatrix} R^T & 0 \\ -R^T \hat{p} & R^T \end{bmatrix} \quad (2.37)$$

which can be verified by direct calculation.

The operator ad_V is defined as the adjoint mapping $ad_V : se(3) \rightarrow se(3)$ on the twist $V := (v, \omega)$ by

$$ad_V = \begin{bmatrix} \hat{\omega} & \hat{v} \\ 0 & \hat{\omega} \end{bmatrix}. \quad (2.38)$$

With direct calculation it can be proven that

$$ad_V^T = (ad_V)^T = \begin{bmatrix} \hat{\omega} & \hat{v} \\ 0 & \hat{\omega} \end{bmatrix}^T = \begin{bmatrix} -\hat{\omega} & 0 \\ -\hat{v} & -\hat{\omega} \end{bmatrix}. \quad (2.39)$$

As mentioned in [17] by Park and Bobrow it can be shown that

$$Ad_T ad_V = ad_{Ad_T V} Ad_T \quad (2.40)$$

by using the definitions (2.38) and (2.36).

3. Geometric model generation

In this chapter an approach developed by Park and Bobrow [17] in 1994 and extended by Chen et al. [4] in 1998 is used. This approach uses lie groups and lie algebra, presented in Chapter 2.3.3. To calculate the dynamics a so call two-step approach based on Newton and Euler, is used. Chen et al. extended the framework by introducing the the so called Assembly Incidence Matrix(AIM) for modelling the structure of a modular robot. The essential part of this approach is the dyad, defined as two successive robot modules connected with a joint. Details on all these techniques will be explained in the next chapter. For simplicity no loops are allowed within the robot structure.

In the first part of this chapter techniques to describe the structure of the robots are introduced. After that the calculation of the kinematics along with a slight extension is presented. Finally the framework as introduced by Chen and Yang is presented.

3.1. The Assembly Incidence Matrix(AIM)

In 1998 Chen et al. [4] introduced the Assembly Incidence Matrix(AIM) as a way to represent a tree-structured robot. The AIM representing a robot with n modules is a $n \times n - 1$ matrix with $2(n - 1)$ non-zero entries. A nonzero entry r_{ij} in the AIM corresponds to the joint e_j being connected to the module v_i on the side r_{ij} . Since every joint connects two modules, every column has exactly two non-zero entries. In Fig.3.1 a tree-structured robot and its corresponding graph representation are shown.

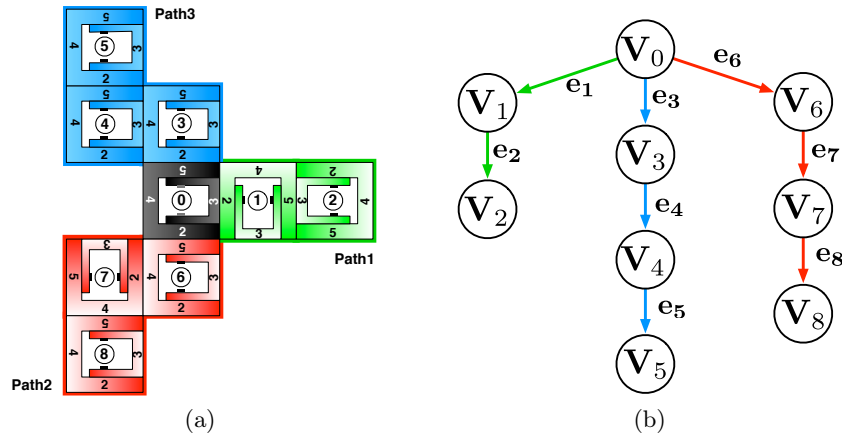


Figure 3.1.: Tree-Structured Robot (a) and the corresponding graph (b)

Fig.3.2 shows the corresponding AIM, to highlight the correlation to both the robot structure and the graph representation the paths are marked within the AIM and within the robot structure. One AIM represents exactly on robot structure and has to be recalculated with each structural change. The last column provides additional information on the module type. In this case it is assumed that only backbone(B) modules are used.

	Path1		Path2		Path3				Type
	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	
v_0	3	0	5	0	0	2	0	0	B
v_1	4	3	0	0	0	0	0	0	B
v_2	0	4	0	0	0	0	0	0	B
v_3	0	0	2	4	0	0	0	0	B
v_4	0	0	0	3	5	0	0	0	B
v_5	0	0	0	0	2	0	0	0	B
v_6	0	0	0	0	0	5	4	0	B
v_7	0	0	0	0	0	0	2	4	B
v_8	0	0	0	0	0	0	0	5	B

Figure 3.2.: The AIM corresponding to the robot structure shown in Fig.3.1a

3.2. The Accessibility Matrix

Along with the AIM the Accessibility Matrix(AM) is defined.

The AM is a $(n + 1) \times (n + 1)$ Matrix containing the information where an entry $r_{ij} = 1$ if there is a connection from v_i to v_j , $r_{ij} = 0$ if there is no connection. These Matrix can be obtained from the AIM with loss of the information about which sides

are connected. The AM derived from Fig.3.2 is shown in (3.1).

$$AM = \begin{matrix} & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \\ \begin{matrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{matrix} & \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad (3.1)$$

3.3. Forward Kinematics

By using the dyads the kinematics can be calculated recursively. Starting at the base module all other positions are calculated with respect to its preceding module using

$$T_{ij}(\theta_j) = T_{ij}(0)e^{\hat{s}_j\theta_j} \quad (3.2)$$

- $\hat{s}_j \in se(3)$ is the skew-symmetric representation of the twist of the joint e_j
- $T_{ij}(0) \in SE(3)$ is the initial position of the module v_j with respect to the frame i of v_i
- $\theta_j \in \mathbb{R}$ is the generalised coordinate of joint e_j , in case of revolute joints this is the angle of rotation

The initial position $T_{ij}(0) \in SE(3)$ is defined by

$$T_{ij}(0) = \begin{bmatrix} R_{ij}(0) & p_{ij}(0) \\ 0 & 1 \end{bmatrix} \quad (3.3)$$

where $R_{ij}(0) \in SO(3)$ is the initial Rotation of the module v_j with respect to v_i . The vector $p_{ij}(0)$ is the vector from the origin of frame i to the origin of frame j with respect to frame i . Thus the kinematics of a serial type robot with $n + 1$ modules can be calculated using

$$T_{0n} = T_{01}(\theta_1)T_{12}(\theta_2) \dots T_{n-1,n}(\theta_n) \quad (3.4)$$

$$= T_{01}(0)e^{\hat{s}_1\theta_1}T_{12}(0)e^{\hat{s}_2\theta_2} \dots T_{n-1,n}(0)e^{\hat{s}_n\theta_n} \quad (3.5)$$

3.3.1. Recursive forward kinematics

Chen and Yang in [3] introduced the `TreeRobotKinematics` algorithm. In [4] they presented the structure of this algorithm as shown in Fig.3.3. The algorithm derived from common graph traversing algorithms, in particular the Depth-First-Search(DFS) algorithm. Starting at the base link the whole robot structure is traversed recursively. The possible transformations are calculated using (3.2) and (3.4).

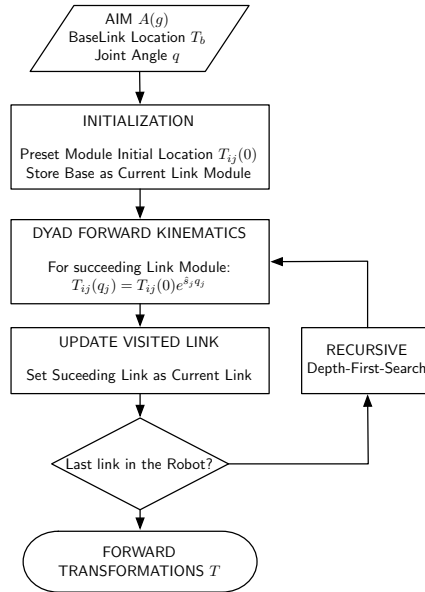


Figure 3.3.: The `TreeRobotKinematics` algorithm

3.3.2. Calculating the initial transformations

The initial transformations defined in (3.3) have been a central topic in the work of Ganzhorn [5].

Since only planar connections are allowed there is an easy and straightforward way to determine both $R_{ij}(0)$ and $p_{ij}(0)$. Given that only connections on side 2,3,4 and 5 are allowed, there are only $4^2 = 16$ different connections that can be divided up into 4 rotations as shown in Fig. 3.1. The initial translation $p_{ij}(0)$ only depends on the connecting side of the module i as shown in Fig.3.4. The MATLAB[®] code for both the evaluation of $R_{ij}(0)$ and $p_{ij}(0)$ which is rather simple can be found in A.2 and A.3.

3. Geometric model generation

Angle(CounterClockw.)	0° CCW	90° CCW	180 CCW	270° CCW
Rotationmatrix	$R_{ij} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$R_{ij} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$R_{ij} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$R_{ij} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Connected sites	(3,4) (5,2) (4,3) (2,5)	(3,5) (5,4) (4,2) (2,3)	(3,3) (5,5) (4,4) (2,2)	(3,2) (5,3) (4,5) (2,4)

Table 3.1.: Initial rotations

Connecting side	$p_{ij}(0)$
2	$p_{ij}(0) = \begin{bmatrix} 0 \\ 0 \\ -L \end{bmatrix}$
3	$p_{ij}(0) = \begin{bmatrix} 0 \\ L \\ 0 \end{bmatrix}$
4	$p_{ij}(0) = \begin{bmatrix} 0 \\ -L \\ 0 \end{bmatrix}$
5	$p_{ij}(0) = \begin{bmatrix} 0 \\ 0 \\ L \end{bmatrix}$

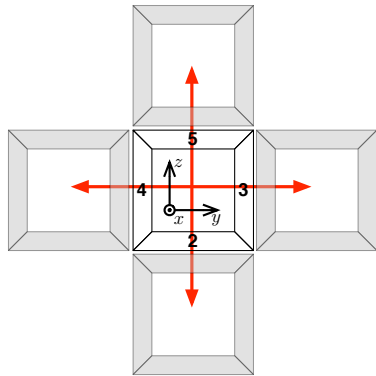


Figure 3.4.: Initial translation

3.3.3. Recursive forward kinematics using lists

Assuming that the structural changes of the robot are rare compared to the runtime, the time needed to evaluate the equations of motion can be reduced by calculating which transformations will be needed. However these speed improvements only take place when the framework is calculated in numerical mode as explained in 4.2. Using this information computationally intensive graph traversing algorithms, like the DFS, have to be used only if the structure of the robot changes.

We can differentiate between two types of transformations:

Definition 1 *direct transformations*

Direct transformations are transformations between two successive modules. These transformations depend on the connecting module sites.

Definition 2 *indirect transformations*

Every transformation that is not a direct transformation is a indirect transformation. These indirect transformations are calculated using transformations of two consecutive modules.

The information about the possible transformations is stored in two lists. One is the Direct-Transformation-List (DTL), storing the direct transformations as well as the connecting sides. The second list is the Indirect-Transformation-List (IDTL) storing the indirect transformations. Tab. 3.5 shows these two list that represent the direct and indirect calculations of the robot shown in Fig.3.1a.

DTL			
T_{ij}		Sides	
i	j	from	to
0	1	3	2
1	2	5	3
0	3	5	2
3	4	4	3
4	5	5	2
0	6	2	5
6	7	4	2
7	8	4	5

IDTL					
T_{ij}		$= T_{ix} \cdot T_{xj}$			
i	j	i	x	x	j
0	2	0	1	1	2
0	4	0	3	3	4
3	5	3	4	4	5
0	5	0	4	4	5
0	7	0	6	6	7
6	8	6	7	7	8
0	8	0	7	7	8

(a) DTL
(b) IDTL

Figure 3.5.: Direct and indirect transformation list referring to the example shown in Fig.3.1.

AIM traversing algorithm

To calculate these list a recursive algorithm has been developed. Since every AIM represents a robot in tree structure as shown in Fig.3.2 and Fig.3.1 it can be traversed using algorithms derived from graph searching algorithms. The algorithm introduced here is based on the DFS and and `TreeRobotKinematics` algorithm by Chen and Yang[3]. In Fig. 3.6 a flowchart of the algorithm is shown.

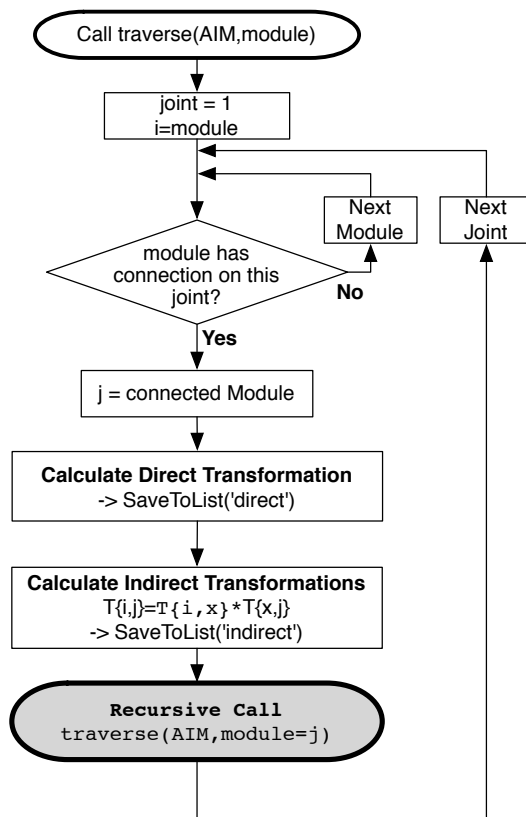


Figure 3.6.: Traversing algorithm

The MATLAB[®] -code of this recursive function is shown in A.5. Due to the traversing order the lists have to be iterated from top to bottom. During runtime every line is just to calculate one direct transformation using

$$T_{ij} = T_{ij}(0)e^{\hat{s}_j q_j} \quad (3.6)$$

where v_i and v_j are two successive modules. $T_{ij}(0)$ is calculated using the information about the connecting sites as described in 3.3.2.

The indirect transformations are calculated by using

$$T_{ij} = T_{ix} \cdot T_{xj} \quad (3.7)$$

since every line represents a calculation that could depend on a transformation calculated in a previous row it is necessary to iterate through the IDTL from top to bottom.

This algorithm has been successfully implemented and tested in MATLAB and was

used in the simulations. The MATLAB[®] -Code to obtain the transformations from a given DTL and IDTL is:

```

1 %CreateT Calculate Transformations from DTL and IDTL
2
3 %Initialize empty Tcell
4 Tcell={};
5
6 [numDirect h]=size(direct);
7 [numInDirect h]=size(indirect);
8
9 % Calculation the direct transformations
10 for j = 1:numDirect
11     i = direct(j,:);
12
13     %if virtual joint or if last was a virtual joint
14     if(any(i(3:4)'==[7 7]') || ...
15        (j > 1 && any(direct(j-1,3:4)' == [7 7]')))
16
17         Tcell{i(1)+1,i(2)+1} = eye(4)*twistexp(twists{i(2)},q(i(2)));
18
19     else %if real joint
20         % if two before was a virtual joint or it is connected
21         % to the base on side 3 or 4
22         if(j==1 || any(direct(j-1,3:4)' == [7 7]'))
23
24             Tcell{i(1)+1,i(2)+1} = ...
25                 eye(4)*twistexp(twists{i(2)},q(i(2)));
26
27         else
28             Tcell{i(1)+1,i(2)+1} = GetTijFromPair(i(3:4)',L(i(2)))...
29                 *twistexp(twists{i(2)},q(i(2)));
30         end
31     end
32 end
33
34 for i = indirect'
35     Tcell{i(1)+1,i(2)+1}= Tcell{i(3)+1,i(4)+1}*Tcell{i(5)+1,i(6)+1};
36 end
37
38 end

```

3.4. Dynamics

In [4] Chen and Yang propose the usage of a recursive Newton-Euler-approach to calculate the dynamics of branching types robots as follows:

3.4.1. Newton-Euler equations of a link assembly

Recalling a link assembly as shown in Fig. 2.6 the Newton-Euler equations are given by

$$F_{j^*} = \begin{bmatrix} f_{j^*} \\ \tau_{j^*} \end{bmatrix} = \begin{bmatrix} m_j I & 0 \\ 0 & J_{j^*} \end{bmatrix} \begin{bmatrix} \dot{v}_{j^*} \\ \dot{\omega}_{j^*} \end{bmatrix} + \begin{bmatrix} \omega_{j^*} \times m_j v_{j^*} \\ \omega_{j^*} \times J_{j^*} \omega_{j^*} \end{bmatrix} \quad (3.8)$$

with $v_{j^*} \times m_j v_{j^*} = 0$.

$$F_{j^*} = \begin{bmatrix} m_j I & 0 \\ 0 & J_{j^*} \end{bmatrix} \begin{bmatrix} \dot{v}_{j^*} \\ \dot{\omega}_{j^*} \end{bmatrix} - \begin{bmatrix} -\hat{\omega}_{j^*} & 0 \\ -\hat{v}_{j^*} & -\hat{\omega}_{j^*} \end{bmatrix} \begin{bmatrix} m_j I & 0 \\ 0 & J_{j^*} \end{bmatrix} \begin{bmatrix} v_{j^*} \\ \omega_{j^*} \end{bmatrix} \quad (3.9)$$

Here $F_{j^*} \in \mathbb{R}^{\mathcal{A} \times \mathcal{K}}$ is the wrench, consisting of the force f_{j^*} and the torque τ_{j^*} , applied on the mass centre as shown in Fig.3.7. J_{j^*} is the inertia tensor with respect to the frame j^* . m_j is the mass of the link assembly, defined as the sum of the link v_j and the joint e_j . 3.7.

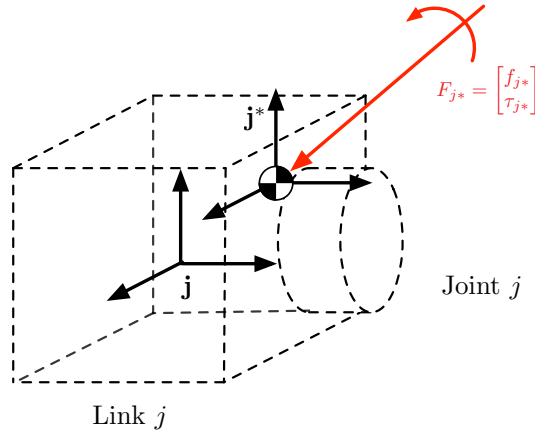


Figure 3.7.: Link assembly and applied wrench according to [4]

By defining:

The generalised mass matrix

$$M_{j^*} = \begin{bmatrix} m_j & 0 \\ 0 & J_{j^*} \end{bmatrix} \quad (3.10)$$

with $M_{j^*} \in \mathbb{R}^{6 \times 6}$.

The generalised body velocity

$$V_{j^*} = \begin{bmatrix} v_{j^*} \\ \omega_{j^*} \end{bmatrix} \quad (3.11)$$

where $V_{j^*} \in \mathbb{R}^{6 \times 1}$ and $v_{j^*}, \omega_{j^*} \in \mathbb{R}^{3 \times 1}$ are the transitional and rotary velocities with respect to j^* .

The generalised body acceleration

$$\dot{V}_{j^*} = \begin{bmatrix} \dot{v}_{j^*} \\ \dot{\omega}_{j^*} \end{bmatrix} \quad (3.12)$$

with $\dot{V}_{j^*} \in \mathbb{R}^{6 \times 1}$. Using this definitions (3.9) can be transformed into the adjoint representation

$$F_{j^*} = M_{j^*} \dot{V}_{j^*} - ad_{V_{j^*}}^T (M_{j^*} V_{j^*}) \quad (3.13)$$

where $ad_{V_{j^*}}^T \in \mathbb{R}^{6 \times 6}$ is the transposed adjoint matrix of $ad_{V_{j^*}}$. The operator ad_V is defined and explained in 2.3.7.

F_{j^*} is transformed into F_j with respect to j with

$$F_j = Ad_{T_{j^*j}}^T F_{j^*} \quad (3.14)$$

The operators Ad_T and ad_V are defined in (2.39) and (2.37).

Using the operator Ad_T the generalised body velocity V_{j^*} as well as the generalised body acceleration \dot{V}_{j^*} can be transformed into V_j and \dot{V}_j with respect to the frame j using

$$V_j = Ad_{T_{j^*j}}^T V_{j^*} \quad (3.15a)$$

$$\dot{V}_j = Ad_{T_{j^*j}}^T \dot{V}_{j^*} \quad (3.15b)$$

The corresponding transformations from frame j to frame j^* are given by

$$V_{j^*} = Ad_{T_{j^*j}} V_j \quad (3.16a)$$

$$\dot{V}_{j^*} = Ad_{T_{j^*j}} \dot{V}_j \quad (3.16b)$$

The generalised mass matrix M_{j^*} with respect to frame j^* defined in (3.10) can be transformed into M_j with respect to the frame j by

$$M_j = Ad_{T_{j^*j}}^T M_{j^*} Ad_{T_{j^*j}} \quad (3.17a)$$

$$= \begin{bmatrix} m_j I & m_j R_{j^*j}^T \hat{p}_{j^*j} R_{j^*j} \\ -m_j R_{j^*j}^T \hat{p}_{j^*j} R_{j^*j} & R_{j^*j}^T (J_{j^*} - m_j \hat{p}_{j^*j}^2) R_{j^*j} \end{bmatrix} \quad (3.17b)$$

If we assume the frame j and frame j^* to be parallel to each other ($R_{j^*j} = I$) this equation is simplified to

$$M_j = \begin{bmatrix} m_j I & m_j \hat{p}_{j^*j} \\ -m_j \hat{p}_{j^*j} & J_{j^*} - m_j \hat{p}_{j^*j}^2 \end{bmatrix} \quad (3.18)$$

We obtain the Newton-Euler equation of the link assembly j with respect to the frame j by substituting the equations (3.14), (3.16a), (3.16b) and 3.17a into (3.13)

$$F_j = M_j \dot{V}_j - \text{ad}_{V_j}^T(M_j V_j) \quad (3.19)$$

3.4.2. The two-step approach

The Newton-Euler-equation of each module is calculated using a two step approach as illustrated in Fig.3.8. In the first step the velocity and acceleration are calculated by starting at the base module and iterating to the end of each branch.

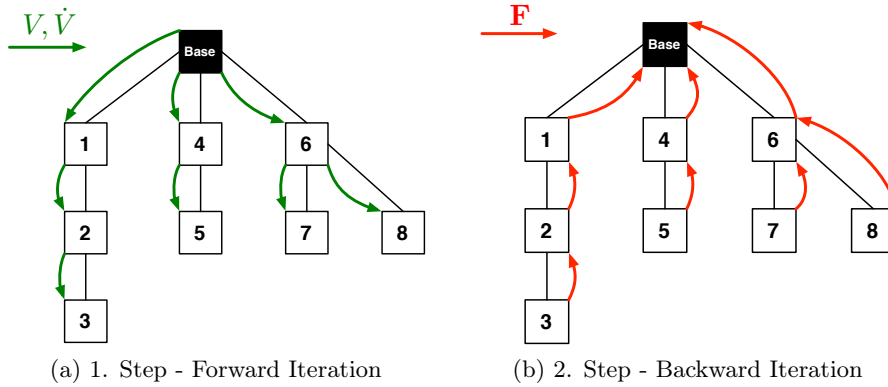


Figure 3.8.: Two step approach

1.Step - Forward Iteration

The forward iteration starts at the base with the base velocity and acceleration

$$V_0 = (0 \ 0 \ 0 \ 0 \ 0 \ 0) \quad (3.20)$$

$$\dot{V}_0 = (0 \ 0 \ -g \ 0 \ 0 \ 0) \quad (3.21)$$

The base acceleration takes gravity in z-Direction into account. Free Objects with respect to the base frame 0 would experience a acceleration of g in z-Direction. This shows that even against possible first thoughts the definition of the base acceleration

as in (3.21) is correct. All velocities and accelerations can be calculated recursively using

$$V_j = \text{Ad}_{T_{ij}^{-1}} V_i + s_j \dot{\theta}_j \quad (3.22)$$

$$\dot{V}_j = \text{Ad}_{T_{ij}^{-1}} \dot{V}_i - \text{ad}_{s_j \dot{\theta}_j} (\text{Ad}_{T_{ij}^{-1}} V_i) \quad (3.23)$$

see [16] and [4] for more details.

2.Step - Backward Iteration

In the backward iteration the wrench of each link i is calculated. This wrench consists of both the wrench that is applied by the preceding links and the external wrench. We define the set \mathcal{V} as the set of all links and $\mathcal{V}_{PD} \subset \mathcal{V}$ as the set of all pendant links. From the Newton-Euler-equation (3.19) follows that for a link assembly d_i with $v_i \in \mathcal{V}_{PD}$ it holds that:

$$F_{d_i} = M_{d_i} \dot{V}_{d_i} - \text{ad}_{V_{d_i}}^T (M_{d_i} V_{d_i}) \quad (3.24)$$

where the total wrench F_{d_i} is defined as

$$F_{d_i} = F_{d_i}^i + F_{d_i}^e \quad (3.25)$$

where $F_{d_i}^i$ is the internal wrench on v_i applied by its predecessors. $F_{d_i}^e$ is the external wrench applied on v_i . From (3.24) and (3.25) follows that

$$F_{d_i} = -F_{d_i}^e + M_{d_i} \dot{V}_{d_i} - \text{ad}_{V_{d_i}}^T (M_{d_i} V_{d_i}) \quad (3.26)$$

As shown in Fig.3.8b the calculation is done by traversing from the tips to the base. The wrench F_i of each module is calculated using

$$F_i = \sum_{j \in \mathcal{V}_{S_i}} \text{Ad}_{T_{ij}^{-1}} F_j - F_i^e + M_i \dot{V}_i - \text{ad}_{V_i}^T (M_i V_i) \quad (3.27)$$

where \mathcal{V}_{S_i} is the set of link that succeed v_i .

If we apply a torque/force directly to the input joint e_i then it holds that

$$\tau_i = s_i^T F_i \quad (3.28)$$

Equations of Motion

By rewriting the (3.20),(3.21),(3.22),(3.23),(3.26),(3.27) and (3.28) into matrix form the generalised velocity, generalised acceleration and generalised force are given by

$$V = GS\dot{\theta} \quad (3.29)$$

$$\dot{V} = G_{T_0} \dot{V}_0 + GS\ddot{\theta} + GA_1 V \quad (3.30)$$

$$F = G^T F^E + G^T M \dot{V} + G^T A_2 M V \quad (3.31)$$

$$\tau = S^T F \quad (3.32)$$

$$\begin{aligned} \dot{q} &= \text{column}[\dot{q}_1, \dot{q}_2, \dots, \dot{q}_n] \in \mathbb{R}^{n \times 1} \\ \ddot{q} &= \text{column}[\ddot{q}_1, \ddot{q}_2, \dots, \ddot{q}_n] \in \mathbb{R}^{n \times 1} \\ V &= \text{column}[V_1, V_2, \dots, V_n] \in \mathbb{R}^{6n \times 1} \\ \dot{V} &= \text{column}[\dot{V}_1, \dot{V}_2, \dots, \dot{V}_n] \in \mathbb{R}^{6n \times 1} \\ F &= \text{column}[F_1, F_2, \dots, F_n] \in \mathbb{R}^{6n \times 1} \\ F^e &= \text{column}[F_1^e, F_2^e, \dots, F_n^e] \in \mathbb{R}^{6n \times 1} \\ \tau &= \text{column}[\tau_1, \tau_2, \dots, \tau_n] \in \mathbb{R}^{n \times 1} \\ S &= \text{diag}[S_1, S_2, \dots, S_n] \in \mathbb{R}^{6n \times n} \\ M &= \text{diag}[M_1, M_2, \dots, M_n] \in \mathbb{R}^{6n \times 6n} \\ ad_{S\dot{\theta}} &= \text{diag}[-ad_{S_1\dot{\theta}_1}, -ad_{S_2\dot{\theta}_2}, \dots, -ad_{S_n\dot{\theta}_n}] \in \mathbb{R}^{6n \times 6n} \\ ad_V^T &= \text{diag}[-ad_{V_1}^T, -ad_{V_2}^T, \dots, -ad_{V_n}^T] \in \mathbb{R}^{6n \times 6n} \end{aligned}$$

$$G_{T_0} = \begin{bmatrix} Ad_{T_{0,1}^{-1}} \\ Ad_{T_{0,2}^{-1}} \\ \vdots \\ Ad_{T_{0,n}^{-1}} \end{bmatrix} \in \mathbb{R}^{6n \times 6} \quad (3.34)$$

$$G = \begin{bmatrix} I_{6 \times 6} & 0_{6 \times 6} & 0_{6 \times 6} & \cdots & 0_{6 \times 6} \\ Ad_{T_{1,2}^{-1}} & I_{6 \times 6} & 0_{6 \times 6} & \cdots & 0_{6 \times 6} \\ Ad_{T_{1,3}^{-1}} & Ad_{T_{2,3}^{-1}} & I_{6 \times 6} & \cdots & 0_{6 \times 6} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ Ad_{T_{1,n}^{-1}} & Ad_{T_{2,n}^{-1}} & Ad_{T_{3,n}^{-1}} & \cdots & I_{6 \times 6} \end{bmatrix} \in \mathbb{R}^{6n \times 6n} \quad (3.35)$$

G is the so called transmission matrix. Note that for non existing Transformations T_{ij} the zero-transformation $T_{ij} = 0^{4 \times 4}$ is used.

Closed from equations of motion By substituting eqs. (3.29)-(3.31) into (3.32) we obtain the closed form of the equation of motion.

$$\boxed{M(q)\ddot{q} + C(q, \dot{q})\dot{q} + N(q) = \tau} \quad (3.36)$$

with the mass matrix $M(q)$ defined by:

$$M(q) = S^T T^T M T S \quad (3.37)$$

the matrix $C(q, \dot{q})$ representing the Coriolis and centrifugal accelerations given by:

$$C(q, \dot{q}) = S^T T^T (M T ad_{S\dot{q}} + ad_V^*) T S \quad (3.38)$$

and the matrix $N(q)$ representing gravitational and external forces defined as:

$$N(q) = S^T T^T M T_{H_0} \dot{V} + S^T T^T F^e \quad (3.39)$$

To calculate the dynamics of a given structure under applied wrenches we can rearrange (4.3) to:

$$\boxed{\ddot{q} = M(q)^{-1}(\tau - C(q, \dot{q})\dot{q} - N(q))} \quad (3.40)$$

This equation in combination with the equations (3.37)-(3.39) will be the main equation during simulation and verification.

4. Implementation

In order to simulate the algorithms of the previous chapter the equations have been implemented using MATLAB[®]. The code developed by Ganzhorn during the work of [5] has been modified and extended.

4.1. Code example

To show how the framework is implemented here the code that is used to simulate a double pendulum structure as shown in Fig.5.2 is presented.

```
1  global direct
2  global indirect
3
4  direct=[]
5  indirect=[]
6
7  %%Create a Pendulum with "orderpendulum"-elements in a row
8  orderpendulum = 2
9  AIM = CreatePendulumAIM(orderpendulum)
10 [numOfModules,numOfJoints] = size(AIM);
11
12 %% Calculate DTL,IDTL
13 CreateTCalcOrder(AIM);
14
15 %% Tau
16 tau=zeros(orderpendulum,1);
17
18 %% Initial Conditions
19 q0 = pi/4*ones(numOfJoints,1);
20 qdot0=zeros(numOfJoints,1);
21 ic = [q0;qdot0];
22
23 %% Parameters lengths and masses
24 L=10*ones(orderpendulum,1);
25 m=ones(orderpendulum,1);
26
27 %% Solve ODE
28 tspan = [0:0.1:10];
29 [t, y] = ode45(@EvaluateNum,tspan,ic,[],tau,L,m);
```


EvaluateNum.m

```

1 function out = EvaluateNum(t,in1,tau,L,m)
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3 %This function simulates the robot
4 %Author: Alexander Gutenkunst
5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6 global AIM
7
8 [~,numOfJoints] = size(AIM);
9
10 q=in1(1:numOfJoints);
11 qdot=in1(numOfJoints+1:numOfJoints*2);
12 %Declare non-symbolic variables now, makes caluclations faster
13
14 g=9.81;
15 %% Calculate p (distances to mass-center)
16 %replace above with
17 pcell = cell(numOfJoints,1);
18 for i=1:numOfJoints
19     pcell{i} = [0 -L(i) 0]'; %distances to mass center
20 end
21
22 %% Inertia tensors
23 Izz=0;
24 Jcell = cell(numOfJoints,1);
25 for i=1:numOfJoints
26     Jcell{i} = diag([Izz 0 0]);
27 end
28
29 %% Masses
30 Massescell = cell(numOfJoints,1);
31 for i=1:numOfJoints
32     Massescell{i} = m(i); %all masses are equal
33 end
34
35 %% Generilized Mass Matrices
36 GMcell = cell(numOfJoints,1);
37 for i=1:numOfJoints
38     GMcell{i} = ...
39     [Massescell{i}*eye(3) Massescell{i}*skew(pcell{i});
40     -Massescell{i}*skew(pcell{i}) Jcell{i}-Massescell{i}*skew(pcell{i})^2];
41 end
42
43 GM = [];
44 for i=1:numOfJoints
45     GM = blkdiag(GM,GMcell{i});
46 end

```

4. Implementation

```
47
48 %% Twists
49 twists = cell(numOfJoints,1);
50 for i = 1:numOfJoints
51     twists{i} = [0,0,0,1,0,0]'; % all twists are the same
52 end
53
54 %% S
55 S = zeros(6*numOfJoints,numOfJoints);
56 for i = 1:numOfJoints
57     S((i-1)*6+1:(i-1)*6+6,i)=twists{i};
58 end
59
60 %% Create T01,T02,T12,... using DTL,IDTL
61 Tcell={};
62 Tcell=CreateTFromOrderNumPen(L,twists,q);
63
64 %% Calculate G_TO
65 G_TO = zeros(numOfJoints*6,6);
66 for i = 1:numOfJoints
67     G_TO(i*6-5:i*6,1:6) = A_AAd_chen(inv(Tcell{1,i+1}));
68 end
69
70 %% Calculate G
71 G=zeros(numOfJoints*6,numOfJoints*6);
72 for i=1:numOfJoints
73     for j=i:numOfJoints
74         rows=(i-1)*6+1:(i-1)*6+6;
75         cols=(j-1)*6+1:(j-1)*6+6;
76         if(i==j)%if diagonal element
77             G(cols,rows) = eye(6);
78         elseif(~cellfun(@isempty,Tcell(i+1,j+1)))
79             G(cols,rows) = A_AAd_chen(inv(Tcell{i+1,j+1}));
80         end
81     end
82 end
83
84 %% External Forces
85 %disp('Calculation External Forces...')
86 Fe(1:6*numOfJoints,1)=zeros(6*numOfJoints,1);
87 for a = 1:numOfJoints %Calculate the force of each Module
88     Fe(((6*a)-5):(6*a),1)=[0, 0, 0, 0, 0, 0]';
89 end
90 V=G*S*qdot;
91 %% Calculate A1
92 A1=[];
93 for i = 1:numOfJoints
```

```

94     A1 = blkdiag(A1,-AkadNonSym(twist(twists{i}*qdot(i))));
95 end
96
97 A2=[];
98 for i = 1:numOfJoints
99     V_temp = V((i-1)*6+1:(i-1)*6+6,1);
100    A2 = blkdiag(A2,-AkadNonSym(twist(V_temp))');
101 end
102 %% Calculate V0
103 V0 = [0 0 0 0 0 0]';
104
105 %% Calculate V0dot
106 V0dot = [0 -g 0 0 0 0]';
107
108 %% Calculate M_of_q
109
110 M_of_q = S'*G'*GM*G*S;
111
112 C = (S'*G'*(GM*A1+A2*GM)*G*S); %(37)
113
114 N = S'*G'*GM*G_T0*V0dot+S'*G'*Fe; %(38)
115
116 qddot = M_of_q\(tau-(C*qdot+N));
117
118 out=[qdot;qddot];
119
120 end

```

It shall be noted that using MATLAB[®] only ODEs of the form

$$\dot{x} = f(x, t) \quad (4.1)$$

can be solved. The second order equations of motion can easily transformed into this form using:

$$\begin{aligned}
 x_1 &= q_1 \\
 &\vdots \\
 x_n &= q_n \\
 x_{n+1} &= \dot{q}_1 \\
 &\vdots \\
 x_{2n} &= \dot{q}_n
 \end{aligned} \quad (4.2)$$

4.2. Comparison of the numerical and the symbolical calculation

During the work of this thesis the question arose whether the calculations should be done symbolically or numerically. Here the aspects of both techniques are shown to provide a remark on this topic for future work.

Since MATLAB[®] supports the usage of symbolic variables all equations that are needed to simulate the framework can be calculated symbolically. As a result the equation of motion

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + N(q) = \tau \quad (4.3)$$

is given completely symbolic. To solve this equation in MATLAB[®] using a ODE-Solver it has to be stored as a function. Functions in MATLAB[®] representing ODEs are structured as shown in Fig.4.1. This equations can be solved by using several

```

1  function xdot = someODE(x)
2      xdot = ...
3  end

```

Figure 4.1.: MATLAB-ODE

ODE-Solver provided by MATLAB[®] such as Runge-Kutta and Adams-Bashforth.

4.2.1. Symbolic calculation

The first approach was to calculate the central equation of motion, namely eq.(3.40), using the MATLAB[®] SYMBOLIC MATH TOOLBOX. This symbolic equation is then transformed into the affine form using (4.2) and saved as a MATLAB[®] -ODE either as file or as an internal representation. Since all intermediate results are symbolic, this approach is quite useful during debugging. An analytic verification, presented in chapter 5.1.3, is only possible with this approach.

4.2.2. Numeric calculation

This approach can be seen as the counterpart to the symbolic calculation, because no symbolic variables are used. Since most terms such as T_{ij} , G , G_{T0} , A_1 , A_2 , ... depend on θ and/or $\dot{\theta}$ they have to be recalculated on every call of the MATLAB[®] -ODE in order to compute $\ddot{\theta}$. Thus all these calculations happen 'inside' the MATLAB[®] -ODE. To avoid the time and effort of a graph-searching-algorithm such as DFS the list DTL and IDTL, introduced in chapter 3.3.3, should be used in the numerical calculations approach.

4.2.3. Comparison

To provide a basis for the decision whether the symbolic or the numerical approach should be used in the actual hardware implementation both approaches have to be compared. The calculation time consists of two parts: The time needed to create the ODE-MATLAB[®]-Function and the time needed to solve it. A multi-pendulum structure has been used as a benchmark for this comparison. Two quantities are used to compare both frameworks: The precalculation time, the time that is needed to create the MATLAB[®]-ODE, and the evaluation time, the time needed to evaluate the ODE at one time. The results shown in Fig.4.2 indicate that the MATLAB[®]-ODE

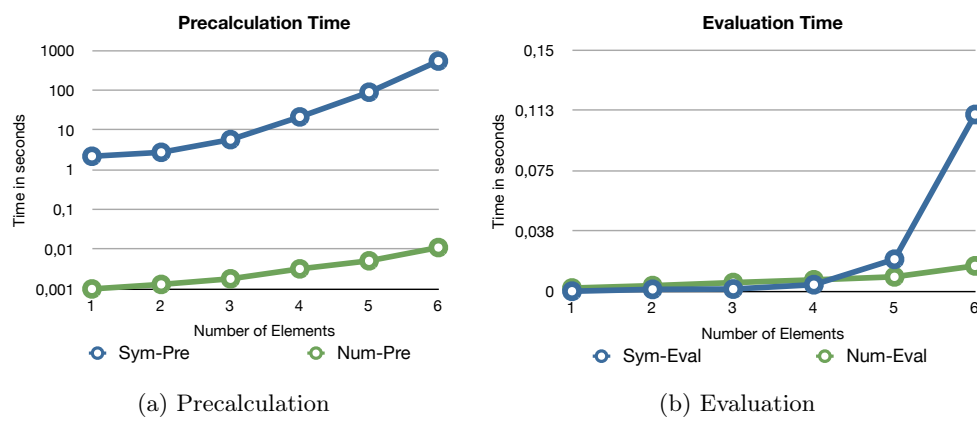


Figure 4.2.: Comparison of the symbolic and numeric approach

can be created much faster using the numerical approach. While the symbolic ODE creation of the pendulum consisting of 6 modules took more than 8 minutes it only took 0.011 seconds using the numerical approach. A rather surprising result is that numerical evaluation of the ODE is only slightly slower on fewer than 5 modules and even faster on larger structures. Although it should be mentioned that the practical calculation time strongly depends on the operating system and the used software. While the numerical approach seems to be superior when using MATLAB[®] it is not necessarily faster than every up-to-date symbolic implementation.

5. Verification

Due to the complexity of the frameworks it is necessary to verify that both implementation and modelling have been correct. In this thesis the verification is done by using two robot structures that represent commonly known physical structures. By calculating the equation of motions both with the framework and the methods of Lagrangian dynamics. Lagrangian dynamics are based on an energetic consideration of a system and form an elegant way to calculate the dynamics of multibody systems.

5.1. Verification using the double pendulum

The double pendulum, as the name suggests, consists of two pendulums connected to each other in row as shown in Fig.5.1. Since the double pendulum is often used to describe chaotic behaviour its equation of motions have been calculated in many ways. One aspect which differs from the common way the double pendulum is represented are the angles.

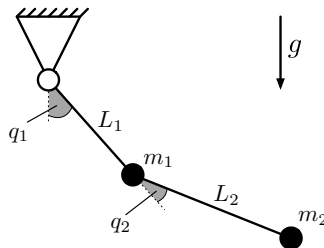


Figure 5.1.: Double pendulum

5.1.1. Modelling

The double pendulum structure can be approximated by a robot structure as shown in Fig. 5.2 This robot structure can be modelled by the framework with the following parameters. The limbs are assumed massless, also friction is neglected, thus $J_1^* = J_2^* = 0$. The masses, located at the joints are m_1 and m_2 . The multiple coordinate frames used to model the double pendulum using the robot modules are illustrated in Fig.5.3. To illustrate their position within the robot structure the robot modules are indicated by dotted lines.

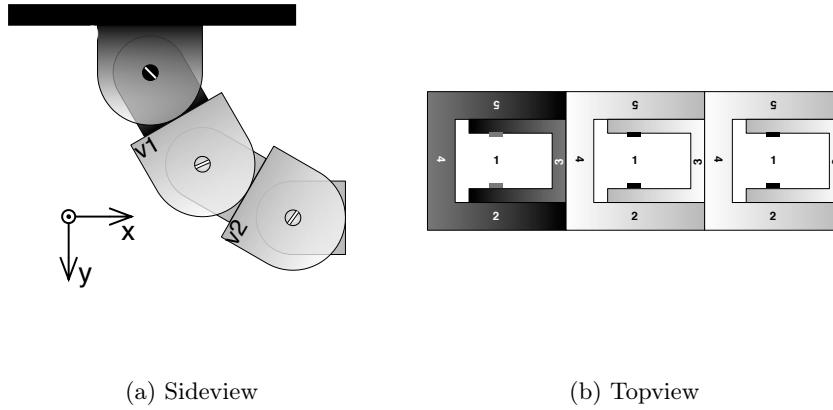


Figure 5.2.: Double pendulum structure

$j_0(\text{Base})$

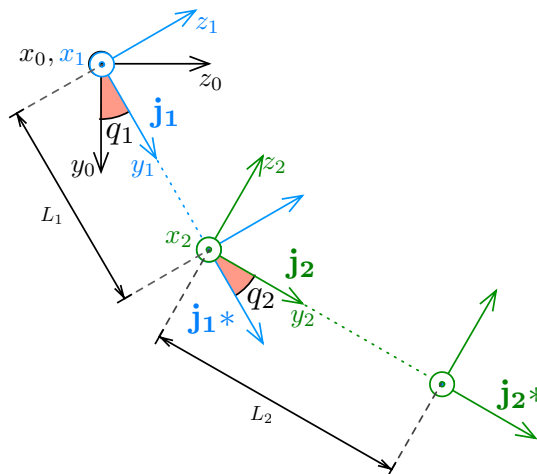


Figure 5.3.: Coordinate frames used to model the double pendulum

The first frame j_0 is the base frame and reference frame for the dynamics. The link assembly j_1 is described with the frame j_1 which is located at the origin of j_0 and rotates around the x-axis. The mass frame j_1^* of link assembly j_1 is located in a distance L_1 in y-direction. Note that j_1^* is fixed with respect to j_1 . Frame j_2 coincides with j_1^* at the centre of the second robot module. Analogue to j_1 and j_1^* the mass frame j_2^* is located in a distance L_2 in y direction from j_2 . Since all robot modules share the same geometry it can be implied that $L_1 = L_2$. Thus this

verification is also done with different lengths.

AIM Since there are two joints and three modules, the AIM representing the robot structure is a 3×2 matrix.

$$AIM = \begin{bmatrix} 3 & 0 \\ 4 & 3 \\ 0 & 4 \end{bmatrix} \quad (5.1)$$

Masses and inertia tensors Since the motors of each robot module are located at the centre, the assumption that each module robot can be seen as a point of mass can be made. Though with regard to Fig.5.3 the mass $m_{1,2}$ is not located at the centre of the link assembly $j_{1,2}$. Using the assumption that the modules can be modelled as mass points, J_{1^*} and J_{2^*} , the inertia tensors with respect to the mass frames are

$$J_{1^*} = J_{2^*} = 0 \quad (5.2)$$

The generalised mass matrices given by (3.17b) are

$$M_1 = \begin{bmatrix} m_1 I & m_1 \hat{p}_{1^*1} \\ -m_1 \hat{p}_{1^*1} & -m_1 \hat{p}_{1^*1}^2 \end{bmatrix} \quad M_2 = \begin{bmatrix} m_2 I & m_2 \hat{p}_{2^*2} \\ -m_2 \hat{p}_{2^*2} & -m_2 \hat{p}_{2^*2}^2 \end{bmatrix} \quad (5.3)$$

where \hat{p}_{1^*1} and \hat{p}_{2^*2} are the skew-symmetric representations of the position vectors of the module frames j_1 and j_2 with respect to the mass frames j_1^* and j_2^* . These vectors are given by

$$p_{1^*1} = \begin{bmatrix} 0 \\ -L_1 \\ 0 \end{bmatrix}, \quad p_{2^*2} = \begin{bmatrix} 0 \\ -L_2 \\ 0 \end{bmatrix} \quad (5.4)$$

Transformations The initial pose of the frame j_1 with respect to j_0 is a 4×4 identity matrix since both matrices coincidence if $\theta_1 = 0$. Therefore

$$T_{01}(0) = I_{4 \times 4} \quad (5.5)$$

The initial pose of j_2 with respect to j_1 is given by

$$T_{12}(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & L_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.6)$$

5.1.2. Simulation

Using the software MATLAB[®] multiple simulations with the model of the double pendulum calculated using the Lagrangian as well as the model calculated by the framework developed by Chen and Yang [Chen1998] have been done. The results showed that the implementation and the modelling has been done correctly since both models behave exactly the same way. The results of one simulation with the initial condition $\varphi_1 = \varphi_2 = \frac{\pi}{2}$, $\dot{\varphi}_1 = \dot{\varphi}_2 = 0$ are shown in Fig. 5.4.

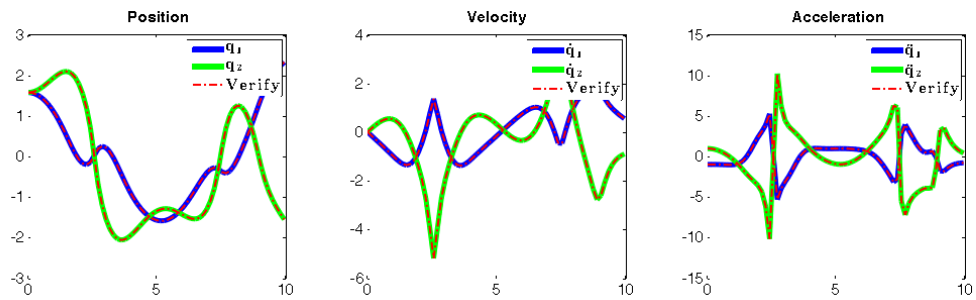


Figure 5.4.: Verification(dashed line) and results of the geometric approach

5.1.3. Analytic verification

absolute/relativ angles

As shown in Fig. 5.5 the angles of a double pendulum can be expressed in both absolute and relative angles. Because the framework uses relative angles, the equations calculated with lagrangian methods also have to be expressed in relative angles to allow direct comparison.

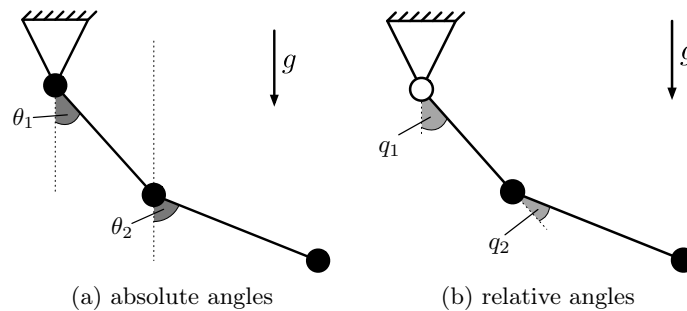


Figure 5.5.: The different angle representations

It holds that:

$$\begin{aligned}
 \theta_1 &= q_1 & q_1 &= \theta_1 \\
 \theta_2 &= q_1 + q_2 & q_2 &= \theta_2 - \theta_1 \\
 \\
 \dot{\theta}_1 &= \dot{q}_1 & \dot{q}_1 &= \dot{\theta}_1 \\
 \dot{\theta}_2 &= \dot{q}_1 + \dot{q}_2 & \dot{q}_2 &= \dot{\theta}_2 - \dot{\theta}_1 \\
 \\
 \ddot{\theta}_1 &= \ddot{q}_1 & \ddot{q}_1 &= \ddot{\theta}_1 \\
 \ddot{\theta}_2 &= \ddot{q}_1 + \ddot{q}_2 & \ddot{q}_2 &= \ddot{\theta}_2 - \ddot{\theta}_1
 \end{aligned} \tag{5.7}$$

The equations of motion given by [27, eq. (13) and (18)] are:

$$\begin{aligned}
 (m_1 + m_2)l_1^2\ddot{\theta}_1 + m_2l_1l_2\ddot{\theta}_2 \cos(\theta_1 - \theta_2) \\
 + m_2l_1l_2\dot{\theta}_2^2 \sin(\theta_1 - \theta_2) + l_1g(m_1 + m_2) \sin \theta_1 = 0
 \end{aligned} \tag{5.8a}$$

$$m_2l_2^2\ddot{\theta}_2 + m_2l_1l_2\ddot{\theta}_1 \cos(\theta_1 - \theta_2) - m_2l_1l_2\dot{\theta}_1^2 \sin(\theta_1 - \theta_2) + l_2m_2g \sin \theta_2 = 0 \tag{5.8b}$$

The interested reader can find a detailed derivation of these equations using the lagrangian method in [27].

Results from Lagrangian method

After transformation into relative angles:

Note:

$$\begin{aligned}
 \cos(\theta_1 - \theta_2) &= \cos(-(\theta_2 - \theta_1)) = \cos(-q_2) = \cos(q_2) \\
 \sin(\theta_1 - \theta_2) &= \sin(-(\theta_2 - \theta_1)) = \sin(-q_2) = -\sin(q_2)
 \end{aligned}$$

$$\begin{aligned}
 m_1 + m_2l_1^2\ddot{q}_1 + m_2l_1l_2(\ddot{q}_1 + \ddot{q}_2) \cos(q_2) \\
 - m_2l_2l_1(\dot{q}_1 + \dot{q}_2)^2 \sin q_2 + l_1gm_1 + m_2 \sin q_1 = 0
 \end{aligned} \tag{5.9a}$$

$$\begin{aligned}
 m_2l_2^2(\ddot{q}_1 + \ddot{q}_2) + m_2l_1l_2\ddot{q}_1 \cos q_2 \\
 + m_2l_1l_2\dot{q}_1^2 \sin q_2 + l_2gm_2 \sin(q_1 + q_2) = 0
 \end{aligned} \tag{5.9b}$$

Expansion and ordering of the terms:

$$\begin{aligned}
 & \underbrace{\begin{bmatrix} (m_1 + m_2)l_1^2\ddot{q}_1 + m_2l_1l_2 \cos q_2\ddot{q}_1 + m_2l_1l_2 \cos q_2\ddot{q}_2 \\ m_2l_2^2\ddot{q}_1 + m_2l_2^2\ddot{q}_2 + m_2l_1l_2\dot{q}_1 \cos(q_2) \end{bmatrix}}_{M_L(q)\ddot{q}} + \\
 & \underbrace{\begin{bmatrix} -l_1l_2m_2\dot{q}_1 \sin(q_2) (\dot{q}_1 + \dot{q}_2) - l_1l_2m_2\dot{q}_2 \sin(q_2) (\dot{q}_1 + \dot{q}_2) \\ m_2l_1l_2\dot{q}_1^2 \sin(q_2) \end{bmatrix}}_{C_L(q,\dot{q})\dot{q}} + \\
 & \underbrace{\begin{bmatrix} l_1g(m_1 + m_2) \sin q_1 \\ l_2gm_2 \sin(q_1 + q_2) \end{bmatrix}}_{N_L(q)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (5.10)
 \end{aligned}$$

This can be rearranged with $q = [q_1 \quad q_2]^T$ to

$$\begin{aligned}
 & \underbrace{\begin{bmatrix} l_1^2(m_1 + m_2) + l_1l_2m_2 \cos(q_2) & l_1l_2m_2 \cos(q_2) \\ l_2^2m_2 + l_1l_2m_2 \cos(q_2) & l_2^2m_2 \end{bmatrix}}_{M_L(q)} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} + \\
 & \underbrace{\begin{bmatrix} -l_1l_2m_2 \sin(q_2)(\dot{q}_1 + \dot{q}_2) & -l_1l_2m_2 \sin(q_2)(\dot{q}_1 + \dot{q}_2) \\ l_1l_2m_2 \sin(q_2)(\dot{q}_1 - \dot{q}_2) & l_1l_2m_2\dot{q}_1 \sin(q_2) \end{bmatrix}}_{C_L(q,\dot{q})} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} + \\
 & \underbrace{\begin{bmatrix} gl_1(m_1 + m_2) \sin(q_1) \\ gl_2m_2 \sin(q_1 + q_2) \end{bmatrix}}_{N_L(q)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (5.11)
 \end{aligned}$$

Geometric Results

The results obtained by calculating the equations of motions using symbolic calculations in MATLAB[®] are:

$$M_G\ddot{q} + C_G\dot{q} + N_G = 0 \quad (5.12)$$

with

$$M_G = \begin{bmatrix} l_1^2(m_1 + m_2) + l_2^2m_2 + 2l_1l_2m_2 \cos(q_2) & l_2m_2(l_2 + l_1 \cos(q_2)) \\ l_2m_2(l_2 + l_1 \cos(q_2)) & l_2^2m_2 \end{bmatrix} \quad (5.13a)$$

$$C_G = \begin{bmatrix} -2l_1l_2m_2\dot{q}_2 \sin(q_2) & -l_1l_2m_2\dot{q}_2 \sin(q_2) \\ l_1l_2m_2 \sin(q_2) (\dot{q}_1 - \dot{q}_2) & l_1l_2m_2\dot{q}_1 \sin(q_2) \end{bmatrix} \quad (5.13b)$$

$$N_G = \begin{bmatrix} g(l_2m_2 \sin(q_1 + q_2) + l_1m_1 \sin(q_1) + l_1m_2 \sin(q_1)) \\ gl_2m_2 \sin(q_1 + q_2) \end{bmatrix} \quad (5.13c)$$

Proof of Equality

Now we want to show that both the equation by Lagrange and the equation given by the geometric approach are actually the same. For both equations it holds that

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + N(q) = 0$$

Since the right side of the equation is zero we are allowed to multiply the whole equation with $T \in \mathbb{R}^{2 \times 2}$ such that

$$T \cdot M(q)\ddot{q} + T \cdot C(q, \dot{q})\dot{q} + T \cdot N(q) = 0$$

The existence of a constant matrix T with

$$M_L = T \cdot M_G \tag{5.14a}$$

$$C_L = T \cdot C_G \tag{5.14b}$$

$$N_L = T \cdot N_G \tag{5.14c}$$

is a sufficient condition for both equations to be the same.

In our case it can be proven by direct calculation that eq.(5.14a)-(5.14c) holds if

$$T = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

thus both equations are equal.

□

5.2. Verification using the crane model

To illustrate and verify the usage of virtual joints the second model is a crane model as shown in Fig.5.6a. The model consist of a wagon with mass m_1 moving linear along one axis. Attached to the wagon is a pendulum with the length l and a mass m_2 attached to its end. The rod is assumed to be massless. Also the rotary pivot at the wagon and the linear movement is assumed to be frictionless. The DOF along the z-Axis is modelled using a virtual joint.

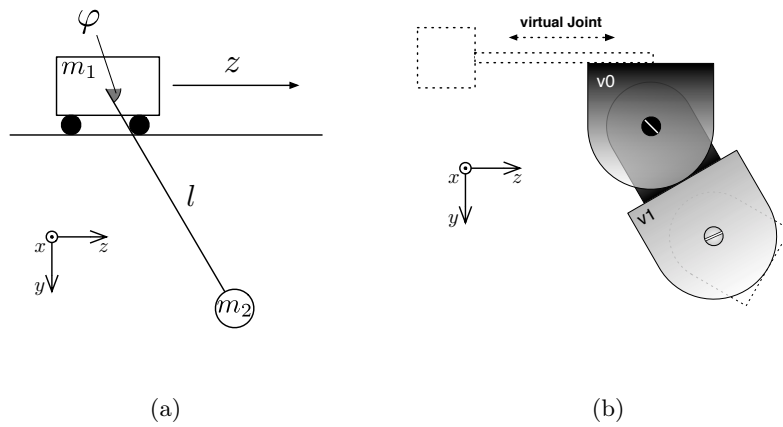


Figure 5.6.: (a) Crane model; (b) Corresponding robot structure using a virtual joint

The equations of motion are obtained from [6, eq.(10a) and (10b)] where they are calculated using Lagrangian methods. The equations of motion, simplified by the assumption of absent of friction and actuation, are given by

$$(m_s + m_L)\ddot{z} + m_l l \ddot{\varphi} \cos \varphi - m_L l \dot{\varphi}^2 \sin \varphi = 0 \quad (5.15a)$$

$$l \ddot{\varphi} + g \sin \varphi + \ddot{z} \cos \varphi = 0 \quad (5.15b)$$

These equations will be used to verify the geometric results. The motion of the wagon is realised using a virtual joint as shown in Fig. 5.6b. Implemented as a virtual prismatic joint it allows the wagon to move freely along the axis.

5.2.1. Modelling

The multiple coordinate frames of the crane model are illustrated in Fig.5.6a.

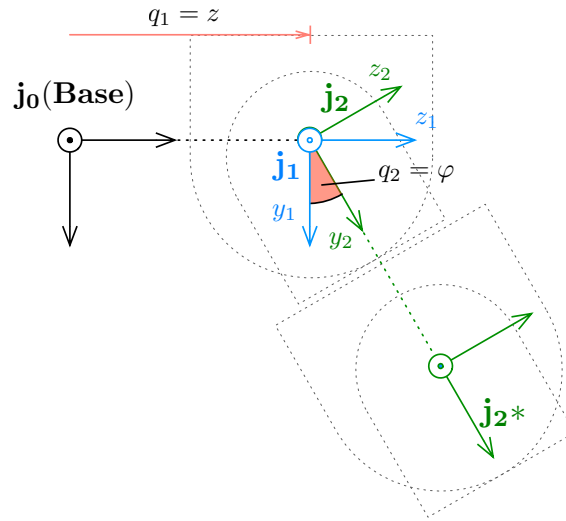


Figure 5.7.: Coordinate frames of the crane model

AIM

Since there are no connecting sides used by the virtual joints, ‘7’ is used as a placeholder. The AIM representing the robot shown in Fig.5.6b is given by

$$AIM = \begin{bmatrix} 7 & 0 \\ 7 & 3 \\ 0 & 4 \end{bmatrix} \quad (5.16)$$

Twists

The virtual twist allows a translational motion parallel to the z -Axis. Its twist is given by:

$$s_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (5.17)$$

The second twist describing the rotation around the x-Axis is defined by:

$$s_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad (5.18)$$

Masses and inertia tensor

While the double pendulum is bounded to the base module, this crane model has one degree of freedom(DOF). This DOF is modelled using a virtual joint. Since the second link assembly represents the pendulum and its mass m_2 . The mass of the wagon m_1 is connected to the first joint, which is the virtual joint.

According to the modelling illustrated in Fig. there are no inertia tensors. From that follows:

$$J_{1*} = J_{2*} = 0 \quad (5.19)$$

Since the module frame j_1 and the mass frame j_{1*} coincidence it holds that:

$$p_{1*1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (5.20)$$

Accordingly to the double pendulum example, for the pendulum connected to the wagon it holds that:

$$p_{2*2} = \begin{bmatrix} 0 \\ -L \\ 0 \end{bmatrix} \quad (5.21)$$

Initial transformations

Because the frame j_1 and j_2 coincidence at $t = 0$:

$$T_{01}(0) = T_{12}(0) = I_{4 \times 4} \quad (5.22)$$

5.2.2. Simulation

Using the settings described in the previous section the crane model can be simulated using MATLAB[®]. Fig. 5.8 shows the simulation of this crane model. The red dotted line shows the simulation of the model created using Lagrangian method. The simulations of the particular simulation are:

$$m_1 = m_2 = 1 \quad L = 5 \quad (5.23)$$

5. Verification

The initial conditions are:

$$z(0) = z_0 = 0 \quad \varphi(0) = \varphi_0 = \frac{\pi}{4} \quad (5.24)$$

The results indicate that the implementation and the modelling has been done

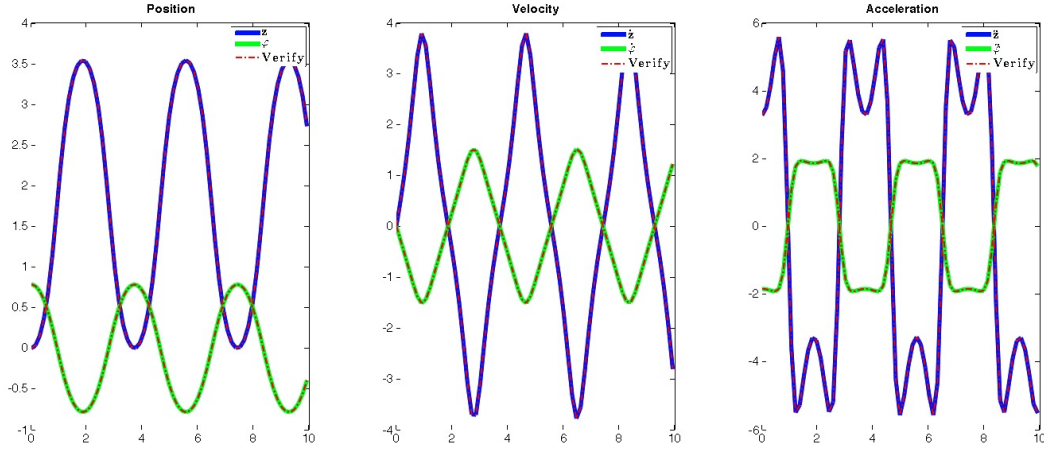


Figure 5.8.: Simulation of the crane model

correctly since the result of the model calculated by the lagrangian method and the model calculated using the geometric approach behave exactly the same. Various simulations with different parameters and initial conditions strengthened this assumption.

5.3. Verification of unbounded robot structures using Newton's first law

Using virtual joints we can design robot structures that are not bounded to the ground. The model generated using the framework introduced in chapter 3 can be easily verified by using a generalised version of Newton's first law of motion. This law, generalised for multibody systems, states that a closed system maintains the velocity of its centre of mass (COM) if no external forces are applied. Thus for the velocity of the centre of mass v_{COM} we obtain:

$$v_{COM} = \text{const} = v_{COM,0} \quad (5.25)$$

where $v_{COM,0}$ is the initial velocity of the mass centre. Since the robot modules are modelled as mass points, the position of the centre of mass r_{COM} is given by:

$$\vec{r}_{COM} = \frac{1}{M} \sum_i m_i \vec{r}_i \quad (5.26)$$

here M is the total mass, r_i are the positions of the mass points with their respective masses m_i .

In order to verify that the modelling of the free robot has been done correctly a unbounded robot with 4 modules has been simulated. To prove that (5.25) holds, the centre of mass (COM) has been calculated using (5.26) and plotted. In order to move the robot, a random torque has been applied on all real joints. Since no external forces are allowed for (5.25) to hold no friction is applied. In short, this is a simulation of a robot moving randomly on a flat frictionless surface. Fig. 5.9 shows the results of this simulation. As one can see, the centre of mass does not move. This approach provides an elegant and fast way to verify such models by checking if a fundamental law of physics holds. Though there is one theoretical catch. This is only a sufficient condition, so all correct models must pass this verification but not all models that pass this verification are necessarily correct.

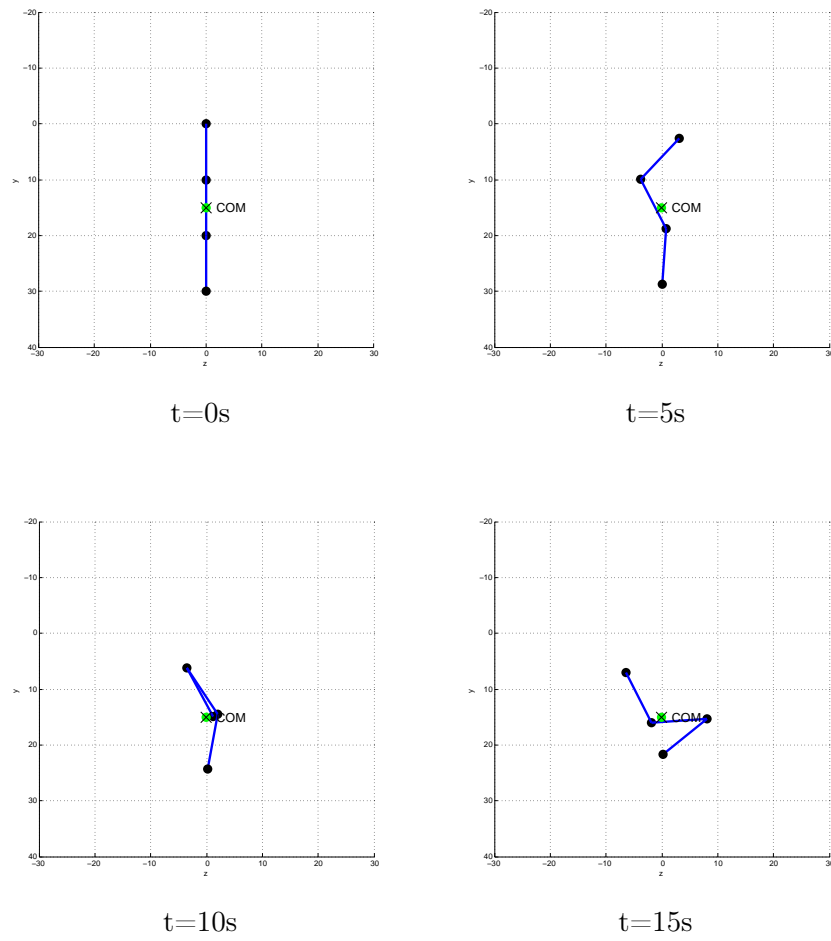


Figure 5.9.: Simulation of moving unbounded robot structure - TopView

6. Controller design

Nonlinear systems are often transformed into linear systems since these transformed systems can be easily controlled using linear techniques. One direct way to linearise a system is to calculate the Jacobian matrix at one operating point. While this technique describes the system exactly at this point, it can be inaccurate on other points resulting in insufficient control accuracy. An other technique to obtain a linear system is the so called *feedback linearisation* also known as *exact linearisation*. In this technique the system is linearised using a feedback that cancels out the nonlinearities. The resulting system is an exact representation of the nonlinear system within a wide operating range. This is achieved by two-steps: Firstly a coordinate transformation is applied. In the second step a feedback which linearises the system is calculated in these transformed coordinates.

6.1. Feedback linearisation

In this section the theory of feedback linearisation based on [8, chapter 5] is presented. For simplicity only square systems with the same number of inputs and outputs are considered. Since this is only a brief summary the interested reader may find further information on feedback linearisation in [8].

6.1.1. Theory

We assume the systems to be in control-affine form.

$$\dot{x} = f(x) + g(x)u \quad (6.1a)$$

$$y = h(x) \quad (6.1b)$$

with

$$g(x) = [g_1(x), \dots, g_m(x)] \quad (6.2)$$

where $f(x), g_1(x), \dots, g_m(x)$ are smooth vector fields defined in \mathbb{R}^m .

Since only square systems are considered the output y is defined by

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} h_1(x) \\ \vdots \\ h_m(x) \end{bmatrix} = h(x) \quad (6.3)$$

Lie derivative

In this theory Lie derivations are widely used. A Lie derivation essentially is the derivation of a function λ along a vector field f . According to [8] we define:

$$L_f \lambda(x) = \sum_{i=1}^n \frac{\partial \lambda}{\partial x_i} f_i(x) \quad (6.4)$$

The k -th derivation along two vector field f and g is defined as

$$L_g L_f \lambda(x) = \frac{\partial(L_f \lambda)}{\partial x} g(x) \quad (6.5)$$

where λ is first derived along f and then derived along g .

The repeating derivation along the same vector field f is defined as

$$L_f^k \lambda(x) = \frac{\partial(L_f^{k-1} \lambda)}{\partial x} f(x) \quad (6.6)$$

Relative degree

The vector

$$r = \begin{bmatrix} r_1 \\ \vdots \\ r_m \end{bmatrix} \quad (6.7)$$

is the relative degree if:

(i)

$$L_{g_j} L_f^k h_i(x) = 0 \quad \forall 1 \leq j \leq m, \forall 1 \leq i \leq m \quad (6.8)$$

for all $k < r_i - 1$

(ii)

$$A(x) = \begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1(x) & \dots & L_{g_m} L_f^{r_1-1} h_1(x) \\ \vdots & & \vdots \\ L_{g_1} L_f^{r_m-1} h_m(x) & \dots & L_{g_m} L_f^{r_m-1} h_m(x) \end{bmatrix} \quad (6.9)$$

is nonsingular.

Practically the relative degree r_i corresponds to the number of times the output y_i has to be differentiated until at least one component of the input vector u is non-zero.

Closed loop

We assume that the system considered are so called ‘flat’ systems with order n , from that follows

$$\sum_{i=1}^m r_i = n. \quad (6.10)$$

In order to linearise the system via feedback a function

$$u = \alpha(x) + \beta(x)v \quad (6.11)$$

with

$$\alpha(x) = \begin{bmatrix} \alpha_1(x) \\ \vdots \\ \alpha_m(x) \end{bmatrix} \quad \beta(x) = \begin{bmatrix} \beta_{11}(x) & \dots & \beta_{1m}(x) \\ \vdots & & \vdots \\ \beta_{m1}(x) & \dots & \beta_{mm}(x) \end{bmatrix} \quad (6.12)$$

has to be determined which linearises the close loop given by:

$$\dot{x} = f(x) + g(x)\alpha(x) + g(x)\beta(x)v \quad (6.13a)$$

$$y = h(x) \quad (6.13b)$$

Coordinate transformation

The transformation into the new coordinates ξ is given by

$$\xi = \Phi(x) \quad (6.14)$$

with

$$\xi = (\xi^1, \dots, \xi^m) \quad (6.15)$$

and

$$\xi^i = \begin{bmatrix} \xi_1^i \\ \vdots \\ \xi_{r_i}^i \end{bmatrix} = \begin{bmatrix} \phi_1^i(x) \\ \vdots \\ \phi_{r_i}^i(x) \end{bmatrix} = \begin{bmatrix} h_i(x) \\ \vdots \\ L_f^{r_i-1} h_i x \end{bmatrix} \quad (6.16)$$

for $1 \leq i \leq m$

The resulting transformed system is given by:

$$\begin{aligned} \dot{\xi}_1^i &= \xi_2^i \\ &\vdots \\ \dot{\xi}_{r_i-1}^i &= \xi_{r_i}^i \\ \dot{\xi}_{r_i}^i &= b_i(\xi) + \sum_{j=1}^m a_{ij}(\xi)u_j \end{aligned} \quad (6.17)$$

$$y_i = \xi_1^i$$

with $1 \leq i \leq m$.

a_{ij} are the entries of the matrix A defined in (6.9) with $x = \Phi^{-1}(\xi)$ and b_i are the entries of

$$b(x)|_{x=\Phi^{-1}} = \begin{bmatrix} L_f^{r_1} h_1(x) \\ L_f^{r_2} h_2(x) \\ \vdots \\ L_f^{r_m} h_m(x) \end{bmatrix}_{x=\Phi^{-1}} \quad (6.18)$$

By defining

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix} = b(\xi) + A(\xi)u \quad (6.19)$$

because $A(\xi)$ is non-singular the feedback can be obtained by

$$u = A^{-1}(\xi)[-b(\xi) + v] \quad (6.20)$$

Applying this feedback on results in a system linear and controllable given by:

$$\begin{aligned} \dot{\xi}_1^i &= \xi_2^i \\ &\vdots \\ \dot{\xi}_{r_1-1}^i &= \xi_{r_i}^i \\ \dot{\xi}_{r_i}^i &= v_i \end{aligned} \quad (6.21)$$

with $1 \leq i \leq m$.

The feedback linearisation of systems in the form (6.1a) the calculation of this feedback is rather complex and involves a coordinate transformation.

Fortunately the systems occurring by using the framework described in this thesis are of the form:

$$\ddot{q} = M(q)^{-1}(\tau - C(q, \dot{q})\dot{q} - N(q)) \quad (6.22)$$

where the applied torque on each joint τ is the input vector. In [22] Seifried discusses the feedback linearisation of systems in this structure. While in [22] especially under-actuated systems are concerned it is assumed in this thesis that the system is fully actuated.

It can be seen directly that systems of the form (3.40) can be linearised using

$$\tau = C(\dot{q}, q)\dot{q} + N(q) + M(q)v \quad (6.23)$$

where v is the new input vector of the linearised system. The resulting linear system, as shown in Fig. 6.1, is given by:

$$v = \ddot{q} \quad (6.24)$$

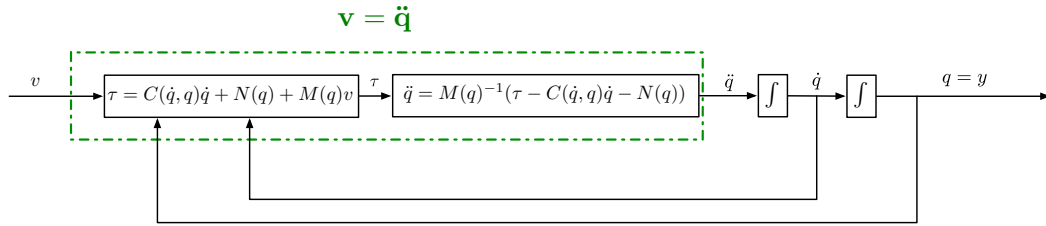


Figure 6.1.: Feedback-Linearisation

This system is not only linearised but also fully decoupled. Such that with

$$\begin{aligned} v_1 &= \ddot{q}_1 = \ddot{y}_1 \\ &\vdots \\ v_n &= \ddot{q}_n = \ddot{y}_n \end{aligned} \tag{6.25}$$

each output y_i is controlled by the input v_i with $1 \leq i \leq n$.

6.1.2. The zero dynamics

In general one has to consider the zero dynamics that can occur during feedback linearisation. The zero dynamics describe the internal dynamics and have to be considered to guarantee internal stability. However since the systems considered in this thesis are flat systems there are no zero dynamics at all. This corresponds to the work of [22] where zero dynamics only exist if there are under-actuated joints. Since in this thesis it is assumed that there are no such under-actuated joints, there is also no zero-dynamics.

6.2. PID-controller

In the previous section it has been shown that the systems concerned in this thesis can easily be linearised and decoupled. As mentioned in [22] (6.24) is already in canonical controllable form, thus it is clearly controllable. Such a system has been simulated to show the advantage how easily a system with these properties can be controlled. The system used here is the double pendulum introduced in chapter 5.1. The simulation itself is done in SIMULINK a MATLAB[®] extension that uses block diagrams to model dynamical systems. Fig.6.2 shows the SIMULINK diagram of the double pendulum which is controlled by two standard PID-controllers. Using the linearised system each controller is only in charge of one joint.

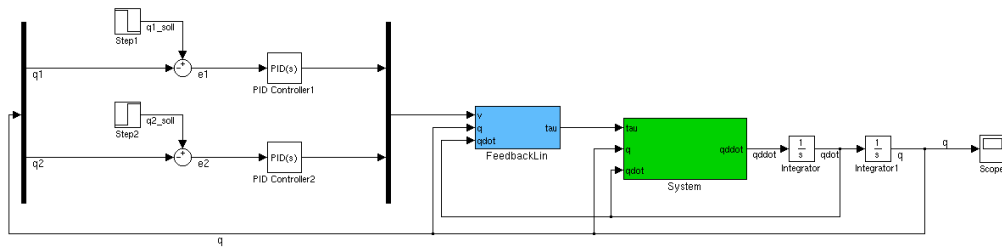


Figure 6.2.: SIMULINK-Diagram

The subsystems *System* and *Feedback-Linearisation* are given by:

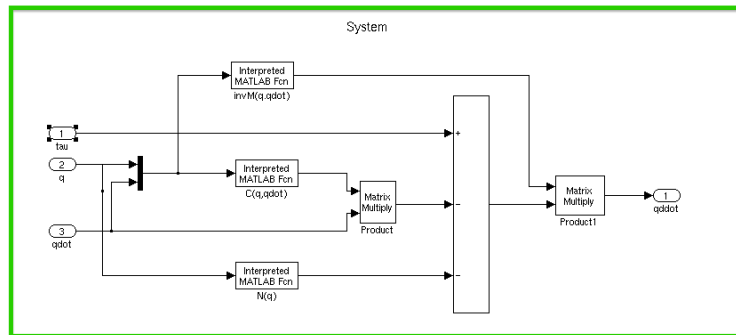
The PID-Controller is defined by three gains K_P , K_I and K_D . These parameters have to be chosen with regards to the desired system behaviour, e.g. a fast response time or low overshoot.

In this example the parameters are

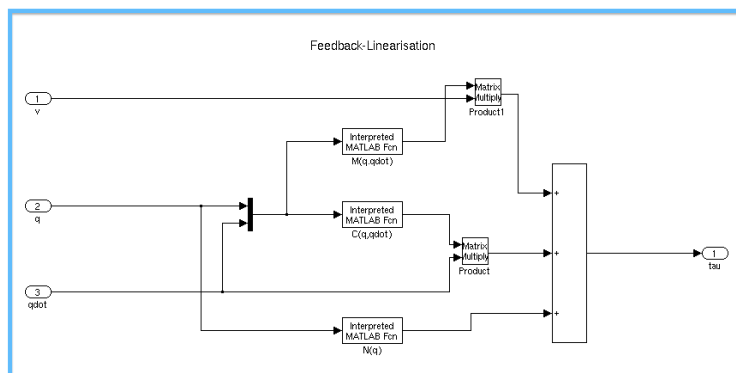
$$\begin{aligned} K_P & 10 \\ K_I & 1 \\ K_D & 4 \end{aligned}$$

Fig. 6.4 shows the step responses of simulations with

$$q_1(0) = q_2(0) = \dot{q}_1(0) = \dot{q}_2(0) = 0 \quad (6.26)$$



(a)



(b)

Figure 6.3.: The System (a) itself and the feedback linearisation (b)

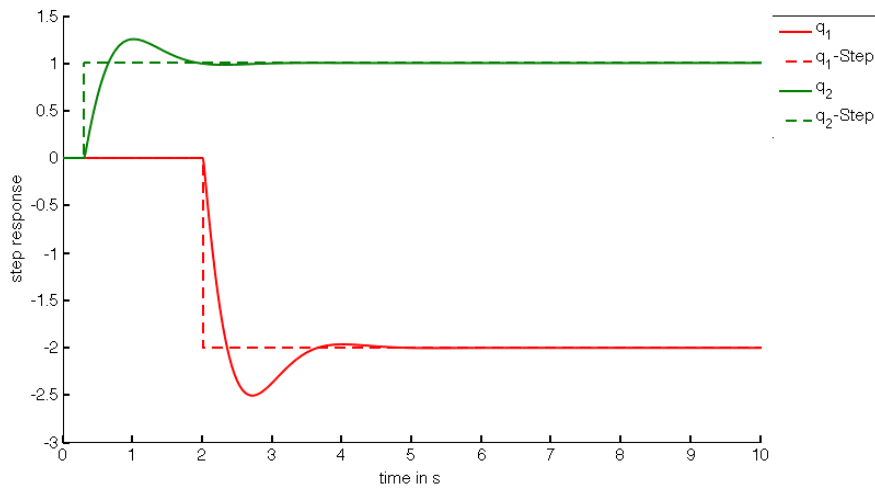


Figure 6.4.: Step response

6.3. Pole placement

Accordingly to [16, p.190ff] and [19, p.100ff] it is shown how a trajectory control based on the linearised system can be calculated. If the robot joints should follow a given trajectory defined by $\tilde{q}_t \in \mathbb{R}^n$, $\dot{\tilde{q}} \in \mathbb{R}^n$, $\ddot{\tilde{q}} \in \mathbb{R}^n$ this can be easily achieved using the linearised system previously calculated as

$$v = \ddot{q} \quad (6.27)$$

and the input

$$v = -K_d(\dot{q} - \dot{\tilde{q}}) - K_p(q - \tilde{q}) + \ddot{\tilde{q}} \quad (6.28)$$

since the system (6.27) is fully decoupled and it holds that

$$\begin{aligned} v_1 &= \ddot{q}_1 = \ddot{y}_1 \\ &\vdots \\ v_n &= \ddot{q}_n = \ddot{y}_n \end{aligned} \quad (6.29)$$

as mentioned before.

The input (6.28) describe for each joint by:

$$v_i = -K_{i,d}(\dot{q}_i - \dot{\tilde{q}}_i) - K_{i,p}(q_i - \tilde{q}_i) + \ddot{\tilde{q}}_i \quad i = 1, \dots, n \quad (6.30)$$

where n is the number of joints.

The error dynamics of each joint is obtained by

$$\ddot{e}_i + K_{d,i}\dot{e}_i + K_{p,i}e_i \quad i = 1, \dots, n \quad (6.31)$$

Using the Hurwitz stability criterion, see [7] for details, it can be seen that $e_i(t)$ is asymptotically stable if $K_{d,i} > 0$ and $K_{p,i} > 0$.

Eq.(6.31) can be transformed into a system of first degree with

$$\begin{aligned} \epsilon_{1,i} &= e_i \\ \epsilon_{2,i} &= \dot{e}_i = \dot{\epsilon}_{1,i} \end{aligned} \quad (6.32)$$

From that follows

$$\begin{bmatrix} \dot{\epsilon}_{1,i} \\ \dot{\epsilon}_{2,i} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -K_{p,i} & -K_{d,i} \end{bmatrix}}_{=A_i} \begin{bmatrix} \epsilon_{1,i} \\ \epsilon_{2,i} \end{bmatrix} \quad (6.33)$$

The eigenvalues, respectively the poles, of A_i can be calculated by

$$\det(A_i - \lambda_i I) = \det \begin{bmatrix} -\lambda_i & 1 \\ -K_{p,i} & -K_{d,i} - \lambda_i \end{bmatrix} = \lambda_i^2 + \lambda_i K_{p,i} + K_{d,i} \quad (6.34)$$

To obtain a desired behaviour of the error dynamics the eigenvalues now can be choose arbitrary. The error dynamic of every joint can be set individually. Defining

$$K_d = \begin{bmatrix} K_{d,1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & K_{d,n} \end{bmatrix} \quad (6.35)$$

$$K_p = \begin{bmatrix} K_{p,1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & K_{p,n} \end{bmatrix} \quad (6.36)$$

for $1 \leq i \leq n$

6.3.1. Example

The same system that was controller by a PID-Controller in the chapter before will now be controlled via pole placement of the error-dynamics. The poles of the joints are set to be

$$\lambda_{1,1} = -1 \quad \lambda_{2,1} = -1 \quad \text{first joint} \quad (6.37a)$$

$$\lambda_{1,2} = -10 \quad \lambda_{2,2} = -5 \quad \text{second joint} \quad (6.37b)$$

Thus we obtain

$$p_1(\lambda) = (2 + \lambda)(3 + \lambda) = \lambda^2 + 5\lambda + 6 \quad (6.38a)$$

$$p_2(\lambda) = (5 + \lambda)(10 + \lambda) = \lambda^2 + 15\lambda + 50 \quad (6.38b)$$

by equating the coefficients of (6.38) and (6.34) the parameters $K_{p,1}$, $K_{p,2}$, $K_{d,1}$ and $K_{d,2}$ are given by

$$K_{p,1} = 5 \quad (6.39)$$

$$K_{d,1} = 6 \quad (6.40)$$

$$K_{p,2} = 15 \quad (6.41)$$

$$K_{d,2} = 50 \quad (6.42)$$

from that follows:

$$K_p = \begin{bmatrix} 5 & 0 \\ 0 & 15 \end{bmatrix} \quad K_d = \begin{bmatrix} 6 & 0 \\ 0 & 50 \end{bmatrix} \quad (6.43)$$

Fig.6.5 shows the SIMULINK-Diagramm with the input (6.28). The subsystems *System* and *Feedbacklinearisation* are still the same as shown in 6.3.

6. Controller design

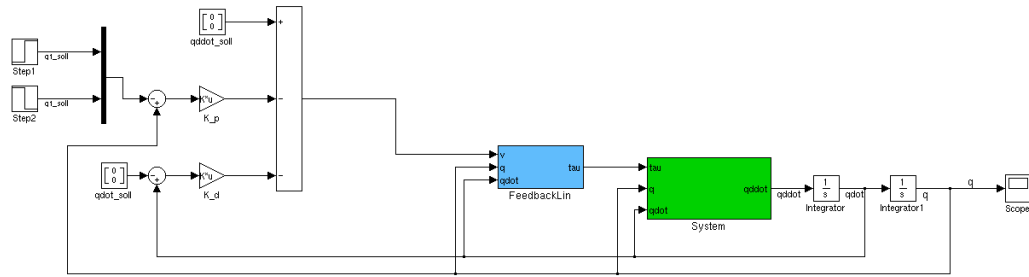


Figure 6.5.: Control via pole placement

Using the same initial conditions as before this system is simulated. The step responses are shown in Fig.6.6. One can see that the second joint reacts much faster since the poles of its error-dynamics are more negative.

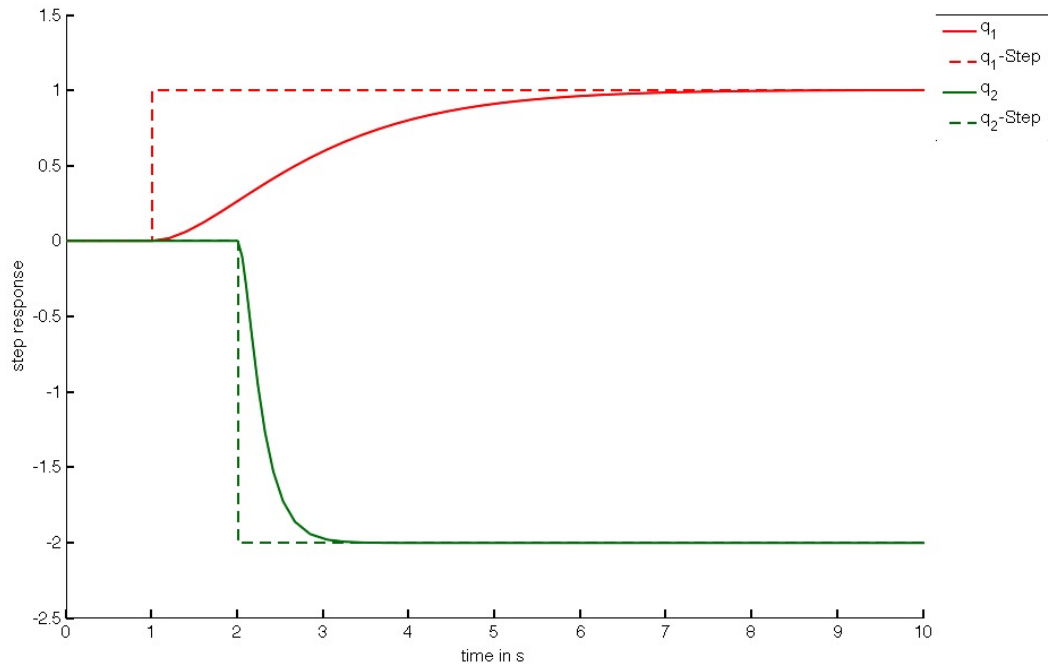


Figure 6.6.: Step response

7. Conclusion

This thesis provides a foundation for further work concerning the robots of the projects ‘SYMBRION’ and ‘REPLICATOR’. Using a framework developed by Chen et al. in [4] based on screw theory in combination with a Newton-Euler approach, the kinematics and dynamics of modular robot structures is calculated. The existing implementation in MATLAB[®] has been further developed.

To verify that the implementation and modelling the modelling has been correct the created models have been compared to models created by a Lagrangian approach.

Although the implementation has been done using MATLAB[®] it provides a basis for the verification of further implementations, e.g. in C#.

The comparison of the symbolic and numeric calculation showed that the choice of the calculation type has a huge influence on the calculation time. This will be a huge topic in the actual hardware implementation.

It has been shown that by using feedback-linearisation the systems that occur in this thesis can be easily linearised and even fully decoupled.

Using both PID-controllers as well as pole placement it has been illustrated that this linearised and decoupled systems can easily be controlled by linear control techniques.

A. Code

A.1. CreatePendulumAIM.m

```
1 function AIM = CreatePendulumAIM(numRealJoints,numVirtJoints)
2 %CREATEPENDULUMAIM Creates a AIM representation of a pendulum with
3 %real and virtual joints, the virtual joints are at the beginning of the
4 %chain
5
6 %if no value for numVirtJoints is set
7 if(nargin == 1)
8     numVirtJoints = 0;
9 end
10
11 numTotalJoints = numRealJoints+numVirtJoints;
12
13 %Create a Pendulum with "orderpendulum"-elements
14 AIM = zeros(numTotalJoints+1,numTotalJoints);
15
16 for i=1:numVirtJoints
17     AIM(i:i+1,i)=[7 7]';
18 end
19
20 for i=numVirtJoints+1:numTotalJoints
21     AIM(i:i+1,i)=[3 4]';
22 end
23 end
```

A.2. GetTijFromPair.m

```
1 function Tij = GetTijFromPair(currentPair,L)
2 %AGETTIJFROMPAIR Get initial Transformation between
3 %module i and j
4     if (currentPair(1) == 2)
5         dij = [0 0 -L]';
6     elseif (currentPair(1) == 3)
7         dij = [0 L 0]';
8     elseif (currentPair(1) == 4)
9         dij = [0 -L 0]';
10    elseif (currentPair(1) == 5)
11        dij = [0 0 L]';
12    else
13        disp('Error occured');
14    end
15
16    Rij = GetRijFromPair(currentPair);
17
18    Tij = zeros(4);
19    Tij = [Rij dij;
20          0 0 0 1];
21
22 end
```

A.3. GetRijFromPair.m

```
1 function Rij = GetRijFromPair(pair)
2 %AGETRIJFROMPAIR Create Rotation using a table
3
4 if(isequal(pair,[3 4]) || isequal(pair,[5 2]) ||...
5     isequal(pair,[4 3]) || isequal(pair,[2 5]))
6     Rij = eye(3)
7
8 elseif(isequal(pair,[3 5]) || isequal(pair,[5 4]) ||...
9     isequal(pair,[4 2]) || isequal(pair,[2 3]))
10
11     Rij = [0 -1 0;
12           1 0 0;
13           0 0 1];
14
15 elseif(isequal(pair,[3 3]) || isequal(pair,[5 5]) ||...
16     isequal(pair,[4 4]) || isequal(pair,[2 2]))
17
18     Rij = [-1 0 0;
19           0 -1 0;
20           0 0 1];
21 else
22
23     Rij = [ 0 1 0;
24           -1 0 0;
25           0 0 1];
26 end
27 end
```


A.4. CreateTCalcWrapper.m

```
1 function [direct indirect] = CreateTCalcWrapper(AIM)
2 %CREATETCALCWRAPPER Wrapper for CreateTCalcOrder.m to avoid global
3 %variables
4     clear global direct
5     clear global indirect
6     clear global AM
7
8     global direct
9     global indirect
10
11     CreateTCalcOrder(AIM);
12
13 end
```

A.5. CreateTCalcOrder.m

```
1 function CreateTCalcOrder(AIM,module)
2 %ACREATETCalcOrder
3 %Calculates DTL and IDTL
4 % %%%% %%%%
5 % % % % %
6 % % i %---->% j %
7 % % % % %
8 % %%%% %%%%
9 % i is connected to j
10
11 %direct transformations
12 global direct
13 %indirect transformations
14 global indirect
15 %save what paths have been taken
16 global AM
17
18 % If no module is specified -> first call and the module is the base(0)
19 [numOfModules,numOfJoints] = size(AIM);
20 if nargin < 2
21     module=0;
22     AM=zeros(numOfModules,numOfModules)
23 end
24
25 i=module
26
27 %iterate through joints
28 for curJoint=1:numOfJoints
29
30     %if this joints connects the i to some other i
31     if(any(AIM(i+1,curJoint)))
32
33         %iterate through modules
34         for j=0:numOfModules-1
35
36             %if other module is found
37             if(any(AIM(j+1,curJoint)) && (j ~= i))
38
39                 %create pair(represents the connected sides)
40                 pair = [AIM(i+1,curJoint) AIM(j+1,curJoint)];
41
42                 %store in direct list
43                 direct=[direct;[i j pair]]
44
45                 %mark path as taken
```

```
46         AM(i+1,j+1)=1
47
48         %Calculate indirect Transformations
49         for k=0:numOfModules-1
50
51             %if there is a possible indirect Transformation
52             if(any(AM(k+1,i+1)))
53
54                 %store in indirect list
55                 indirect=[indirect;[k j k i i j]]
56
57                 %mark path as taken
58                 AM(k+1,j+1)=1
59             end
60         end
61
62         %Delete Module From AIM
63         AIM(i+1,curJoint)=0;
64
65         %Recursive call
66         CreateTCalcOrder(AIM,j)
67     end
68 end
69 end
70 end
71 end
```

Bibliography

- [1] Aldebaran Robotics. Nao.
<http://www.aldebaran-robotics.com/en/Pressroom/Photography/nao.html>.
- [2] R. S. Ball. *A Treatise on the Theory of Screws*. Cambridge University Press, 1900.
- [3] I.-M. Chen and G. Yang. Configuration independent kinematics for modular robots. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 2, pages 1440–1445 vol.2, apr 1996.
- [4] I.-M. Chen and G. Yang. Automatic model generation for modular reconfigurable robot dynamics. *ASME Journal of Dynamic Systems, Measurement, and Control*, 120:346–352, 1998.
- [5] J. Ganzhorn. Analysis of kinematics and dynamics of modular robots. 2011.
- [6] Hertkorn, Kögel, Scheu, and Wieland. *Regelung der horizontalen Bewegung eines Brückenkrans*. Universität Stuttgart Institut für Systemtheorie und Regelungstechnik, 2006.
- [7] M. Horn. *Regelungstechnik*. Pearson Studium, 2003.
- [8] A. Isidori. *Nonlinear Control Systems: An Introduction (Communications and Control Engineering)*. Springer, 1994.
- [9] KUKA Roboter GmbH. Kuka KR 16 S.
http://www.kuka-robotics.com/en/products/industrial_robots/low/kr16_2_s.
- [10] Lynxmotion. Bh3-r walking robots.
<http://www.lynxmotion.com/images/jpg/bh3r01.jpg>.
- [11] MATLAB. *version 7.14.0.739 (R2012a)*. The MathWorks Inc., Natick, Massachusetts, United States of America, 2012.
- [12] E. Meister. Kinematics and dynamics for robot organisms. In P. Levi and S. Kernbach, editors, *Symbiotic Multi-Robot Organisms*, pages 329–339. Springer-Verlag, 2010.
- [13] E. Meister and A. Gutenkunst. Self-adaptive framework for modular and self-reconfigurable robotic systems. *ADAPTIVE 2012*, in-press.

- [14] E. Meister and D. Kryvovkhov. An artificial tactile sensor skin for modular robots. In P. F. J.C. Samin, editor, *Proc. of MULTIBODY DYNAMICS 2011, ECCOMAS Thematic Conference*, Brussels, Belgium, 2011.
- [15] E. Meister, S. Stepanenko, and S. Kernbach. Adaptive locomotion of multibody snake-like robot. *Multibody Dynamics 2011*, 2011.
- [16] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1 edition, 3 1994.
- [17] F. Park and J. Bobrow. A recursive algorithm for robot dynamics using lie groups. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 1535 –1540 vol.2, may 1994.
- [18] W. D. Pietruszka. *MATLAB und Simulink in der Ingenieurpraxis: Modellbildung, Berechnung und Simulation (German Edition)*. Vieweg+Teubner Verlag, 2006.
- [19] S. Ploen. *Geometric Algorithms for the Dynamics and Control of Multibody Systems*. PhD thesis, Univerity of California, 1997.
- [20] PolyBot.
<http://www2.parc.com/spl/projects/modrobots/polybot/demonstrations/loopcut.jpg>.
- [21] PolyPod. <http://robotics.stanford.edu/users/mark/tpoly.gif>.
- [22] R. Seifried. Optimization-based design of feedback linearizable underactuated multibody systems. In *Proceedings of the ECCOMAS Thematic Conference Multibody Dynamics 2009, Warsaw, Poland, 29. June - 2. July 2009, paper ID 121*, 2009.
- [23] SINTEF. Anna Konda.
<http://www.sintef.no/home/Information-and-Communication-Technology-ICT/Applied-Cybernetics/Projects/Our-snake-robots/Anna-Konda-The-fire-fighting-snake-robot/>.
- [24] K. Stoy, D. Brandt, and D. J. Christensen. *Self-Reconfigurable Robots: An Introduction*. The MIT Press, new edition edition, 1 2010.
- [25] SuperBot.
<http://www.isi.edu/robots/superbot/pictures/slides/P2210020.html>.
- [26] SYMBRION/REPLICATOR. <http://www.symbion.eu>.
- [27] E. W. Weisstein.
<http://scienceworld.wolfram.com/physics/DoublePendulum.html>, 2007.

Declaration

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

(Alexander Gutenkunst)