

Robot Dynamics Lecture Notes

Robotic Systems Lab, ETH Zurich

HS 2021

Contents

1	Introduction	1
1.1	Nomenclature	2
1.2	Operators	3
2	Kinematics	5
2.1	Introduction	5
2.2	Position	6
2.2.1	Representation of Positions	6
	Cartesian coordinates	6
	Cylindrical coordinates	7
	Spherical coordinates	7
2.3	Linear Velocity	7
2.3.1	Representation of Linear Velocities	7
	Cartesian Coordinates	8
	Cylindrical Coordinates	8
	Spherical Coordinates	8
2.4	Rotation	9
2.4.1	Rotation Matrices	9
2.4.2	Active vs. Passive Rotation	11
	Passive Rotation	11
	Active Rotation	11
2.4.3	Elementary Rotations	12
2.4.4	Composition of Rotations	12
2.4.5	Representation of Rotations	13
	Euler Angles	13
	Angle Axis	18
	Unit Quaternions	19
2.5	Angular Velocity	21
2.5.1	Time Derivatives of Rotation Parameterizations	22
	Time Derivatives of Euler Angles ZYX \Leftrightarrow Angular Velocity .	23
	Time Derivatives of Euler Angles XYZ \Leftrightarrow Angular Velocity .	24
	Time Derivatives of Euler Angles ZYZ \Leftrightarrow Angular Velocity .	24
	Time Derivatives of Euler Angles ZXZ \Leftrightarrow Angular Velocity .	24
	Time Derivative of Rotation Quaternion \Leftrightarrow Angular Velocity .	24
	Time Derivative of Angle Axis \Leftrightarrow Angular Velocity	25
	Time Derivative of Rotation Vector \Leftrightarrow Angular Velocity . . .	25
2.6	Transformation	26
2.7	Velocity in Moving Bodies	28

	Some Notes on Vector Differentiation	29
2.8	Kinematics of Systems of Bodies	30
2.8.1	Generalized Coordinates and Joint Configuration	30
2.8.2	Task-Space Coordinates	31
	End-Effector Configuration Parameters	31
	Operational Space Coordinates	32
2.8.3	Forward Kinematics	33
2.8.4	Differential Kinematics and Analytical Jacobian	34
	Position and Rotation Jacobian	35
	Dependency on Parameterization	35
2.8.5	Geometric or Basic Jacobian	37
	Addition and Subtraction of Geometric Jacobians	37
	Calculation of geometric Jacobian using Rigid Body Formulation	38
2.8.6	Relation between Geometric and Analytic Jacobian Matrix	41
2.9	Kinematic Control Methods	42
2.9.1	Inverse Differential Kinematics	42
	Singularities	43
	Redundancy	44
2.9.2	Multi-task Inverse Differential Kinematics Control	44
	Multi-task with Equal Priority	45
	Multi-task with Prioritization	45
2.9.3	Inverse Kinematics	48
	Analytical Solution	49
	Numerical Solution	49
	Appropriate Rotation Error	52
2.9.4	Trajectory Control	54
	Position Trajectory Control	54
	Orientation Trajectory Control	54
2.10	Floating Base Kinematics	55
2.10.1	Generalized Velocity and Acceleration	55
2.10.2	Forward Kinematics	56
2.10.3	Differential Kinematics of Floating Base Systems	56
2.10.4	Contacts and Constraints	57
	Point Contacts - Quadruped	58
	Extended Contacts - Humanoid	58
2.10.5	Support Consistent Inverse Kinematics	58
3	Dynamics	61
3.1	Introduction	61
3.2	Foundations from Classical Mechanics	62
3.2.1	Newton's Law for Particles	62
3.2.2	Virtual Displacements	63
3.2.3	Virtual Displacement of Single Rigid Bodies	63
3.2.4	Virtual Displacement of Multi-Body Systems	64
3.2.5	Principle of Virtual Work	64
3.3	Newton-Euler Method	65
3.3.1	Newton-Euler for Single Bodies	65
3.3.2	Newton-Euler for Multi-Body Systems	66
3.4	Lagrange Method	67
3.4.1	Introduction	67

3.4.2	Kinetic Energy	68
3.4.3	Potential Energy	69
3.4.4	External Forces	69
3.4.5	Additional Constraints	69
3.5	Projected Newton-Euler Method	70
3.5.1	Introduction	70
3.5.2	Deriving Generalized Equations of Motion	71
3.5.3	External Forces & Actuation	72
3.6	Summary and Relation between Methods	73
3.7	Dynamics of Floating Base Systems	73
3.7.1	Contact Forces	74
	Soft Contact Model	74
	Contact Forces from Constraints	75
3.7.2	Constraint Consistent Dynamics	75
3.7.3	Contact Switches and Impact Collisions	75
	Impulse Transfer	76
	Energy Loss	76
3.8	Joint-space Dynamic Control	77
3.8.1	Joint Impedance Regulation	77
	Gravity Compensation	77
	Inverse Dynamics Control	77
3.9	Task-space Dynamics Control	78
3.9.1	Multi-task Decomposition	78
3.9.2	End-effector Dynamics	79
3.9.3	End-effector Motion Control	79
	Alternative Notation	79
3.9.4	Operational Space Control	80
3.10	Inverse Dynamics for Floating-Base Systems	81
3.10.1	Quadratic Problems	81
3.10.2	Iterative Null-Space Projection	81
3.10.3	Sequence of Constrained Optimization	82
3.10.4	Task-Space Control for Floating Base Systems as QP	83
3.11	Quasi-static (Virtual Model) Control	83
A	Matlab Code for Examples	87
A.1	Multi-task Control	87
A.2	Inverse Kinematics for Rotations	88
B	Matlab Code for 3D rotations	93
B.1	Euler angles to rotation matrix	93
B.2	Rotation matrix to Euler angles	94

Chapter 1

Introduction

The course "Robot Dynamics" provides an overview on how to model robotic systems and gives a first insight in how to use these models in order to control the systems. It tries to foster the understanding of the similarities between different types of robots, such as robot arms, legged and wheeled machines, or flying systems, that can be modeled using the same techniques. In fact, most robots can be described (accurately enough) by a single body or a set of bodies on which different forces act. However, these forces can come from different sources. A robot arm moving in free space is driven by the actuator forces acting on the joints, while a legged robot additionally encounters interaction forces at its feet and flying vehicles are kept in the air due to aerodynamic forces.

In general, we need to distinguish between two categories of robots, namely **fixed base** and **floating base** systems. The former category includes all kinds of robotic machines that are rigidly bolted to the ground. The classical example are industrial robot arms. Such systems mostly feature an actuator in every joint, which means that the degree of freedom (DOF) is equal to the number of actuators in the systems. The latter category encompasses mobile systems, i.e. robots that have a moving base. In contrast to the robot arm example, the motion of the floating base can typically not directly be controlled through actuators but only through external forces acting on the system such as contact or aerodynamic forces.

This lecture script gives a compact overview about the underlying theory. As the course is still in development, this script is neither complete nor extensive. As general readings for people interested in this topic, we refer to two great books that are available through SpringerLink (free access from ETH), namely

- Handbook of Robotics (Siciliano, Khatib) [7]
<http://link.springer.com/referencework/10.1007/978-3-540-30301-5>
- Robotics – Modelling, Planning and Control (Siciliano, Sciavicco, Villani, Oriolo) [8]
<http://link.springer.com/book/10.1007%2F978-1-84628-642-1>

1.1 Nomenclature

The following list holds as reference table for the entire script. While almost every textbook uses a different nomenclature, we try to stick here to the IEEE standards for the style and use index parameter to ensure a complete description

\mathbf{r}	vector (bold small letter)
\mathbf{B}	matrix (bold capital letter)
$\mathbf{e}_x^{\mathcal{A}}, \mathbf{e}_y^{\mathcal{A}}, \mathbf{e}_z^{\mathcal{A}}$	unitary basis vectors of coordinate frame \mathcal{A}
\mathcal{A}	coordinate frame \mathcal{A} (caligraph letter)
A	origin of coordinate system \mathcal{A}
$\mathbf{e}_x^{\mathcal{I}}, \mathbf{e}_y^{\mathcal{I}}, \mathbf{e}_z^{\mathcal{I}}$	global / inertial / world coordinate system (never moves)
${}^{\mathcal{A}}\mathbf{r}_{AP}$	position vector of point P w.r.t. the origin of frame \mathcal{A} expressed in frame \mathcal{A}
ϕ_{AB}	describes the orientation of frame \mathcal{B} w.r.t. \mathcal{A} , which is $\in SO(3)$
χ_P	stacked parameters of position representation
χ_R	stacked parameters of orientation representation
$\chi_{P,0}$	minimal-dimensional stacked parameters of position representation
$\chi_{R,0}$	minimum-dimensional stacked parameters of orientation representation
\mathbf{v}_P	(absolute) velocity of point P w.r.t. inertial frame
\mathbf{a}_P	(absolute) acceleration of point P w.r.t. inertial frame
ω_{AB}	angular velocity of frame \mathcal{B} w.r.t. \mathcal{A}
$\Omega_{\mathcal{B}} = {}^{\mathcal{I}}\omega_{\mathcal{IB}}$	absolute angular velocity of frame \mathcal{B}
$\Psi_{\mathcal{B}} = \dot{\Omega}_{\mathcal{B}}$	absolute angular acceleration of frame \mathcal{B}
n_q	total number of generalized coordinates, respectively velocities
n_j	number of joints
n_l	number of links
n_{τ}	number of actuated joints
n_b	number of base coordinates
n_c	number of contact points
m	number of end-effector configuration parameters
m_0	number of operational space coordinates = minimal number of end-effector configuration parameters
ϵ	machine precision
\mathbf{x}^*	the star stands for a desired value
\mathbf{q}	generalized coordinates
\mathbf{J}	Jacobian matrix
\mathbf{J}_A	analytical Jacobian matrix
\mathbf{J}_0	basic Jacobian matrix
$\mathbf{N} = \mathcal{N}(\mathbf{J})$	null-space projector matrix

1.2 Operators

There exist a number of special operators such as

$$\text{Cross product/skew/unskew} \quad \mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [\mathbf{a}]_{\times} \mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\text{Euclidean norm} \quad \|\mathbf{a}\| = \sqrt{\mathbf{a}^T \mathbf{a}} = \sqrt{a_1^2 + \dots + a_n^2}$$

$$\text{Absolute vector} \quad |\mathbf{a}| = (|a_1| \quad \dots \quad |a_n|)$$

$$\text{Null space projector} \quad \mathcal{N}(\mathbf{J})$$

Chapter 2

Kinematics

2.1 Introduction

Kinematics is the description of the motion of *points*, *bodies*, and *systems of bodies*. It does only describe **how** things are moving, but **not why**. To describe the kinematics of a moving point, we will refer to position vectors, which are generically defined in \mathbb{R}^3 , and their derivatives. For an extended body, we need to additionally take into account rotations $\phi \in SO(3)$ to completely define its configuration.

In the following, we will first discuss the basics of kinematics by describing the motion of points and single bodies before moving on to serial systems of bodies in section 2.8.

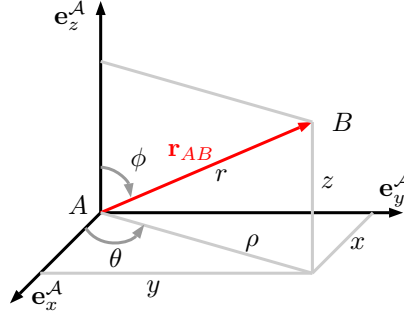


Figure 2.1: Representation of positions using Cartesian, cylindrical, or spherical coordinates.

2.2 Position

The position of a point B relative to point A can be written as

$$\mathbf{r}_{AB}. \quad (2.1)$$

For points in the three dimensional space, positions are represented by vectors $\mathbf{r} \in \mathbb{R}^3$. In order to numerically express the components of a vector, it is necessary to define a reference frame \mathcal{A} and to express the vector in this frame:

$${}^{\mathcal{A}}\mathbf{r}_{AB}. \quad (2.2)$$

The unit vectors $(\mathbf{e}_x^{\mathcal{A}}, \mathbf{e}_y^{\mathcal{A}}, \mathbf{e}_z^{\mathcal{A}})$ of frame \mathcal{A} form an ortho-normal basis of \mathbb{R}^3 .

2.2.1 Representation of Positions

Representing a position in the three dimensional space requires three parameters.

Cartesian coordinates

The most common approach is to work with Cartesian coordinates and hence parameterize the position by

$$\chi_{Pc} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad (2.3)$$

which implies that a position vector is simply given by

$$\mathbf{r} = x\mathbf{e}_x + y\mathbf{e}_y + z\mathbf{e}_z = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (2.4)$$

Cylindrical coordinates

A second approach is to work with cylindrical coordinates

$$\chi_{Pz} = \begin{pmatrix} \rho \\ \theta \\ z \end{pmatrix}, \quad (2.5)$$

which implies that a position vector is given by

$${}_{\mathcal{A}}\mathbf{r} = \begin{pmatrix} \rho \cos \theta \\ \rho \sin \theta \\ z \end{pmatrix}. \quad (2.6)$$

Spherical coordinates

A third method is to use spherical coordinates ¹

$$\chi_{Ps} = \begin{pmatrix} r \\ \theta \\ \phi \end{pmatrix}, \quad (2.7)$$

which implies that a position vector is given by

$${}_{\mathcal{A}}\mathbf{r} = \begin{pmatrix} r \cos \theta \sin \phi \\ r \sin \theta \sin \phi \\ r \cos \phi \end{pmatrix}. \quad (2.8)$$

Note: All the previously introduced parameterizations require three parameters to describe a position in 3D space, meaning that they are at the same time also minimal representations. This will be different for rotations as shown in section 2.4.5. In most of the cases, people work with Cartesian coordinates due to the simple properties for vector calculus.

2.3 Linear Velocity

The velocity of point B relative to point A is given by

$$\dot{\mathbf{r}}_{AB}. \quad (2.9)$$

For three dimensional space, velocities are represented by vectors $\dot{\mathbf{r}} \in \mathbb{R}^3$. For performing vector algebra, the same rules as introduced for the positions need to hold.

2.3.1 Representation of Linear Velocities

There exists a linear mapping $\mathbf{E}_P(\chi)$ between velocities $\dot{\mathbf{r}}$ and the derivatives of the representation $\dot{\chi}_P$:

$$\dot{\mathbf{r}} = \mathbf{E}_P(\chi_P) \dot{\chi}_P \quad (2.10)$$

$$\dot{\chi}_P = \mathbf{E}_P^{-1}(\chi_P) \dot{\mathbf{r}} \quad (2.11)$$

¹What is drawn in Fig. 2.1 is the azimuthal angle θ , and polar angle ϕ . This is often referred to as "mathematical notation", whereby we use ϕ instead of φ in order to limit conflicts with other use of φ in this lecture notes. The common notation in physics (which is also the ISO standard) uses φ for the azimuthal angle and θ for the polar angle. Be also careful when using Matlab as it outputs typically elevation angle instead of azimuthal angle.

Cartesian Coordinates

For Cartesian coordinates, the mapping is simply the identity:

$$\mathbf{E}_{Pc}(\chi_{Pc}) = \mathbf{E}_{Pc}^{-1}(\chi_{Pc}) = \mathbb{I} \quad (2.12)$$

Cylindrical Coordinates

For cylindrical coordinates we get

$$\dot{\mathbf{r}}(\chi_{Pz}) = \begin{pmatrix} \dot{\rho} \cos \theta - \rho \dot{\theta} \sin \theta \\ \dot{\rho} \sin \theta + \rho \dot{\theta} \cos \theta \\ \dot{z} \end{pmatrix}, \quad (2.13)$$

which can be solved for

$$\dot{\chi}_{Pz} = \begin{pmatrix} \dot{\rho} \\ \dot{\theta} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} \dot{x} \cos \theta + \dot{y} \sin \theta \\ -\dot{x} \sin \theta / \rho + \dot{y} \cos \theta / \rho \\ \dot{z} \end{pmatrix} = \underbrace{\begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta / \rho & \cos \theta / \rho & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{E}_{Pz}^{-1}} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix}. \quad (2.14)$$

The inverse is

$$\mathbf{E}_{Pz}(\chi_{Pz}) = \frac{\partial \mathbf{r}(\chi_{Pz})}{\partial \chi_{Pz}} = \begin{bmatrix} \cos \theta & -\rho \sin \theta & 0 \\ \sin \theta & \rho \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (2.15)$$

Spherical Coordinates

Using the same approach for spherical coordinates results in:

$$\mathbf{E}_{Ps} = \begin{bmatrix} \cos \theta \sin \phi & -r \sin \phi \sin \theta & r \cos \phi \cos \theta \\ \sin \phi \sin \theta & r \cos \theta \sin \phi & r \cos \phi \sin \theta \\ \cos \phi & 0 & -r \sin \phi \end{bmatrix}, \quad (2.16)$$

$$\mathbf{E}_{Ps}^{-1} = \begin{bmatrix} \cos \theta \sin \phi & \sin \phi \sin \theta & \cos \phi \\ -\sin \theta / (r \sin \phi) & \cos \theta / (r \sin \phi) & 0 \\ (\cos \phi \cos \theta) / r & (\cos \phi \sin \theta) / r & -\sin \phi / r \end{bmatrix}. \quad (2.17)$$

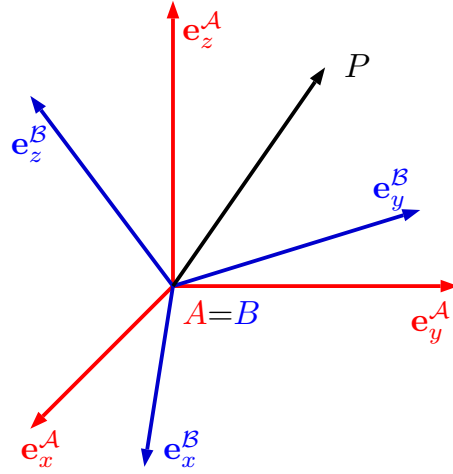


Figure 2.2: Generic 3D rotation between two frames \mathcal{A} and \mathcal{B} .

2.4 Rotation

While the configuration of a point is fully described by a position, bodies additionally require a *rotation* to define their pose. As theoretical abstraction of rotations,

$$\phi_{\mathcal{AB}} \in SO(3) \quad (2.18)$$

is often used to indicate the orientation of body fixed frame \mathcal{B} with respect to a reference frame \mathcal{A} . It is important to understand that, since $\phi_{\mathcal{AB}}$ lives in $SO(3)$, there is no numerical equivalent to a position such as "angular position". Instead, the orientation $\phi_{\mathcal{AB}}$ can be parametrized in several ways. For a better understanding, we will start by defining the mapping between coordinate frames by means of *rotation matrices* and then show how these relate to different parameterizations.

2.4.1 Rotation Matrices

Consider the situation depicted in Fig. 2.2 with a reference frame \mathcal{A} . The position vector of a point P which is fixed in this frame is written as

$${}_{\mathcal{A}}\mathbf{r}_{AP} = \begin{pmatrix} {}_{\mathcal{A}}r_{AP_x} \\ {}_{\mathcal{A}}r_{AP_y} \\ {}_{\mathcal{A}}r_{AP_z} \end{pmatrix}. \quad (2.19)$$

Consider now a reference frame \mathcal{B} which is rotated w.r.t. \mathcal{A} . The origin B of frame \mathcal{B} coincides with the origin A of frame \mathcal{A} . The position vector of point P, this time expressed in frame \mathcal{B} , is

$${}_{\mathcal{B}}\mathbf{r}_{AP} = \begin{pmatrix} {}_{\mathcal{B}}r_{AP_x} \\ {}_{\mathcal{B}}r_{AP_y} \\ {}_{\mathcal{B}}r_{AP_z} \end{pmatrix}. \quad (2.20)$$

By writing the unit vectors of \mathcal{B} expressed in frame \mathcal{A} as $[{}_{\mathcal{A}}\mathbf{e}_x^{\mathcal{B}}, {}_{\mathcal{A}}\mathbf{e}_y^{\mathcal{B}}, {}_{\mathcal{A}}\mathbf{e}_z^{\mathcal{B}}]$, we can write the mapping between the two position vectors ${}_{\mathcal{A}}\mathbf{r}_{AP}$ and ${}_{\mathcal{B}}\mathbf{r}_{AP}$ as

$${}_{\mathcal{A}}\mathbf{r}_{AP} = {}_{\mathcal{A}}\mathbf{e}_x^{\mathcal{B}} \cdot {}_{\mathcal{B}}r_{AP_x} + {}_{\mathcal{A}}\mathbf{e}_y^{\mathcal{B}} \cdot {}_{\mathcal{B}}r_{AP_y} + {}_{\mathcal{A}}\mathbf{e}_z^{\mathcal{B}} \cdot {}_{\mathcal{B}}r_{AP_z}. \quad (2.21)$$

The mapping shown in (2.21) can be rewritten in compact form as

$$\begin{aligned}\mathcal{A}\mathbf{r}_{AP} &= [\mathcal{A}\mathbf{e}_x^{\mathcal{B}} \quad \mathcal{A}\mathbf{e}_y^{\mathcal{B}} \quad \mathcal{A}\mathbf{e}_z^{\mathcal{B}}] \cdot \mathcal{B}\mathbf{r}_{AP} \\ &= \mathbf{C}_{\mathcal{AB}} \cdot \mathcal{B}\mathbf{r}_{AP}.\end{aligned}\tag{2.22}$$

The term $\mathbf{C}_{\mathcal{AB}}$ is a 3×3 matrix called *rotation matrix*. Since the columns of $\mathbf{C}_{\mathcal{AB}}$ are orthogonal unit vectors, $\mathbf{C}_{\mathcal{AB}}$ is orthogonal, meaning that

$$\mathbf{C}_{\mathcal{AB}}^T \cdot \mathbf{C}_{\mathcal{AB}} = \mathbb{I}_3\tag{2.23}$$

A consequence of (2.23) is that $\mathbf{C}_{\mathcal{BA}} = \mathbf{C}_{\mathcal{AB}}^{-1} = \mathbf{C}_{\mathcal{AB}}^T$. The rotation matrix $\mathbf{C}_{\mathcal{AB}}$ belongs to the *special orthogonal group* $SO(3)$. This requires to apply a special type of algebra that is different from classical \mathbb{R}^3 vector algebra.

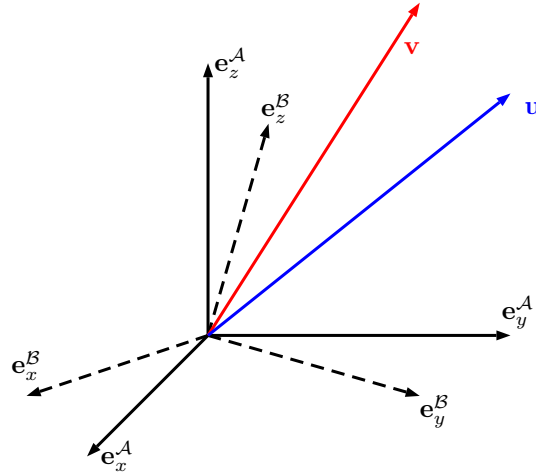


Figure 2.3: A *passive* rotation can be interpreted as the rotation of a coordinate frame, and an *active* rotation as the rotation of an object \mathbf{u} which yields \mathbf{v} .

2.4.2 Active vs. Passive Rotation

Rotations can have two different interpretations, which lead to the definition of the so-called *active* and *passive* rotations.

Passive Rotation

Passive rotations, also known as rotation transformations, correspond to a mapping between coordinate frames as shown in (2.21). A passive rotation $\mathbf{C}_{\mathcal{A}\mathcal{B}}$ maps the same object \mathbf{u} from frame \mathcal{B} to frame \mathcal{A} :

$${}_{\mathcal{A}}\mathbf{u} = \mathbf{C}_{\mathcal{A}\mathcal{B}} \cdot {}_{\mathcal{B}}\mathbf{u} \quad (2.24)$$

Active Rotation

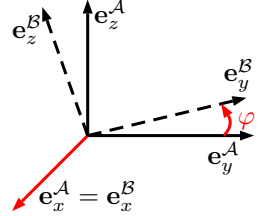
An active rotation, often indicated with a 3×3 matrix \mathbf{R} , is an *operator* that rotates a vector ${}_{\mathcal{A}}\mathbf{u}$ to a vector ${}_{\mathcal{A}}\mathbf{v}$ in the same reference frame \mathcal{A} :

$${}_{\mathcal{A}}\mathbf{v} = \mathbf{R} \cdot {}_{\mathcal{A}}\mathbf{u}. \quad (2.25)$$

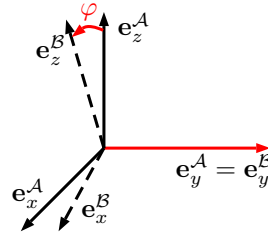
Active rotations are not very relevant in robot dynamics and are hence not used in the course of this lecture.

2.4.3 Elementary Rotations

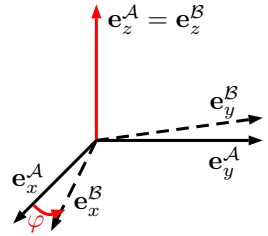
The most simple and at the same time most often appearing rotations are elementary rotations, i.e. rotations around one of the basis vectors \mathbf{e}_x^A , \mathbf{e}_y^A or \mathbf{e}_z^A . Given a rotation angle φ , the three elementary rotations are:



$$\mathbf{C}_{AB} = \mathbf{C}_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \varphi & -\sin \varphi \\ 0 & \sin \varphi & \cos \varphi \end{bmatrix} \quad (2.26)$$



$$\mathbf{C}_{AB} = \mathbf{C}_y(\varphi) = \begin{bmatrix} \cos \varphi & 0 & \sin \varphi \\ 0 & 1 & 0 \\ -\sin \varphi & 0 & \cos \varphi \end{bmatrix} \quad (2.27)$$



$$\mathbf{C}_{AB} = \mathbf{C}_z(\varphi) = \begin{bmatrix} \cos \varphi & -\sin \varphi & 0 \\ \sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.28)$$

2.4.4 Composition of Rotations

Consider three reference frames \mathcal{A} , \mathcal{B} and \mathcal{C} . The coordinates of vector \mathbf{u} can be mapped from \mathcal{B} to \mathcal{A} by writing

$${}_{\mathcal{A}}\mathbf{u} = \mathbf{C}_{AB} \cdot {}_{\mathcal{B}}\mathbf{u}. \quad (2.29)$$

We can also write

$${}_{\mathcal{B}}\mathbf{u} = \mathbf{C}_{BC} \cdot {}_{\mathcal{C}}\mathbf{u}. \quad (2.30)$$

By combining the last two equations, we can write

$$\begin{aligned} {}_{\mathcal{A}}\mathbf{u} &= \mathbf{C}_{AB} \cdot (\mathbf{C}_{BC} \cdot {}_{\mathcal{C}}\mathbf{u}) \\ &= \mathbf{C}_{AC} \cdot {}_{\mathcal{C}}\mathbf{u}. \end{aligned} \quad (2.31)$$

The resulting rotation matrix $\mathbf{C}_{AC} = \mathbf{C}_{AB} \cdot \mathbf{C}_{BC}$ can be interpreted as the rotation obtained by rotating frame \mathcal{A} until it coincides with frame \mathcal{B} , and then rotating frame \mathcal{B} until it coincides with frame \mathcal{C} .

2.4.5 Representation of Rotations

As discussed in the previous sections, generic rotations in the three-dimensional space are represented by 3×3 rotation matrices, i.e. by means of 9 parameters. These parameters, however, are not independent, but constrained by the orthogonality conditions shown in (2.23). Hence, only three independent parameters such as *Euler angles* are needed to obtain a minimal representation of rotations in space. Other non-minimal representations can be derived, namely the *angle-axis* and the *unit quaternion* representation. We will briefly discuss advantages and disadvantages of each parameterization, as well as derive the mapping from one implementation to the other. For a more detailed analysis of three dimensional rotations (which goes beyond the scope of this lecture), the reader is referred to [2].

Euler Angles

A rotation in space can be understood as a sequence of three elementary rotations defined in (2.26) to (2.28). To fully describe all possible orientations, two successive rotations should not be made around parallel axes. When the first and third rotations are made around the same axis, the parameterization is called *proper* Euler angles. When all three angles are different, we typically refer to *Tait–Bryan*, *Cardan* or *roll-pitch-yaw* angles. The latter ones are often used in robotics.

Note: Please note again that there applies a different type of algebra to rotations than what we know from typical position vectors. Hence, never add, subtract or simply multiply Euler angles, angle-axis, or quaternions.

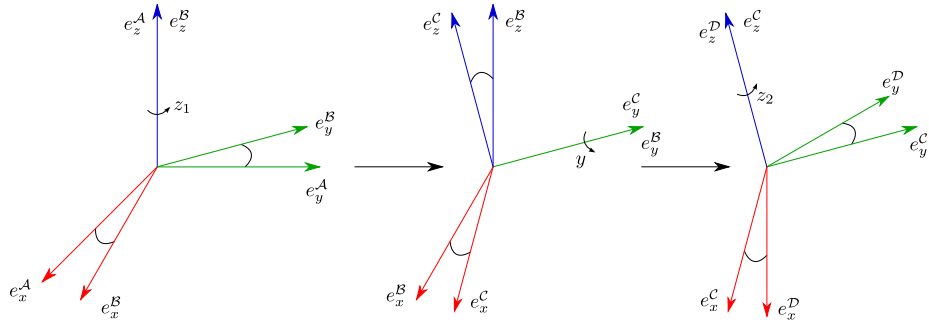


Figure 2.4: ZYZ Euler Angles as three successive rotations around z , y , and z axes. Rotation from \mathcal{A} -frame to \mathcal{D} -frame: $(z-y'-z'')$ – (yaw–pitch–yaw)

ZYZ Euler Angles ZYZ Euler Angles are also known as *proper* Euler Angles. The rotation angles can be collected in a parameter vector

$$\chi_{R,euler ZYZ} = \begin{pmatrix} z_1 \\ y \\ z_2 \end{pmatrix}. \quad (2.32)$$

The resulting rotation matrix is obtained by a concatenation of elementary rotations (see Fig.2.4) given by

$$\begin{aligned} \mathbf{C}_{\mathcal{AD}} &= \mathbf{C}_{\mathcal{AB}}(z_1) \mathbf{C}_{\mathcal{BC}}(y) \mathbf{C}_{\mathcal{CD}}(z_2) \Rightarrow \mathcal{A}\mathbf{r} = \mathbf{C}_{\mathcal{AD}}\mathcal{D}\mathbf{r} \\ &= \begin{bmatrix} \cos z_1 & -\sin z_1 & 0 \\ \sin z_1 & \cos z_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos y & 0 & \sin y \\ 0 & 1 & 0 \\ -\sin y & 0 & \cos y \end{bmatrix} \begin{bmatrix} \cos z_2 & -\sin z_2 & 0 \\ \sin z_2 & \cos z_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c_y c_{z_1} c_{z_2} - s_{z_1} s_{z_2} & -c_{z_2} s_{z_1} - c_y c_{z_1} s_{z_2} & c_{z_1} s_y \\ c_{z_1} s_{z_2} + c_y c_{z_2} s_{z_1} & c_{z_1} c_{z_2} - c_y s_{z_1} s_{z_2} & s_y s_{z_1} \\ -c_{z_2} s_y & s_y s_{z_2} & c_y \end{bmatrix}. \end{aligned} \quad (2.33)$$

Analyzing (2.33) allows to find the solution of the inverse problem. Given a rotation matrix

$$\mathbf{C}_{\mathcal{AD}} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}, \quad (2.34)$$

the ZYZ Euler angles are given by

$$\chi_{R,euler ZYZ} = \begin{pmatrix} z_1 \\ y \\ z_2 \end{pmatrix} = \begin{pmatrix} \text{atan2}(c_{23}, c_{13}) \\ \text{atan2}\left(\sqrt{c_{13}^2 + c_{23}^2}, c_{33}\right) \\ \text{atan2}(c_{32}, -c_{31}) \end{pmatrix}. \quad (2.35)$$

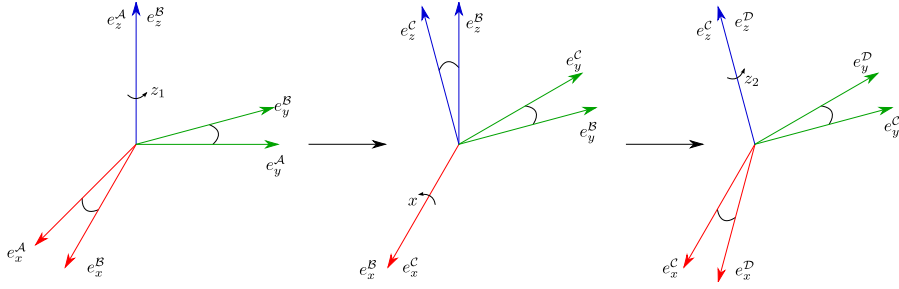


Figure 2.5: ZXZ Euler Angles as three successive rotations around z, x, and z axes. Rotation from \mathcal{A} -frame to \mathcal{D} -frame: (z-x'-z'') – (yaw-roll-yaw)

ZXZ Euler Angles ZXZ Euler Angles are also known as *proper* Euler Angles. The rotation angles can be collected in a parameter vector

$$\chi_{R,eulerZXZ} = \begin{pmatrix} z_1 \\ x \\ z_2 \end{pmatrix}. \quad (2.36)$$

The resulting rotation matrix is obtained by a concatenation of elementary rotations (Fig.2.5) given by

$$\begin{aligned} \mathbf{C}_{AD} &= \mathbf{C}_{AB}(z_1) \mathbf{C}_{BC}(x) \mathbf{C}_{CD}(z_2) \Rightarrow \mathcal{A}\mathbf{r} = \mathbf{C}_{AD}\mathcal{D}\mathbf{r} \\ &= \begin{bmatrix} \cos z_1 & -\sin z_1 & 0 \\ \sin z_1 & \cos z_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos x & -\sin x \\ 0 & \sin x & \cos x \end{bmatrix} \begin{bmatrix} \cos z_2 & -\sin z_2 & 0 \\ \sin z_2 & \cos z_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c_{z1}c_{z2} - c_x s_{z1}s_{z2} & -c_{z1}s_{z2} - c_x c_{z2}s_{z1} & s_x s_{z1} \\ c_{z2}s_{z1} + c_x c_{z1}s_{z2} & c_x c_{z1}c_{z2} - s_{z1}s_{z2} & -c_{z1}s_x \\ s_x s_{z2} & c_{z2}s_x & c_x \end{bmatrix}. \end{aligned} \quad (2.37)$$

Analyzing (2.37) allows to find the solution of the inverse problem. Given a rotation matrix

$$\mathbf{C}_{AD} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}, \quad (2.38)$$

the Euler Angles are given by

$$\chi_{R,eulerZXZ} = \begin{pmatrix} z_1 \\ x \\ z_2 \end{pmatrix} = \begin{pmatrix} \text{atan2}(c_{13}, -c_{23}) \\ \text{atan2}\left(\sqrt{c_{13}^2 + c_{23}^2}, c_{33}\right) \\ \text{atan2}(c_{31}, c_{32}) \end{pmatrix}. \quad (2.39)$$

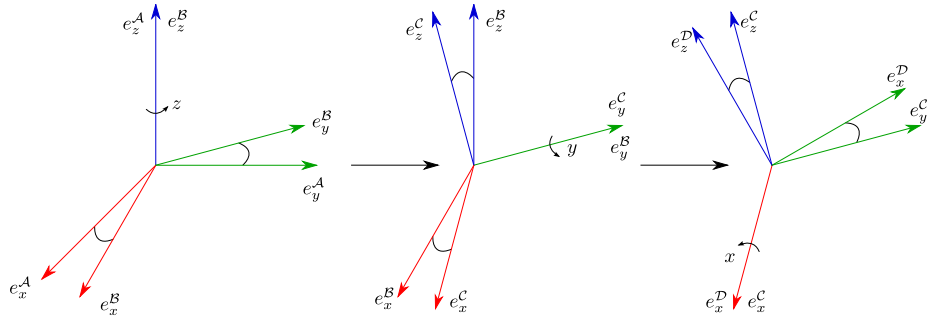


Figure 2.6: ZYX Euler Angles as three successive rotations around z , y , and x . Rotation from \mathcal{A} -frame to \mathcal{D} -frame: (z - y - x) – (yaw–pitch–roll)

ZYX Euler Angles ZYX Euler Angles, also known as Tait-Bryan angles, are often used for flying vehicles and called yaw-pitch-roll. The rotation angles can be collected in a parameter vector

$$\chi_{R,eulerZYX} = \begin{pmatrix} z \\ y \\ x \end{pmatrix}. \quad (2.40)$$

The resulting rotation matrix is obtained by a concatenation of elementary rotations given by

$$\begin{aligned} \mathbf{C}_{\mathcal{AD}} &= \mathbf{C}_{\mathcal{AB}}(z) \mathbf{C}_{\mathcal{BC}}(y) \mathbf{C}_{\mathcal{CD}}(x) \Rightarrow \mathcal{A}\mathbf{r} = \mathbf{C}_{\mathcal{AD}}\mathcal{D}\mathbf{r} \\ &= \begin{bmatrix} \cos z & -\sin z & 0 \\ \sin z & \cos z & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos y & 0 & \sin y \\ 0 & 1 & 0 \\ -\sin y & 0 & \cos y \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos x & -\sin x \\ 0 & \sin x & \cos x \end{bmatrix} \\ &= \begin{bmatrix} c_y c_z & c_z s_x s_y - c_x s_z & s_x s_z + c_x c_z s_y \\ c_y s_z & c_x c_z + s_x s_y s_z & c_x s_y s_z - c_z s_x \\ -s_y & c_y s_x & c_x c_y \end{bmatrix}. \end{aligned} \quad (2.41)$$

Given a rotation matrix as in (2.38), the inverse solution is

$$\chi_{R,eulerZYX} = \begin{pmatrix} z \\ y \\ x \end{pmatrix} = \begin{pmatrix} \text{atan2}(c_{21}, c_{11}) \\ \text{atan2}(-c_{31}, \sqrt{c_{32}^2 + c_{33}^2}) \\ \text{atan2}(c_{32}, c_{33}) \end{pmatrix}. \quad (2.42)$$

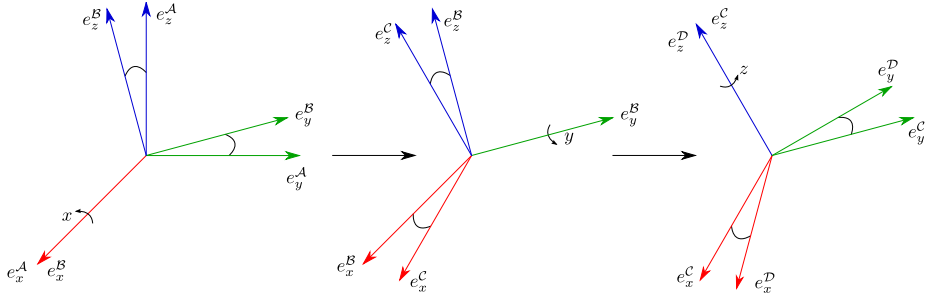


Figure 2.7: XYZ Euler Angles as three successive rotations around x, y, and z. Rotation from \mathcal{A} -frame to \mathcal{D} -frame: (x-y'-z'') – (roll-pitch-yaw)

XYZ Euler Angles XYZ Euler Angles are also known as Cardan angles. The rotation angles can be collected in a parameter vector

$$\chi_{R,eulerXYZ} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}. \quad (2.43)$$

The resulting rotation matrix is obtained by a concatenation of elementary rotations (see Fig.2.7) given by

$$\begin{aligned} \mathbf{C}_{\mathcal{AD}} &= \mathbf{C}_{\mathcal{AB}}(x) \mathbf{C}_{\mathcal{BC}}(y) \mathbf{C}_{\mathcal{CD}}(z) \Rightarrow \mathcal{A}\mathbf{r} = \mathbf{C}_{\mathcal{AD}} \mathbf{r} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos x & -\sin x \\ 0 & \sin x & \cos x \end{bmatrix} \begin{bmatrix} \cos y & 0 & \sin y \\ 0 & 1 & 0 \\ -\sin y & 0 & \cos y \end{bmatrix} \begin{bmatrix} \cos z & -\sin z & 0 \\ \sin z & \cos z & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.44) \\ &= \begin{bmatrix} c_y c_z & -c_y s_z & s_y \\ c_x s_z + c_z s_x s_y & c_x c_z - s_x s_y s_z & -c_y s_x \\ s_x s_z - c_x c_z s_y & c_z s_x + c_x s_y s_z & c_x c_y \end{bmatrix}. \end{aligned}$$

Given a rotation matrix as in (2.38), the inverse solution is

$$\chi_{R,eulerXYZ} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \text{atan2}(-c_{23}, c_{33}) \\ \text{atan2}(c_{13}, \sqrt{c_{11}^2 + c_{12}^2}) \\ \text{atan2}(-c_{12}, c_{11}) \end{pmatrix}. \quad (2.45)$$

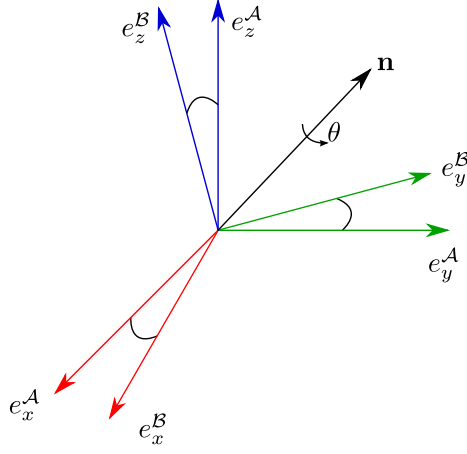


Figure 2.8: The angle-axis representation defines the orientation of two coordinate frames as a rotation of an angle θ around an axis \mathbf{n} .

Angle Axis

The *angle-axis* is a non-minimal implementation of rotations which is defined by an angle θ and an axis \mathbf{n} . The vector $\mathbf{n} \in \mathbb{R}^3$ defines the direction around which the rotation is made, while the scalar $\theta \in \mathbb{R}$ defines the rotation magnitude:

$$\chi_{R, AngleAxis} = \begin{pmatrix} \theta \\ \mathbf{n} \end{pmatrix} \quad (2.46)$$

This representation features four parameters and the unitary length constraint $\|\mathbf{n}\| = 1$. It is possible to combine these two quantities to obtain a *rotation vector*, or *Euler vector*, defined as

$$\varphi = \theta \cdot \mathbf{n} \in \mathbb{R}^3. \quad (2.47)$$

It is important to note that, although φ belongs to \mathbb{R}^3 , the composition of two rotations is not equal to the result of the sum of the two corresponding rotation vectors.

With angle axis parameters φ_{AB} , the rotation matrix results to

$$\mathbf{C}_{AB} = \begin{bmatrix} n_x^2(1 - c_\theta) + c_\theta & n_x n_y(1 - c_\theta) - n_z s_\theta & n_x n_z(1 - c_\theta) + n_y s_\theta \\ n_x n_y(1 - c_\theta) + n_z s_\theta & n_y^2(1 - c_\theta) + c_\theta & n_y n_z(1 - c_\theta) - n_x s_\theta \\ n_x n_z(1 - c_\theta) - n_y s_\theta & n_y n_z(1 - c_\theta) + n_x s_\theta & n_z^2(1 - c_\theta) + c_\theta \end{bmatrix}. \quad (2.48)$$

Given a rotation matrix as in (2.38), the angle axis parameters are:

$$\theta = \cos^{-1} \left(\frac{c_{11} + c_{22} + c_{33} - 1}{2} \right), \quad (2.49)$$

$$\mathbf{n} = \frac{1}{2\sin(\theta)} \begin{pmatrix} c_{32} - c_{23} \\ c_{13} - c_{31} \\ c_{21} - c_{12} \end{pmatrix}. \quad (2.50)$$

As it can be seen from (2.50), this representation encounters a problem for $\theta = 0$ and $\theta = \pi$ since $\sin(\theta) = 0$, meaning that the rotation vector is not defined. For $\theta = 0$ it can have any direction, for $\theta = \pi$ it can point in two opposite directions.

Unit Quaternions

A non-minimal representation of rotations which does not suffer from the disadvantage encountered with the angle axis is provided by *unit quaternions*, also known as Euler parameters. Considering a rotational vector $\boldsymbol{\varphi} \in \mathbb{R}^3$, a unit quaternion $\boldsymbol{\xi}$ is defined by

$$\boldsymbol{\chi}_{R,quat} = \boldsymbol{\xi} = \begin{pmatrix} \xi_0 \\ \check{\boldsymbol{\xi}} \end{pmatrix} \in \mathbb{H}, \quad (2.51)$$

where

$$\begin{aligned} \xi_0 &= \cos\left(\frac{\|\boldsymbol{\varphi}\|}{2}\right) = \cos\left(\frac{\theta}{2}\right) \\ \check{\boldsymbol{\xi}} &= \sin\left(\frac{\|\boldsymbol{\varphi}\|}{2}\right) \frac{\boldsymbol{\varphi}}{\|\boldsymbol{\varphi}\|} = \sin\left(\frac{\theta}{2}\right) \mathbf{n} = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}. \end{aligned} \quad (2.52)$$

The first parameter ξ_0 is called the real part of the quaternion, the latter $\check{\boldsymbol{\xi}}$ the imaginary part. The unit quaternion fulfills the constraint

$$\xi_0^2 + \xi_1^2 + \xi_2^2 + \xi_3^2 = 1. \quad (2.53)$$

Similarly to the angle axis, the rotation matrix calculated from quaternions is

$$\begin{aligned} \mathbf{C}_{\mathcal{AD}} &= \mathbb{I}_{3 \times 3} + 2\xi_0 [\check{\boldsymbol{\xi}}]_{\times} + 2[\check{\boldsymbol{\xi}}]_{\times}^2 = (2\xi_0^2 - 1) \mathbb{I}_{3 \times 3} + 2\xi_0 [\check{\boldsymbol{\xi}}]_{\times} + 2\check{\boldsymbol{\xi}}\check{\boldsymbol{\xi}}^T \\ &= \begin{bmatrix} \xi_0^2 + \xi_1^2 - \xi_2^2 - \xi_3^2 & 2\xi_1\xi_2 - 2\xi_0\xi_3 & 2\xi_0\xi_2 + 2\xi_1\xi_3 \\ 2\xi_0\xi_3 + 2\xi_1\xi_2 & \xi_0^2 - \xi_1^2 + \xi_2^2 - \xi_3^2 & 2\xi_2\xi_3 - 2\xi_0\xi_1 \\ 2\xi_1\xi_3 - 2\xi_0\xi_2 & 2\xi_0\xi_1 + 2\xi_2\xi_3 & \xi_0^2 - \xi_1^2 - \xi_2^2 + \xi_3^2 \end{bmatrix}. \end{aligned} \quad (2.54)$$

Given a rotation matrix as in (2.38), the corresponding quaternions are

$$\boldsymbol{\chi}_{R,quat} = \boldsymbol{\xi}_{\mathcal{AD}} = \frac{1}{2} \begin{pmatrix} \sqrt{c_{11} + c_{22} + c_{33} + 1} \\ \text{sgn}(c_{32} - c_{23})\sqrt{c_{11} - c_{22} - c_{33} + 1} \\ \text{sgn}(c_{13} - c_{31})\sqrt{c_{22} - c_{33} - c_{11} + 1} \\ \text{sgn}(c_{21} - c_{12})\sqrt{c_{33} - c_{11} - c_{22} + 1} \end{pmatrix}. \quad (2.55)$$

In this formulation, $\text{sgn}(x) = 1$ for $x \geq 0$ and $\text{sgn}(x) = -1$ for $x < 0$. In this formulation we implicitly assume $c_{11} + c_{22} + c_{33} + 1 \geq 0$, which corresponds to an angle $[-\pi, \pi]$ and thus any rotation can be described.

When working with quaternions, there exists a special algebra that allows to directly work with the quaternion parameterization (and not only the rotation matrices). The interested reader should have a look at [9], which provides a very complete, yet compact and well understandable introduction to quaternions. In short, the following rules are important. For inversion with $\boldsymbol{\xi}^{-1}$ parameterizing $\mathbf{C}^{-1} = \mathbf{C}^T$ it holds that:

$$\boldsymbol{\xi} = \begin{pmatrix} \xi \\ \check{\boldsymbol{\xi}} \end{pmatrix} \xrightarrow{\text{inverse}} \boldsymbol{\xi}^{-1} = \boldsymbol{\xi}^T = \begin{pmatrix} \xi \\ -\check{\boldsymbol{\xi}} \end{pmatrix} \quad (2.56)$$

If $\boldsymbol{\xi}_{\mathcal{AB}}$ and $\boldsymbol{\xi}_{\mathcal{BC}}$ represent the quaternions corresponding to $\mathbf{C}_{\mathcal{AB}}$ and $\mathbf{C}_{\mathcal{BC}}$, their multi-

plication associated with $\mathbf{C}_{AC} = \mathbf{C}_{AB}\mathbf{C}_{BC}$ is

$$\xi_{AC} = \xi_{AB} \otimes \xi_{BC} = \begin{pmatrix} \xi_{0,AB} \cdot \xi_{0,BC} - \check{\xi}_{AB}^T \cdot \check{\xi}_{BC} \\ \xi_{0,AB} \cdot \check{\xi}_{BC} + \xi_{0,BC} \cdot \check{\xi}_{AB} + [\check{\xi}_{AB}]_{\times} \cdot \check{\xi}_{BC} \end{pmatrix} \quad (2.57)$$

$$= \underbrace{\begin{bmatrix} \xi_0 & -\check{\xi}^T \\ \check{\xi} & \xi_0 \mathbb{I} + [\check{\xi}]_{\times} \end{bmatrix}}_{\mathbf{M}_l(\xi_{AB})} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}_{BC} \quad (2.58)$$

$$= \begin{bmatrix} \xi_0 & -\xi_1 & -\xi_2 & -\xi_3 \\ \xi_1 & \xi_0 & -\xi_3 & \xi_2 \\ \xi_2 & \xi_3 & \xi_0 & -\xi_1 \\ \xi_3 & -\xi_2 & \xi_1 & \xi_0 \end{bmatrix}_{AB} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}_{BC}, \quad (2.59)$$

whereby $\mathbf{M}_l(\xi)$ represents the left matrix of a quaternion. The same quaternion multiplication can also be written using the right quaternion, which gives:

$$\xi_{AB} \otimes \xi_{BC} = \underbrace{\begin{bmatrix} \xi_0 & -\check{\xi}^T \\ \check{\xi} & \xi_0 \mathbb{I} - [\check{\xi}]_{\times} \end{bmatrix}}_{\mathbf{M}_r(\xi_{BC})} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}_{AB} \quad (2.60)$$

$$= \begin{bmatrix} \xi_0 & -\xi_1 & -\xi_2 & -\xi_3 \\ \xi_1 & \xi_0 & \xi_3 & -\xi_2 \\ \xi_2 & -\xi_3 & \xi_0 & \xi_1 \\ \xi_3 & \xi_2 & -\xi_1 & \xi_0 \end{bmatrix}_{BC} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}_{AB}, \quad (2.61)$$

Using the special algebra for quaternions can be directly used to rotate vectors. The pure (imaginary) quaternion of a coordinate vector ${}_B\mathbf{r}$ expressed in frame \mathcal{B} is given by

$$\mathbf{p}({}_B\mathbf{r}) = \begin{pmatrix} 0 \\ {}_B\mathbf{r} \end{pmatrix}. \quad (2.62)$$

Given the unit quaternion ξ_{AB} representing the orientation of \mathcal{A} w.r.t. \mathcal{B} , one can show that [9]:

$$\mathbf{p}({}_A\mathbf{r}) = \xi_{AB} \otimes \mathbf{p}({}_B\mathbf{r}) \otimes \xi_{AB}^T \quad (2.63)$$

$$= \mathbf{M}_l(\xi_{AB}) \mathbf{M}_r(\xi_{AB}^T) \mathbf{p}({}_B\mathbf{r}) \quad (2.64)$$

Example 2.4.1: Rotation with quaternion multiplication

Given a vector ${}_A\mathbf{r} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ in frame \mathcal{A} . What is the vector expressed in frame \mathcal{B} , which is rotated by $\theta = \pi/3$ w.r.t. \mathcal{A} around the x-axis?

The quaternion corresponding to the rotation from \mathcal{B} to \mathcal{A} is

$$\xi_{\mathcal{AB}} = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\mathbf{n} \end{pmatrix}, = \frac{1}{2} \begin{pmatrix} \sqrt{3} \\ 1 \\ 0 \\ 0 \end{pmatrix}. \quad (2.65)$$

Hence, the quaternion corresponding to the inverse rotation is

$$\xi_{\mathcal{BA}} = \xi_{\mathcal{AB}}^T = \frac{1}{2} \begin{pmatrix} \sqrt{3} \\ -1 \\ 0 \\ 0 \end{pmatrix} \quad (2.66)$$

In order to express vector \mathbf{r} in \mathcal{B} we can apply the following calculation:

$$\mathbf{p}({}_{\mathcal{B}}\mathbf{r}) = \xi_{\mathcal{BA}} \otimes \mathbf{p}({}_{\mathcal{A}}\mathbf{r}) \otimes \xi_{\mathcal{BA}}^T \quad (2.67)$$

$$= \mathbf{M}_l(\xi_{\mathcal{BA}}) \mathbf{M}_r(\xi_{\mathcal{BA}}^T) \mathbf{p}({}_{\mathcal{A}}\mathbf{r}) \quad (2.68)$$

$$= \frac{1}{2} \begin{bmatrix} \sqrt{3} & 1 & 0 & 0 \\ -1 & \sqrt{3} & 0 & 0 \\ 0 & 0 & \sqrt{3} & 1 \\ 0 & 0 & -1 & \sqrt{3} \end{bmatrix} \frac{1}{2} \begin{bmatrix} \sqrt{3} & -1 & 0 & 0 \\ 1 & \sqrt{3} & 0 & 0 \\ 0 & 0 & \sqrt{3} & 1 \\ 0 & 0 & -1 & \sqrt{3} \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \frac{1}{2} \\ -\frac{\sqrt{3}}{2} \end{pmatrix} \quad (2.69)$$

2.5 Angular Velocity

Consider a frame \mathcal{B} which is moving with respect to a fixed frame \mathcal{A} . The *angular velocity* ${}_{\mathcal{A}}\omega_{\mathcal{AB}}$, which describes the rotational motion of \mathcal{B} w.r.t. \mathcal{A} , is defined by the limit

$${}_{\mathcal{A}}\omega_{\mathcal{AB}} = \lim_{\epsilon \rightarrow 0} \frac{{}_{\mathcal{A}}\varphi_{\mathcal{B}(t)\mathcal{B}(t+\epsilon)}}{\epsilon}. \quad (2.70)$$

As discussed in the last section, a rotational vector φ is, in general, not a proper vector. However, for $\epsilon \rightarrow 0$, the angular velocity is defined as the ratio of a proper vector ${}_{\mathcal{A}}\varphi_{\mathcal{B}(t)\mathcal{B}(t+\epsilon)}$ and a scalar. Hence, angular velocities can be summed according to the rules of vector summation. Consequently, the relative rotation of \mathcal{A} w.r.t. \mathcal{B} is

$$\omega_{\mathcal{AB}} = -\omega_{\mathcal{BA}}. \quad (2.71)$$

It can be shown that the relationship between the angular velocity vector ${}_{\mathcal{A}}\omega_{\mathcal{AB}}$ and a time varying rotation matrix $\mathbf{C}_{\mathcal{AB}}(t)$ is defined by

$$[{}_{\mathcal{A}}\omega_{\mathcal{AB}}]_{\times} = \dot{\mathbf{C}}_{\mathcal{AB}} \cdot \mathbf{C}_{\mathcal{AB}}^T, \quad (2.72)$$

with $[{}_{\mathcal{A}}\omega_{\mathcal{AB}}]_{\times}$ being the skew symmetric matrix of ${}_{\mathcal{A}}\omega_{\mathcal{AB}}$:

$$[{}_{\mathcal{A}}\omega_{\mathcal{AB}}]_{\times} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}, \quad {}_{\mathcal{A}}\omega_{\mathcal{AB}} = \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}. \quad (2.73)$$

Angular velocities can be transformed like other vectors:

$${}^B\boldsymbol{\omega}_{AB} = \mathbf{C}_{BA} \cdot {}^A\boldsymbol{\omega}_{AB}. \quad (2.74)$$

The corresponding cross-product matrix $[\boldsymbol{\omega}]_{\times}$ is transformed by

$$[{}^B\boldsymbol{\omega}_{AB}]_{\times} = \mathbf{C}_{BA} \cdot [{}^A\boldsymbol{\omega}_{AB}]_{\times} \cdot \mathbf{C}_{AB}. \quad (2.75)$$

The angular velocity of consecutive coordinate systems is given by

$${}^D\boldsymbol{\omega}_{AC} = {}^D\boldsymbol{\omega}_{AB} + {}^D\boldsymbol{\omega}_{BC} \quad (2.76)$$

Similarly to the vector addition used before, it is important that all vectors are expressed in the same reference system \mathcal{D} .

Example 2.5.1: Angular velocity from rotation matrix

Determine the angular velocity of \mathcal{B} with respect to \mathcal{A} in case of an elementary rotation with $\alpha(t)$ around \mathbf{e}_x^A using the corresponding rotation matrix:

$$\mathbf{C}_{AB}(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha(t)) & -\sin(\alpha(t)) \\ 0 & \sin(\alpha(t)) & \cos(\alpha(t)) \end{bmatrix} \quad (2.77)$$

$$[{}^A\boldsymbol{\omega}_{AB}]_{\times} = \dot{\mathbf{C}}_{AB} \mathbf{C}_{AB}^T \quad (2.78)$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\dot{\alpha} \sin \alpha & -\dot{\alpha} \cos \alpha \\ 0 & \dot{\alpha} \cos \alpha & -\dot{\alpha} \sin \alpha \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & -\sin \alpha & \cos \alpha \end{bmatrix} \quad (2.79)$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -\dot{\alpha} \\ 0 & \dot{\alpha} & 0 \end{bmatrix} \quad (2.80)$$

Un-skewing this matrix results in:

$${}^A\boldsymbol{\omega}_{AB} = \begin{pmatrix} \dot{\alpha} \\ 0 \\ 0 \end{pmatrix}. \quad (2.81)$$

2.5.1 Time Derivatives of Rotation Parameterizations

As introduced in section 2.4.5, there exist different parameterizations for a rotation. Similarly to what we have seen for linear velocity, their derivatives can be mapped to angular velocity:

$${}^A\boldsymbol{\omega}_{AB} = \mathbf{E}_R(\boldsymbol{\chi}_R) \cdot \dot{\boldsymbol{\chi}}_R. \quad (2.82)$$

In the following, these mappings will be derived and discussed.

Time Derivatives of Euler Angles ZYX \Leftrightarrow Angular Velocity

Given a set of ZYX Euler angles $\chi_{R,eulerZYX} = [z \ y \ x]^T$ and their time derivatives $\dot{\chi}_{R,eulerZYX} = [\dot{z} \ \dot{y} \ \dot{x}]^T$, we wish to find the mapping $\mathbf{E}_{R,eulerZYX} = \mathbf{E}_R(\chi_{R,eulerZYX}) \in \mathbb{R}^{3 \times 3}$ that maps $\dot{\chi}$ to ${}_{\mathcal{A}}\omega_{AB}$. The columns of $\mathbf{E}(\chi_{R,eulerZYX})$ are the components of the unit vectors around which the angular velocities are applied expressed in a fixed frame \mathcal{A} . These are obtained by selecting the columns of a rotation matrix which is built up by successive elementary rotations specified by the Euler angle parameterization.

Starting from the reference frame \mathcal{A} , the first rotation will be an elementary rotation around ${}_{\mathcal{A}}\mathbf{e}_z^A$, which is simply given by

$${}_{\mathcal{A}}\mathbf{e}_z^A = \mathbb{I}_{3 \times 3} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \quad (2.83)$$

After an elementary rotation around ${}_{\mathcal{A}}\mathbf{e}_z^A$, the y axis ${}_{\mathcal{A}}\mathbf{e}_y^{A'}$ will be expressed by

$${}_{\mathcal{A}}\mathbf{e}_y^{A'} = \mathbf{C}_{\mathcal{A}\mathcal{A}'}(z) \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(z) & -\sin(z) & 0 \\ \sin(z) & \cos(z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -\sin(z) \\ \cos(z) \\ 0 \end{bmatrix}. \quad (2.84)$$

After an elementary rotation around ${}_{\mathcal{A}}\mathbf{e}_y^{A'}$, the x axis ${}_{\mathcal{A}}\mathbf{e}_x^{A''}$ will be expressed by

$$\begin{aligned} {}_{\mathcal{A}}\mathbf{e}_x^{A''} &= \mathbf{C}_{\mathcal{A}\mathcal{A}'}(z) \cdot \mathbf{C}_{\mathcal{A}'\mathcal{A}''}(y) \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \cos(z) & -\sin(z) & 0 \\ \sin(z) & \cos(z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(y) & 0 & \sin(y) \\ 0 & 1 & 0 \\ -\sin(y) & 0 & \cos(y) \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \cos(y) \cos(z) \\ \cos(y) \sin(z) \\ -\sin(y) \end{bmatrix}. \end{aligned} \quad (2.85)$$

Finally, the mapping $\mathbf{E}(\chi_R)$ will be computed as:

$$\mathbf{E}_{R,eulerZYX} = [{}_{\mathcal{A}}\mathbf{e}_z^A \quad {}_{\mathcal{A}}\mathbf{e}_y^{A'} \quad {}_{\mathcal{A}}\mathbf{e}_x^{A''}] = \begin{bmatrix} 0 & -\sin(z) & \cos(y) \cos(z) \\ 0 & \cos(z) & \cos(y) \sin(z) \\ 1 & 0 & -\sin(y) \end{bmatrix}. \quad (2.86)$$

It is easy to find that $\det(\mathbf{E}_{R,eulerZYX}) = -\cos(y)$. The mapping then becomes singular when $y = \pi/2 + k\pi, \forall k \in \mathbb{Z}$. This means that although we can always describe an angular velocity using Euler Angle time derivatives, the inverse is not always possible. The inverse mapping is given by:

$$\mathbf{E}_{R,eulerZYX}^{-1} = \begin{bmatrix} \frac{\cos(z) \sin(y)}{\cos(y)} & \frac{\sin(y) \sin(z)}{\cos(y)} & 1 \\ -\sin(z) & \cos(z) & 0 \\ \frac{\cos(z)}{\cos(y)} & \frac{\sin(z)}{\cos(y)} & 0 \end{bmatrix}. \quad (2.87)$$

Time Derivatives of Euler Angles XYZ \Leftrightarrow Angular Velocity

Analog to the previous derivation, the projection matrix for XYZ Euler Angles and its inverse are

$$\mathbf{E}_{R,eulerXYZ} = \begin{bmatrix} 1 & 0 & \sin(y) \\ 0 & \cos(x) & -\cos(y)\sin(x) \\ 0 & \sin(x) & \cos(x)\cos(y) \end{bmatrix}, \quad (2.88)$$

$$\mathbf{E}_{R,eulerXYZ}^{-1} = \begin{bmatrix} 1 & \frac{\sin(x)\sin(y)}{\cos(y)} & \frac{-\cos(x)\sin(y)}{\cos(y)} \\ 0 & \cos(x) & \sin(x) \\ 0 & \frac{-\sin(x)}{\cos(y)} & \frac{\cos(x)}{\cos(y)} \end{bmatrix}. \quad (2.89)$$

Time Derivatives of Euler Angles ZYZ \Leftrightarrow Angular Velocity

The projection matrix for ZYZ Euler angles and its inverse are

$$\mathbf{E}_{R,eulerZYZ} = \begin{bmatrix} 0 & -\sin(z_1) & \cos(z_1)\sin(y) \\ 0 & \cos(z_1) & \sin(z_1)\sin(y) \\ 1 & 0 & \cos(y) \end{bmatrix}, \quad (2.90)$$

$$\mathbf{E}_{R,eulerZYZ}^{-1} = \begin{bmatrix} \frac{-\cos(y)\cos(z_1)}{\sin(y)} & \frac{-\cos(y)\sin(z_1)}{\sin(y)} & 1 \\ -\sin(z_1) & \cos(z_1) & 0 \\ \frac{\cos(z_1)}{\sin(y)} & \frac{\sin(z_1)}{\sin(y)} & 0 \end{bmatrix}. \quad (2.91)$$

Time Derivatives of Euler Angles ZXZ \Leftrightarrow Angular Velocity

The projection matrix for ZXZ Euler angles and its inverse are

$$\mathbf{E}_{R,eulerZXZ} = \begin{bmatrix} 0 & \cos(z_1) & \sin(z_1)\sin(x) \\ 0 & \sin(z_1) & -\cos(z_1)\sin(x) \\ 1 & 0 & \cos(x) \end{bmatrix}, \quad (2.92)$$

$$\mathbf{E}_{R,eulerZXZ}^{-1} = \begin{bmatrix} \frac{-\cos(x)\sin(z_1)}{\sin(x)} & \frac{\cos(x)\cos(z_1)}{\sin(x)} & 1 \\ \cos(z_1) & \sin(z_1) & 0 \\ \frac{\sin(z_1)}{\sin(x)} & \frac{-\cos(z_1)}{\sin(x)} & 0 \end{bmatrix}. \quad (2.93)$$

Time Derivative of Rotation Quaternion \Leftrightarrow Angular Velocity

For quaternions it can be shown that the following relations hold:

$$\mathcal{I}\boldsymbol{\omega}_{\mathcal{B}} = 2\mathbf{H}(\boldsymbol{\xi}_{\mathcal{IB}})\dot{\boldsymbol{\xi}}_{\mathcal{IB}} = \mathbf{E}_{R,quat}\dot{\boldsymbol{\chi}}_{R,quat} \quad (2.94)$$

$$\dot{\boldsymbol{\xi}}_{\mathcal{IB}} = \frac{1}{2}\mathbf{H}(\boldsymbol{\xi}_{\mathcal{IB}})^T \mathcal{I}\boldsymbol{\omega}_{\mathcal{B}} = \mathbf{E}_{R,quat}^{-1}\dot{\boldsymbol{\chi}}_{R,quat}, \quad (2.95)$$

with

$$\mathbf{H}(\xi) = \begin{bmatrix} -\check{\xi} & [\check{\xi}]_{\times} + \xi_0 \mathbb{I}_{3 \times 3} \end{bmatrix} \in \mathbb{R}^{3 \times 4} \quad (2.96)$$

$$= \begin{bmatrix} -\xi_1 & \xi_0 & -\xi_3 & \xi_2 \\ -\xi_2 & \xi_3 & \xi_0 & -\xi_1 \\ -\xi_3 & -\xi_2 & \xi_1 & \xi_0 \end{bmatrix}. \quad (2.97)$$

Hence, the mapping matrix \mathbf{E}_R and its inverse for a quaternion representation (2.51) are

$$\mathbf{E}_{R,quat} = 2\mathbf{H}(\xi), \quad (2.98)$$

$$\mathbf{E}_{R,quat}^{-1} = \frac{1}{2}\mathbf{H}(\xi)^T. \quad (2.99)$$

Time Derivative of Angle Axis \Leftrightarrow Angular Velocity

For angle axis it can be shown that the following relations hold:

$${}_I\boldsymbol{\omega}_{IB} = \mathbf{n}\dot{\theta} + \dot{\mathbf{n}} \sin \theta + [\mathbf{n}]_{\times} \dot{\mathbf{n}}(1 - \cos \theta) \quad (2.100)$$

$$\dot{\theta} = \mathbf{n}^T {}_I\boldsymbol{\omega}_{IB}, \quad \dot{\mathbf{n}} = \left(-\frac{1}{2} \frac{\sin \theta}{1 - \cos \theta} [\mathbf{n}]_{\times}^2 - \frac{1}{2} [\mathbf{n}]_{\times} \right) {}_I\boldsymbol{\omega}_{IB} \quad \forall \theta \in \mathbb{R} \setminus \{0\} \quad (2.101)$$

Hence, the mapping matrix \mathbf{E}_R and its inverse for the angle axis (2.46) are

$$\mathbf{E}_{R,angleaxis} = [\mathbf{n} \quad \sin \theta \mathbb{I}_{3 \times 3} + (1 - \cos \theta) [\mathbf{n}]_{\times}] \quad (2.102)$$

$$\mathbf{E}_{R,angleaxis}^{-1} = \begin{bmatrix} \mathbf{n}^T \\ -\frac{1}{2} \frac{\sin \theta}{1 - \cos \theta} [\mathbf{n}]_{\times}^2 - \frac{1}{2} [\mathbf{n}]_{\times} \end{bmatrix} \quad (2.103)$$

Time Derivative of Rotation Vector \Leftrightarrow Angular Velocity

For a rotation vector it can be shown that the following relations hold:

$${}_I\boldsymbol{\omega}_{IB} = \left[\mathbb{I}_{3 \times 3} + [\boldsymbol{\varphi}]_{\times} \left(\frac{1 - \cos \|\boldsymbol{\varphi}\|}{\|\boldsymbol{\varphi}\|^2} \right) + [\boldsymbol{\varphi}]_{\times}^2 \left(\frac{\|\boldsymbol{\varphi}\| - \sin \|\boldsymbol{\varphi}\|}{\|\boldsymbol{\varphi}\|^3} \right) \right] \dot{\boldsymbol{\varphi}} \quad \forall \|\boldsymbol{\varphi}\| \in \mathbb{R} \setminus \{0\} \quad (2.104)$$

$$\dot{\boldsymbol{\varphi}} = \left[\mathbb{I}_{3 \times 3} - \frac{1}{2} [\boldsymbol{\varphi}]_{\times} + [\boldsymbol{\varphi}]_{\times}^2 \frac{1}{\|\boldsymbol{\varphi}\|^2} \left(1 - \frac{\|\boldsymbol{\varphi}\|}{2} \frac{\sin \|\boldsymbol{\varphi}\|}{1 - \cos \|\boldsymbol{\varphi}\|} \right) \right] {}_I\boldsymbol{\omega}_{IB} \quad \forall \|\boldsymbol{\varphi}\| \in \mathbb{R} \setminus \{0\} \quad (2.105)$$

Hence, the mapping matrix \mathbf{E}_R and its inverse for the rotation vector (2.47) are

$$\mathbf{E}_{R,rotationvector} = \left[\mathbb{I}_{3 \times 3} + [\boldsymbol{\varphi}]_{\times} \left(\frac{1 - \cos \|\boldsymbol{\varphi}\|}{\|\boldsymbol{\varphi}\|^2} \right) + [\boldsymbol{\varphi}]_{\times}^2 \left(\frac{\|\boldsymbol{\varphi}\| - \sin \|\boldsymbol{\varphi}\|}{\|\boldsymbol{\varphi}\|^3} \right) \right] \quad (2.106)$$

$$\mathbf{E}_{R,rotationvector}^{-1} = \left[\mathbb{I}_{3 \times 3} - \frac{1}{2} [\boldsymbol{\varphi}]_{\times} + [\boldsymbol{\varphi}]_{\times}^2 \frac{1}{\|\boldsymbol{\varphi}\|^2} \left(1 - \frac{\|\boldsymbol{\varphi}\|}{2} \frac{\sin \|\boldsymbol{\varphi}\|}{1 - \cos \|\boldsymbol{\varphi}\|} \right) \right] \quad (2.107)$$

2.6 Transformation

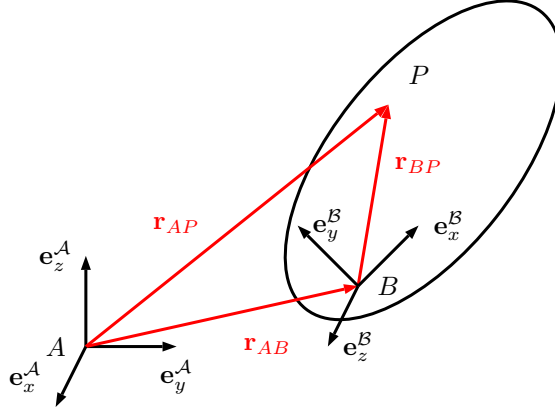


Figure 2.9: Single body with body fixed frame \mathcal{B} .

In the most general case, two reference frames have a position offset and relative rotation (see Fig. 2.9). As a result, a point P can be transformed from one frame to another using a homogeneous transformation matrix \mathbf{T} which is a combined translation and rotation:

$$\mathbf{r}_{AP} = \mathbf{r}_{AB} + \mathbf{r}_{BP} \quad (2.108)$$

$$\mathcal{A}\mathbf{r}_{AP} = \mathcal{A}\mathbf{r}_{AB} + \mathcal{A}\mathbf{r}_{BP} = \mathcal{A}\mathbf{r}_{AB} + \mathbf{C}_{\mathcal{AB}} \cdot \mathcal{B}\mathbf{r}_{BP} \quad (2.109)$$

$$\begin{pmatrix} \mathcal{A}\mathbf{r}_{AP} \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} \mathbf{C}_{\mathcal{AB}} & \mathcal{A}\mathbf{r}_{AB} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}}_{\mathbf{T}_{\mathcal{AB}}} \begin{pmatrix} \mathcal{B}\mathbf{r}_{BP} \\ 1 \end{pmatrix} \quad (2.110)$$

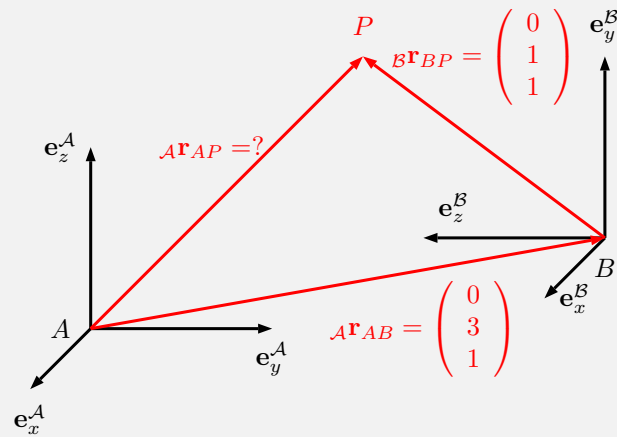
The inverse of the homogeneous transformation can be calculated as

$$\mathbf{T}_{\mathcal{AB}}^{-1} = \begin{bmatrix} \mathbf{C}_{\mathcal{AB}}^T & \overbrace{-\mathbf{C}_{\mathcal{AB}}^T \mathcal{A}\mathbf{r}_{AB}}^{\mathcal{B}\mathbf{r}_{BA}} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}. \quad (2.111)$$

Consecutive homogeneous transformations are given by

$$\mathbf{T}_{\mathcal{AC}} = \mathbf{T}_{\mathcal{AB}} \mathbf{T}_{\mathcal{BC}}. \quad (2.112)$$

Example 2.6.1: Homogeneous transformation



What is the homogeneous transformation matrix \mathbf{T}_{AB} and the position vector ${}^A\mathbf{r}_{AP}$?

The homogeneous transformation matrix is

$$\mathbf{T}_{AB} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.113)$$

Correspondingly, the position vector can be calculated as:

$$\begin{pmatrix} {}^A\mathbf{r}_{AP} \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 2 \\ 1 \end{pmatrix}. \quad (2.114)$$

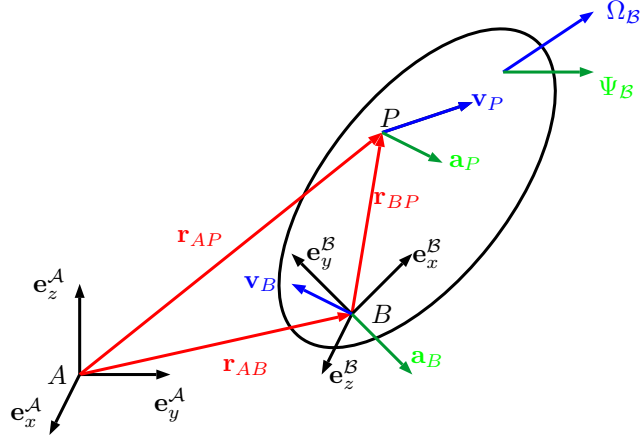


Figure 2.10: Rigid body with velocity and acceleration.

2.7 Velocity in Moving Bodies

Figure 2.10 depicts the velocities of a single body. Assuming \mathcal{A} being in inertial fixed frame, we have:

- \mathbf{v}_P : the absolute velocity of P
- $\mathbf{a}_P = \dot{\mathbf{v}}_P$: the absolute acceleration of P
- $\boldsymbol{\Omega}_B = \boldsymbol{\omega}_{AB}$: (absolute) angular velocity of body \mathcal{B}
- $\boldsymbol{\Psi}_B = \dot{\boldsymbol{\Omega}}_B$: (absolute) angular acceleration of body \mathcal{B}

At this point it is important to understand the difference between the velocity, i.e. the absolute time variation of a position, expressed in a frame \mathcal{C} :

$${}_{\mathcal{C}}(\dot{\mathbf{r}}_{AP}) = {}_{\mathcal{C}}\left(\frac{d}{dt}\mathbf{r}_{AP}\right) = {}_{\mathcal{C}}\mathbf{v}_{AP}, \quad (2.115)$$

and the time differentiation of the coordinates of a position vector:

$$({}_{\mathcal{C}}\dot{\mathbf{r}}_{AP}) = ({}_{\mathcal{C}}\mathbf{r}_{AP})' = \frac{d}{dt}({}_{\mathcal{C}}\mathbf{r}_{AP}). \quad (2.116)$$

They are only equal in case \mathcal{C} is selected as an inertial frame. Following the transformation introduced before, we can write the position of P as:

$${}_{\mathcal{A}}\mathbf{r}_{AP} = {}_{\mathcal{A}}\mathbf{r}_{AB} + {}_{\mathcal{A}}\mathbf{r}_{BP} = {}_{\mathcal{A}}\mathbf{r}_{AB} + \mathbf{C}_{AB} \cdot {}_{\mathcal{B}}\mathbf{r}_{BP}. \quad (2.117)$$

Differentiating with respect to time results in

$${}_{\mathcal{A}}\dot{\mathbf{r}}_{AP} = {}_{\mathcal{A}}\dot{\mathbf{r}}_{AB} + \dot{\mathbf{C}}_{AB} \cdot {}_{\mathcal{B}}\mathbf{r}_{BP} + \mathbf{C}_{AB} \cdot {}_{\mathcal{B}}\dot{\mathbf{r}}_{BP} \quad (2.118)$$

Since P is a point on the rigid body \mathcal{B} , the relative velocity ${}_{\mathcal{B}}\dot{\mathbf{r}}_{BP} = 0$. Furthermore, from (2.72) it can be seen that $\dot{\mathbf{C}}_{AB} = [{}_{\mathcal{A}}\boldsymbol{\omega}_{AB}]_{\times} \cdot \mathbf{C}_{AB}$, yielding

$${}_{\mathcal{A}}\dot{\mathbf{r}}_{AP} = {}_{\mathcal{A}}\dot{\mathbf{r}}_{AB} + [{}_{\mathcal{A}}\boldsymbol{\omega}_{AB}]_{\times} \cdot \mathbf{C}_{AB} \cdot {}_{\mathcal{B}}\mathbf{r}_{BP} \quad (2.119)$$

$$= {}_{\mathcal{A}}\dot{\mathbf{r}}_{AB} + {}_{\mathcal{A}}\boldsymbol{\omega}_{AB} \times {}_{\mathcal{A}}\mathbf{r}_{BP} \quad (2.120)$$

This is the very famous *rigid body formulation* for velocities, also known as velocity composition rule. It can be reformulated to

$$\mathbf{v}_P = \mathbf{v}_B + \boldsymbol{\Omega} \times \mathbf{r}_{BP}. \quad (2.121)$$

Applying the same calculation rules for accelerations results in

$$\mathbf{a}_P = \mathbf{a}_B + \dot{\boldsymbol{\Omega}} \times \mathbf{r}_{BP} + \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \mathbf{r}_{BP}). \quad (2.122)$$

Some Notes on Vector Differentiation

Be careful with vector differentiation in moving coordinate systems. In particular, it is not true that the velocity generally equals the time derivative of the position:

$$\mathbf{v}_P \neq \dot{\mathbf{r}}_{AP} \quad (2.123)$$

Equality only holds if the differentiation is done in non-moving systems represented by \mathcal{A} :

$$\mathcal{A}\mathbf{v}_P = \mathcal{A}\dot{\mathbf{r}}_{AP} \quad (2.124)$$

In case a moving system \mathcal{B} is used for representation, the *Euler differentiation rule* must be applied

$${}_{\mathcal{B}}\mathbf{v}_P = \mathbf{C}_{\mathcal{B}\mathcal{A}} \cdot \frac{d}{dt} (\mathbf{C}_{\mathcal{A}\mathcal{B}} \cdot {}_{\mathcal{B}}\mathbf{r}_{AP}) \quad (2.125)$$

$$= \mathbf{C}_{\mathcal{B}\mathcal{A}} \cdot \left(\mathbf{C}_{\mathcal{A}\mathcal{B}} \cdot {}_{\mathcal{B}}\dot{\mathbf{r}}_{AP} + \dot{\mathbf{C}}_{\mathcal{A}\mathcal{B}} \cdot {}_{\mathcal{B}}\mathbf{r}_{AP} \right) \quad (2.126)$$

$$= \mathbf{C}_{\mathcal{B}\mathcal{A}} \cdot \left(\mathbf{C}_{\mathcal{A}\mathcal{B}} \cdot {}_{\mathcal{B}}\dot{\mathbf{r}}_{AP} + [{}_{\mathcal{A}}\boldsymbol{\omega}_{\mathcal{A}\mathcal{B}}]_{\times} \cdot \mathbf{C}_{\mathcal{A}\mathcal{B}} \cdot {}_{\mathcal{B}}\mathbf{r}_{AP} \right) \quad (2.127)$$

$$= {}_{\mathcal{B}}\dot{\mathbf{r}}_{AP} + \mathbf{C}_{\mathcal{B}\mathcal{A}} \cdot [{}_{\mathcal{A}}\boldsymbol{\omega}_{\mathcal{A}\mathcal{B}}]_{\times} \cdot \mathbf{C}_{\mathcal{A}\mathcal{B}} \cdot {}_{\mathcal{B}}\mathbf{r}_{AP} \quad (2.128)$$

$$\stackrel{(2.75)}{=} {}_{\mathcal{B}}\dot{\mathbf{r}}_{AP} + {}_{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{A}\mathcal{B}} \times {}_{\mathcal{B}}\mathbf{r}_{AP} \quad (2.129)$$

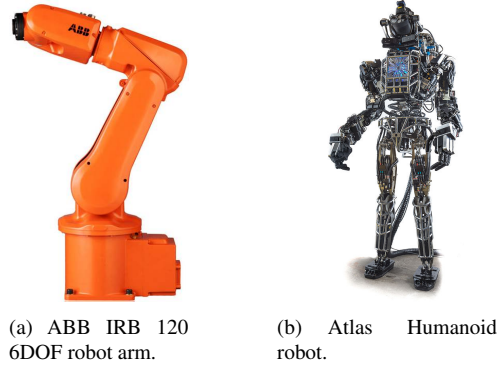


Figure 2.11: Fixed base (a) and floating base (b) systems.

2.8 Kinematics of Systems of Bodies

Most robotic systems can be modeled as open kinematic structures composed of $n_l = n_j + 1$ links connected by n_j joints (prismatic or revolute) with one degree of freedom each. Since there is a single joint with displacement q_i between two successive bodies, a simple transformation relates both bodies:

$$\mathbf{T}_{\mathcal{B}_{i-1}\mathcal{B}_i} = \mathbf{T}_{\mathcal{B}_{i-1}\mathcal{B}_i}(q_i) \quad (2.130)$$

There are two different types, namely *fixed base* and *floating base* systems whereby the root link is either connected to the ground or freely moving. In the following when discussing general aspects of multi-body kinematics we will focus on fixed base systems such as manipulators. Floating base systems will be specifically covered in section 2.10. When dealing with fixed base systems, the frame attached to the root link is often selected to be identical with the world fixed (inertial) frame.

2.8.1 Generalized Coordinates and Joint Configuration

The configuration of a robot such as a manipulator can be described by the generalized coordinate vector

$$\mathbf{q} = \begin{pmatrix} q_1 \\ \vdots \\ q_n \end{pmatrix}. \quad (2.131)$$

This set of scalar values must completely describe the configuration of the system, i.e. for constant values of \mathbf{q} , the robot cannot move anymore. In most cases, one chooses coordinates that are independent, which implies that the number of generalized coordinates corresponds to the number of degrees of freedom. For a fixed base system without additional kinematic constraints, this minimal set of generalized coordinates are then called *minimal coordinates*. It is important to understand that the choice of generalized coordinates is not unique. However, in most applications the generalized coordinates correspond to the degrees of freedom of the robot: For revolute joints, the single degree of freedom q_i corresponds to the rotation angle of the joint. In case of a prismatic joint, q_i represents the linear displacement.

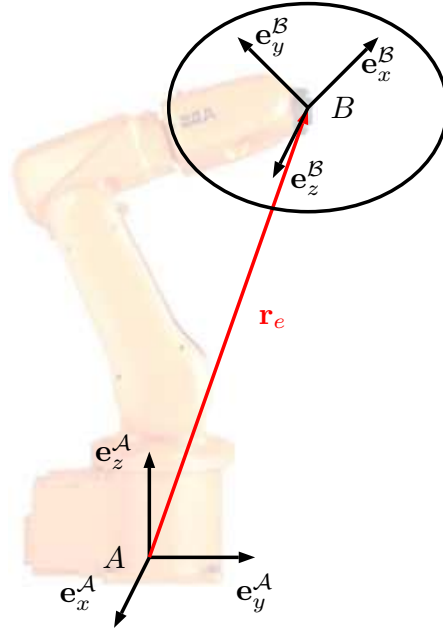


Figure 2.12: Example of task space coordinates corresponding to the end-effector of a manipulator.

Example 2.8.1: Generalized Coordinates and Joint Configuration

The generalized coordinates of a SCARA robot arm are:

$$\mathbf{q} = (\alpha \quad \beta \quad \gamma \quad \zeta)^T, \quad (2.132)$$

with the rotation angles α , β , and γ around the global vertical axis and the linear displacement ζ .

2.8.2 Task-Space Coordinates

The configuration of the end-effector of a robot arm as depicted in Fig. 2.12 can be described by its relative position and orientation w.r.t. a reference frame. The reference frame is often selected as the inertial or root frame.

End-Effector Configuration Parameters

As we have seen in section 2.2.1 and section 2.4.5, the position $\mathbf{r}_e \in \mathbb{R}^3$ and rotation $\phi_e \in SO(3)$ of a frame with respect to a base can be parameterized by χ_P respectively χ_R . Hence, the combined position and orientation (of the end-effector) is given by

$$\mathbf{x}_e = \begin{pmatrix} \mathbf{r}_e \\ \phi_e \end{pmatrix} \in SE(3), \quad (2.133)$$

which can be parameterized by

$$\chi_e = \begin{pmatrix} \chi_{e_P} \\ \chi_{e_R} \end{pmatrix} = \begin{pmatrix} \chi_1 \\ \vdots \\ \chi_m \end{pmatrix} \in \mathbb{R}^m. \quad (2.134)$$

The number m varies depending on the parameterization. Please remember at this point that the rotation ϕ_e is only a theoretical abstraction of the orientation, for which no numerical equivalent such as "angular position" exists (see section 2.4).

Example 2.8.2: End-effector Configuration

To describe the end-effector in 3D space using Cartesian position parameters (3) as well as Euler Angles (3), gives a total of $m = 6$ parameters. In case one uses spherical position parameters (3) and all elements of the direction cosine matrix associated with the rotation (9), $m = 12$ parameters will be necessary.

Operational Space Coordinates

The end-effector of a manipulator operates in the so-called operational space, which depends on the geometry and structure of the arm. The operational space can be described by

$$\chi_o = \begin{pmatrix} \chi_{o_P} \\ \chi_{o_R} \end{pmatrix} = \begin{pmatrix} \chi_1 \\ \vdots \\ \chi_{m_0} \end{pmatrix}, \quad (2.135)$$

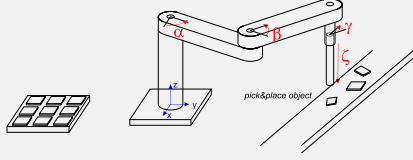
whereby $\chi_1, \chi_2, \dots, \chi_{m_0}$ are *independent* operational space coordinates². Hence, they can be understood as a minimal selection of end-effector configuration parameters. Note that $m_0 \leq n_j$ since the degree of mobility at the end-effector is certainly not larger than the number of joints in the system.

Example 2.8.3: Operational Space Coordinates 1

To describe the end-effector in the most general case of a six-dimensional operational space requires $m_0 = 6$ parameters. Hence, only Euler Angles are a valid parameterization while the choice of quaternions or rotation matrix is not possible.

²Please note that there are different definitions for operational space coordinates in literature and some use them as equivalent to end-effector coordinates. We introduce this here as minimal representation to properly define things like e.g. singularities.

Example 2.8.4: Operational Space Coordinates 2



For a SCARA robot arm as depicted on the left, the operational space is only 4DOF, namely the three positions and the rotation around the vertical axis.

$$\chi_o = \begin{pmatrix} x \\ y \\ z \\ \varphi \end{pmatrix} \quad (2.136)$$

Note: In the example for the SCARA robot arm it is relatively simple to define four operational space coordinates for a 4DOF arm. However, in case of an arm with four non co-linear rotation axes, it can be impossible to select four operational space coordinates. In such an example, operational space coordinates remain a rather theoretical concept.

In the following, we will not work with operational space coordinates but focus on the more generic concept of end-effector configuration parameters.

2.8.3 Forward Kinematics

Forward kinematics describes the mapping between joint (generalized) coordinates \mathbf{q} and the end-effector configuration χ_e :

$$\chi_e = \chi_e(\mathbf{q}). \quad (2.137)$$

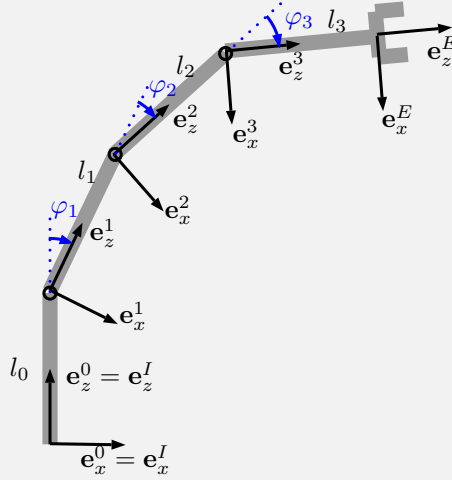
This relation can be obtained through the evaluation of (2.130) from the base to the end-effector. For a serial linkage system with n_j joints, this is

$$\mathbf{T}_{\mathcal{IE}}(\mathbf{q}) = \mathbf{T}_{\mathcal{I}0} \cdot \left(\prod_{k=1}^{n_j} \mathbf{T}_{k-1,k}(q_k) \right) \cdot \mathbf{T}_{n_j\mathcal{E}} = \begin{bmatrix} \mathbf{C}_{\mathcal{IE}}(\mathbf{q}) & {}^{\mathcal{I}}\mathbf{r}_{IE}(\mathbf{q}) \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}. \quad (2.138)$$

When talking about fixed base robots, the first coordinate frame of the robot 0 is not moving with respect to an inertial frame such that $\mathbf{T}_{\mathcal{I}0}$ is a constant transformation. Furthermore, in most cases, an end-effector frame \mathcal{E} is introduced, which is rigidly connected to the last link but which does not have to be identical with the last body coordinate frame. Hence, also $\mathbf{T}_{n_j\mathcal{E}}$ is constant.

In order to create the representation in form of (2.137), it is necessary to transform the rotation matrix $\mathbf{C}_{\mathcal{IE}}(\mathbf{q})$ and the position vector ${}^{\mathcal{I}}\mathbf{r}_{IE}(\mathbf{q})$ into end-effector parameters χ_e . While this is straight forward for the position, i.e. $\chi_{eP}(\mathbf{q}) = {}^{\mathcal{I}}\mathbf{r}_{IE}$, transferring the rotation matrix $\mathbf{C}_{\mathcal{IE}}(\mathbf{q})$ can be significantly more difficult depending on the choice of parameterization (see section 2.5.1).

Example 2.8.5: Forward Kinematics



Find the forward kinematics for a planar 3DOF robot arm.

The generalized coordinates are

$$\mathbf{q} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \end{pmatrix}. \quad (2.139)$$

With this we calculate the end-effector position and orientation:

$$\chi_e(\mathbf{q}) = \begin{pmatrix} \chi_{eP}(\mathbf{q}) \\ \chi_{eR}(\mathbf{q}) \end{pmatrix} \quad (2.140)$$

$$\chi_{eP}(\mathbf{q}) = \begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) + l_3 \sin(q_1 + q_2 + q_3) \\ l_0 + l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) + l_3 \cos(q_1 + q_2 + q_3) \end{pmatrix} \quad (2.141)$$

$$\chi_{eR}(\mathbf{q}) = \chi_{eR}(\mathbf{q}) = q_1 + q_2 + q_3 \quad (2.142)$$

2.8.4 Differential Kinematics and Analytical Jacobian

Very often, we are interested in local changes or local changes per time (i.e. velocities) which is known as *differential or instantaneous kinematics*. A common approach is to linearize the forward kinematics:

$$\chi_e + \delta\chi_e = \chi_e(\mathbf{q} + \delta\mathbf{q}) = \chi_e(\mathbf{q}) + \frac{\partial\chi_e(\mathbf{q})}{\partial\mathbf{q}}\delta\mathbf{q} + O(\delta\mathbf{q}^2), \quad (2.143)$$

which results in the first order approximation

$$\delta\chi_e \approx \frac{\partial\chi_e(\mathbf{q})}{\partial\mathbf{q}}\delta\mathbf{q} = \mathbf{J}_{eA}(\mathbf{q})\delta\mathbf{q}, \quad (2.144)$$

where

$$\mathbf{J}_{eA}(\mathbf{q}) = \begin{bmatrix} \frac{\partial \chi_1}{\partial q_1} & \dots & \frac{\partial \chi_1}{\partial q_{n_j}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \chi_m}{\partial q_1} & \dots & \frac{\partial \chi_m}{\partial q_{n_j}} \end{bmatrix} \quad (2.145)$$

is the $m \times n_j$ *analytical Jacobian* matrix. The Jacobian matrix is very often used in kinematics and dynamics of robotic systems. It relates differences from joint to configuration space. While it represents an approximation in the context of finite differences:

$$\Delta \chi_e \approx \mathbf{J}_{eA}(\mathbf{q}) \Delta \mathbf{q}, \quad (2.146)$$

it results in an exact relation between velocities:

$$\dot{\chi}_e = \mathbf{J}_{eA}(\mathbf{q}) \dot{\mathbf{q}}. \quad (2.147)$$

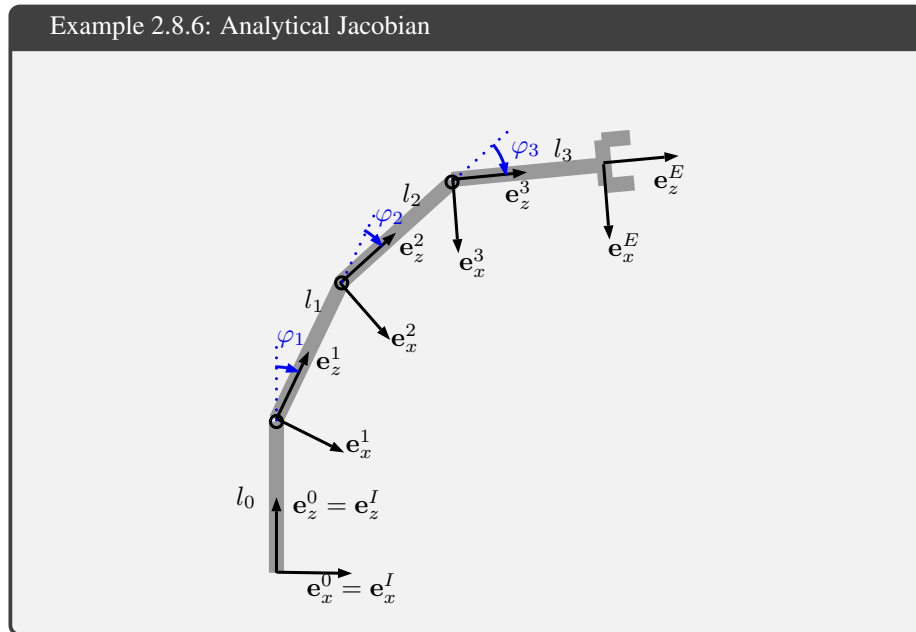
Position and Rotation Jacobian

Since the end-effector configuration (2.134) is parameterized by the stacked vector of end-effector position χ_{eP} and orientation χ_{eR} , literature often talks about position and rotation Jacobian:

$$\mathbf{J}_{eA} = \begin{bmatrix} \mathbf{J}_{eAP} \\ \mathbf{J}_{eAR} \end{bmatrix} = \begin{bmatrix} \frac{\partial \chi_{eP}}{\partial \mathbf{q}} \\ \frac{\partial \chi_{eR}}{\partial \mathbf{q}} \end{bmatrix}. \quad (2.148)$$

Dependency on Parameterization

As we have seen in section 2.2.1 and section 2.4.5, this Jacobian strongly depends on the selected parameterization. For example, when using Euler Angles the dimension of \mathbf{J}_{eAR} is $3 \times n_j$, in case of quaternions it is $4 \times n_j$, and for the full rotation matrix parameters $9 \times n_j$.



Find the analytical position and rotation Jacobian for the end-effector of a planar 3DOF robot arm.

Differentiation of $\chi_{eP}(\mathbf{q})$ given in (2.141) with respect to the generalized coordinates \mathbf{q} given in (2.139) is:

$$\begin{aligned}\mathbf{J}_{eAP}(\mathbf{q}) &= \frac{\partial \chi_{eP}}{\partial \mathbf{q}} \\ &= \begin{bmatrix} l_1 c_1 + l_2 c_{12} + l_3 c_{123} & l_2 c_{12} + l_3 c_{213} & l_3 c_{213} \\ -l_1 s_1 - l_2 s_{12} - l_3 s_{123} & -l_2 s_{12} - l_3 s_{213} & -l_3 s_{213} \end{bmatrix} \in \mathbb{R}^{2 \times 3}\end{aligned}\quad (2.149)$$

with $c_{123} = \cos(q_1 + q_2 + q_3)$ and $s_{123} = \sin(q_1 + q_2 + q_3)$.

Differentiation of $\chi_{eR}(\mathbf{q})$ given in (2.142) with respect to the generalized coordinates \mathbf{q} given in (2.139) is:

$$\mathbf{J}_{eAR}(\mathbf{q}) = \frac{\partial \chi_{eR}}{\partial \mathbf{q}} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \in \mathbb{R}^{1 \times 3} \quad (2.150)$$

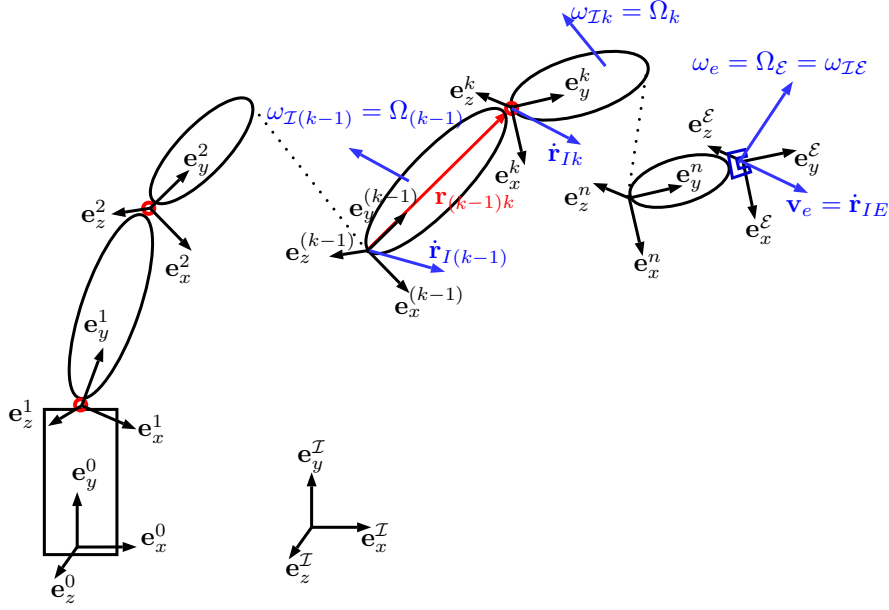


Figure 2.13: Serial linkage arm with n moving links.

2.8.5 Geometric or Basic Jacobian

As we have seen in (2.147), a Jacobian maps generalized velocities (in joint space) to time-derivatives of the end-effector configuration representation (which is not the linear and angular velocity!). The associated partial differentiations of the end-effector configuration $\mathbf{J}_{eA} = \frac{\partial \mathbf{x}_e}{\partial \mathbf{q}}$ depends on the selected parameterization, especially on the parameterization of the rotation.

However, as we have learned previously, a body has a unique linear velocity \mathbf{v}_e and angular velocity $\boldsymbol{\omega}_e$. Hence, there must exist a *unique* Jacobian that relates the generalized velocity $\dot{\mathbf{q}}$ to the velocity of the end-effector (linear \mathbf{v}_e and angular $\boldsymbol{\omega}_e$):

$$\mathbf{w}_e = \begin{pmatrix} \mathbf{v}_e \\ \boldsymbol{\omega}_e \end{pmatrix} = \mathbf{J}_{e0}(\mathbf{q}) \dot{\mathbf{q}}. \quad (2.151)$$

\mathbf{J}_{e0} is called the geometric (or basic) Jacobian and it has in the most general cases the dimension $6 \times n_j$. Please also note at this point that the geometric Jacobian has a basis \mathcal{A} as it maps generalized velocities to end-effector velocities represented in a specific coordinate frame

$${}_{\mathcal{A}}\mathbf{w}_e = {}_{\mathcal{A}}\mathbf{J}_{e0}(\mathbf{q}) \dot{\mathbf{q}}. \quad (2.152)$$

Addition and Subtraction of Geometric Jacobians

From basic kinematics we know that the velocity of a point C can be calculated from the velocity of a point B and the relative velocity between B and C :

$$\begin{aligned} \mathbf{w}_C &= \begin{pmatrix} \mathbf{v}_C \\ \boldsymbol{\omega}_C \end{pmatrix} = \mathbf{w}_B + \mathbf{w}_{BC} \\ \mathbf{J}_C \dot{\mathbf{q}} &= \mathbf{J}_B \dot{\mathbf{q}} + \mathbf{J}_{BC} \dot{\mathbf{q}} \end{aligned} \quad (2.153)$$

From this we can identify that geometric Jacobians can be simply added

$${}^{\mathcal{A}}\mathbf{J}_C = {}^{\mathcal{A}}\mathbf{J}_B + {}^{\mathcal{A}}\mathbf{J}_{BC}, \quad (2.154)$$

as long as they are represented with respect to the same reference frame.

Calculation of geometric Jacobian using Rigid Body Formulation

From the analysis of the velocities of points on moving bodies (c.f. (2.121)) applied to the serial linkage depicted in Fig. 2.13, it follows that the velocity of linkage k is given by

$$\dot{\mathbf{r}}_{Ik} = \dot{\mathbf{r}}_{I(k-1)} + \boldsymbol{\omega}_{\mathcal{I}(k-1)} \times \mathbf{r}_{(k-1)k}. \quad (2.155)$$

Please keep again in mind that for numerical addition it is crucial that all the vectors are expressed in same coordinate system. When denoting the base frame as 0 and the end-effector frame as $n+1$, the end-effector velocity can be written as

$$\dot{\mathbf{r}}_{IE} = \sum_{k=1}^n \boldsymbol{\omega}_{\mathcal{I}k} \times \mathbf{r}_{k(k+1)}. \quad (2.156)$$

Using \mathbf{n}_k to represent the rotation axis of joint k such that

$$\boldsymbol{\omega}_{(k-1)k} = \mathbf{n}_k \dot{q}_k \quad (2.157)$$

and recalling that

$$\boldsymbol{\omega}_{\mathcal{I}(k)} = \boldsymbol{\omega}_{\mathcal{I}(k-1)} + \boldsymbol{\omega}_{(k-1)k}, \quad (2.158)$$

the angular velocity of body k can be written as

$$\boldsymbol{\omega}_{\mathcal{I}k} = \sum_{i=1}^k \mathbf{n}_i \dot{q}_i. \quad (2.159)$$

Substituting this in (2.156) and rearranging terms results to

$$\dot{\mathbf{r}}_{IE} = \sum_{k=1}^n \left(\sum_{i=1}^k (\mathbf{n}_i \dot{q}_i) \times \mathbf{r}_{k(k+1)} \right) \quad (2.160)$$

$$= \sum_{k=1}^n \mathbf{n}_k \dot{q}_k \times \sum_{i=k}^n \mathbf{r}_{i(i+1)} \quad (2.161)$$

$$= \sum_{k=1}^n \mathbf{n}_k \dot{q}_k \times \mathbf{r}_{k(n+1)} \quad (2.162)$$

Bringing this in matrix formulation yields the geometric Jacobian

$$\dot{\mathbf{r}}_{IE} = \underbrace{\begin{bmatrix} \mathbf{n}_1 \times \mathbf{r}_{1(n+1)} & \mathbf{n}_2 \times \mathbf{r}_{2(n+1)} & \dots & \mathbf{n}_n \times \mathbf{r}_{n(n+1)} \end{bmatrix}}_{\mathbf{J}_{e0_P}} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{pmatrix} \quad (2.163)$$

Given (2.159), the rotation Jacobian is

$$\boldsymbol{\omega}_{\mathcal{IE}} = \sum_{i=1}^n \mathbf{n}_i \dot{q}_i = \underbrace{\begin{bmatrix} \mathbf{n}_1 & \mathbf{n}_2 & \dots & \mathbf{n}_n \end{bmatrix}}_{\mathbf{J}_{e0_R}} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_n \end{pmatrix} \quad (2.164)$$

Combining these two expressions yields the combined geometric Jacobian:

$$\mathbf{J}_{e0} = \begin{bmatrix} \mathbf{J}_{e0_P} \\ \mathbf{J}_{e0_R} \end{bmatrix} = \begin{bmatrix} \mathbf{n}_1 \times \mathbf{r}_{1(n+1)} & \mathbf{n}_2 \times \mathbf{r}_{2(n+1)} & \cdots & \mathbf{n}_n \times \mathbf{r}_{n(n+1)} \\ \mathbf{n}_1 & \mathbf{n}_2 & \cdots & \mathbf{n}_n \end{bmatrix} \quad (2.165)$$

As stated at the beginning of this section, it is important that we need to define this Jacobian with respect to a basis, e.g. \mathcal{I} (or any other frame):

$${}^{\mathcal{I}}\mathbf{J}_{e0} = \begin{bmatrix} {}^{\mathcal{I}}\mathbf{J}_{e0_P} \\ {}^{\mathcal{I}}\mathbf{J}_{e0_R} \end{bmatrix} = \begin{bmatrix} {}^{\mathcal{I}}\mathbf{n}_1 \times {}^{\mathcal{I}}\mathbf{r}_{1(n+1)} & {}^{\mathcal{I}}\mathbf{n}_2 \times {}^{\mathcal{I}}\mathbf{r}_{2(n+1)} & \cdots & {}^{\mathcal{I}}\mathbf{n}_n \times {}^{\mathcal{I}}\mathbf{r}_{n(n+1)} \\ {}^{\mathcal{I}}\mathbf{n}_1 & {}^{\mathcal{I}}\mathbf{n}_2 & \cdots & {}^{\mathcal{I}}\mathbf{n}_n \end{bmatrix}. \quad (2.166)$$

In this formulation, the rotation axis is given by

$${}^{\mathcal{I}}\mathbf{n}_k = \mathbf{C}_{\mathcal{I}(k-1)} \cdot {}^{(k-1)}\mathbf{n}_k. \quad (2.167)$$

Example 2.8.7: Basic Jacobian

Determine the basic Jacobian for this planar robot arm

The generalized coordinates are

$$\mathbf{q} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \end{pmatrix} \quad (2.168)$$

Step 1: determine the rotation matrices

$$\mathbf{C}_{I1} = \begin{bmatrix} c_1 & 0 & s_1 \\ 0 & 1 & 0 \\ -s_1 & 0 & c_1 \end{bmatrix} \quad (2.169)$$

$$\mathbf{C}_{I2} = \mathbf{C}_{I1} \cdot \begin{bmatrix} c_2 & 0 & s_2 \\ 0 & 1 & 0 \\ -s_2 & 0 & c_2 \end{bmatrix} = \begin{bmatrix} c_{12} & 0 & s_{12} \\ 0 & 1 & 0 \\ -s_{12} & 0 & c_{12} \end{bmatrix} \quad (2.170)$$

$$\mathbf{C}_{I3} = \mathbf{C}_{I2} \cdot \begin{bmatrix} c_3 & 0 & s_3 \\ 0 & 1 & 0 \\ -s_3 & 0 & c_3 \end{bmatrix} = \begin{bmatrix} c_{123} & 0 & s_{123} \\ 0 & 1 & 0 \\ -s_{123} & 0 & c_{123} \end{bmatrix}, \quad (2.171)$$

with $s_{123} = \sin(q_1 + q_2 + q_3)$ and $c_{123} = \cos(q_1 + q_2 + q_3)$.

Step 2: determine the local rotation axis ${}_{(k-1)}\mathbf{n}_k$:

$${}_0\mathbf{n}_1 = {}_1\mathbf{n}_2 = {}_2\mathbf{n}_3 = \mathbf{e}_y \quad (2.172)$$

Step 3: determine the rotation axis ${}_I\mathbf{n}_k = \mathbf{C}_{I(k-1)} \cdot {}_{(k-1)}\mathbf{n}_k$:

$${}_I\mathbf{n}_1 = {}_0\mathbf{n}_1 = \mathbf{e}_y \quad (2.173)$$

$${}_I\mathbf{n}_2 = \mathbf{C}_{I1} \cdot {}_1\mathbf{n}_2 = \mathbf{e}_y \quad (2.174)$$

$${}_I\mathbf{n}_3 = \mathbf{C}_{I2} \cdot {}_2\mathbf{n}_3 = \mathbf{e}_y \quad (2.175)$$

Step 4: determine the position vectors from joint to end-effector:

$${}_I\mathbf{r}_{1E} = {}_I\mathbf{r}_{12} + {}_I\mathbf{r}_{23} + {}_I\mathbf{r}_{3E} \quad (2.176)$$

$$= \mathbf{C}_{I1} \cdot {}_1\mathbf{r}_{12} + \mathbf{C}_{I2} \cdot {}_2\mathbf{r}_{23} + \mathbf{C}_{I3} \cdot {}_3\mathbf{r}_{3E} \quad (2.177)$$

$$= l_1 \begin{pmatrix} s_{q_1} \\ 0 \\ c_{q_1} \end{pmatrix} + l_2 \begin{pmatrix} s_{12} \\ 0 \\ c_{12} \end{pmatrix} + l_3 \begin{pmatrix} s_{123} \\ 0 \\ c_{123} \end{pmatrix} \quad (2.178)$$

$${}_I\mathbf{r}_{2E} = {}_I\mathbf{r}_{23} + {}_I\mathbf{r}_{3E} \quad (2.179)$$

$$= \mathbf{C}_{I2} \cdot {}_2\mathbf{r}_{23} + \mathbf{C}_{I3} \cdot {}_3\mathbf{r}_{3E} \quad (2.180)$$

$$= l_2 \begin{pmatrix} s_{12} \\ 0 \\ c_{12} \end{pmatrix} + l_3 \begin{pmatrix} s_{123} \\ 0 \\ c_{123} \end{pmatrix} \quad (2.181)$$

$${}_I\mathbf{r}_{3E} = {}_I\mathbf{r}_{3E} \quad (2.182)$$

$$= \mathbf{C}_{I3} \cdot {}_3\mathbf{r}_{3E} \quad (2.183)$$

$$= l_3 \begin{pmatrix} s_{123} \\ 0 \\ c_{123} \end{pmatrix} \quad (2.184)$$

Step 5a: determine the position Jacobian:

$${}_I\mathbf{J}_{e0P} = \begin{bmatrix} {}_I\mathbf{n}_1 \times {}_I\mathbf{r}_{1E} & {}_I\mathbf{n}_2 \times {}_I\mathbf{r}_{2E} & {}_I\mathbf{n}_3 \times {}_I\mathbf{r}_{3E} \end{bmatrix} \quad (2.185)$$

$$= \begin{bmatrix} l_1 c_1 + l_2 c_{12} + l_3 c_{123} & l_2 c_{12} + l_3 c_{123} & l_3 c_{123} \\ 0 & 0 & 0 \\ -l_1 s_1 - l_2 s_{12} - l_3 s_{123} & -l_2 s_{12} - l_3 s_{123} & -l_3 s_{123} \end{bmatrix} \quad (2.186)$$

Step 5b: determine the rotation Jacobian:

$$\mathcal{I}\mathbf{J}_{e0_R} = [\mathcal{I}\mathbf{n}_1 \quad \mathcal{I}\mathbf{n}_2 \quad \mathcal{I}\mathbf{n}_3] \quad (2.187)$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad (2.188)$$

2.8.6 Relation between Geometric and Analytic Jacobian Matrix

As introduced in section 2.2.1 and section 2.5.1, there exists a mapping between the differentials of the end-effector representation parameters $\dot{\chi}_e$ and the twist \mathbf{w}_e consisting of linear and angular velocities. This relationship also shows up in the mapping of the representation dependent Jacobians given by a partial differentiation of position and rotation with respect to generalized coordinates $\mathbf{J}_{eA} = \frac{\partial \chi_e}{\partial \mathbf{q}} \in \mathbb{R}^{m \times n_j}$ and the geometric Jacobian $\mathbf{J}_{e0} \in \mathbb{R}^{6 \times n_j}$. Given that

$$\dot{\chi}_e = \mathbf{J}_{eA}(\mathbf{q}) \dot{\mathbf{q}} \quad \text{with } \mathbf{J}_{eA}(\mathbf{q}) \in \mathbb{R}^{m_e \times n_j} \quad (2.189)$$

$$\mathbf{w}_e = \mathbf{J}_{e0}(\mathbf{q}) \dot{\mathbf{q}} \quad \text{with } \mathbf{J}_{e0}(\mathbf{q}) \in \mathbb{R}^{6 \times n_j} \quad (2.190)$$

$$\mathbf{w}_e = \mathbf{E}_e(\chi_e) \dot{\chi}_e \quad \text{with } \mathbf{E}_e(\chi_e) = \begin{bmatrix} \mathbf{E}_P & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_R \end{bmatrix} \in \mathbb{R}^{m_e \times 6} \quad (2.191)$$

the following mapping holds

$$\mathbf{J}_{e0}(\mathbf{q}) = \mathbf{E}_e(\chi) \mathbf{J}_{eA}(\mathbf{q}) \quad (2.192)$$

Please note that the parameterization dependent matrices \mathbf{E} and \mathbf{E}^{-1} were derived earlier:

- Cartesian coordinates (2.12)
- Cylindrical coordinates (2.14) and (2.15)
- Spherical coordinates (2.16) and (2.17)
- Euler Angles
 - XYZ: (2.88) and (2.89)
 - ZYX: (2.86) and (2.87)
 - ZYZ: (2.90) and (2.91)
- Quaternions (2.98) and (2.99)
- Angle Axis (2.102) and (2.103)
- Rotation Vector (2.106) and (2.107)

Literature often does not distinguish between geometric and analytic Jacobian. Mostly, when writing \mathbf{J} it is referred to the geometric Jacobian. The same holds for this course. For planar systems, the analytical and geometric Jacobian are identical.

2.9 Kinematic Control Methods

The previously introduced relationship between end-effector and joint space configuration respectively is often used for kinematic control.

2.9.1 Inverse Differential Kinematics

As we have seen, the Jacobian $\mathbf{J}_{e0}(\mathbf{q})$ performs a simple mapping between joint space velocity $\dot{\mathbf{q}}$ and end-effector velocity \mathbf{w}_e :

$$\mathbf{w}_e = \mathbf{J}_{e0} \dot{\mathbf{q}} \quad (2.193)$$

However, in many cases and particularly in control, we are interested to solve the inverse problem. Given a (desired) end-effector velocity \mathbf{w}_e^* , what is the corresponding joint velocity $\dot{\mathbf{q}}$? The obvious way to do this, is simply to invert the Jacobian or, more general, to take the pseudo-inverse of the Jacobian

$$\dot{\mathbf{q}} = \mathbf{J}_{e0}^+ \mathbf{w}_e^*. \quad (2.194)$$

However, as we know, there exist different kinds of pseudo-inverse methods and the solution is not unique but depending on the properties of \mathbf{J}_{e0} .

Infobox: Generalized Inverse and Pseudo Inverse

The Moore-Penrose Pseudo Inverse A^+ of the matrix A is the most widely known generalization of the inverse for non-square matrices. A generalized inverse B is defined as the matrix fulfilling the following condition:

$$ABA = A \quad (2.195)$$

In the general case there is no unique solution to the generalized inverse B . The Moore-Penrose inverse addresses this by adding additional three conditions to the generalized inverse. The four conditions are collectively known as the Moore-Penrose conditions and result in a unique solution A^+ for the matrix A . For a real matrix $A \in \mathbf{R}^{m \times n}$ the four conditions are:

$$AA^+A = A \quad (2.196)$$

$$A^+AA^+ = A^+ \quad (2.197)$$

$$(A^+A)^T = A^+A \quad (2.198)$$

$$(AA^+)^T = AA^+ \quad (2.199)$$

Matrices with full column rank For matrices with full column rank (also called tall matrices, $m \geq n$) the inverse $(A^T A)^{-1}$ is defined and the pseudo inverse, fulfilling the four Moore-Penrose conditions, can be obtained as follows:

$$A^+ = (A^T A)^{-1} A^T \quad (2.200)$$

It follows, that A^+ is the left inverse of A : $A^+A = \mathbb{I}$. Furthermore, by analyzing the linear set of equations $Ax = b$. Since A is a tall matrix there are more

equations than variables in x and there exists no exact solution. It can be shown that A^+ minimizes the least squares error:

$$e(x) = \|Ax - b\|_2^2 = (Ax - b)^T(Ax - b) \quad (2.201)$$

The error function is convex and therefore the minimum is found by setting the gradient of the error to zero:

$$x = \arg \min_x e(x) \quad (2.202)$$

$$\frac{\partial}{\partial x} (Ax - b)^T(Ax - b) = 2A^T Ax - 2A^T b = 0 \quad (2.203)$$

$$x = (A^T A)^{-1} A^T b = A^+ b \quad (2.204)$$

Matrices with full row rank For matrices with full row rank (also called wide matrices, $m \leq n$) the inverse $(AA^T)^{-1}$ is defined and the pseudo inverse, fulfilling the four Moore-Penrose conditions, can be obtained as follows:

$$A^+ = A^T (AA^T)^{-1} \quad (2.205)$$

It follows, that A^+ is the right inverse of A : $AA^+ = \mathbb{I}$. It can then be shown for a set of linear equations $Ax = b$, which has due to A being wide multiple solutions:

$$x = A^+ b + N_A x_0 = A^+ b + (\mathbb{I} - A^+ A) x_0 \quad (2.206)$$

N_A denotes the null space projection matrix of A fulfilling $AN_A = 0$. $x = A^+ b$ is the solution that minimizes $\|x\|_2$ while ensuring that $Ax = b$ is satisfied. The interested reader is referred to [10] for a formal proof.

Damped pseudo inverse To cope with the problem of a badly conditioned matrix A with singular values close to zero, a common approach is to use a damped version of the Moore-Penrose pseudo-inverse:

$$A_d^+ = A^T (AA^T + \lambda^2 \mathbb{I})^{-1} \quad (2.207)$$

This inversion method minimizes the error $\|Ax - b\|_2^2 + \lambda^2 \|\dot{\mathbf{q}}\|_2^2$.

Singularities

In case the robot is in a configuration \mathbf{q}_s such that the Jacobian $\mathbf{J}_{e0}(\mathbf{q}_s)$ has $rank < m_0$, with m_0 being the number of operational space coordinates (number of controllable end-effector DOFs), the configuration is called *singular*. A singularity implies that for a desired end-effector velocity \mathbf{w}_e^* there exists no generalized velocity $\dot{\mathbf{q}}$ that fulfills $\mathbf{w}_e^* = \mathbf{J}_{e0} \dot{\mathbf{q}}$. By taking the Moore-Penrose pseudo inverse, the solution $\dot{\mathbf{q}} = \mathbf{J}_{e0}^+ \mathbf{w}_e^*$ minimizes the least square error $\|\mathbf{w}_e^* - \mathbf{J}_{e0} \dot{\mathbf{q}}\|^2$.

Unfortunately, if a robot is close to a singular configuration, \mathbf{J}_{e0} becomes badly conditioned with singular values close to 0. This implies that small desired velocities

\mathbf{w}_e^* in corresponding directions will lead to extremely high joint velocities $\dot{\mathbf{q}}$. This behavior is particularly problematic for inverse kinematics algorithms as introduced in section 2.9.3.

Some of the singularities are obvious (e.g. in case the robot arm is at the limit of its workspace) and hence simple to prevent. However, there can also be cases of end-effector configurations that are perfectly within the workspace but where some joints are in singular alignment. These situations are hard to prevent and require careful motion planning. This is particularly the case when working with a 6DOF arm and controlling the same number of end-effector coordinates. A commonly used method to prevent the robot from getting stuck in singularities while following desired end-effector poses, is to use robot arms with 7DOF. Thereby, the redundancy of the system can be used to keep the robot configuration away from singularities, while following the same end-effector pose trajectory.

Damped Solution To cope with the problem of a badly conditioned \mathbf{J}_{e0} close to singularities, a common approach is to use a damped version of the Moore-Penrose pseudo-inverse:

$$\dot{\mathbf{q}} = \mathbf{J}_{e0}^T (\mathbf{J}_{e0} \mathbf{J}_{e0}^T + \lambda^2 \mathbb{I})^{-1} \mathbf{w}_e^* \quad (2.208)$$

with λ being a damping parameter. The larger the damping behavior, the more stable the solution, but the slower the convergence. For more details, the interested reader is referred to this manuscript [3].

Redundancy

In case the robot is in a configuration \mathbf{q}^* such that the Jacobian $\mathbf{J}_{e0}(\mathbf{q}^*)$ has $\text{rank} < n$, the configuration is called *redundant*. When taking the Moore-Penrose pseudo inverse, as in the previous cases, the solution

$$\dot{\mathbf{q}} = \mathbf{J}_{e0}^T (\mathbf{J}_{e0} \mathbf{J}_{e0}^T)^{-1} \mathbf{w}_e^* \quad (2.209)$$

minimizes $\|\dot{\mathbf{q}}\|_2$ while satisfying $\mathbf{w}_e^* = \mathbf{J}_{e0} \dot{\mathbf{q}}$. Redundancy in the system implies that there exist an infinite number of solutions

$$\dot{\mathbf{q}} = \mathbf{J}_{e0}^+ \mathbf{w}_e^* + \mathbf{N} \dot{\mathbf{q}}_0, \quad (2.210)$$

with $\mathbf{N} = \mathcal{N}(\mathbf{J}_{e0})$ being the null-space projection matrix of \mathbf{J}_{e0} fulfilling $\mathbf{J}_{e0} \mathbf{N} = \mathbf{0}$. This allows modifying the generalized velocity $\dot{\mathbf{q}}$ by an arbitrary choice of $\dot{\mathbf{q}}_0$ without changing the end-effector motion

$$\mathbf{J}_{e0} (\mathbf{J}_{e0}^+ \mathbf{w}_e^* + \mathbf{N} \dot{\mathbf{q}}_0) = \mathbf{w}_e^* \quad \forall \dot{\mathbf{q}}_0. \quad (2.211)$$

There are different methods to determine the null-space projection matrix. The simplest projection is given by

$$\mathbf{N} = \mathbb{I} - \mathbf{J}_{e0}^+ \mathbf{J}_{e0}. \quad (2.212)$$

2.9.2 Multi-task Inverse Differential Kinematics Control

For task- or operational space objectives such as following a certain trajectory, reaching an end-effector orientation, ensuring kinematic constraints, etc., the task Jacobian and desired task velocity are defined by:

$$\text{task}_i := \{\mathbf{J}_i, \mathbf{w}_i^*\}. \quad (2.213)$$

Multi-task with Equal Priority

In case all n_t tasks have the same priority, the generalized velocity is given by

$$\dot{\mathbf{q}} = \underbrace{\begin{bmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_{n_t} \end{bmatrix}}_{\bar{\mathbf{J}}}^+ \underbrace{\begin{pmatrix} \mathbf{w}_1^* \\ \vdots \\ \mathbf{w}_{n_t}^* \end{pmatrix}}_{\bar{\mathbf{w}}}. \quad (2.214)$$

In case the row-rank of the stacked Jacobian matrix $\bar{\mathbf{J}}$ is larger than the column rank, the tasks are only fulfilled in a least square optimal sense $\|\bar{\mathbf{w}} - \bar{\mathbf{J}}\dot{\mathbf{q}}\|_2$. A way to weight some tasks higher than others is to use a weighted pseudo inverse matrix

$$\bar{\mathbf{J}}^{+W} = (\bar{\mathbf{J}}^T \mathbf{W} \bar{\mathbf{J}})^{-1} \bar{\mathbf{J}}^T \mathbf{W} \quad (2.215)$$

with a weighting matrix $\mathbf{W} = \text{diag}(w_1, \dots, w_m)$. This corresponds to the minimization of the error $\|\mathbf{W}^{1/2}(\bar{\mathbf{w}} - \bar{\mathbf{J}}\dot{\mathbf{q}})\|_2$. Since \mathbf{W} is a diagonal matrix, the Cholesky factor $1/2$ corresponds to element-wise square root.

Multi-task with Prioritization

An approach in order to clearly prioritize certain tasks with respect to others is to use consecutive null-space projection in order to determine the solution. Assume that the n_t tasks have decreasing priority. As we have seen in (2.210), the solution to $task_1$ is

$$\dot{\mathbf{q}} = \mathbf{J}_1^+ \mathbf{w}_1^* + \mathbf{N}_1 \dot{\mathbf{q}}_0. \quad (2.216)$$

Hence, in order not to violate the first objective, it must hold that

$$\mathbf{w}_2 = \mathbf{J}_2 \dot{\mathbf{q}} = \mathbf{J}_2 (\mathbf{J}_1^+ \mathbf{w}_1^* + \mathbf{N}_1 \dot{\mathbf{q}}_0). \quad (2.217)$$

This can be solved for $\dot{\mathbf{q}}_0$

$$\dot{\mathbf{q}}_0 = (\mathbf{J}_2 \mathbf{N}_1)^+ (\mathbf{w}_2^* - \mathbf{J}_2 \mathbf{J}_1^+ \mathbf{w}_1^*) \quad (2.218)$$

and back-substituted in (2.216) resulting in

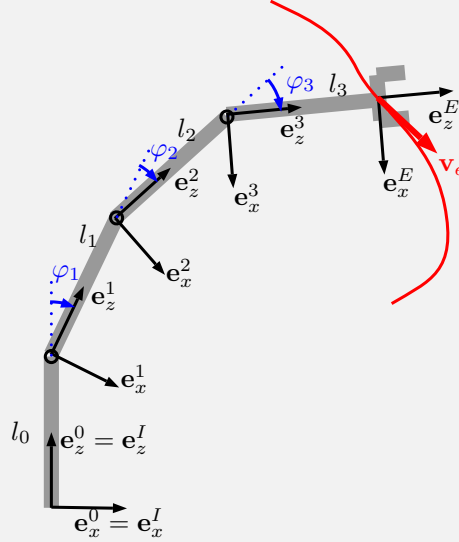
$$\dot{\mathbf{q}} = \mathbf{J}_1^+ \mathbf{w}_1^* + \mathbf{N}_1 (\mathbf{J}_2 \mathbf{N}_1)^+ (\mathbf{w}_2^* - \mathbf{J}_2 \mathbf{J}_1^+ \mathbf{w}_1^*). \quad (2.219)$$

For n_t tasks, the solution can be written in a recursive way as

$$\dot{\mathbf{q}} = \sum_{i=1}^{n_t} \bar{\mathbf{N}}_{i-1} \dot{\mathbf{q}}_i, \quad \text{with} \quad \dot{\mathbf{q}}_i = (\mathbf{J}_i \bar{\mathbf{N}}_{i-1})^+ \left(\mathbf{w}_i^* - \mathbf{J}_i \sum_{k=1}^{i-1} \bar{\mathbf{N}}_{k-1} \dot{\mathbf{q}}_k \right), \quad (2.220)$$

whereby $\bar{\mathbf{N}}_i$ is the null-space projection of the stacked Jacobian $\bar{\mathbf{J}}_i = [\mathbf{J}_1^T \dots \mathbf{J}_{i-1}^T]^T$.

Example 2.9.1: Multi-task Control



This example discusses again the 3DOF planar robot arm which has unitary link lengths. Given is a desired end-effector velocity ${}^0\dot{\mathbf{r}}_E^*(t)$ defined in \mathbb{R}^2 . For numerical evaluation, we consider an instant t when the robot is in configuration $\mathbf{q}_t = (\pi/6, \pi/3, \pi/3)^T$ and the desired end-effector velocity is ${}^0\dot{\mathbf{r}}_{E,t}^* = (1, 1)^T$.

What is the analytical solution to this end-effector tracking problem?

The desired end-effector motion is:

$$\mathbf{w}_E^* = {}^0\dot{\mathbf{r}}_E^*. \quad (2.221)$$

Using \mathbf{J}_E given by the x and z row of the Jacobians calculated in Example 2.8.6 and 2.8.7, the generalized velocity can be calculated as:

$$\dot{\mathbf{q}} = \mathbf{J}_E^+ \mathbf{w}_E^* \quad (2.222)$$

What is the numerical solution to this end-effector tracking problem?

The target velocity for this instant is

$$\mathbf{w}_{E,t}^* = {}^0\dot{\mathbf{r}}_{E,t}^* = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad (2.223)$$

Evaluation of the Jacobians for $\mathbf{q}_t = (\pi/6, \pi/3, \pi/3)^T$ gives

$$\mathbf{J}_{E,t} = \mathbf{J}_E(\mathbf{q}_t) = \frac{1}{2} \begin{bmatrix} 0 & -\sqrt{3} & -\sqrt{3} \\ -4 & -3 & -1 \end{bmatrix} \quad (2.224)$$

Taking the Moore-Penrose pseudo inverse results in

$$\dot{\mathbf{q}}_t^{single} = \mathbf{J}_{E,t}^+ \mathbf{w}_{E,t}^* = \begin{pmatrix} 0.069 \\ -0.560 \\ -0.595 \end{pmatrix}. \quad (2.225)$$

As we can see, these generalized velocities yield the exact task space (or end-effector) error velocity

$$\mathbf{w}_{E,t} = \mathbf{J}_{E,t} \dot{\mathbf{q}}_t^{single} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \mathbf{w}_{E,t}^* \quad (2.226)$$

Beside moving the end-effector with the desired velocity, we want that the first and second joint velocity is zero. What is the general solution if all tasks should be fulfilled with equal priority?

In addition to the previously defined task, we introduce an additional task to keep all joint velocities equal:

$$\mathbf{w}_j = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{J}_j = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad (2.227)$$

The stacked problem can now be solved for :

$$\dot{\mathbf{q}} = \begin{bmatrix} \mathbf{J}_E \\ \mathbf{J}_j \end{bmatrix}^+ \begin{pmatrix} \mathbf{w}_E^* \\ \mathbf{w}_j^* \end{pmatrix} \quad (2.228)$$

What is the numerical solution to the stacked problem?

$$\dot{\mathbf{q}}_t^{stack} = \begin{bmatrix} \mathbf{J}_{E,t} \\ \mathbf{J}_{j,t} \end{bmatrix}^+ \begin{pmatrix} \mathbf{w}_E^* \\ \mathbf{w}_j^* \end{pmatrix} \quad (2.229)$$

$$= \begin{pmatrix} -0.133 \\ -0.067 \\ -1.132 \end{pmatrix} \quad (2.230)$$

We notice that this joint velocity results in

$$\begin{bmatrix} \mathbf{J}_{E,t} \\ \mathbf{J}_{j,t} \end{bmatrix} \dot{\mathbf{q}}_t^{stack} = \begin{pmatrix} 1.039 \\ 0.933 \\ -0.133 \\ -0.067 \end{pmatrix} \neq \begin{pmatrix} \mathbf{w}_E^* \\ \mathbf{w}_{\Delta\dot{q}}^* \end{pmatrix} \quad (2.231)$$

It is interesting to have a look at the errors on both tasks, which are

$$\|\mathbf{w}_{E,t}^* - \mathbf{J}_{E,t} \dot{\mathbf{q}}_t^{stack}\|^2 = 0.0059 \quad (2.232)$$

$$\|\mathbf{w}_{j,t}^* - \mathbf{J}_{j,t} \dot{\mathbf{q}}_t^{stack}\|^2 = 0.0223. \quad (2.233)$$

As we can see, both tasks will be equally violated and neither of them is fulfilled. In comparison, the solution without joint velocity constraint (2.225) yields

$$\|\mathbf{w}_{E,t}^* - \mathbf{J}_{E,t}\dot{\mathbf{q}}_t^{single}\|^2 = 0 \quad (2.234)$$

$$\|\mathbf{w}_{j,t}^* - \mathbf{J}_{j,t}\dot{\mathbf{q}}_t^{single}\|^2 = 0.319. \quad (2.235)$$

The error in the second task (2.235) is larger than the sum of errors of both tasks (2.232) and (2.233) of the joint optimization. In other words, when optimizing for both tasks, we sacrifice end-effector tracking performance in order to reduce the velocity in joint 1 and 2.

What is the general solution if the end-effector tracking task has higher priority than keeping the first two joint velocities to zero?

To this end we use the formalism given in (2.219)

$$\dot{\mathbf{q}}_t^{prio} = \mathbf{J}_1^+ \mathbf{w}_1^* + \mathbf{N}_1 (\mathbf{J}_2 \mathbf{N}_1)^+ (\mathbf{w}_2^* - \mathbf{J}_2 \mathbf{J}_1^+ \mathbf{w}_1^*). \quad (2.236)$$

with $\mathbf{J}_1 = \mathbf{J}_{E,t}$, $\mathbf{J}_2 = \mathbf{J}_{j,t}$, $\mathbf{w}_1^* = \mathbf{w}_{E,t}^*$ and $\mathbf{w}_2^* = \mathbf{w}_{j,t}^*$. A possible null-space projector matrix is

$$\mathbf{N}_1 = \mathbb{I} - \mathbf{J}_1^+ \mathbf{J}_1 = \frac{1}{9} \begin{bmatrix} 1 & -2 & 2 \\ -2 & 4 & -4 \\ 2 & -4 & 4 \end{bmatrix} \quad (2.237)$$

Using \mathbf{N}_1 , which has $rank = 1$, the generalized velocity results in

$$\dot{\mathbf{q}}_t^{prio} = \begin{pmatrix} -0.169 \\ -0.085 \\ -1.070 \end{pmatrix} \quad (2.238)$$

Again, let us have a look at the errors

$$\|\mathbf{w}_{E,t}^* - \mathbf{J}_{E,t}\dot{\mathbf{q}}_t^{prio}\|^2 = 0 \quad (2.239)$$

$$\|\mathbf{w}_{j,t}^* - \mathbf{J}_{j,t}\dot{\mathbf{q}}_t^{prio}\|^2 = 0.036 \quad (2.240)$$

In comparison to the stacked solution, the total error is larger but the end-effector task is tracked accurately. When comparing to the single task optimization, it can be seen that the total error is reduced. In fact, the single dimensional null-space was used in an optimal sense in order to reduce the joint velocities without changing end-effector tracking performance.

2.9.3 Inverse Kinematics

The goal of *inverse kinematics* is to find a joint configuration \mathbf{q} as a function of a given end-effector configuration χ_e^* :

$$\mathbf{q} = \mathbf{q}(\chi_e^*) \quad (2.241)$$

Analytical Solution

There exist analytic solutions for a large class of mechanisms. In fact, many robot arms are particularly designed such that inverse kinematics can be analytically solved. For a classical 6DOF robot arm, the necessary condition for an analytical solution is that three neighboring axes intersect. There exist two widely used approaches to solve an inverse kinematics problem analytically. The *geometric* approach is to decompose the spacial geometry of the manipulator into several plane problems and apply geometric laws. The *algebraic* approach is to manipulate the transformation matrix to get the angles. For more information, the interested reader is referred to [8].

Numerical Solution

With increasing computational power, numerical approaches are a common tool to solve the inverse kinematics problem. As we have seen in section 2.8.4, differences in joint space coordinates $\Delta \mathbf{q}$ can be directly mapped to differences in end-effector coordinates $\Delta \chi_e$ using the analytical Jacobian:

$$\Delta \chi_e = \mathbf{J}_{eA} \Delta \mathbf{q}. \quad (2.242)$$

This relationship can be used to iteratively solve the inverse kinematics problem for a given desired end-effector configuration χ_e^* and start configuration \mathbf{q}^0 (see algorithm 1). The algorithm iterates through these steps until the target location \mathbf{q}^* is reached with a certain tolerance on $\|\Delta \chi_e\| = \|\chi_e^* - \chi_e(\mathbf{q}^*)\| < tol$.

Algorithm 1 Numerical Inverse Kinematics

```

1:  $\mathbf{q} \leftarrow \mathbf{q}^0$  ▷ Start configuration
2: while  $\|\chi_e^* - \chi_e(\mathbf{q})\| > tol$  do ▷ While the solution is not reached
3:    $\mathbf{J}_{eA} \leftarrow \mathbf{J}_{eA}(\mathbf{q}) = \frac{\partial \chi_e}{\partial \mathbf{q}}(\mathbf{q})$  ▷ Evaluate Jacobian for  $\mathbf{q}$ 
4:    $\mathbf{J}_{eA}^+ \leftarrow (\mathbf{J}_{eA})^+$  ▷ Calculate the pseudo inverse
5:    $\Delta \chi_e \leftarrow \chi_e^* - \chi_e(\mathbf{q})$  ▷ Find the end-effector configuration error vector
6:    $\mathbf{q} \leftarrow \mathbf{q} + \mathbf{J}_{eA}^+ \Delta \chi_e$  ▷ Update the generalized coordinates
7: end while

```

Unfortunately, this approach has some problems. First, if the error between the target and actual configuration $\Delta \chi_e^i$ is getting large, the error linearization as implemented by analytical Jacobian is not accurate enough. A simple way to deal with this issue is to scale each update step:

$$\mathbf{q} \leftarrow \mathbf{q} + k \mathbf{J}_{eA}^+ \Delta \chi_e \quad (2.243)$$

with $0 < k < 1$ in order to remain within the validity region of the linearization and avoid overshooting or divergence. However, this leads to a slower convergence of the solution. When working with very small k gains, the solution follows the local linearization represented by the analytical Jacobian until full convergence.

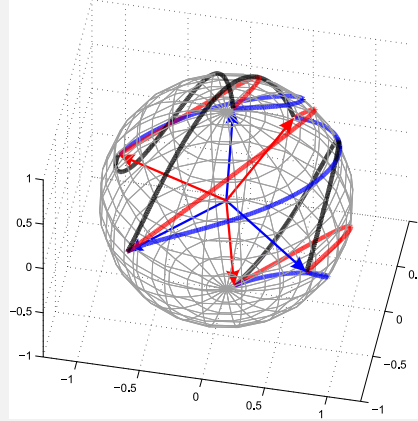
Second, in case the target is close to a singularity position the Jacobian inversion becomes a badly conditioned problem. As illustrated in section 2.9.1, one can for example use the damped inverse in order to overcome this problem. Another approach, for which a detailed explanation is given in [3], is to use the Jacobi-transposed method to replace the update step 6 by

$$\mathbf{q} \leftarrow \mathbf{q} + \alpha \mathbf{J}_{eA}^T \Delta \chi_e. \quad (2.244)$$

When choosing the parameter α small enough, convergence of the problem can be guaranteed. At the same time, there is no time-consuming inversion required.

At this point it is important to understand that the choice of parameterization leads to very different numerical behaviors as illustrated in Example 2.9.7.

Example 2.9.7: Inverse Kinematics of Rotations



As an example to illustrate the effect of different parameterizations, we consider an example with a simple robot arm that features three successive rotational joints with rotation axis z , y , and x respectively:

$$\mathbf{q} = \begin{pmatrix} q_z \\ q_y \\ q_x \end{pmatrix} \quad (2.245)$$

The selection of this set of generalized coordinates results in the rotation matrix given in (2.41) with $z = q_z$, $y = q_y$, and $x = q_x$. Following (2.164), the basic Jacobian is

$$\mathbf{J}_{e0_R} = \begin{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} & \mathbf{C}_z(q_z) \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} & \mathbf{C}_z(q_z) \mathbf{C}_y(q_y) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \end{bmatrix} \quad (2.246)$$

$$= \begin{bmatrix} 0 & -\sin(z) & \cos(y) \cos(z) \\ 0 & \cos(z) & \cos(y) \sin(z) \\ 1 & 0 & -\sin(y) \end{bmatrix}. \quad (2.247)$$

The initial conditions are selected as $\mathbf{q}^0 = (-0.7 \quad 0 \quad 1.5)^T$. For this configuration, the rotation of the end-effector expressed in Euler ZYX coordinates corresponds to $\chi_{R,eulerZYX} = (-0.7 \quad 0 \quad 1.5)^T$ indicated by the blue coordinate frame aligned with the center of the unit sphere. Please note: Since the Euler Angle parameterization has exactly the same order as joint rotations, we have $\chi_{R,eulerZYX} = \mathbf{q}$. The target orientation of the end-effector is selected as $\chi_{R,eulerZYX} = (0.7 \quad 1.5 \quad -0.5)^T$ (red frame). For parameterization of the rotation we select:

1. Euler angles ZYX: $\chi_{R,eulerZYX} = (z \quad y \quad x)^T$
2. Euler angles XYZ: $\chi_{R,eulerXYZ} = (x \quad y \quad z)^T$

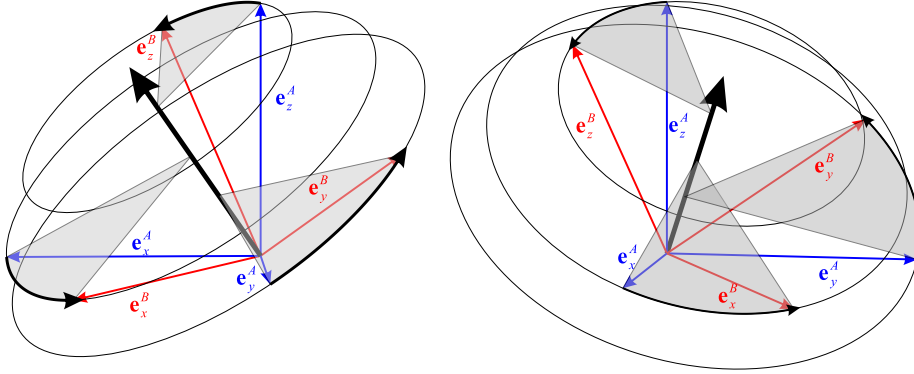


Figure 2.14: A 3D rotation from a start to a target coordinate system (A to B) can be represented by a rotation angle and a 3D rotation axis.

3. Rotation Vector: $\chi_{rotvec} = (\varphi_x \quad \varphi_y \quad \varphi_z)^T$

For every parameterization, the following steps have to be applied:

1. Evaluate the rotation matrix $\mathbf{C}^i = \mathbf{C}(\mathbf{q})$ for the actual generalized coordinates \mathbf{q}^i
2. Transform \mathbf{C}^i into end-effector parameters χ_R^i using
 - Euler ZYX: (2.42)
 - Euler XYZ: (2.45)
 - Rotation vector: (2.47), (2.49) and (2.50)
3. Evaluate the analytical Jacobian $\mathbf{J}_{eA_R}^i = \mathbf{E}_{\chi_R}(\chi_R^i)^{-1} \mathbf{J}_{e0_R}(\mathbf{q}^i)$ with \mathbf{E}_{χ_R} given by
 - Euler ZYX: (2.86)
 - Euler XYZ: (2.89)
 - Rotation vector: (2.106)
4. Calculate difference $\Delta \chi_R^i = \chi_R^* - \chi_R^i$
5. Update the generalized coordinates $\mathbf{q}^{i+1} = \mathbf{q}^i + k \mathbf{J}_{eA_R}^+ \Delta \chi_R^i$, with k being a small scaling factor to follow always the local linearization.

Depending on the robot kinematics and the choice of rotation parameterization (blue: Euler ZYX, red: Euler XYZ, black: rotation vector), the coordinate frame is rotated on different ways to the target system. The matlab code for this example can be found in appendix A.2

Appropriate Rotation Error

As we have seen in the previous example, the selection of rotation parameterization changes the convergence from the start to the goal configuration. However, ideally, we would always take the "shortest path", irrespective of the parametrization. Working with rotation errors that give the shortest possible rotation between two orientations will allow this. In order to better understand rotation errors in 3D we can have a look at Fig. 2.14. As we already know from section 2.4.5, a possible rotation parameterization is angle axis, i.e. an axis giving the direction of the rotation (black arrow) and an angle defining the magnitude of the rotation. Because the rotation axis remains constant throughout the rotation, this can be understood as the "shortest possible rotation". Hence, we choose rotation error $\Delta\phi \in \mathbb{R}^3$ to be a rotation vector as defined in (2.47),

$$\Delta\phi = \Delta\varphi, \quad (2.248)$$

which parameterizes the relative orientation of the target frame \mathcal{B} w.r.t. the current frame \mathcal{A} . The rotation matrix $\mathbf{C}_{\mathcal{AB}}$ associated with this relative orientation between the starting orientation $\mathbf{C}_{\mathcal{IA}}(\varphi^t)$ and the goal orientation $\mathbf{C}_{\mathcal{IB}}(\varphi^*)$ is given by

$$\mathbf{C}_{\mathcal{AB}}(\Delta\varphi) = \mathbf{C}_{\mathcal{IA}}(\varphi^t)^T \mathbf{C}_{\mathcal{IB}}(\varphi^*). \quad (2.249)$$

$\Delta\varphi$ is thus not simply $\varphi^* - \varphi^t$, but is calculated from $\mathbf{C}_{\mathcal{AB}}$ using (2.47), (2.49) and (2.50). It is important to note what frame the orientation error is represented in. Since the rotation vector is parallel to the axis of rotation, its representation in frame \mathcal{A} and \mathcal{B} is equal and given by

$${}_{\mathcal{A}}\Delta\varphi = {}_{\mathcal{B}}\Delta\varphi = \text{rotVec}(\mathbf{C}_{\mathcal{AB}}). \quad (2.250)$$

Afterwards, the rotation error can be mapped to the frame of interest, e.g. the inertial frame.

Alternatively, the rotation error represented in the inertial frame can be directly found through

$${}_{\mathcal{I}}\Delta\varphi = \text{rotVec}(\mathbf{C}_{\mathcal{IB}}\mathbf{C}_{\mathcal{IA}}^T). \quad (2.251)$$

Note here that $\mathbf{C}_{\mathcal{IB}}\mathbf{C}_{\mathcal{IA}}^T$ does not correspond to the usual concatenation of rotations. It represents the relative orientation of frame \mathcal{B} w.r.t. \mathcal{A} frame *as seen from the inertial frame*. The presented equation (2.251) can be derived as follows:

$${}_{\mathcal{I}}\Delta\varphi = \mathbf{C}_{\mathcal{IA}} {}_{\mathcal{A}}\Delta\varphi = \mathbf{C}_{\mathcal{IA}} \text{rotVec}(\mathbf{C}_{\mathcal{AB}}) \quad (2.252)$$

$$= \text{rotVec}(\mathbf{C}_{\mathcal{IA}}\mathbf{C}_{\mathcal{AB}}\mathbf{C}_{\mathcal{IA}}^T) \quad (2.253)$$

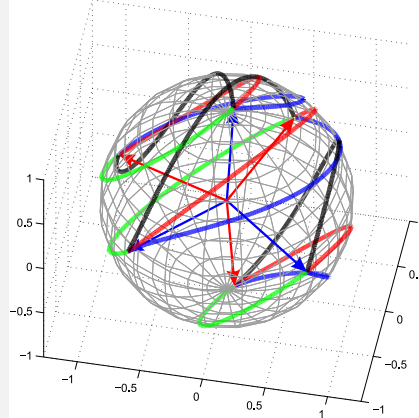
$$= \text{rotVec}(\mathbf{C}_{\mathcal{IB}}\mathbf{C}_{\mathcal{IA}}^T) \quad (2.254)$$

Given the definition of the angular velocity in (2.70), we can change the update step 6 of algorithm 1 to

$$\mathbf{q} \leftarrow \mathbf{q} + k_{p_R} {}_{\mathcal{I}}\mathbf{J}_{e0_R}^+ {}_{\mathcal{I}}\Delta\varphi. \quad (2.255)$$

In case k_{p_R} is taken very small, the algorithm will rotate on the "shortest path" from the start to the target configuration. Please note two things: First, this update step takes the geometric instead of the analytical Jacobian. Second, in contrast to the previous formulations, $\Delta\varphi$ is not equivalent to the difference in rotation vectors of start and goal rotation, but rather represents the rotation vector of the "difference rotation".

Example 2.9.8: Inverse Kinematics using Rotation Vector



This example considers the same problem as in example 2.9.7, but applies the inverse kinematics approach using a rotation vector description of the rotation error.

Given the target rotation matrix \mathbf{C}^* , the following steps are applied:

1. Get the rotation matrix $\mathbf{C}^i = \mathbf{C}(\mathbf{q})$ for the actual generalized coordinates \mathbf{q}^i .
2. Calculate the relative rotation $\mathbf{C}^{rel} = \mathbf{C}^* \cdot \mathbf{C}^{iT}$.
3. Calculate the rotation vector $\Delta\varphi$ of the relative rotation using (2.47), (2.49) and (2.50).
4. update the generalized coordinates $\mathbf{q}^{i+1} = \mathbf{q}^i + k\mathbf{J}_{e0R}^+ \Delta\varphi$, with k being a small scaling factor to follow always the local linearization.

As we can see, the solution follows the green path (= "direct line") from the start to the goal location. The matlab code for this example can be found in appendix A.2

2.9.4 Trajectory Control

Pure inverse differential kinematics is often insufficient to follow a predefined task-space trajectory, as the pose will drift away if no feedback is provided for position or rotation. Hence, trajectory control can be achieved by stabilizing inverse differential kinematics through a weighted tracking error feedback. This results in smooth and drift-free motion.

Position Trajectory Control

For position tracking with a given pre-planned position $\mathbf{r}_e^*(t)$ and velocity $\dot{\mathbf{r}}_e^*(t)$, the problem is trivial. The feedback control part should bring

$$\Delta \mathbf{r}_e^t = \mathbf{r}_e^*(t) - \mathbf{r}_e(\mathbf{q}^t) \quad (2.256)$$

to zero, which results in the following trajectory controller:

$$\dot{\mathbf{q}}^* = \mathbf{J}_{e0_P}^+(\mathbf{q}^t) \cdot (\dot{\mathbf{r}}_e^*(t) + k_{p_P} \Delta \mathbf{r}_e^t). \quad (2.257)$$

Thereby, k_{p_P} represents the position feedback, defining how quickly the actual position converges to the target position. If this algorithm is implemented as digital control problem with fixed time step Δt , one can provide stability boundaries for k_{p_P} by an Eigenvalue analysis (see e.g. [8]).

Orientation Trajectory Control

Things are getting more complicated when tracking orientations $\chi_R^*(t)$ and angular velocities $\omega^*(t)$. As we have seen in the inverse kinematics example 2.9.7, the selection of parameterization leads to different trajectories. The best approach is to follow the shortest rotation approach, which is equivalent to the position tracking. With this we can write:

$$\dot{\mathbf{q}}^* = \mathbf{J}_{e0_R}^+(\mathbf{q}^t) \cdot (\omega_e^*(t) + k_{p_R} \Delta \varphi), \quad (2.258)$$

whereby $\Delta \varphi$ is calculated as described in section 2.9.3. Please note that this formulation does not require an analytical Jacobian.

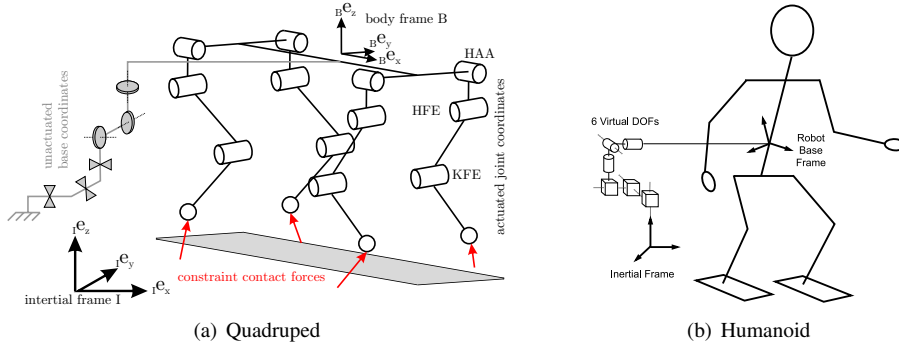


Figure 2.15: Free-floating robots have an un-actuated base.

2.10 Floating Base Kinematics

Free-floating robots like the quadraped and humanoid depicted in Fig. 2.15 are described by n_b un-actuated base coordinates \mathbf{q}_b and n_j actuated joint coordinates \mathbf{q}_j :

$$\mathbf{q} = \begin{pmatrix} \mathbf{q}_b \\ \mathbf{q}_j \end{pmatrix} \quad (2.259)$$

The unactuated base is free in translation and rotation

$$\mathbf{q}_b = \begin{pmatrix} \mathbf{q}_{b_P} \\ \mathbf{q}_{b_R} \end{pmatrix} \in \mathbb{R}^3 \times SO(3), \quad (2.260)$$

whereby the position \mathbf{q}_{b_P} and rotation \mathbf{q}_{b_R} can be parameterized using different representations as seen in sections 2.2.1 and 2.4.5. Hence, the dimension of the generalized coordinate vector of a floating base system $n_b + n_j$ depends on the parameterization of the rotation, whereby the minimal number of generalized coordinates for the base is $n_{b0} = 6$.

There are several challenges linked with floating base systems. First, there are typically no (onboard) sensors that allow to directly measure the base position and orientation. To cope with this, some research areas use motion tracking systems, i.e. external cameras and markers on the robot that allows measuring the pose of the base. Otherwise, it is necessary to use sensor fusion algorithms in order to estimate the pose from different other sensor information. Second, since the base is not directly actuated, the motion of the system of bodies (i.e. the total linear and angular impulse) can only be changed through additional external forces resulting from contacts (see also section 2.10.4).

2.10.1 Generalized Velocity and Acceleration

Since differentiation in $SO(3)$ is different from \mathbb{R}^3 , people often introduce generalized velocity and acceleration vectors

$$\mathbf{u} = \begin{pmatrix} I \mathbf{v}_B \\ {}_B \boldsymbol{\omega}_{IB} \\ \dot{\varphi}_1 \\ \vdots \\ \dot{\varphi}_{n_j} \end{pmatrix} \in \mathbb{R}^{6+n_j} = \mathbb{R}^{n_u} \quad \dot{\mathbf{u}} = \begin{pmatrix} I \mathbf{a}_B \\ {}_B \dot{\boldsymbol{\psi}}_{IB} \\ \ddot{\varphi}_1 \\ \vdots \\ \ddot{\varphi}_{n_j} \end{pmatrix} \in \mathbb{R}^{6+n_j} \quad (2.261)$$

As shown in section 2.5.1, there exists a direct mapping from rotational velocity ω to time derivatives of end-effector rotation coordinates $\dot{\mathbf{q}}_{bR}$ such that

$$\mathbf{u} = \mathbf{E}_{fb} \cdot \dot{\mathbf{q}}, \text{ with } \mathbf{E}_{fb} = \begin{bmatrix} \mathbb{I}_{3 \times 3} & 0 & 0 \\ 0 & \mathbf{E}_{\chi_R} & 0 \\ 0 & 0 & \mathbb{I}_{n_j \times n_j} \end{bmatrix} \quad (2.262)$$

whereby the rotation parameterization dependent matrices \mathbf{E}_{χ_R} and its inverse $\mathbf{E}_{\chi_R}^{-1}$ are given in

- Euler Angles
 - XYZ: (2.88) and (2.89)
 - ZYX: (2.86) and (2.87)
 - ZYZ: (2.90) and (2.91)
- Quaternions (2.98) and (2.99)
- Angle Axis (2.102) and (2.103)
- Rotation Vector (2.106) and (2.107)

Please note that many text-books write $\dot{\mathbf{q}}$, but implicitly mean \mathbf{u} and not the time-derivative of the parameterization. For fixed base systems, there is typically no difference. However, whenever working with floating base systems, you must be aware of the difference. In these lecture notes we try to be consistent in this context and hence always use \mathbf{u} when specifically talking about floating base systems.

2.10.2 Forward Kinematics

We wish to derive the relationship between the generalized velocities \mathbf{u} and the operational space velocities ${}^{\mathcal{I}}\mathbf{v}_Q$ of a point Q , which is fixed at the end of a kinematic chain that stems from a floating base B . The position vector ${}^{\mathcal{I}}\mathbf{r}_{IQ} = {}^{\mathcal{I}}\mathbf{r}_{IQ}(\mathbf{q})$ of a point w.r.t. the inertial frame \mathcal{I} is given by:

$${}^{\mathcal{I}}\mathbf{r}_{IQ}(\mathbf{q}) = {}^{\mathcal{I}}\mathbf{r}_{IB}(\mathbf{q}) + \mathbf{C}_{TB}(\mathbf{q}) \cdot {}^B\mathbf{r}_{BQ}(\mathbf{q}), \quad (2.263)$$

where the rotation matrix $\mathbf{C}_{TB}(\mathbf{q})$ describes the orientation of the floating base \mathcal{B} w.r.t. the inertial frame \mathcal{I} , ${}^{\mathcal{I}}\mathbf{r}_{IB}(\mathbf{q})$ represents the position of the floating base \mathcal{B} w.r.t. the inertial frame \mathcal{I} expressed in the inertial frame, ${}^B\mathbf{r}_{BQ}(\mathbf{q})$ represents the position of Q w.r.t. the floating Base B expressed in frame \mathcal{B} , and $\mathbf{q} = \mathbf{q}(t)$ is a function of time t .

2.10.3 Differential Kinematics of Floating Base Systems

Time differentiation of the position vector (2.263) yields:

$$\begin{aligned} {}^{\mathcal{I}}\mathbf{v}_Q &= {}^{\mathcal{I}}\mathbf{v}_B + \dot{\mathbf{C}}_{TB} \cdot {}^B\mathbf{r}_{BQ} + \mathbf{C}_{TB} \cdot {}^B\dot{\mathbf{r}}_{BQ} \\ &= {}^{\mathcal{I}}\mathbf{v}_B + \mathbf{C}_{TB} \cdot [{}^B\boldsymbol{\omega}_{TB}]_{\times} \cdot {}^B\mathbf{r}_{BQ} + \mathbf{C}_{TB} \cdot {}^B\dot{\mathbf{r}}_{BQ} \\ &= {}^{\mathcal{I}}\mathbf{v}_B - \mathbf{C}_{TB} \cdot [{}^B\mathbf{r}_{BQ}]_{\times} \cdot {}^B\boldsymbol{\omega}_{TB} + \mathbf{C}_{TB} \cdot {}^B\dot{\mathbf{r}}_{BQ} \\ &= {}^{\mathcal{I}}\mathbf{v}_B - \mathbf{C}_{TB} \cdot [{}^B\mathbf{r}_{BQ}]_{\times} \cdot {}^B\boldsymbol{\omega}_{TB} + \mathbf{C}_{TB} \cdot {}^B\mathbf{J}_{P_{q_j}}(\mathbf{q}_j) \cdot \dot{\mathbf{q}}_j \\ &= \begin{bmatrix} \mathbb{I}_{3 \times 3} & -\mathbf{C}_{TB} \cdot [{}^B\mathbf{r}_{BQ}]_{\times} & \mathbf{C}_{TB} \cdot {}^B\mathbf{J}_{P_{q_j}}(\mathbf{q}_j) \end{bmatrix} \cdot \mathbf{u} \end{aligned} \quad (2.264)$$

If we attach a frame at $\mathcal{I}r_{IQ}$, we can derive a similar mapping for angular velocities. The orientation of frame \mathcal{Q} w.r.t. the inertial frame \mathcal{I} is described by:

$$\mathbf{C}_{IQ} = \mathbf{C}_{IB} \cdot \mathbf{C}_{BQ} \quad (2.265)$$

Time differentiation of both sides of (2.265) yields:

$$\begin{aligned} [\mathcal{I}\boldsymbol{\omega}_{IQ}]_{\times} \cdot \mathbf{C}_{IQ} &= [\mathcal{I}\boldsymbol{\omega}_{IB}]_{\times} \cdot \mathbf{C}_{IB} \cdot \mathbf{C}_{BQ} + \mathbf{C}_{IB} \cdot [\mathcal{B}\boldsymbol{\omega}_{BQ}]_{\times} \cdot \mathbf{C}_{BQ} \\ &= [\mathcal{I}\boldsymbol{\omega}_{IB}]_{\times} \cdot \mathbf{C}_{IQ} + \mathbf{C}_{IB} \cdot \mathbf{C}_{BI} \cdot [\mathcal{I}\boldsymbol{\omega}_{BQ}]_{\times} \cdot \mathbf{C}_{BI}^T \cdot \mathbf{C}_{BQ} \\ &= [\mathcal{I}\boldsymbol{\omega}_{IB}]_{\times} \cdot \mathbf{C}_{IQ} + [\mathcal{I}\boldsymbol{\omega}_{BQ}]_{\times} \cdot \mathbf{C}_{IQ}, \end{aligned} \quad (2.266)$$

which gives finally:

$$\begin{aligned} \mathcal{I}\boldsymbol{\omega}_{IQ} &= \mathcal{I}\boldsymbol{\omega}_{IB} + \mathcal{I}\boldsymbol{\omega}_{BQ} \\ &= \begin{bmatrix} \mathbf{0}_{3 \times 3} & \mathbf{C}_{IB} & \mathbf{C}_{IB} \cdot \mathcal{B}\mathbf{J}_{R_{q_j}}(\mathbf{q}_j) \end{bmatrix} \cdot \mathbf{u} \end{aligned} \quad (2.267)$$

Hence, the mapping from generalized velocities \mathbf{u} to the operational space twist $[\mathcal{I}\mathbf{v}_Q^T \quad \mathcal{I}\boldsymbol{\omega}_{IQ}^T]^T$ of frame \mathcal{Q} is realized by the spatial Jacobian:

$$\begin{aligned} \mathcal{I}\mathbf{J}_Q(\mathbf{q}) &= \begin{bmatrix} \mathcal{I}\mathbf{J}_P \\ \mathcal{I}\mathbf{J}_R \end{bmatrix} \\ &= \begin{bmatrix} \mathbb{I}_{3 \times 3} & -\mathbf{C}_{IB} \cdot [\mathcal{B}\mathbf{r}_{BQ}]_{\times} & \mathbf{C}_{IB} \cdot \mathcal{B}\mathbf{J}_{P_{q_j}}(\mathbf{q}_j) \\ \mathbf{0}_{3 \times 3} & \mathbf{C}_{IB} & \mathbf{C}_{IB} \cdot \mathcal{B}\mathbf{J}_{R_{q_j}}(\mathbf{q}_j) \end{bmatrix} \end{aligned} \quad (2.268)$$

2.10.4 Contacts and Constraints

In kinematics, contacts between the robot and its environment can be modeled as kinematic constraints. Every point C_i that is in contact with the environment (attached to coordinate frame \mathcal{I}) imposes three constraints

$$\mathcal{I}\mathbf{r}_{IC_i} = \text{const}, \quad \mathcal{I}\dot{\mathbf{r}}_{IC_i} = \mathcal{I}\ddot{\mathbf{r}}_{IC_i} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \quad (2.269)$$

These contact constraints can be expressed as a function of the generalized velocities and accelerations using the contact point Jacobian

$$\mathcal{I}\mathbf{J}_{C_i}\mathbf{u} = \mathbf{0}, \quad \mathcal{I}\mathbf{J}_{C_i}\dot{\mathbf{u}} + \mathcal{I}\dot{\mathbf{J}}_{C_i}\mathbf{u} = \mathbf{0} \quad (2.270)$$

In case there are n_c active contacts, the constraints are simply stacked to

$$\mathbf{J}_c = \begin{bmatrix} \mathbf{J}_{C_1} \\ \vdots \\ \mathbf{J}_{C_{n_c}} \end{bmatrix} \in \mathbb{R}^{3n_c \times n_n}. \quad (2.271)$$

The $\text{rank}(\mathbf{J}_c)$ indicates the number of independent contact constraints. This stacked Jacobian can be split into

$$\mathbf{J}_c = [\mathbf{J}_{c,b} \quad \mathbf{J}_{c,j}] \quad (2.272)$$

whereby $\mathbf{J}_{c,b}$ indicates the relation between base motion and contact constraints. In case the rank of $\mathbf{J}_{c,b}$ (i.e. number of base constraints) is full ($\text{rank}(\mathbf{J}_{c,b}) = 6$), the system features enough constraints such that the base motion can be controlled from joint motion. The difference between the number of independent contact constraints $\text{rank}(\mathbf{J}_c)$ and the number of base constraints $\text{rank}(\mathbf{J}_{c,b})$ is the number of internal kinematic constraints that must be fulfilled.

Point Contacts - Quadruped

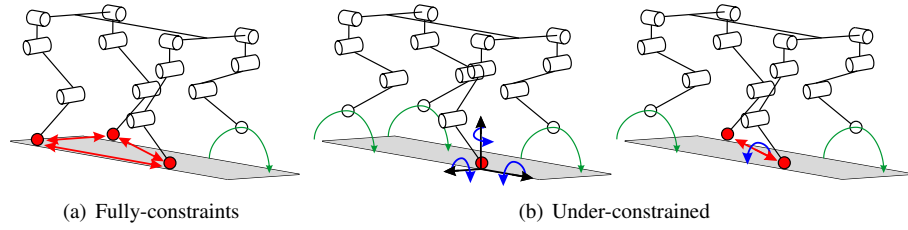


Figure 2.16: Depending on the number and arrangement of point contacts, a quadruped is fully-constrained (a) or under-constrained (b).

The point feet of a quadrupedal robot impose three (independent) constraints each. In case of two point contacts, the stacked contact Jacobian has $\text{rank}(\mathbf{J}_c) = 6$ but the Jacobian w.r.t. base coordinates has only $\text{rank}(\mathbf{J}_{c,b}) = 5$. This implies that the system is under-actuated and the base can not be arbitrarily moved by the joints. This becomes intuitively clear when looking at Fig. 2.16(b), as the robot cannot change the orientation around the line of support.

In contrast thereto, three point contacts as illustrated in Fig. 2.16(a) imply $\text{rank}(\mathbf{J}_c) = 9$ and $\text{rank}(\mathbf{J}_{c,b}) = 6$. This means that the body position and orientation is fully controllable through the joints. At the same time, there are three internal constraints that can be interpreted by the fact that the three legs cannot be moved one with respect to another.

Extended Contacts - Humanoid

For systems with extended feet, additional constraints are required in order to limit the foot rotation. One possible option is to introduce a rotational Jacobian. Much more common is to assign multiple contact points on the same link. A single contact point (Fig. 2.18, left) imposes three constraints as already seen in the previous section. In case of two contact points, the rank of the constraints is $\text{rank}(\mathbf{J}_c) = \text{rank}(\mathbf{J}_{c,b}) = 5$ and in case of three points assigned to the same element we get $\text{rank}(\mathbf{J}_c) = \text{rank}(\mathbf{J}_{c,b}) = 6$ although $\mathbf{J}_c \in \mathbb{R}^{9 \times n_j}$.

2.10.5 Support Consistent Inverse Kinematics

Applying inverse kinematics to floating base systems answers the question how to move individual joints in order to achieve certain task-space motion without violating contact constraints. This seeks for the application of a multi-task approach with prioritization, whereby the contact constraints are considered to have higher priority

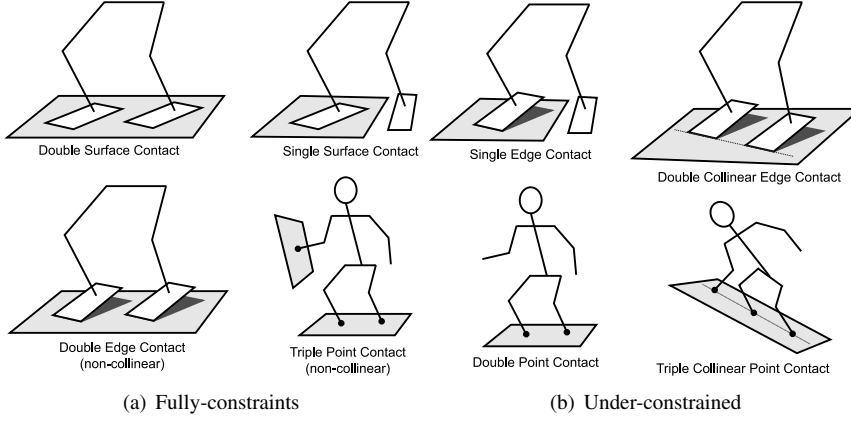


Figure 2.17: Depending on the number and arrangement of point contacts, a humanoid is fully-constrained (a) or under-constrained (b).

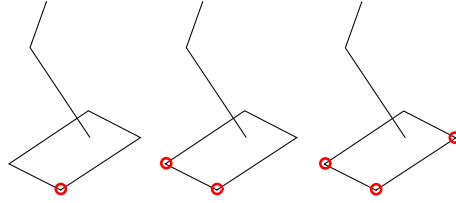


Figure 2.18: Multiple contact points attached to a single foot.

than the task-space motion. As introduced above, contact constraints of the n_c legs in ground contact are given by

$$\mathbf{J}_c \mathbf{u} = \mathbf{0}, \quad (2.273)$$

which implies that the motion of the system in contact is given by

$$\mathbf{u} = \mathbf{J}_c^+ \mathbf{0} + \mathcal{N}(\mathbf{J}_c) \mathbf{u}_0 = \mathbf{N}_c \mathbf{u}_0. \quad (2.274)$$

Hence, given a demanded task space motion

$$\mathbf{w}_t = \mathbf{J}_t \mathbf{u}_i \quad (2.275)$$

the joint velocity required to achieve this is

$$\mathbf{u} = \mathbf{N}_c (\mathbf{J}_t \mathbf{N}_c)^+ \mathbf{w}_t \quad (2.276)$$

Chapter 3

Dynamics

Mathematical models of a robot's dynamics provide a description of *why things move* when forces are generated in and applied on the system. They play an important role for both *simulation* and *control*. This chapter presents a concise overview regarding different approaches for modeling the dynamics of a robot and provides insight into model-based control techniques.

3.1 Introduction

For many applications with fixed-based robots we need to find a multi-body dynamics formulated as

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{J}_c(\mathbf{q})^T \mathbf{F}_c \quad (3.1)$$

consisting of the following components:

$\mathbf{M}(\mathbf{q})$	$\in \mathbb{R}^{n_q \times n_q}$	Generalized mass matrix (orthogonal).
$\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$	$\in \mathbb{R}^{n_q}$	Generalized position, velocity and acceleration vectors.
$\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}})$	$\in \mathbb{R}^{n_q}$	Coriolis and centrifugal terms.
$\mathbf{g}(\mathbf{q})$	$\in \mathbb{R}^{n_q}$	Gravitational terms.
$\boldsymbol{\tau}$	$\in \mathbb{R}^{n_q}$	External generalized forces.
\mathbf{F}_c	$\in \mathbb{R}^{3 \times n_c}$	External Cartesian forces (e.g. from contacts).
$\mathbf{J}_c(\mathbf{q})$	$\in \mathbb{R}^{n_c \times n_q}$	Geometric Jacobian corresponding to the external forces.

Please note: For simplicity and compactness, we use here the formulation for fixed base systems with $\ddot{\mathbf{q}}$ instead of $\ddot{\mathbf{u}}$ as introduced for floating base systems. Section 3.7 will specifically deal with the dynamics of floating base systems.

In literature, different methods exist to compute the so-called Equations of Motion (EoM) of a given system, i.e., a closed-form mathematical model of the system dynamics. All such methods are usually based on *Newtonian* and/or *Lagrangian* mechanics formulations, but despite the different approaches taken, all methods will result in *equivalent* descriptions of the dynamics.

In this text we present the most common methods used in robotics. The first such approach presented in section 3.3, is the well-known classical *Newton-Euler* method, which essentially applies the principles of conservation of linear and angular momentum for all links of a robot, and considers the motion explicitly in Cartesian space. The second approach presented in section 3.4, known as the *Lagrange Method*, is an analytical technique which utilizes scalar energy-based functions over the the space of

generalized coordinates which adhere to certain minimization principles, thus resulting in trajectories which automatically satisfy the kinematic constraints of the system. The third and arguably most useful approach of all is presented in section 3.5 and is that of the *Projected Newton-Euler* method, which manages to combine the advantages of both the Newton-Euler and Lagrange methods, thus constituting a reformulation of the Newton-Euler method in terms of the generalized coordinates and hence directly considers the feasible motions of the system, i.e., those satisfying the applicable kinematic constraints.

Although the methods are often presented to be quite different, and may appear as such initially, their equivalence will become quite clear by the end. In the end, it is up to the reader to identify the appropriate approach to use for modeling a given robotic system or problem setting.

3.2 Foundations from Classical Mechanics

Prior to presenting each of the aforementioned methods for deriving the EoM, there are certain principles and necessary analytical tools which one must be acquainted with first. These are necessary for understanding the foundations upon which the presented methods are based.

3.2.1 Newton's Law for Particles

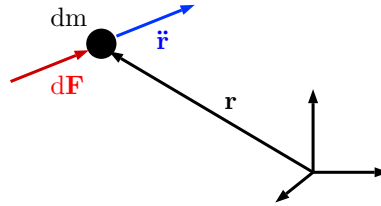


Figure 3.1: Force acting on single particle

The most basic formulation of Newtonian mechanics describes the motion of point-masses, i.e., *particles* with mass m and of infinitely small dimensions, where the entire mass is concentrated at the single point defined by the position vector \mathbf{r} . Keep in mind that a point-mass does not have an orientation as it is impossible to define any geometry. This means that we need only consider the Newton's second law to fully describe the motion of the system:

$$\ddot{\mathbf{r}}m = \mathbf{F} \quad (3.2)$$

Considering again the fact that we have defined the mass to be of infinitely small dimensions, we can think of it as an *infinitesimal mass* dm subject to *infinitesimal forces* $d\mathbf{F}$ concentrated at the position of the particle:

$$\ddot{\mathbf{r}}dm = d\mathbf{F} \quad (3.3)$$

As we will see in the sections that follow, these relations still play an important role in the derived methods. (3.2) still applies for the case of computing the linear dynamics of the Center of Mass (CoM) of rigid bodies, while (3.3) is used for to describe the effects of internal forces in a system via the Principle of Virtual Work.

3.2.2 Virtual Displacements

We will now introduce the concept of *variational notation* via the δ operator. Essentially, this operator behaves exactly as the differential d operator except that it has a completely different meaning. The differential is defined to describe an infinitesimal quantity which is then used to define “rates of change” of one quantity with respect to another. On the contrary the variation of a quantity describes, for a fixed instant in time, all possible directions the quantity may move in while adhering to the applicable constraints.

The most important fact to remember regarding a variation, is that if it taken with respect to quantity which is a function of time, then any temporal dependence is disregarded completely. To demonstrate this property, we will define the *admissible* variation of a position vector, called a *virtual displacement* $\delta \mathbf{r}$, and assume that the position is a function of generalized coordinates \mathbf{q} and time t , thus $\mathbf{r} = \mathbf{r}(\mathbf{q}, t)$. By applying the chain rule over the n_q elements of \mathbf{q} we compute the variation $\delta \mathbf{r}$:

$$\delta \mathbf{r}(\mathbf{q}, t) = \sum_{k=1}^{n_q} \frac{\partial \mathbf{r}}{\partial q_k} \delta q_k \quad (3.4)$$

Notice that the expression in (3.4) does not include $\frac{\partial \mathbf{r}}{\partial t}$, and can thus be understood to be a purely geometric interpretation of displacement.

3.2.3 Virtual Displacement of Single Rigid Bodies

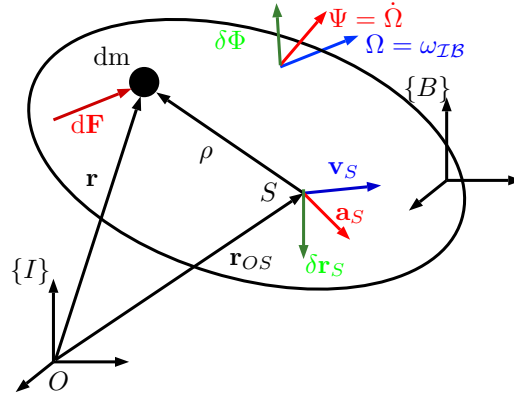


Figure 3.2: Kinematics of a single body

A body with mass existing in 3D Cartesian space, as shown in Fig. 3.2, is very simply a large number of particles closely placed together to form a single rigid body. Each infinitesimal point-mass dm is subject to the motion of the total body and hence can be assigned an absolute position and velocity at any instant in time. Lets now consider another point S on the body, such that we can define the relative position ρ of dm w.r.t. the point S .

Thus one can then use the rigid body kinematics formulation introduced in section 2.7 to describe the motion an arbitrary point-mass dm defined on the body \mathcal{B} via another body point S :

$$\mathbf{r} = \mathbf{r}_{OS} + \boldsymbol{\rho} \quad (3.5)$$

$$\dot{\mathbf{r}} = \mathbf{v}_S + \boldsymbol{\Omega} \times \boldsymbol{\rho} = \begin{bmatrix} \mathbb{I}_{3 \times 3} & -[\boldsymbol{\rho}]_{\times} \end{bmatrix} \begin{pmatrix} \mathbf{v}_S \\ \boldsymbol{\Omega} \end{pmatrix} \quad (3.6)$$

$$\ddot{\mathbf{r}} = \mathbf{a}_S + \boldsymbol{\Psi} \times \boldsymbol{\rho} + \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \boldsymbol{\rho}) = \begin{bmatrix} \mathbb{I}_{3 \times 3} & -[\boldsymbol{\rho}]_{\times} \end{bmatrix} \begin{pmatrix} \mathbf{a}_S \\ \boldsymbol{\Psi} \end{pmatrix} + [\boldsymbol{\Omega}]_{\times} [\boldsymbol{\Omega}]_{\times} \boldsymbol{\rho} \quad (3.7)$$

Where, r_{OS} is the absolute position of body point S , $\boldsymbol{\rho}$ is the relative position of dm w.r.t S , and \mathbf{v}_S , $\boldsymbol{\Omega}$, \mathbf{a}_S , and $\boldsymbol{\Psi}$ are the absolute velocities and accelerations of point S . Applying (3.4) to (3.5), we get the expression for the virtual displacement of the body element dm :

$$\delta \mathbf{r} = \delta \mathbf{r}_S + \delta \boldsymbol{\Phi} \times \boldsymbol{\rho} = \begin{bmatrix} \mathbb{I}_{3 \times 3} & -[\boldsymbol{\rho}]_{\times} \end{bmatrix} \begin{pmatrix} \delta \mathbf{r}_S \\ \delta \boldsymbol{\Phi} \end{pmatrix} \quad (3.8)$$

and $\delta \boldsymbol{\Phi}$ is the variation of the infinitesimal rotation of the local body-fixed frame defined at S and describes all constraint-compatible changes in orientation of the local frame at S . Note the bold face over the entire quantity of $\delta \boldsymbol{\Phi}$; this is due to the fact that the variation is not taken w.r.t. an orientation quantity, just like angular velocity is not the result of time-differentiating an orientation.

3.2.4 Virtual Displacement of Multi-Body Systems

Multi-body systems can only exhibit motions that are compatible with the constraints enforced by the joints, which limit the relative motion between links. As we have seen section 2.8, the motion is typically described using generalized coordinates \mathbf{q} :

$$\begin{pmatrix} \mathbf{v}_S \\ \boldsymbol{\Omega} \end{pmatrix} = \begin{bmatrix} \mathbf{J}_P \\ \mathbf{J}_R \end{bmatrix} \dot{\mathbf{q}} \quad (3.9)$$

$$\begin{pmatrix} \mathbf{a}_S \\ \boldsymbol{\Psi} \end{pmatrix} = \begin{bmatrix} \mathbf{J}_P \\ \mathbf{J}_R \end{bmatrix} \ddot{\mathbf{q}} + \begin{bmatrix} \dot{\mathbf{J}}_P \\ \dot{\mathbf{J}}_R \end{bmatrix} \dot{\mathbf{q}} \quad (3.10)$$

Again applying (3.4) to (3.9) and (3.10), the concept of virtual displacements now becomes relevant to multi-body systems, and implies that virtual displacements that are consistent with the joints must have the form:

$$\begin{pmatrix} \delta \mathbf{r}_S \\ \delta \boldsymbol{\Phi} \end{pmatrix} = \begin{bmatrix} \mathbf{J}_P \\ \mathbf{J}_R \end{bmatrix} \delta \mathbf{q} \quad (3.11)$$

3.2.5 Principle of Virtual Work

A fundamental principle in mechanics is the *principle of virtual work* which describes the fact that configuration constraints actually define forces which do not perform work in the direction of the virtual displacements. For a constraint force \mathbf{F}_c applied at point \mathbf{r}_c , contributed by an arbitrary joint, and equally onto the relevant bodies (action-reaction principle), the principle of virtual work states that:

$$\delta W = \delta \mathbf{r}_c^T \cdot \mathbf{F}_c = 0 \quad (3.12)$$

We can extend this further by considering again an infinitesimal dm in a body \mathcal{B} . By also considering *d'Alembert's Principle* describing dynamic equilibrium for particles, the virtual work integrated over the entire geometry of body \mathcal{B} becomes:

$$\delta W = \int_{\mathcal{B}} \delta \mathbf{r}^T \cdot (\ddot{\mathbf{r}} dm - d\mathbf{F}_{ext}) = 0, \quad \forall \delta \mathbf{r} \quad (3.13)$$

Where the quantities are defined as follows:

dm	infinitesimal mass element
$d\mathbf{F}_{ext}$	external forces acting on element dm
$\ddot{\mathbf{r}}$	acceleration of element dm
$\delta \mathbf{r}$	virtual displacement of dm
\mathcal{B}	Body system containing infinitesimal particles dm

Thus the virtual work is zero for all displacements $\delta \mathbf{r}$, those changes in configuration which are consistent with the constraints of the system. Although this formulation might seem somewhat complicated, it will become clearer in the continuation that it describes most of the well known concepts in a quite compact and well understandable manner.

3.3 Newton-Euler Method

3.3.1 Newton-Euler for Single Bodies

Evaluating the principle of virtual work (3.13) for a single body results to:

$$\begin{aligned} 0 = \delta W &= \int_{\mathcal{B}} \begin{pmatrix} \delta \mathbf{r}_s \\ \delta \Phi \end{pmatrix}^T \begin{bmatrix} \mathbb{I}_{3 \times 3} \\ [\rho]_{\times} \end{bmatrix} \left(\begin{bmatrix} \mathbb{I}_{3 \times 3} & -[\rho]_{\times} \end{bmatrix} \begin{pmatrix} \mathbf{a}_s \\ \Psi \end{pmatrix} dm + [\Omega]_{\times}^2 \rho dm - d\mathbf{F}_{ext} \right) \\ &= \begin{pmatrix} \delta \mathbf{r}_s \\ \delta \Phi \end{pmatrix}^T \int_{\mathcal{B}} \left(\begin{bmatrix} \mathbb{I}_{3 \times 3} dm & [\rho]_{\times}^T dm \\ [\rho]_{\times} dm & -[\rho]_{\times}^2 dm \end{bmatrix} \begin{pmatrix} \mathbf{a}_s \\ \Psi \end{pmatrix} + \begin{pmatrix} [\Omega]_{\times}^2 \rho dm \\ [\rho]_{\times} [\Omega]_{\times}^2 \rho dm \end{pmatrix} - \begin{pmatrix} d\mathbf{F}_{ext} \\ [\rho]_{\times} d\mathbf{F}_{ext} \end{pmatrix} \right) \end{aligned} \quad (3.14)$$

Please note that this formulation must hold for arbitrary virtual displacements as there are no active constraints from joints or contacts. Knowing the computation rule $\mathbf{a} \times (\mathbf{b} \times (\mathbf{b} \times \mathbf{a})) = -\mathbf{b} \times (\mathbf{a} \times (\mathbf{a} \times \mathbf{b}))$ and introducing

$$\int_{\mathcal{B}} dm =: m \quad \text{body mass} \quad (3.15)$$

$$\int_{\mathcal{B}} \rho dm =: \mathbf{0} \quad \text{since } S = \text{COG} \quad (3.16)$$

$$\int_{\mathcal{B}} -[\rho]_{\times}^2 dm = \int_{\mathcal{B}} [\rho]_{\times} [\rho]_{\times}^T dm =: \Theta_S \quad \text{Inertia matrix around COG} \quad (3.17)$$

we get

$$0 = \delta W = \begin{pmatrix} \delta \mathbf{r}_s \\ \delta \Phi \end{pmatrix}^T \left(\begin{bmatrix} \mathbb{I}_{3 \times 3} m & \mathbf{0} \\ \mathbf{0} & \Theta_S \end{bmatrix} \begin{pmatrix} \mathbf{a}_s \\ \Psi \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ [\Omega]_{\times} \Theta_S \Omega \end{pmatrix} - \begin{pmatrix} \mathbf{F}_{ext} \\ \mathbf{T}_{ext} \end{pmatrix} \right) \quad \forall \begin{pmatrix} \delta \mathbf{r}_s \\ \delta \Phi \end{pmatrix}. \quad (3.18)$$

In order to define the laws of conservation of linear and angular momentum we introduce the definitions:

$$\mathbf{p}_S = m\mathbf{v}_S \quad \text{linear momentum} \quad (3.19)$$

$$\mathbf{N}_S = \mathbf{\Theta}_S \cdot \mathbf{\Omega} \quad \text{angular momentum around COG} \quad (3.20)$$

$$\dot{\mathbf{p}}_S = m\mathbf{a}_S \quad \text{change in linear momentum} \quad (3.21)$$

$$\dot{\mathbf{N}}_S = \mathbf{\Theta}_S \cdot \mathbf{\Psi} + \mathbf{\Omega} \times \mathbf{\Theta}_S \cdot \mathbf{\Omega} \quad \text{change in angular momentum} \quad (3.22)$$

A free moving body needs to fulfill that the change in linear momentum equals the sum of all external forces

$$0 = \delta W = \begin{pmatrix} \delta \mathbf{r}_s \\ \delta \mathbf{\Phi} \end{pmatrix}^T \left(\begin{pmatrix} \dot{\mathbf{p}}_S \\ \dot{\mathbf{N}}_S \end{pmatrix} - \begin{pmatrix} \mathbf{F}_{ext} \\ \mathbf{T}_{ext} \end{pmatrix} \right) \quad \forall \begin{pmatrix} \delta \mathbf{r}_s \\ \delta \mathbf{\Phi} \end{pmatrix}, \quad (3.23)$$

which results in the well-known formulations by Newton and Euler:

$$\dot{\mathbf{p}}_S = \mathbf{F}_{ext,S} \quad (3.24)$$

$$\dot{\mathbf{N}}_S = \mathbf{T}_{ext} \quad (3.25)$$

where $\mathbf{F}_{ext,S}$ are the resultant external forces that act through the COG and \mathbf{T}_{ext} are the resultant external torques. External forces which do not act through the COG need to be shifted to an equivalent force/moment pair of which the force acts through the COG. Please note again that for numerical calculation, the terms of the change in linear and angular momentum must be expressed in the same coordinate system. For the inertia tensor $\mathbf{\Theta}$ we must apply ${}_B\mathbf{\Theta} = \mathbf{C}_{BA} \cdot {}_A\mathbf{\Theta} \cdot \mathbf{C}_{BA}^T$.

3.3.2 Newton-Euler for Multi-Body Systems

When dealing with multi-body systems, a valid approach is to separate all bodies at the joints as depicted in Fig. 3.3 and to consider every body as a single unit. Thereby, the constraint forces \mathbf{F}_i at the joints must be introduced as external forces acting on each of the bodies when cut free. For all these bodies, we must then apply conservation of linear (3.24) and angular momentum (3.25) in all DoFs, subject to external forces (which now include the joint forces \mathbf{F}_i , too). For a general 3D case and a fixed base, this results in a $6n_j$ -dimensional systems of equations. Additionally, there are $5n_j$ motion constraint due to the ideal joints. They ensure that the two connected bodies only move along the direction of the joint but don't move in all other directions that are blocked by the joint.

Many simulation packages build upon such formalism and enforce the joint motion constraints using hard or soft constraints. The latter method often speeds up simulation due to the type of solvers that can be applied. However, it fulfills the ideal joint constraints only approximately, which can lead to physically inconsistent phenomena in simulation.

When solving such a system by hand, the approach is to describe the motion only as a function generalized coordinates. This still leads to $6n_j$ equations ($3n_j$ for planar systems) but ensures already the ideal joint constraints. The $6n_j$ equations linearly depend on n_j generalized coordinates as well as on the $5n_j$ joint constraint forces. Algebraic manipulation of the system of equations allows to eliminate all constraints forces.

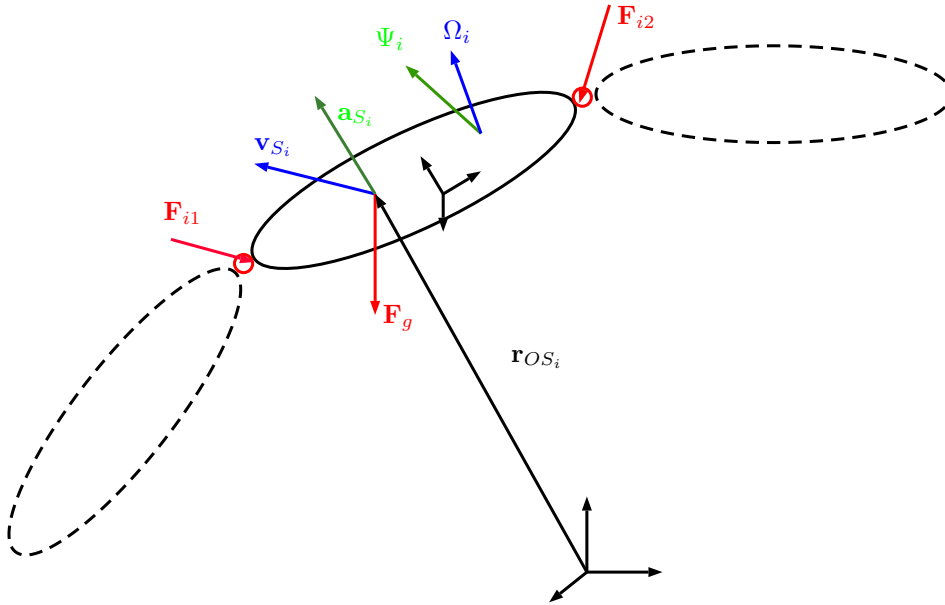


Figure 3.3: Free-cut of multi-body system with constraint forces due to joints

Example 3.3.1: Cart-Pendulum with Newton-Euler

3.4 Lagrange Method

3.4.1 Introduction

Another common approach for deriving the equations of motion of a system is that of using the so-called *Lagrange Method*. It originates from the sub-field of physics known as *Analytical Mechanics* [1][4], and is closely tied to both the d'Alembert and Hamilton principles, as it is one of the analytical methods used to describe the motion of physical systems. The method is centered around three fundamental concepts:

1. The definition of generalized coordinates \mathbf{q} and generalized velocities $\dot{\mathbf{q}}$, which may or may not encode the information regarding the constraints applicable to the system.
2. A scalar function called the *Lagrangian* function \mathcal{L} . For mechanical systems, it is exactly the difference between the total kinetic energy \mathcal{T} and the total potential energy \mathcal{U} , of the system at each instant:

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \quad (3.26)$$

3. The so-called *Euler-Lagrange* equation, also known as the *Euler-Lagrange of the second kind*, which applies to the Lagrangian function \mathcal{L} and to the total external

generalized forces τ :

$$\frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} \right)^T - \left(\frac{\partial \mathcal{L}}{\partial \mathbf{q}} \right)^T = \tau \quad (3.27)$$

In the most general case, the Lagrangian is a function of the generalized coordinates and velocities \mathbf{q} and $\dot{\mathbf{q}}$, and it may also have an explicit dependence on time t , hence we redefine the aforementioned scalar energy functions as $\mathcal{T} = \mathcal{T}(t, \mathbf{q}, \dot{\mathbf{q}})$ and $\mathcal{U} = \mathcal{U}(t, \mathbf{q})$, thus $\mathcal{L} = \mathcal{L}(t, \mathbf{q}, \dot{\mathbf{q}})$.

In the end, one of the most notable properties of this formulation is the capacity to eliminate all internal reaction forces of the system from the final EoM, in contrast to the Newton-Euler formulation where they are explicitly accounted for. To apply this method to derive EoM of a complex multi-body system, there are additional aspects which must be considered. These are presented in a concise overview at the end of this section, and are explained in the immediate continuation.

3.4.2 Kinetic Energy

The kinetic energy of a system of n_b bodies is defined as:

$$\mathcal{T} = \sum_{i=1}^{n_b} \left(\frac{1}{2} m_i \dot{\mathbf{r}}_{S_i}^T \dot{\mathbf{r}}_{S_i} + \frac{1}{2} \boldsymbol{\Omega}_{S_i}^T \cdot \boldsymbol{\Theta}_{S_i} \cdot \boldsymbol{\Omega}_{S_i} \right) \quad (3.28)$$

For every body \mathcal{B}_i in the system, although the linear part may be computed while expressed in some frame \mathcal{A} , it may be more convenient to compute the rotational kinetic energy using expressions in another frame \mathcal{B} , rotated w.r.t to \mathcal{A} , where the inertia matrix $\boldsymbol{\Theta}_{S,i}$ may have a diagonal form, i.e. the basis vectors of \mathcal{B} are principle w.r.t. the mass distribution. This computation will yield correct results as long as both linear and angular velocities $\dot{\mathbf{r}}_{S,i}$ and $\boldsymbol{\Omega}_{S,i}$ express the *absolute* velocities of the body, i.e. velocities w.r.t. to the inertial frame.

We now need to express the kinetic energy as a function of the generalized quantities. To achieve this, we make use of the Jacobian matrices described by (2.163) and (2.164), but computed for each body \mathcal{B}_i instead of the end-effector. This then allows us to use the following kinematic relationships:

$$\dot{\mathbf{r}}_{S_i} = \mathbf{J}_{S_i} \dot{\mathbf{q}} \quad (3.29)$$

$$\boldsymbol{\Omega}_{S_i} = \mathbf{J}_{R_i} \dot{\mathbf{q}} \quad (3.30)$$

Replacing these relationships into the definition of the kinetic energy in (3.28), results in the kinetic energy expressed in the generalized coordinates:

$$\mathcal{T}(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2} \dot{\mathbf{q}}^T \underbrace{\left(\sum_{i=1}^{n_b} (\mathbf{J}_{S_i}^T m \mathbf{J}_{S_i} + \mathbf{J}_{R_i}^T \boldsymbol{\Theta}_{S_i} \mathbf{J}_{R_i}) \right)}_{\mathbf{M}(\mathbf{q})} \dot{\mathbf{q}} \quad (3.31)$$

The underlined quantity $\mathbf{M}(\mathbf{q})$ is defined as the *generalized mass matrix* or *generalized inertia matrix*, and as we will see in the continuation, is solely responsible for generating both the inertial and non-linear centrifugal and Coriolis force terms in the final EoM.

3.4.3 Potential Energy

In typical mechanics problems, there are two basic contributions to the potential energy in a system; masses contributing via their gravitational potential energy, and elastic elements via the energy stored when deflected from rest.

In the first case, each body \mathcal{B}_i holds potential energy due to the effect of the gravitational potential field of the earth. Although not linear on large scales, in most cases we approximate its effect on bodies via a uniform and unidirectional potential field defined along the unit vector \mathbf{e}_g acting through the CoM of each body. Knowing the position \mathbf{r}_{S_i} of the CoM of each body This then allows us to compute the potential energy of each body as:

$$\mathbf{F}_{g_i} = m_i g \mathbf{e}_g \quad (3.32)$$

$$\mathcal{U}_g = - \sum_{i=1}^{n_b} \mathbf{r}_{S_i}^T \mathbf{F}_{g_i} \quad (3.33)$$

Note that the zero energy level can be arbitrarily chosen. In addition to gravitational potential energy contributions, many applications involve elastic elements such as springs or other compliant components. If such a element E_j can be reasonably approximated to have a linear deflection-to-force or deflection-to-torque relationship, then the potential energy contribution can be described by the following relation:

$$\mathcal{U}_{E_j} = \frac{1}{2} k_j (d(\mathbf{q}) - d_0)^2 \quad (3.34)$$

Where $d(\mathbf{q})$ expresses the instantaneous configuration of the elastic element (e.g. the length of a spring) as a function of generalized coordinates. d_0 is defined as the *rest* configuration where no forces are exerted by the element, e.g., the unsprung length of a linear spring. We call the difference $(d(\mathbf{q}) - d_0)$ the *deflection* of the elastic element.

Note: Spring forces may equivalently be considered as external forces whose magnitude and possibly direction depends on the generalized coordinates \mathbf{q} . The formalism is similar to introducing actuator torques.

3.4.4 External Forces

All external forces that do work on the system are accounted by the generalized force vector $\boldsymbol{\tau}$. Please refer to section 3.5.3 for a derivation.

3.4.5 Additional Constraints

The final case we have to account for is that of when additional constraints are imposed on the system. When we define generalized coordinates, in most cases, we aim to describe the system using a *minimal* set of coordinates so that we can express the the DoFs which we can control or at least measure.

Consider the case presented in section 2.10.4, when contact constraints are imposed on a floating-based system. These constraints may not be active at all times, e.g., as in the case of a walking robot or a ball bouncing on a table. Such a contact constraint cannot, in general, be explicitly accounted for when we defining the vector \mathbf{q} since it may not always apply on the system. The most common way to overcome this to use *Lagrange Multipliers*, usually denoted by a vector $\boldsymbol{\lambda}$. Constraints which are explicitly defined

using velocities, are known as *motion constraints* and are usually expressed as linear combinations of the general velocities:

$$\sum_{k=1}^{n_q} a_{k,j}(\mathbf{q}) \dot{q}_k + a_{0,j}(t), \quad j = 1, 2, \dots, n_{c,m} \quad (3.35)$$

For $n_{c,m}$ motion constraints, we can stack all $a_{k,j}(\mathbf{q})$ coefficients to a motion constraint Jacobian matrix \mathbf{J}_m :

$$\mathbf{J}_m = \begin{bmatrix} a_{1,1} & \dots & a_{1,n_{c,m}} \\ \vdots & \dots & \vdots \\ a_{n_q,1} & \dots & a_{n_q,n_{c,m}} \end{bmatrix} \quad (3.36)$$

On the other hand, there are situations where constraints will be defined as functions of the configuration, i.e. as scalar functions $f_j(\mathbf{q}) : \mathbb{R}^{n_q} \rightarrow \mathbb{R}$, and are known as *configuration constraints*. In these cases, computing the gradient of such a function w.r.t the generalized coordinates results in the constraint Jacobian matrix described in section 2.10.4.

In total, incorporating these constraints results in a description of the *constrained* EoM, since the additional constraints being applied for by the parameters λ are not account for in the definition of \mathbf{q} . Thus the final EoM, are derived using the so-called *Constrained Euler-Lagrange* (CEL) equation, also known as the *Euler-Lagrange of the first kind*:

$$\frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} \right) - \left(\frac{\partial \mathcal{L}}{\partial \mathbf{q}} \right) + \mathbf{J}_m^T \boldsymbol{\lambda}_m + \left(\frac{\partial \mathbf{f}_c}{\partial \mathbf{q}} \right)^T \boldsymbol{\lambda}_c = \boldsymbol{\tau} \quad (3.37)$$

Where $\boldsymbol{\lambda}_m \in \mathbb{R}^{n_{c,m}}$ is the vector of Lagrangian multipliers for the motion constraints, while $\boldsymbol{\lambda}_c \in \mathbb{R}^{n_{c,c}}$ is the vector of Lagrangian multipliers for the configuration constraints.

In conclusion, we note that it should be clear why this method is also referred to as being energy-based, as the equations of motion are derived from a scalar energy function, and thus, we did not make explicit use of any Cartesian vector quantities. All the kinematic information is encoded into the definition of the generalized coordinates, which are then used to define the *scalar* Lagrangian, and then applied to (3.27) in order to produce the EoM. This is in stark contrast to Newton-Euler methods which explicitly deal with Cartesian vector quantities.

3.5 Projected Newton-Euler Method

3.5.1 Introduction

The final method we will describe for deriving EoM of multi-body systems is that which makes use of the *Projected Newton-Euler* (PNE) formulation. Essentially, this method combines the classical Newton-Euler equation for dynamic equilibrium in Cartesian coordinates, with the constraint compliant Lagrange formulation using generalized coordinates. In fact, as we will see in the continuation, one can derive the PNE equations from both of the other two formulations.

Let us briefly recapitulate what the resulting EoM using PNE will look like:

$$\mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{J}_c^T \mathbf{F}_c \quad (3.38)$$

3.5.2 Deriving Generalized Equations of Motion

As outlined in section 3.2.4, the use of generalized coordinates allows to describe the motion and virtual displacements of bodies in ways which are consistent with the applicable constraints. Thus, we can rewrite principles of linear and angular momentum applied to each body \mathcal{B}_i as:

$$\begin{aligned} \begin{pmatrix} \dot{\mathbf{p}}_{S_i} \\ \dot{\mathbf{N}}_{S_i} \end{pmatrix} &= \begin{pmatrix} m\mathbf{a}_{S_i} \\ \boldsymbol{\Theta}_{S_i}\boldsymbol{\Psi}_{S_i} + \boldsymbol{\Omega}_{S_i} \times \boldsymbol{\Theta}_{S_i}\boldsymbol{\Omega}_{S_i} \end{pmatrix} \\ &= \begin{pmatrix} m\mathbf{J}_{S_i} \\ \boldsymbol{\Theta}_{S_i}\mathbf{J}_{R_i} \end{pmatrix} \ddot{\mathbf{q}} + \begin{pmatrix} m\dot{\mathbf{J}}_{S_i}\dot{\mathbf{q}} \\ \boldsymbol{\Theta}_{S_i}\dot{\mathbf{J}}_{R_i}\dot{\mathbf{q}} + \mathbf{J}_{R_i}\dot{\mathbf{q}} \times \boldsymbol{\Theta}_{S_i}\mathbf{J}_{R_i}\dot{\mathbf{q}} \end{pmatrix} \end{aligned} \quad (3.39)$$

Furthermore, using the expressions for virtual displacements of generalized coordinates, we can rewrite the principle of virtual work as follows:

$$0 = \delta W = \sum_{i=1}^{n_b} \begin{pmatrix} \delta \mathbf{r}_{S_i} \\ \delta \boldsymbol{\Phi}_{S_i} \end{pmatrix}^T \left(\begin{pmatrix} \dot{\mathbf{p}}_{S_i} \\ \dot{\mathbf{N}}_{S_i} \end{pmatrix} - \begin{pmatrix} \mathbf{F}_{ext,i} \\ \mathbf{T}_{ext,i} \end{pmatrix} \right) \quad \forall \begin{pmatrix} \delta \mathbf{r}_s \\ \delta \boldsymbol{\Phi}_{S_i} \end{pmatrix}_{\text{consistent}}, \quad (3.40)$$

$$= \delta \mathbf{q}^T \sum_{i=1}^{n_b} \begin{pmatrix} \mathbf{J}_{S_i} \\ \mathbf{J}_{R_i} \end{pmatrix}^T \left(\begin{pmatrix} \dot{\mathbf{p}}_{S_i} \\ \dot{\mathbf{N}}_{S_i} \end{pmatrix} - \begin{pmatrix} \mathbf{F}_{ext,i} \\ \mathbf{T}_{ext,i} \end{pmatrix} \right) \quad \forall \delta \mathbf{q} \quad (3.41)$$

Combining (3.39) and (3.41) directly yields:

$$0 = \sum_{i=1}^{n_b} \begin{pmatrix} \mathbf{J}_{S_i} \\ \mathbf{J}_{R_i} \end{pmatrix}^T \begin{pmatrix} m\mathbf{J}_{S_i} \\ \boldsymbol{\Theta}_{S_i}\mathbf{J}_{R_i} \end{pmatrix} \ddot{\mathbf{q}} + \begin{pmatrix} \mathbf{J}_{S_i} \\ \mathbf{J}_{R_i} \end{pmatrix}^T \begin{pmatrix} m\dot{\mathbf{J}}_{S_i}\dot{\mathbf{q}} \\ \boldsymbol{\Theta}_{S_i}\dot{\mathbf{J}}_{R_i}\dot{\mathbf{q}} + \mathbf{J}_{R_i}\dot{\mathbf{q}} \times \boldsymbol{\Theta}_{S_i}\mathbf{J}_{R_i}\dot{\mathbf{q}} \end{pmatrix} - \begin{pmatrix} \mathbf{J}_{P_i} \\ \mathbf{J}_{R_i} \end{pmatrix}^T \begin{pmatrix} \mathbf{F}_{ext,i} \\ \mathbf{T}_{ext,i} \end{pmatrix} \quad (3.42)$$

A final re-grouping of terms results in the components of (3.38):

$$\mathbf{M} = \sum_{i=1}^{n_b} (\mathbf{J}_{S_i}^T m \mathbf{J}_{S_i} + \mathbf{J}_{R_i}^T \boldsymbol{\Theta}_{S_i} \mathbf{J}_{R_i}) \quad (3.43)$$

$$\mathbf{b} = \sum_{i=1}^{n_b} \left(\mathbf{J}_{S_i}^T m \dot{\mathbf{J}}_{S_i} \dot{\mathbf{q}} + \mathbf{J}_{R_i}^T \left(\boldsymbol{\Theta}_{S_i} \dot{\mathbf{J}}_{R_i} \dot{\mathbf{q}} + \boldsymbol{\Omega}_{S_i} \times \boldsymbol{\Theta}_{S_i} \boldsymbol{\Omega}_{S_i} \right) \right)_i \quad (3.44)$$

$$\mathbf{g} = \sum_{i=1}^{n_b} -\mathbf{J}_{S_i}^T \mathbf{F}_{g,i} \quad (3.45)$$

Again, we must stress the importance of selecting an appropriate choice of coordinate frame to express all Cartesian vectors. We apply the same reasoning as we did in computing the kinetic energy in section 3.4.2, where, we stated that both angular and linear velocities must be absolute, i.e., measured w.r.t an. inertial frame. It does not matter then if linear and angular are expressed differently, however, an careful selection can simplify the final expressions:

$$\mathbf{M} = \sum_{i=1}^{n_b} (\mathcal{A} \mathbf{J}_{S_i}^T \cdot m \cdot \mathcal{A} \mathbf{J}_{S_i} + \mathcal{B} \mathbf{J}_{R_i}^T \cdot \mathcal{B} \boldsymbol{\Theta}_{S_i} \cdot \mathcal{B} \mathbf{J}_{R_i}) \quad (3.46)$$

$$\mathbf{b} = \sum_{i=1}^{n_b} \left(\mathcal{A} \mathbf{J}_{S_i}^T \cdot m \cdot \mathcal{A} \dot{\mathbf{J}}_{S_i} \cdot \dot{\mathbf{q}} + \mathcal{B} \mathbf{J}_{R_i}^T \cdot \left(\mathcal{B} \boldsymbol{\Theta}_{S_i} \cdot \mathcal{B} \dot{\mathbf{J}}_{R_i} \cdot \dot{\mathbf{q}} + \mathcal{B} \boldsymbol{\Omega}_{S_i} \times \mathcal{B} \boldsymbol{\Theta}_{S_i} \cdot \mathcal{B} \boldsymbol{\Omega}_{S_i} \right) \right) \quad (3.47)$$

$$\mathbf{g} = \sum_{i=1}^{n_b} (-\mathcal{A} \mathbf{J}_{S_i}^T \mathcal{A} \mathbf{F}_{g,i}) \quad (3.48)$$

3.5.3 External Forces & Actuation

As we have seen in section 3.4.4, in order to account for external forces in EoM expressed in generalized coordinates, we can project Cartesian forces and torques onto the space of generalized coordinates using appropriate Jacobian matrices. The latter project the effect of the forces onto the subspace of the generalized coordinates on which work is done. We will now consider how to effectively account for all acting forces and torques on a multi-body system.

Lets assume we know that $n_{f,ext}$ external forces \mathbf{F}_j , $j \in \{1, \dots, n_{f,ext}\}$ and $n_{m,ext}$ external torques \mathbf{T}_k , $k \in \{1, \dots, n_{m,ext}\}$ act on the system (on any body). The generalized forces $\tau_{F,ext}$ due to the external forces can be calculated in the following way: Assume the Cartesian force \mathbf{F}_j acts on point P_j and the translational Jacobian of that point is $\mathbf{J}_{P,j}$. Then the generalized forces are computed as

$$\tau_{F,ext} = \sum_{j=1}^{n_{f,ext}} \mathbf{J}_{P,j}^T \mathbf{F}_j . \quad (3.49)$$

Similarly, the generalized forces $\tau_{T,ext}$ can be evaluated by projecting each Cartesian torque by the rotational Jacobian of the body they act on, which we shall call $\mathbf{J}_{B,k,rot}$:

$$\tau_{T,ext} = \sum_{k=1}^{n_{m,ext}} \mathbf{J}_{R,k}^T \mathbf{T}_{ext,k} . \quad (3.50)$$

Finally, the contributions of external forces and torques can simply be added as they are both represented in the space of generalized coordinates:

$$\tau_{ext} = \tau_{F,ext} + \tau_{T,ext} \quad (3.51)$$

For the special case of actuator forces or torques that act between two body links, we need only consider the kinematic relations defined between those two (successive) body links. Thus, an actuator acting between body links \mathcal{B}_{k-1} and \mathcal{B}_k , imposes a force $\mathbf{F}_{a,k}$ and/or torque $\mathbf{T}_{a,k}$ on both equally and in opposite directions, i.e., *action-reaction*. What thus need to compute are the Jacobian matrices to appropriately project onto the dimensions of \mathbf{q} .

To this end, we recall that Jacobian matrices, when expressed in the same reference frame, can be simply added or subtracted, and thus having computed $\mathbf{J}_{S_{k-1}}$, \mathbf{J}_{S_k} , $\mathbf{J}_{R_{k-1}}$, and \mathbf{J}_{R_k} from our previous analysis of the kinematics, we can compute the contribution of the actuators to the generalized forces as:

$$\tau_{a,k} = (\mathbf{J}_{S_k} - \mathbf{J}_{S_{k-1}})^T \mathbf{F}_{a,k} + (\mathbf{J}_{R_k} - \mathbf{J}_{R_{k-1}})^T \mathbf{T}_{a,k} \quad (3.52)$$

In the most simple, and in fact also the most common, case of an actuator acting in the direction of a generalized coordinate q_j , the difference between the Jacobian at the point of action and reaction multiplied with the force or torque provides the single entry in the generalized force vector τ_j . Thus, the vector of total external generalized forces is simply a combination of the aforementioned parts, accounting for n_A joint actuators and n_B body links:

$$\tau = \sum_{k=1}^{n_A} \tau_{a,k} + \tau_{ext} \quad (3.53)$$

Comments:

- The translational Jacobian of an arbitrary point on a body can be obtained by starting with the translational Jacobian of the CoM of that body and applying (2.154).
- Alternatively, one may transform all forces that do not act through the CoM of a body into an equivalent force/moment pair which allows usage of the CoM translational and rotational Jacobians.
- Actuator forces/torques are no different to any other external forces/torques except that both the torque/force and its reaction act on the system.

3.6 Summary and Relation between Methods

When looking at three methods, we can identify

- All need definition of generalized coordinates
- All of them need Jacobians of CoGs
- Evaluate one or the other equation

To check correctness of the implementation, it can help to observe the time evolution of the total energy (Hamiltonian \mathcal{H})

$$\mathcal{H} = \mathcal{T} + \mathcal{U} . \quad (3.54)$$

In absence of friction, actuation, and external forces the total energy in the system should remain constant.

3.7 Dynamics of Floating Base Systems

$$\mathbf{M}(\mathbf{q}) \dot{\mathbf{u}} + \mathbf{b}(\mathbf{q}, \mathbf{u}) + \mathbf{g}(\mathbf{q}) = \mathbf{S}^T \boldsymbol{\tau} + \mathbf{J}_{ext}^T \mathbf{F}_{ext} \quad (3.55)$$

consisting of the following components:

$\mathbf{M}(\mathbf{q})$	$\in \mathbb{R}^{n_q \times n_q}$	Mass matrix (orthogonal)
\mathbf{q}	$\in \mathbb{R}^{n_q}$	generalized coordinates
\mathbf{u}	$\in \mathbb{R}^{n_q}$	generalized velocity
$\dot{\mathbf{u}}$	$\in \mathbb{R}^{n_q}$	generalized acceleration
$\mathbf{b}(\mathbf{q}, \mathbf{u})$	$\in \mathbb{R}^{n_q}$	Coriolis and centrifugal terms
$\mathbf{g}(\mathbf{q})$	$\in \mathbb{R}^{n_q}$	gravitational terms
\mathbf{S}	$\in \mathbb{R}^{n_\tau \times n_q}$	selection matrix of actuated joints
$\boldsymbol{\tau}$	$\in \mathbb{R}^{n_\tau}$	generalized torques acting in direction of generalized coordinates
\mathbf{F}_{ext}	$\in \mathbb{R}^{n_c}$	external forces acting
\mathbf{J}_{ext}	$\in \mathbb{R}^{n_c \times n_q}$	(geometric) Jacobian of location where external forces apply

As we have seen in section 2.10, the generalized coordinates of a floating base systems consist of actuated joint coordinates \mathbf{q}_j and unactuated base coordinates \mathbf{q}_b respectively the corresponding velocities $\mathbf{u}_j = \dot{\mathbf{q}}_j \in \mathbb{R}^{n_j}$ and $\mathbf{u}_b \in \mathbb{R}^6$. Please note again at this place that \mathbf{u}_b is not equal to the time derivative of the position and orientation

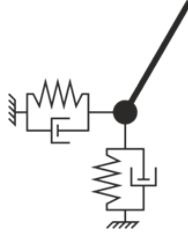


Figure 3.4: In soft contact model, the contact force is modeled as a spring-damper force

parameterization $\dot{\mathbf{q}}$, since no angular position exists but only different ways of parameterizing the orientation. The selection matrix \mathbf{S} selects the actuated joints according to

$$\mathbf{u}_j = \mathbf{S}\mathbf{u} = \mathbf{S} \begin{pmatrix} \mathbf{u}_b \\ \mathbf{u}_j \end{pmatrix} = \begin{bmatrix} \mathbf{0}_{n_j \times 6} & \mathbb{I}_{n_j \times n_j} \end{bmatrix} \begin{pmatrix} \mathbf{u}_b \\ \mathbf{u}_j \end{pmatrix} \quad (3.56)$$

In order to control also the unactuated base coordinates \mathbf{q}_b , *external forces* \mathbf{F}_{ext} are necessary. Depending on the type of robot, they can come from very different sources. For example, this force can originate from contacts (interaction) with the environment (e.g. legged robots) or from aerodynamics (e.g. flying robots). In many text books and in particular when working with contact forces (e.g. legged robots), people use the alternative notation

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{u}} + \mathbf{b}(\mathbf{q}, \mathbf{u}) + \mathbf{g}(\mathbf{q}) + \mathbf{J}_c^T \mathbf{F}_c = \mathbf{S}^T \boldsymbol{\tau}, \quad (3.57)$$

with \mathbf{F}_c representing a force that the robot exerts on its environment.

3.7.1 Contact Forces

There exist two fundamentally different methods to model contact forces. The *soft contact* method models the interaction by force elements (i.e. spring-damper) where the force is only a function of the location and velocity of the point in contact. The *hard contact* method treats the contact as a kinematic constraint.

Soft Contact Model

For the soft contact model, we typically identify the point when first making contact with the environment as \mathbf{r}_{c0} . When using a linear spring-damper model for the environment, the contact force (exerted by the robot on its environment) is

$$\mathbf{F}_c = k_p (\mathbf{r}_c - \mathbf{r}_{c0}) + k_d \dot{\mathbf{r}}_c. \quad (3.58)$$

While modeling the environment like this seems logic and physically correct from a first point of view, it turns out that finding physically correct spring and damping parameters to simulate the system dynamics is almost impossible. One of the big problems is that the combined differential equations for multi-body system and contacts become very stiff (slow multi-body dynamics and very fast contact dynamics). Solving such problems either result in poor speed or low accuracy. To overcome this problem, the contact parameters for stiffness and damping are often tuned as numerical simulation parameters that have nothing to do with the actual physical parameters.

Contact Forces from Constraints

Instead of contact forces resulting from force elements at the contact, contacts can also be handled as *kinematics constraints*. If a point C with position \mathbf{r}_c is in contact, it is not allowed to move anymore:

$$\mathbf{r}_c = \text{const} \quad (3.59)$$

$$\dot{\mathbf{r}}_c = \mathbf{J}_c \mathbf{u} = \mathbf{0} \quad (3.60)$$

$$\ddot{\mathbf{r}}_c = \mathbf{J}_c \dot{\mathbf{u}} + \dot{\mathbf{J}}_c \mathbf{u} = \mathbf{0} \quad (3.61)$$

This formulation can be extended to include rotational constraints when two bodies exhibit multiple simultaneous contacts between them, such as in the case of contacting surfaces. From the constraint (3.61) and the equation of motion (3.57) we can compute the contact force as

$$\mathbf{F}_c = (\mathbf{J}_c \mathbf{M}^{-1} \mathbf{J}_c^T)^{-1} (\mathbf{J}_c \mathbf{M}^{-1} (\mathbf{S}^T \boldsymbol{\tau} - \mathbf{b} - \mathbf{g}) + \dot{\mathbf{J}}_c \mathbf{u}) \quad (3.62)$$

The previous equation can prove quite useful in practical applications, as it allows us to estimate the reaction forces produced on a multi-body system, such as a robotic arm or leg, when in contact with the environment. Thus, solely utilizing the description of the multi-body system dynamics, we can measure contact forces without needing external sensors to directly measure forces and moments due to contacts.

3.7.2 Constraint Consistent Dynamics

We can define the dynamically consistent support null-space matrix as

$$\mathbf{N}_c = \mathbb{I} - \mathbf{M}^{-1} \mathbf{J}_c^T (\mathbf{J}_c \mathbf{M}^{-1} \mathbf{J}_c^T)^{-1} \mathbf{J}_c. \quad (3.63)$$

\mathbf{N}_c defines a generalized space of motion with no acceleration or force coupling effects on the supporting links. Substituting the solution for the contact force (3.62) into the equation of motion (3.57) results in

$$\mathbf{M} \dot{\mathbf{u}} + \mathbf{N}_c^T (\mathbf{b} + \mathbf{g}) + \mathbf{J}_c^T (\mathbf{J}_c \mathbf{M}^{-1} \mathbf{J}_c^T)^{-1} \dot{\mathbf{J}}_c \mathbf{u} = \mathbf{N}_c^T \mathbf{S}^T \boldsymbol{\tau}. \quad (3.64)$$

By further including the support constraint (3.61) which implies $\dot{\mathbf{J}}_c \mathbf{u} = -\mathbf{J}_c \dot{\mathbf{u}}$, the constraint consistent equations of motion can be compactly formulated as

$$\mathbf{N}_c^T (\mathbf{M} \dot{\mathbf{u}} + \mathbf{b} + \mathbf{g}) = \mathbf{N}_c^T \mathbf{S}^T \boldsymbol{\tau}. \quad (3.65)$$

3.7.3 Contact Switches and Impact Collisions

A hard contact model requires subdividing the analysis of the system dynamics into two intervals, before and after a change in the contact situation respectively an impact. The impact itself is a complex physical phenomenon which occurs when two or more bodies collide with each other. The characteristic of an impact is a very short duration with high peak forces that results in a rapid dissipation of energy and large accelerations. To model the process of energy transfer and dissipation, various coefficients are employed, such as the coefficient of restitution and the impulse ratio. Idealizing the process, respectively considering the impact as an infinitesimally short process requires to include instantaneous changes in velocities if bodies are making contact.

Impulse Transfer

To resolve the contact impulse, we use the integrated equation of motion over a single point in time t_0

$$\int_{\{t_0\}} (\mathbf{M}\dot{\mathbf{u}} + \mathbf{b} + \mathbf{g} + \mathbf{J}_c^T \mathbf{F}_c - \mathbf{S}^T \boldsymbol{\tau}) dt = \mathbf{M} (\mathbf{u}^+ - \mathbf{u}^-) + \mathbf{J}_c^T \mathcal{F}_c = \mathbf{0}, \quad (3.66)$$

with the impulsive force \mathcal{F}_c and the pre- respectively post-impact generalized velocities \mathbf{u}^- and \mathbf{u}^+ . Assuming a perfect inelastic collision with a Newtonian collision law, all contact points that are considered part of the collision instantaneously come to rest ($\dot{\mathbf{r}}_c^+ = \mathbf{J}_c \mathbf{u}^+ = \mathbf{0}$). Combining this post-impact constraint with the integrated equation of motion, we can solve for the impulsive force as

$$\mathcal{F}_c = (\mathbf{J}_c \mathbf{M}^{-1} \mathbf{J}_c^T)^{-1} \mathbf{J}_c \dot{\mathbf{q}}^- = \boldsymbol{\Lambda}_c \dot{\mathbf{r}}_c^-. \quad (3.67)$$

Analyzing this formalism a bit more in detail by considering the basic mechanics that defines *impulse = mass · speed*, we identify the inertia that is seen at the support point as the so called end-effector inertia:

$$\boldsymbol{\Lambda}_c = (\mathbf{J}_c \mathbf{M}^{-1} \mathbf{J}_c^T)^{-1} \quad (3.68)$$

Substituting (3.68) into (3.67) yields the instantaneous change in generalized velocities:

$$\Delta \mathbf{u} = \mathbf{u}^+ - \mathbf{u}^- = -\mathbf{M}^{-1} \mathbf{J}_c^T (\mathbf{J}_c \mathbf{M}^{-1} \mathbf{J}_c^T)^{-1} \mathbf{J}_c \mathbf{u}^-. \quad (3.69)$$

Using again the nomenclature introduced previously for the dynamically consistent support null-space projector \mathbf{N}_c , the post-impact generalized velocities are determined by

$$\mathbf{u}^+ = \left(\mathbb{I} - \mathbf{M}^{-1} \mathbf{J}_c^T (\mathbf{J}_c \mathbf{M}^{-1} \mathbf{J}_c^T)^{-1} \mathbf{J}_c \right) \mathbf{u}^- = \mathbf{N}_c \mathbf{u}^-. \quad (3.70)$$

The result that is obtained by satisfying the post impact contact constraint is intuitively clear: Using the support null-space projector \mathbf{N}_c , the pre-impact velocity \mathbf{u}^- is projected onto the support consistent manifold.

Energy Loss

The instantaneous change in the contact situation is always associated with a kinetic energetic loss. This can be quantified in generalized coordinates or as a function of the end-effector inertia and the change in velocity at the support point by

$$E_{loss} = \Delta E_{kin} = -\frac{1}{2} \Delta \mathbf{u}^T \mathbf{M} \Delta \mathbf{u} \quad (3.71)$$

$$= -\frac{1}{2} \Delta \dot{\mathbf{r}}_c^T \boldsymbol{\Lambda}_c \Delta \dot{\mathbf{r}}_c = -\frac{1}{2} \dot{\mathbf{r}}_c^{-T} \boldsymbol{\Lambda}_c \dot{\mathbf{r}}_c^-. \quad (3.72)$$

3.8 Joint-space Dynamic Control

Current industrial robots rely almost exclusively on the concept of joint position control. They build upon PID controllers to independently regulate the position or velocity of every joint of the robot. Such a controller compensates for disturbances in the actuators and the entire robot and leads in the ideal case to perfect tracking of a desired motion. Only by additionally sensing of the joint torque (e.g., by measuring it with a load cell or by estimating it from motor current and actuator models) it becomes possible to integrate model-based load compensation.

3.8.1 Joint Impedance Regulation

In case of torque controllable actuators, the joint feedback gains for joint position \mathbf{k}_p and velocity \mathbf{k}_d correspond to joint stiffness and damping and the desired actuator torque can be calculated as

$$\boldsymbol{\tau}^* = \mathbf{k}_p (\mathbf{q}^* - \mathbf{q}) + \mathbf{k}_d (\dot{\mathbf{q}}^* - \dot{\mathbf{q}}), \quad (3.73)$$

with \mathbf{q}^* and $\dot{\mathbf{q}}^*$ representing the desired joint position and velocity, respectively. When applying this control law to the robot arm, we get a steady-state tracking error:

$$\cancel{\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}^*} + \cancel{\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}})} + \mathbf{g}(\mathbf{q}) = \mathbf{k}_p (\mathbf{q}^* - \mathbf{q}) + \mathbf{k}_d (\dot{\mathbf{q}}^* - \dot{\mathbf{q}}) \quad (3.74)$$

Gravity Compensation

In order to compensate for steady-state offset and to adjust the joint impedance, a common approach is to select the desired actuator torque as

$$\boldsymbol{\tau}^* = \mathbf{k}_p (\mathbf{q}^* - \mathbf{q}) + \mathbf{k}_d (\dot{\mathbf{q}}^* - \dot{\mathbf{q}}) + \hat{\mathbf{g}}(\mathbf{q}), \quad (3.75)$$

with $\hat{\mathbf{g}}(\mathbf{q})$ representing the estimated gravity effects. Unfortunately, since the inertia seen at each joint varies with the robot configuration, the PD gains must be selected for some average configuration in the workspace. This reduces the performance when dynamic effects become significant. There is still a steady-state error, because the model is never perfectly accurate.

Inverse Dynamics Control

A simple way to get around these short-comings is to implement an inverse dynamics control method. Thereby, dynamic decoupling and motion control is achieved by selecting the joint torque as

$$\boldsymbol{\tau}^* = \hat{\mathbf{M}}(\mathbf{q}) \ddot{\mathbf{q}}^* + \hat{\mathbf{b}}(\mathbf{q}, \dot{\mathbf{q}}) + \hat{\mathbf{g}}(\mathbf{q}), \quad (3.76)$$

where $\hat{\mathbf{M}}(\mathbf{q})$, $\hat{\mathbf{b}}(\mathbf{q}, \dot{\mathbf{q}})$, and $\hat{\mathbf{g}}(\mathbf{q})$ represent the estimates of $\mathbf{M}(\mathbf{q})$, $\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}})$, and $\mathbf{g}(\mathbf{q})$. In case of a perfectly modeled plant, the closed-loop dynamics of the system (3.1) with control (3.76) results in

$$\mathbb{I} \ddot{\mathbf{q}} = \ddot{\mathbf{q}}^*. \quad (3.77)$$

In other words, this method allows to directly shape the decoupled dynamics of every joint. Similar to the impedance law introduced in (3.73), a common approach is to select the desired acceleration according to

$$\ddot{\mathbf{q}}^* = \mathbf{k}_p (\mathbf{q}^* - \mathbf{q}) + \mathbf{k}_d (\dot{\mathbf{q}}^* - \dot{\mathbf{q}}), \quad (3.78)$$

which corresponds to a linear mass-spring-damper system with unitary mass. As a great benefit, the tuning and selection of feedback gains \mathbf{k}_p and \mathbf{k}_d becomes intuitively clear since they represent physical parameters of a decoupled point mass oscillator. Hence, the eigenfrequency and a dimensionless damping value of the system are given by

$$\omega = \sqrt{k_p}, \quad (3.79)$$

$$D = \frac{k_d}{2\sqrt{k_p}}. \quad (3.80)$$

Critical damping is achieved for $D = 1$, overcritical damping for $D > 1$ and undercritical damping for $D < 1$. The compliance of the controller can be adjusted by varying k_p . For example, assuming that the time constant respectively oscillation frequency around the nominal point should be 3 Hz, the ideal control gain k_p is 350. Furthermore, critical damping requires $k_d = 37$. This holds as good starting values for controller gain tuning.

3.9 Task-space Dynamics Control

So far we have only considered model-based joint space control. However, in most situation, we want to move a specific point in task-space, i.e., in world-fixed frame. The linear and rotational acceleration of the end-effector e (or any other point and link) is coupled to the generalized accelerations through the geometric Jacobians:

$$\dot{\mathbf{w}}_e = \begin{pmatrix} \ddot{\mathbf{r}} \\ \dot{\dot{\boldsymbol{\omega}}} \end{pmatrix}_e = \mathbf{J}_e \ddot{\mathbf{q}} + \dot{\mathbf{J}}_e \dot{\mathbf{q}}. \quad (3.81)$$

3.9.1 Multi-task Decomposition

Similar to the kinematic multi-tasks control outlined in section 2.9.2, we can perform inverse dynamics while fulfilling multiple tasks. Given a set of motion objectives by the desired task space acceleration and the corresponding Jacobian, we can treat all goals with the same priority:

$$\ddot{\mathbf{q}} = \begin{bmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_{n_t} \end{bmatrix}^+ \left(\begin{pmatrix} \dot{\mathbf{w}}_1 \\ \vdots \\ \dot{\mathbf{w}}_{n_t} \end{pmatrix} - \begin{bmatrix} \dot{\mathbf{J}}_1 \\ \vdots \\ \dot{\mathbf{J}}_{n_t} \end{bmatrix} \dot{\mathbf{q}} \right) \quad (3.82)$$

In case some tasks have higher priority, we can determine the solution using a similar recursive algorithm as outlined in (2.220):

$$\ddot{\mathbf{q}} = \sum_{i=1}^{n_t} \mathbf{N}_i \ddot{\mathbf{q}}_i, \quad \text{with} \quad \ddot{\mathbf{q}}_i = (\mathbf{J}_i \mathbf{N}_i)^+ \left(\dot{\mathbf{w}}_i^* - \dot{\mathbf{J}}_i \dot{\mathbf{q}} - \mathbf{J}_i \sum_{k=1}^{i-1} \mathbf{N}_k \ddot{\mathbf{q}}_k \right), \quad (3.83)$$

whereby \mathbf{N}_i is the null-space projection of the stacked Jacobian $\bar{\mathbf{J}}_i = [\mathbf{J}_1^T \quad \dots \quad \mathbf{J}_{i-1}^T]^T$.

3.9.2 End-effector Dynamics

An interesting method for analysis and control description is to have a look at the end-effector dynamics. To this end we can solve (3.1) for $\ddot{\mathbf{q}}$, insert into (3.81):

$$\dot{\mathbf{w}}_e = \mathbf{J}_e \mathbf{M}^{-1} (\boldsymbol{\tau} - \mathbf{b} - \mathbf{g}) + \dot{\mathbf{J}}_e \dot{\mathbf{q}}. \quad (3.84)$$

Substituting the joint torque with the end-effector force pre-multiplied with the Jacobian transposed

$$\boldsymbol{\tau} = \mathbf{J}_e^T \mathbf{F}_e \quad (3.85)$$

yields the end-effector dynamics

$$\boldsymbol{\Lambda}_e \dot{\mathbf{w}}_e + \boldsymbol{\mu} + \mathbf{p} = \mathbf{F}_e, \quad (3.86)$$

whereby

$$\boldsymbol{\Lambda}_e = (\mathbf{J}_e \mathbf{M}^{-1} \mathbf{J}_e^T)^{-1} \quad (3.87)$$

$$\boldsymbol{\mu} = \boldsymbol{\Lambda}_e \mathbf{J}_e \mathbf{M}^{-1} \mathbf{b} - \boldsymbol{\Lambda}_e \dot{\mathbf{J}}_e \dot{\mathbf{q}} \quad (3.88)$$

$$\mathbf{p} = \boldsymbol{\Lambda}_e \mathbf{J}_e \mathbf{M}^{-1} \mathbf{g} \quad (3.89)$$

represent the end-effector inertia, centrifugal/Coriolis, and gravitational terms in the task space. For more details, the interested reader is referred to [5].

3.9.3 End-effector Motion Control

Similar to joint-space inverse dynamics control, we can use the task-space equations of motion in inverse form to define the desired joint torque. Combining (3.85) and (3.86) results in

$$\boldsymbol{\tau}^* = \hat{\mathbf{J}}^T \left(\hat{\boldsymbol{\Lambda}}_e \dot{\mathbf{w}}_e^* + \hat{\boldsymbol{\mu}} + \hat{\mathbf{p}} \right). \quad (3.90)$$

We use this formalism together with control strategy for the desired acceleration $\dot{\mathbf{w}}^*$ such as e.g.:

$$\dot{\mathbf{w}}_e^* = \mathbf{k}_p \begin{pmatrix} \mathbf{r}_e^* - \mathbf{r}_e \\ \Delta \phi_e \end{pmatrix} + \mathbf{k}_d (\mathbf{w}_e^* - \mathbf{w}_e). \quad (3.91)$$

Please note that we use $\Delta \phi_e$ to represent the end-effector rotation error. Remember that, for rotation errors, we can not simply subtract two rotations but need follow the procedure introduced in section 2.9.3. For small errors we can make use of the error-approximation

$$\begin{pmatrix} \mathbf{r}_e^* - \mathbf{r}_e \\ \Delta \phi_e \end{pmatrix} = \begin{bmatrix} \mathbb{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_R \end{bmatrix} \begin{pmatrix} \mathbf{r}^* - \mathbf{r} \\ \boldsymbol{\chi}_R^* - \boldsymbol{\chi}_R \end{pmatrix}, \quad (3.92)$$

whereby the matrix $\mathbf{E}_R(\boldsymbol{\chi})$ is used due to the different methods of rotation parametrization.

Alternative Notation

In many papers you will find the expression $\boldsymbol{\Lambda} = \mathbf{J}_e^{-T} \mathbf{M} \mathbf{J}_e^{-1}$ although \mathbf{J}_e is not necessarily invertible. However, this simplifies algebraic manipulation of the equations. In

fact, (3.90) can be straight forward simplified to

$$\boldsymbol{\tau}^* = \mathbf{J}^T (\boldsymbol{\Lambda}_e \dot{\mathbf{w}}_e^* + \boldsymbol{\mu} + \mathbf{p}) \quad (3.93)$$

$$= \mathbf{J}^T \boldsymbol{\Lambda}_e \dot{\mathbf{w}}_e^* + \mathbf{J}^T \mathbf{J}_e^{-T} \mathbf{M} \mathbf{J}_e^{-1} \mathbf{J}_e \mathbf{M}^{-1} \mathbf{b} - \mathbf{J}^T \boldsymbol{\Lambda}_e \dot{\mathbf{J}}_e \dot{\mathbf{q}} + \mathbf{J}^T \mathbf{J}_e^{-T} \mathbf{M} \mathbf{J}_e^{-1} \mathbf{J}_e \mathbf{M}^{-1} \mathbf{g} \quad (3.94)$$

$$= \mathbf{J}^T \boldsymbol{\Lambda}_e \dot{\mathbf{w}}_e^* + \underbrace{\mathbf{b} - \mathbf{J}^T \boldsymbol{\Lambda}_e \dot{\mathbf{J}}_e \dot{\mathbf{q}}}_{\tilde{\mathbf{b}}} + \mathbf{g} \quad (3.95)$$

$$= \mathbf{J}^T \boldsymbol{\Lambda}_e \dot{\mathbf{w}}_e^* + \tilde{\mathbf{b}} + \mathbf{g} \quad (3.96)$$

The following operational space formulation can be equally written in this formulation.

3.9.4 Operational Space Control

There exist many situations where the robot should apply a force in some directions while it needs to move in other directions. An example is cleaning a window, whereby the robot applies a specific pressure force in normal direction and controls the motion in all other directions. Another example would be inserting a pin into a hole: Whereby, the pin should be moved in direction of the pin and should not rotate round the main axis of the pin, while the forces and moments around the two other axis should be kept zero. In order to achieve this, we make use of so-called operational space control. Following the work of Khatib [5], we can define selection matrices \mathbf{S}_M and \mathbf{S}_F for the motion and force directions, yielding the combined control problem

$$\boldsymbol{\tau}^* = \hat{\mathbf{J}}^T \left(\hat{\boldsymbol{\Lambda}} \mathbf{S}_M \dot{\mathbf{w}}_e + \mathbf{S}_F \mathbf{F}_c + \hat{\boldsymbol{\mu}} + \hat{\mathbf{p}} \right). \quad (3.97)$$

Following [5], we can define specification matrices for position and orientation

$$\boldsymbol{\Sigma}_p = \begin{bmatrix} \sigma_{px} & 0 & 0 \\ 0 & \sigma_{py} & 0 \\ 0 & 0 & \sigma_{pz} \end{bmatrix} \quad \boldsymbol{\Sigma}_r = \begin{bmatrix} \sigma_{rx} & 0 & 0 \\ 0 & \sigma_{ry} & 0 \\ 0 & 0 & \sigma_{rz} \end{bmatrix} \quad (3.98)$$

where σ_i are binary numbers assigned the value 1 when a free motion is specified along (linear) or around (rotation) specific axis, and zero otherwise. In case the contact force coordinate frame is rotated with respect to the inertial frame described by the rotation transformation matrix \mathbf{C} , we need to transform the selection matrix. The two selection matrices \mathbf{S}_F and \mathbf{S}_M are then defined as

$$\mathbf{S}_M = \begin{bmatrix} \mathbf{C}^T \boldsymbol{\Sigma}_p \mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}^T \boldsymbol{\Sigma}_r \mathbf{C} \end{bmatrix} \quad \mathbf{S}_F = \begin{bmatrix} \mathbf{C}^T (\mathbb{I}_3 - \boldsymbol{\Sigma}_p) \mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}^T (\mathbb{I}_3 - \boldsymbol{\Sigma}_r) \mathbf{C} \end{bmatrix} \quad (3.99)$$

3.10 Inverse Dynamics for Floating-Base Systems

When working with floating-base systems, we need to perform inverse dynamics while ensuring that the contact constraints (3.61) are satisfied. Given a desired acceleration $\dot{\mathbf{u}}^*_{consistent}$, we can invert the constraint consistent equation of motion (3.65):

$$\boldsymbol{\tau}^* = (\mathbf{N}_c^T \mathbf{S}^T)^+ \mathbf{N}_c^T (\mathbf{M}\dot{\mathbf{u}} + \mathbf{b} + \mathbf{g}). \quad (3.100)$$

Notice that we need to take the pseudo-inverse $(\mathbf{N}_c^T \mathbf{S}^T)^+$. Recall the multi-task solution from kinematics: depending on the structure of the matrix of which we take the pseudo-inverse, there exists a null-space that allows to modify $\boldsymbol{\tau}^*$:

$$\boldsymbol{\tau}^* = (\mathbf{N}_c^T \mathbf{S}^T)^+ \mathbf{N}_c^T (\mathbf{M}\dot{\mathbf{u}}^* + \mathbf{b} + \mathbf{g}) + \mathcal{N}(\mathbf{N}_c^T \mathbf{S}^T) \boldsymbol{\tau}_0^*, \quad (3.101)$$

while the support consistent equation of motion $\mathbf{N}_c^T \mathbf{S}^T \boldsymbol{\tau}^* = \mathbf{N}_c^T (\mathbf{M}\dot{\mathbf{u}} + \mathbf{b} + \mathbf{g})$ is still valid. In other words, there exist different joint torque distributions which all lead to the same motion $\dot{\mathbf{u}}^*$ of the system. In fact, the different torque distributions change the contact force distribution. In case of multiple contacts, we can create internal forces between the contacts which does not change the net force and moment on the robot and hence does not create any additional acceleration. When taking the pseudo-inverse as in (3.100), the solution is the least square minimal torque $\boldsymbol{\tau}^*$ that fulfills the constraint consistent equation of motion.

3.10.1 Quadratic Problems

There exist many approaches that tackle the problem of simultaneously controlling different operational-space objectives. These tasks involve motion at selected locations (e.g. end-effector, COG, etc.), desired contact forces, or joint torques.

A very comprehensive method is to understand operational space control as sequential least square optimization problem of linear objectives. To prepare for prioritized task-space inverse dynamics, we introduce in this section the concept of hierarchical least square optimization of a set of n_T linear equations

$$\mathbf{A}_i \mathbf{x} = \mathbf{b}_i, \quad (3.102)$$

with the optimization variable \mathbf{x} . Problems of the same priority $i \geq 1$, with $i = 1$ being the highest priority, are stacked in matrix \mathbf{A}_i and vector \mathbf{b}_i . As it will be shown later, motion tasks as well as joint torque and contact force tasks can be brought into this linear form whereby the optimization variable is the joint acceleration and joint torque, respectively. In the proposed hierarchical framework, the goal is to solve each task as good as possible in a least square sense

$$\min_{\mathbf{x}} \|\mathbf{A}_i \mathbf{x} - \mathbf{b}_i\|_2, \quad (3.103)$$

without influencing task of higher priority. There exist different methods to solve this problem such as e.g. iterative null-space projection (section 3.10.2) or as a sequence of constrained quadratic programs (QP) using standard numerical solvers.

3.10.2 Iterative Null-Space Projection

The requirement that a task is not allowed to influence any task with higher priority can be formulated by defining \mathbf{x} as a sum of task specific \mathbf{x}_i pre-multiplied with the

null-space projection matrix \mathbf{N}_i of higher prioritized tasks:

$$\mathbf{x} = \sum_{k=1}^{n_T} \mathbf{N}_k \mathbf{x}_k. \quad (3.104)$$

The null-space projector \mathbf{N}_i is defined as $\mathbf{N}_i = \mathcal{N} \left([\mathbf{A}_1^T \ \dots \ \mathbf{A}_{i-1}^T]^T \right)$ with $\mathbf{N}_1 = \mathbb{I}$ and the sufficient property

$$\mathbf{A}_i \mathbf{N}_j = \mathbf{0} \quad \forall i < j. \quad (3.105)$$

As we have seen in the kinematics part, there exist different methods for null-space projector calculation. Using property (3.105), the prioritized minimization problem (3.103) can be solved for each task individually by inserting (3.104) and solving for \mathbf{x}_i :

$$\mathbf{A}_i \mathbf{x} - \mathbf{b}_i = \mathbf{A}_i \sum_{k=1}^{n_T} \mathbf{N}_k \mathbf{x}_k - \mathbf{b}_i \quad (3.106)$$

$$\mathbf{x}_i = (\mathbf{A}_i \mathbf{N}_i)^+ \left(\mathbf{b}_i - \mathbf{A}_i \sum_{k=1}^{i-1} \mathbf{N}_k \mathbf{x}_k \right). \quad (3.107)$$

The calculation sequence of the optimization procedure is implemented as:

Algorithm 2 Hierarchical Least Square Optimization

```

 $n_T$  = Number of Tasks
 $\mathbf{x} = \mathbf{0}$  ▷ initial optimal solution
 $\mathbf{N}_1 = \mathbb{I}$  ▷ initial null-space projector
for  $i = 1 \rightarrow n_T$  do
     $\mathbf{x}_i = (\mathbf{A}_i \mathbf{N}_i)^+ (\mathbf{b}_i - \mathbf{A}_i \mathbf{x})$ 
     $\mathbf{x} = \mathbf{x} + \mathbf{N}_i \mathbf{x}_i$ 
     $\mathbf{N}_{i+1} = \mathcal{N} \left( [\mathbf{A}_1^T \ \dots \ \mathbf{A}_i^T]^T \right)$ 
end for

```

3.10.3 Sequence of Constrained Optimization

Every single step of the hierarchical least square optimization corresponds to a quadratic optimization with the linear constraint that tasks of lower priority are not allowed to change:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \|\mathbf{A}_i \mathbf{x} - \mathbf{b}_i\|_2 \\ \text{s.t.} \quad & \underbrace{\begin{bmatrix} \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_{i-1} \end{bmatrix}}_{\hat{\mathbf{A}}_{i-1}} \mathbf{x} - \underbrace{\begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_{i-1} \end{pmatrix}}_{\hat{\mathbf{b}}_{i-1}} = \mathbf{c}. \end{aligned} \quad (3.108)$$

Note: As long as $\hat{\mathbf{A}}_{i-1}$ has full column rank, the cost \mathbf{c} is $\mathbf{0}$.

This sequence of constrained quadratic optimization can be solved using a standard QP solver.

3.10.4 Task-Space Control for Floating Base Systems as QP

The complex behaviors of robotic systems can be described as multi-tasks control problems with the optimization variable as stacked vector of generalized acceleration $\dot{\mathbf{u}}$, contact force \mathbf{F}_c , and joint torque $\boldsymbol{\tau}$:

$$\mathbf{x} = \begin{pmatrix} \dot{\mathbf{u}} \\ \mathbf{F}_c \\ \boldsymbol{\tau} \end{pmatrix} \quad (3.109)$$

Please note: Our formulation here is for floating base systems using generalized velocities \mathbf{u} , but the approach can be also used for fixed base systems. Using this optimization variable, the equation of motion $\mathbf{M}\dot{\mathbf{u}} + \mathbf{b} + \mathbf{g} + \mathbf{J}_c^T \mathbf{F}_c = \mathbf{S}^T \boldsymbol{\tau}$ can be formulated as least square problem with

$$\mathbf{A} = [\hat{\mathbf{M}} \quad \hat{\mathbf{J}}_c^T \quad -\mathbf{S}^T] \quad \mathbf{b} = -\hat{\mathbf{b}} - \hat{\mathbf{g}}. \quad (3.110)$$

To achieve a desired acceleration at a point of interest $\mathbf{J}\dot{\mathbf{u}} + \dot{\mathbf{J}}\mathbf{u} = \dot{\mathbf{w}}^*$, we can simply define a task

$$\mathbf{A} = [\hat{\mathbf{J}}_i \quad \mathbf{0} \quad \mathbf{0}] \quad \mathbf{b} = \dot{\mathbf{w}}^* - \hat{\mathbf{J}}_i \mathbf{u} \quad (3.111)$$

There are also many other objectives of interest. In the following we give some examples but there are many more that can be achieved. In case we would like to push with a specific force or moment $\mathbf{F}_i = \mathbf{F}_i^*$ at certain location, we can define

$$\mathbf{A} = [\mathbf{0} \quad \mathbb{I} \quad \mathbf{0}] \quad \mathbf{b} = \mathbf{F}_i^* \quad (3.112)$$

Moreover, if we want to find a solution that minimizes the overall joint torque, we can define a task:

$$\mathbf{A} = [\mathbf{0} \quad \mathbf{0} \quad \mathbb{I}] \quad \mathbf{b} = \mathbf{0} \quad (3.113)$$

Example 3.10.1: Inverse joint space dynamics as QP

Given desired accelerations $\ddot{\mathbf{q}}^*$, please formulate joint space inverse dynamics control for a fixed base manipulator that is not in contact with the environment. Note: this is a purely academic example since solving this problem using the optimization approach presented in this section is an overkill.

Task 1: EoM

$$\mathbf{A} = [\hat{\mathbf{M}} \quad \mathbf{0} \quad -\mathbb{I}] \quad \mathbf{b} = -\hat{\mathbf{b}} - \hat{\mathbf{g}}. \quad (3.114)$$

Task 2: Achieve desired joint space acceleration

$$\mathbf{A} = [\mathbb{I} \quad \mathbf{0} \quad \mathbf{0}] \quad \mathbf{b} = \ddot{\mathbf{q}}^*. \quad (3.115)$$

3.11 Quasi-static (Virtual Model) Control

Note: This section is not part of this years lecture but is still provided for the interested reader.

For relatively slow maneuvers, the predominant forces are external loads acting on the robot. In such situation, we can look at the quasistatic equations of motion which neglect the terms $\mathbf{M}\ddot{\mathbf{q}}$ and $\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}})$. A widely used quasi-static control approach is Virtual model control (VMC) as introduced by Pratt et al. [6] This method generates desired joint torques to produce the same effect on the system as if there were external (virtual) forces \mathbf{F}_{v_i} acting at specified locations \mathbf{r}_{v_i} . To derive a generalized form of VMC for floating base systems, we apply the *principle of virtual work* (see section 3.2.5) which states that variations in work must cancel for all virtual displacements of the multi-body system. We remember that a virtual displacement at an arbitrary point p_i of a floating base systems can be written as

$$\delta \mathbf{r}_{p_i} = \delta \mathbf{r}_b + \delta \boldsymbol{\varphi}_b \times \mathbf{r}_{bp_i} + \delta \mathbf{r}_{bp_i} = \begin{bmatrix} \mathbb{I} & -[\mathbf{r}_{bp_i}]_{\times} & \mathbf{J}_{bp_i} \end{bmatrix} \begin{pmatrix} \delta \mathbf{r}_b \\ \delta \boldsymbol{\varphi}_b \\ \delta \mathbf{q}_r \end{pmatrix}, \quad (3.116)$$

with $\delta \mathbf{r}_b$ and $\delta \boldsymbol{\varphi}_b$ being the variation in position and rotation of the base node b with respect to an inertial frame. The skew-symmetric matrix $[\mathbf{r}_{bp_i}]_{\times} = -[\mathbf{r}_{bp_i}]_{\times}^T$ corresponds to the cross-multiplication matrix $[\mathbf{r}_{bp_i}]_{\times} \boldsymbol{\varphi}_b = \mathbf{r}_{bp_i} \times \boldsymbol{\varphi}_b$ of the relative position vector \mathbf{r}_{bp_i} from base b to point p_i . The relative position variation $\delta \mathbf{r}_{bp_i} = \mathbf{J}_{bp_i} \delta \mathbf{q}_r$ is expressed by a variation in generalized joint coordinates $\delta \mathbf{q}_r$ projected by the relative Jacobian $\mathbf{J}_{bp_i} = \frac{\partial \mathbf{r}_{bp_i}}{\partial \mathbf{q}_r}$. With this parametrization of an arbitrary field of variations the virtual work generated by external and internal forces results in

$$\begin{aligned} \delta W &= \delta \mathbf{q}_r^T \boldsymbol{\tau} + \sum_i \delta \mathbf{r}_{p_i}^T \mathbf{F}_{p_i} \\ &= \begin{bmatrix} \delta \mathbf{r}_b^T & \delta \boldsymbol{\varphi}_b^T & \delta \mathbf{q}_r^T \end{bmatrix} \left(\begin{pmatrix} 0 \\ 0 \\ \boldsymbol{\tau} \end{pmatrix} + \sum_i \begin{bmatrix} \mathbb{I} \\ [\mathbf{r}_{bp_i}]_{\times} \\ \mathbf{J}_{bp_i}^T \end{bmatrix} \mathbf{F}_{p_i} \right) = \mathbf{0} \quad \forall \begin{pmatrix} \delta \mathbf{r}_b \\ \delta \boldsymbol{\varphi}_b \\ \delta \mathbf{q}_r \end{pmatrix}, \end{aligned}$$

$$\rightarrow 0 = \sum_i \mathbf{F}_{p_i}, \quad (3.117)$$

$$\rightarrow 0 = \sum_i \mathbf{r}_{bp_i} \times \mathbf{F}_{p_i}, \quad (3.118)$$

$$\rightarrow 0 = \boldsymbol{\tau} + \sum_i \mathbf{J}_{bp_i}^T \mathbf{F}_{p_i}, \quad (3.119)$$

with $\sum_i \mathbf{F}_{p_i} = \sum_i \mathbf{F}_{g_i} - \sum_i \mathbf{F}_{v_i} - \sum_i \mathbf{F}_{c_i}$ representing all external forces such as the gravitational forces (\mathbf{F}_{g_i}), virtual control forces ($-\mathbf{F}_{v_i}$), and the contact forces ($-\mathbf{F}_{c_i}$). Equations (3.117) and (3.118) correspond to the force respectively torque equilibrium of all external loads and are used to determine the unknown ground contact forces \mathbf{F}_{c_i} . In most cases, this is done by a pseudo-inversion according to

$$\begin{pmatrix} \mathbf{F}_{c_1} \\ \vdots \\ \mathbf{F}_{c_{n_c}} \end{pmatrix} = \begin{bmatrix} \mathbb{I} & \cdots & \mathbb{I} \\ [\mathbf{r}_{c_1}]_{\times} & \cdots & [\mathbf{r}_{c_{n_c}}]_{\times} \end{bmatrix}^+ \begin{bmatrix} \sum \mathbf{F}_{g_i} - \sum \mathbf{F}_{v_i} \\ \sum \mathbf{r}_{g_i} \times \mathbf{F}_{g_i} - \sum \mathbf{r}_{v_i} \times \mathbf{F}_{v_i} \end{bmatrix}. \quad (3.120)$$

Given all external forces, the desired joint torques are extracted from (3.119):

$$\boldsymbol{\tau} = - \sum_i \mathbf{J}_{bg_i}^T \mathbf{F}_{g_i} + \sum_i \mathbf{J}_{bv_i}^T \mathbf{F}_{v_i} + \sum_i \mathbf{J}_{bc_i}^T \mathbf{F}_{c_i}. \quad (3.121)$$

For simplicity of notation, we avoided the inclusion of external moments τ_i^a . They can be simply added in the formalism with the corresponding rotational Jacobian as $\sum_i \mathbf{J}_{R_i}^T \tau_i^a$.

Appendix A

Matlab Code for Examples

This chapter collects all MATLAB code snippets for the individual examples.

A.1 Multi-task Control

Code for example 2.9.1.

```
clc
clear all
close all

% define the end-effector jacobian
J_E = @(q) [
    cos(q(1))+cos(q(1)+q(2))+cos(q(1)+q(2)+q(3)),...
    cos(q(1)+q(2))+cos(q(1)+q(2)+q(3)), cos(q(1)+q(2)+q(3)) ;
    -sin(q(1))-sin(q(1)+q(2))-sin(q(1)+q(2)+q(3)),...
    -sin(q(1)+q(2))-sin(q(1)+q(2)+q(3)), -sin(q(1)+q(2)+q(3)) ];
w_E_des = [1;1];

% numerical evaluation at time t
q_t = [pi/6,pi/3,pi/3];
J_Et = J_E(q_t);

% define the jacobian s.t. joint 1 has zero velocity
J_j1 = [1,0,0];
w_j1_des = [0];

% define the jacobian s.t. joint 2 has zero velocity
J_j2 = [0,1,0];
w_j2_des = [0];

% combined joint Jacobian
Jj = [J_j1;J_j2];
wj = [w_j1_des;w_j2_des];

%% Fulfill only the first task
disp('EX 1: fulfill only end-effector tracking')
dq_single = pinv(J_Et)*w_E_des;
printmat(pinv(J_Et),'pinv(J_Et)')
printmat(dq_single,'dq_single=pinv(J_Et)*w_E_des')
% Error calculation
printmat(J_Et*dq_single,'J_Et*dq_single')
```

```

printmat(abs(w_E_des - J_Et*dq_single),'|w_E_des - J_Et*dq_single|')
disp(['||w_E_des - J_Et*dq_single||^2 = ',...
      num2str((w_E_des - J_Et*dq_single)'*(w_E_des - J_Et*dq_single))]) ;

printmat(Jj*dq_single,'Jj*dq_single')
printmat(abs(wj - Jj*dq_single),'|wj - Jj*dq_single|')
disp(['||wj - Jj*dq_single||^2 = ',...
      num2str((wj - Jj*dq_single)'*(wj - Jj*dq_single))]) ;

%% Fulfill all tasks with equal priority
disp('EX 2: fulfill all tasks with equal priority')
dq_stack = pinv([J_Et;Jj])*[w_E_des;wj];
printmat(pinv([J_Et;Jj]),'pinv([J_Et;Jj])')
printmat(dq_stack,'dq_stack=pinv([J_Et;Jj])*[w_E_des;wj]')
% Error calculation
printmat(J_Et*dq_stack,'J_Et*dq_stack')
printmat(abs(w_E_des - J_Et*dq_stack),'|w_E_des - J_Et*dq_stack|')
disp(['||w_E_des - J_Et*dq_stack||^2 = ',...
      num2str((w_E_des - J_Et*dq_stack)'*(w_E_des - J_Et*dq_stack))]);

printmat(Jj*dq_stack,'Jj*dq_stack')
printmat(abs(wj - Jj*dq_stack),'|wj - Jj*dq_stack|')
disp(['||wj - Jj*dq_stack||^2 = ',...
      num2str((wj - Jj*dq_stack)'*(wj - Jj*dq_stack))]);

%% Fulfill both tasks but give the tracking a higher priority
disp('EX 3: fulfill tracking task with higher priority')
J1 = J_Et;
J2 = Jj;
w1 = w_E_des;
w2 = wj;
% calculate the null-space matrix
N1 = eye(3)-pinv(J1)*J1;
printmat(N1,'N1');
disp(['rank(N1) = ',num2str(rank(N1))]);
dq_prio = pinv(J1)*w1+N1*pinv(J2*N1,1e-6)*(wj-J2*pinv(J1)*w1);
printmat(dq_prio,'dq_prio = pinv(J1)*w1+N1*pinv(J2*N1)*(w2-J2*pinv(J1)*w1)')
% Error calculation
printmat(J_Et*dq_prio,'J_Et*dq_prio')
printmat(abs(w_E_des - J_Et*dq_prio),'|w_E_des - J_Et*dq_prio|')
disp(['||w_E_des - J_Et*dq_prio||^2 = ',...
      num2str((w_E_des - J_Et*dq_prio)'*(w_E_des - J_Et*dq_prio))]) ;

printmat(Jj*dq_prio,'Jj*dq_prio')
printmat(abs(wj - Jj*dq_prio),'|wj - Jj*dq_prio|')
disp(['||wj - Jj*dq_prio||^2 = ',...
      num2str((wj - Jj*dq_prio)'*(wj - Jj*dq_prio))]) ;

```

A.2 Inverse Kinematics for Rotations

Code for example 2.9.7.

```

clc
clear all
close all

%%

```

```

% consider a robot with three successive joints rotating around z, y, z
% the end-effector rotation matrix is
Cx = @(x) [1,0,0;0,cos(x),-sin(x);0,sin(x),cos(x)];
Cy = @(y) [cos(y),0,sin(y);0,1,0;-sin(y),0,cos(y)];
Cz = @(z) [cos(z),-sin(z),0;sin(z),cos(z),0;0,0,1];
Czyx = @(q) Cz(q(1))*Cy(q(2))*Cx(q(3));

% basic Jacobian is given by
w1 = @(q) [0;0;1];
w2 = @(q) Cz(q(1))*[0;1;0];
w3 = @(q) Cz(q(1))*Cy(q(2))*[1;0;0];
J0 = @(q) [w1(q),w2(q),w3(q)];

%% analytical Jacobian and E matrix for ZYX Euler angles
% chi_EulerZYX = [z;y;x];
% omega = E(chi)*dchi
% J0 = E*JA <=> JA=E\J0
E_eulerZYX = @(chi) [
0,-sin(chi(1)),cos(chi(2))*cos(chi(1));
0,cos(chi(1)),cos(chi(2))*sin(chi(1));
1,0,-sin(chi(2))];
JA_eulerZYX = @(chi,q) E_eulerZYX(chi)\J0(q);
rotmat2euleranglesZYX = @(C) [
atan2(C(2,1),C(1,1));
-atan2(C(3,1),sqrt(C(3,2)^2+C(3,3)^2));
atan2(C(3,2),C(3,3))];

%% analytical Jacobian and E matrix for XYZ Euler angles
% chi_EulerXYZ = [x;y;z];
E_eulerXYZ = @(chi) [
1,0,sin(chi(2));
0,cos(chi(1)), -cos(chi(2))*sin(chi(1));
0,sin(chi(1)), cos(chi(1))*cos(chi(2))];
JA_eulerXYZ = @(chi,q) E_eulerXYZ(chi)\J0(q);
rotmat2euleranglesXYZ = @(C) [
atan2(-C(2,3),C(3,3));
atan2(C(1,3),sqrt(C(1,1)^2+C(1,2)^2));
atan2(-C(1,2),C(1,1))];

%% analytical Jacobian and E matrix for rotation vector
% chi_rotvec = [rx;ry;rz]
skew = @(a) [0,-a(3),a(2);a(3),0,-a(1);-a(2),a(1),0];
E_rotvec = @(x) ...
eye(3)+skew(x)*(1-cos(norm(x)))/norm(x)^2+...
skew(x)*skew(x)*(norm(x)-sin(norm(x)))/norm(x)^3;
JA_rotvec = @(chi,q) E_rotvec(chi)\J0(q);
rotmat2rotvec = @(C) ...
acos((C(1,1)+C(2,2)+C(3,3)-1)/2)*1/...
(2*sin(acos((C(1,1)+C(2,2)+C(3,3)-1)/2)))*...
[C(3,2)-C(2,3);C(1,3)-C(3,1);C(2,1)-C(1,2)];
rotvec2rotmat = @(x) ...
eye(3)+sin(norm(x))/norm(x)*skew(x)+...
(1-cos(norm(x)))*skew(x)*skew(x)/norm(x)^2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Numerical evaluation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% start conditions
q_0 = [-0.7;0;1.5];
q_goal = [0.7;1.5;-0.5];

```

```

C0 = Cxyz(q_0);
Cgoal = Cxyz(q_goal);

% define k parameter
k = 0.001;

%% ZYX euler angles
q_eulerZYX_v(:,1) = q_0;
chi_eulerZYX_goal = rotmat2euleranglesZYX(Cxyz(q_goal));
for i=1:10000
    chi_eulerZYX_v(:,i) = rotmat2euleranglesZYX(Cxyz(q_eulerZYX_v(:,i)));
    JA_eulerZYX_eval = JA_eulerZYX(chi_eulerZYX_v(:,i),q_eulerZYX_v(:,i));
    Δ_chi_eulerZYX = chi_eulerZYX_goal-chi_eulerZYX_v(:,i);
    q_eulerZYX_v(:,i+1) = ...
        q_eulerZYX_v(:,i) + k*(JA_eulerZYX_eval\Δ_chi_eulerZYX);

    C_eulerZYX_mat(i, :, :) = Cxyz(q_eulerZYX_v(:,i));

    if norm(Δ_chi_eulerZYX)<0.01
        break
    end
end

%% XYZ euler angles
q_eulerXYZ_v(:,1) = q_0;
chi_eulerXYZ_goal = rotmat2euleranglesXYZ(Cxyz(q_goal));
for i=1:10000
    chi_eulerXYZ_v(:,i) = rotmat2euleranglesXYZ(Cxyz(q_eulerXYZ_v(:,i)));
    JA_eulerXYZ_eval = JA_eulerXYZ(chi_eulerXYZ_v(:,i),q_eulerXYZ_v(:,i));
    Δ_chi_eulerXYZ = chi_eulerXYZ_goal-chi_eulerXYZ_v(:,i);
    q_eulerXYZ_v(:,i+1) = ...
        q_eulerXYZ_v(:,i) + k*(JA_eulerXYZ_eval\Δ_chi_eulerXYZ);

    C_eulerXYZ_mat(i, :, :) = Cxyz(q_eulerXYZ_v(:,i));

    if norm(Δ_chi_eulerXYZ)<0.01
        break
    end
end

%% rotvector
q_rotvec_v(:,1) = q_0;
chi_rotvec_goal = rotmat2rotvec(Cxyz(q_goal));
for i=1:10000
    chi_rotvec_v(:,i) = rotmat2rotvec(Cxyz(q_rotvec_v(:,i)));
    JA_rotvec_eval = JA_rotvec(chi_rotvec_v(:,i),q_rotvec_v(:,i));
    Δ_chi_rotvec = chi_rotvec_goal-chi_rotvec_v(:,i);
    q_rotvec_v(:,i+1) = ...
        q_rotvec_v(:,i) + k*(JA_rotvec_eval\Δ_chi_rotvec);

    C_rotvec_mat(i, :, :) = Cxyz(q_rotvec_v(:,i));

    if norm(Δ_chi_rotvec)<0.01
        break
    end
end

%% error function calculated from relative rotation
q_omega_v(:,1) = q_0;
for i=1:10000

```

```

chi_omega_v(:,i) = rotmat2rotvec(Czyx(q_omega_v(:,i)));
J0e = J0(q_omega_v(:,i));
CΔ = Czyx(q_goal)*Czyx(q_omega_v(:,i))';
chi_CΔ = rotmat2rotvec(CΔ);
q_omega_v(:,i+1) = q_omega_v(:,i) + k*(J0e\chi_CΔ);

C_omega_mat(i, :, :) = Czyx(q_omega_v(:,i));
if norm(chi_CΔ)<0.01
    break
end
end

%% plotting
figure
hold on
axis equal
grid on

[xs,ys,zs] = sphere;
surf(xs,ys,zs,'FaceColor',[0.8 0.8 0.8],'FaceAlpha',0.5)

for i=1:3
    % initial CS
    plot3([0,C0(1,i)],[0,C0(2,i)],[0,C0(3,i)],'b','linewidth',2)
    % goal CS
    plot3([0,Cgoal(1,i)],[0,Cgoal(2,i)],[0,Cgoal(3,i)],'r','linewidth',2)

    % trajectories
    plot3(C_eulerZYX_mat(:,1,i),C_eulerZYX_mat(:,2,i),...
        C_eulerZYX_mat(:,3,i),'b','linewidth',4)
    plot3(C_eulerXYZ_mat(:,1,i),C_eulerXYZ_mat(:,2,i),...
        C_eulerXYZ_mat(:,3,i),'k','linewidth',4)
    plot3(C_rotvec_mat(:,1,i),C_rotvec_mat(:,2,i),...
        C_rotvec_mat(:,3,i),'r','linewidth',4)
    plot3(C_omega_mat(:,1,i),C_omega_mat(:,2,i),...
        C_omega_mat(:,3,i),'g','linewidth',4)
end

```


Appendix B

Matlab Code for 3D rotations

This chapter collects functions which implement the theory discussed for rotations in the three dimensional space. This code is part of the open-source project *Kindr - Kinematics and Dynamics for Robotics*.

B.1 Euler angles to rotation matrix

This section implements the composition of elementary rotations to obtain a rotation matrix which is associated to a given set of Euler angles (ZYZ, ZXZ, ZYX and XYZ).

```
function R = mapEulerAnglesXYZToRotationMatrix(angles)
% MAPEULERANGLESXYZTOROTATIONMATRIX(angles) maps a set of Euler angles to a
% rotation matrix in SO(3). The Euler angles represent a set successive
% rotations around X-Y'-Z''. This is equivalent to rotating around the
% fixed axes in Z-Y-X order.
%
% Author(s): Dario Bellicoso

x = angles(1);
y = angles(2);
z = angles(3);

R = getRotationMatrixX(x)*getRotationMatrixY(y)*getRotationMatrixZ(z);

if isa(angles, 'sym')
    R = simplify(R);
end

end
```

```
function R = mapEulerAnglesZXZToRotationMatrix(angles)
% MAPEULERANGLESZXZTOROTATIONMATRIX(angles) maps a set of Euler angles to a
% rotation matrix in SO(3). The Euler angles represent a set successive
% rotations around Z-X'-Z''.
%
% Author(s): Dario Bellicoso

z1 = angles(1);
x = angles(2);
```

```

z2 = angles(3);

R = getRotationMatrixZ(z1)*getRotationMatrixX(x)*getRotationMatrixZ(z2);

if isa(angles, 'sym')
    R = simplify(R);
end

end

```

```

function R = mapEulerAnglesZYXToRotationMatrix(angles)
% MAPEULERANGLESZYXTOROTATIONMATRIX(angles) maps a set of Euler angles to a
% rotation matrix in SO(3). The Euler angles represent a set successive
% rotations around Z-Y'-X''. This is equivalent to rotating around the
% fixed axes in X-Y-Z order.
%
% Author(s): Dario Bellicoso

z = angles(1);
y = angles(2);
x = angles(3);

if isa(angles, 'sym')
    R = simplify(getRotationMatrixZ(z)*getRotationMatrixY(y)*getRotationMatrixX(x));
else
    R = getRotationMatrixZ(z)*getRotationMatrixY(y)*getRotationMatrixX(x);
end

end

```

```

function R = mapEulerAnglesZYZToRotationMatrix(angles)
% MAPEULERANGLESZYZTOROTATIONMATRIX(angles) maps a set of Euler angles to a
% rotation matrix in SO(3). The Euler angles represent a set successive
% rotations around Z-Y'-Z''.
%
% Author(s): Dario Bellicoso

z1 = angles(1);
y = angles(2);
z2 = angles(3);

R = getRotationMatrixZ(z1)*getRotationMatrixY(y)*getRotationMatrixZ(z2);

if isa(angles, 'sym')
    R = simplify(R);
end

end

```

B.2 Rotation matrix to Euler angles

This section implements the extraction of Euler angles (ZYZ, ZXZ, ZYX and XYZ) from a given rotation matrix.

```
function ph = getEulAngXYZFromRotationMatrix(C)
% GETEULANGXYZFROMROTATIONMATRIX(C) extracts XYZ Euler angles from a
% rotation matrix.
%
% Author(s): Dario Bellicoso

x = atan2(-C(2,3),C(3,3));
y = atan2(C(1,3), sqrt(C(1,1)^2+C(1,2)^2));
z = atan2(-C(1,2),C(1,1));

ph = [x y z]';
```

```
function ph = getEulAngZXZFromRotationMatrix(C)
% GETEULANGZYXFROMROTATIONMATRIX(C) extracts ZXZ Euler angles from a
% rotation matrix.
%
% Author(s): Dario Bellicoso

z1 = atan2(C(1,3),-C(2,3));
x = atan2(sqrt(C(1,3)^2+C(2,3)^2), C(3,3));
z2 = atan2(C(3,1),C(3,2));

ph = [z1 x z2]';
```

```
function ph = getEulAngZYXFromRotationMatrix(C)
% GETEULANGZYXFROMROTATIONMATRIX(C) extracts ZYX Euler angles from a
% rotation matrix.
%
% Author(s): Dario Bellicoso

z = atan2(C(2,1),C(1,1));
y = atan2(-C(3,1), sqrt(C(3,2)^2+C(3,3)^2));
x = atan2(C(3,2),C(3,3));

ph = [z y x]';
```

```
function ph = getEulAngZYZFromRotationMatrix(C)
% GETEULANGZYXFROMROTATIONMATRIX(C) extracts ZYZ Euler angles from a
% rotation matrix.
%
% Author(s): Dario Bellicoso

z1 = atan2(C(2,3),C(1,3));
x = atan2(sqrt(C(1,3)^2+C(2,3)^2), C(3,3));
z2 = atan2(C(3,2),-C(3,1));

ph = [z1 x z2]';
```


Bibliography

- [1] Haim Baruh. *Analytical Dynamics*. WCB/McGraw-Hill Boston, 1999. ISBN 0073659770. URL <http://coewww.rutgers.edu/~baruh/book.html>.
- [2] Michael Bloesch, Hannes Sommer, Tristan Laidlow, Michael Burri, Gabriel Nuetzi, Péter Fankhauser, Dario Bellicoso, Christian Gehring, Stefan Leutenegger, Marco Hutter, and Roland Siegwart. A Primer on the Differential Calculus of 3D Orientations. Technical report, jun 2016. URL <http://arxiv.org/abs/1606.05285>.
- [3] S. R. Buss. Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods. Technical report, 2004. URL <http://math.ucsd.edu/~sbuss/ResearchWeb>.
- [4] H. Goldstein, C.P. Poole, and J.L. Safko. *Classical Mechanics*. Addison Wesley, 2002. ISBN 9780201657029.
- [5] O Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, 3(1):43–53, 1987. URL <http://dx.doi.org/10.1109/JRA.1987.1087068>.
- [6] Jerry Pratt, Chee-Meng Chew, Ann Torres, Peter Dilworth, and Gill Pratt. Virtual model control: An intuitive approach for bipedal locomotion. *International Journal of Robotics Research (IJRR)*, 20(2):129–143, 2001. doi: 10.1177/02783640122067309. URL <http://dx.doi.org/10.1177/02783640122067309>.
- [7] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-23957-4. doi: 10.1007/978-3-540-30301-5. URL <http://link.springer.com/10.1007/978-3-540-30301-5>.
- [8] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics - Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer London, London, 2009. ISBN 978-1-84628-641-4. doi: 10.1007/978-1-84628-642-1. URL <http://link.springer.com/10.1007/978-1-84628-642-1>.
- [9] Joan Sola. Quaternion Kinematics for the Error-State KF. Technical report, 2016. URL <http://www.iri.upc.edu/people/jsola/JoanSola/objectes/notes/kinematics.pdf>.

- [10] Carlo Tomasi. Linear Systems. Technical report, 2015. URL <https://www2.cs.duke.edu/courses/fall15/compsci527/notes/linear-systems.pdf>.