Introduction to Robotics

Module: Trajectory generation and robot programming

(summer term 2000)

Recommended literature

Text Books

- Craig,J.J.: Introduction to Robotics. Addison-Wesley, 2nd. Ed, 1989 (3rd edition will be available 2000 ?)
- Craig,J.: Adaptive Control of Mechanical Manipulators. Addison-Wesley, Reading(Mass.), 1988
- R.P. Paul: Robot Manipulators, MIT Press 1981

Journals

- Transactions on Robotics and Automation, IEEE
- IEEE Conference on Robotics and Automation

Acknowledgement

This handout has been partially derived from the lecture notes from Wolfgang Weber teaching robotics at the Fachhochschule Darmstadt, Germany and the above mentioned textbooks from J.J. Craig and R.P. Paul.

Revisions

Revision	Date	Author	History
0.9	12-Mar-99	Th. Horsch	1st draft
1.0	20.2.2000	Th. Horsch	1st release

CONTENT

1. Motivation

1.1. Why study robotics

Robots are mainly used for

- Automation of industrial tasks
  - Spot welding
  - Arc welding
  - Machine transfer and machine tending
  - Pick and Place
  - Clueing and sealing
  - Deburring
  - Assembly/Disassembly
- Testbed for Artificial Intelligence (AI)

Robotics is an interdisciplinary subject requiring

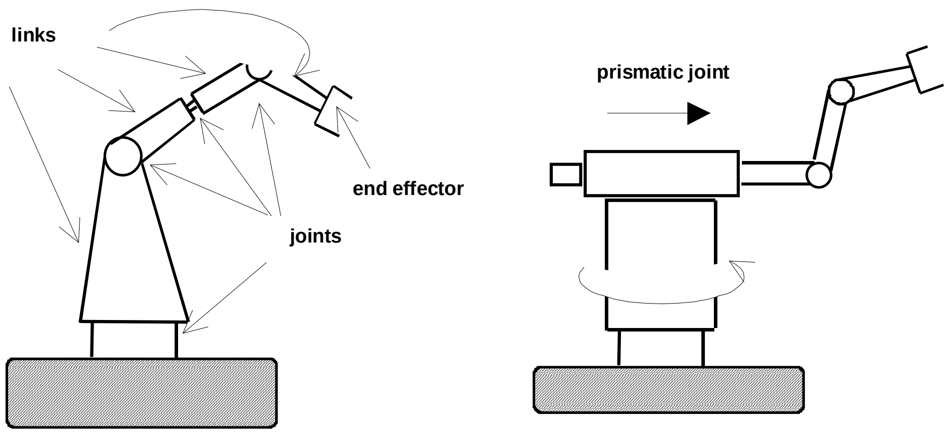
Mechanical Engineering	(addressed in module Components of a robot arm)
Electrical/Electronic Engineering	(addressed in module Control of a robot arm)
Computer Science/Mathematics	(addressed in this module)

It is worth noting that many problems in computer graphics animation are related to robotics problems (much of the material in this course is relevant to animation).

1.2. Terminology

We begin with some robot terminology.

Exactly what constitutes a robot is sometimes debated. Numerically controlled (milling) machines are usually not referred as robots. The distiction lies somewhere in the sophistication of the programmability of the device – if a mechanical device can be programmed to perfom a wide variety of applications, it is probably an indstrial robot. Machines which are for the most part relegated to one class of task are considered fixed automation.



Industrial robots constist of (nearly) rigid links which are connected with **joints** which allow relative motion of neighboring **links**. These joints are usually instrumented with position sensors which allow the relative position of neighboring links to be measured. In case of rotary or **revolute** joints, these displacements are called joint angles. Some robots contain sliding, or **prismatic** joints.

The number of **degrees of freedom** that a robot possesses is the number of independant position varaibles which would have to be specified in order to locate all parts of the mechanism. Usually a robot is an open kinematic chain. This implies, that each joint variable is usually defined with a single variable and the number of joints equals the number of degrees of freedom.

At the free end of the chain of links which make up the robot is the **end effector**. Depending on the intended application of the robot, the end effector may be a gripper, welding torch or any other device. We generally describe the position of the manipulator by giving a description of the **tool frame** or sometimes called the **TCP** frame (TCP=Tool Center Point), which is attached to the end-effector, relative to the **base frame** which is attached to the nonmoving base of the manipulator.

Frames (coordinate systems) are used to describe the position and orientation of a rigid body in space. They serve as a reference system within which to express the position and orientation of a body.

Exercises with EASY-ROB:

- a 6R robot with intersecting wrist axes (->load ->robot ->rv6.rob in EASY-ROB)
- a 2 degree of freedom (DOF) robot with rotational joints (2R planar robot) (->load ->robot ->dh2.rob in EASY-ROB)
- a 7R robot with rotational joints (->load ->robot ->dh_manth7.rob in EASY-ROB)

You can consider more examples with EASY-ROB. Robots from different robot vendors are available: Fanuc, ABB, KUKA, Reis (->load ->extern in EASY-ROB)

Kinematics is the science of motion which treats motion without regard to the forces which cause it. Within the science of kinematics one studies the position, velocity, acceleration, and all higher order derivatives of the position variables. Hence, the study of the kinematics of robots refers to all geometrical and time-based properties of the motion.

A very basic problem to be solved is:

How to relate the robot's **configuration** or pose to the position and orientation of its end effector. A configuration of an n-degree of freedom robot is an n-vector $(q_1, q_2, ..., q_n)$, where each q_i is either a rotational joint angle or a prismatic joint translation.
This is known as the **forward kinematics** of the robot. This is the static geometrical problem of computing the position and orientation of the end-effector of the robot. Specifically, given a set of joint angles, the forward kinematic problem is to compute the position and orientation of the TCP relative to the base frame. Sometimes we think of this as changing the representation of robot position from joint space description into a cartesian space description.

The following problem is considered the **inverse kinematics**: Given the position and orientation of the end-effector of the robot, calculate all possible sets of joint angles which could be used to attain the this given position and orientation. The inverse kinematics is not as simple as the forward kinematics. Because the kinematic equations are nonlinear, their solution is not always easy or even possible in a closed form. The existence of a kinematic solution defines the workspace of a given robot. The lack of a solution means that the robot cannot attain the desired position and orientation because it lies outside the robot's workspace.

In addition to dealing with static positioning problems, we may wish to analyse robots in motion. Often in performing velocity analysis of a mechanism it is convenient to define a matrix quantity called the **jacobian** of the robot. The jacobian specifies a mapping from velocities in joint space to velocities in cartesian space. The nature of this mapping changes as the configuration of the robot varies. At certain points, called singularities, this mapping is not invertible. An understanding of this phenomenon is important to designers and users of robots.

A common way of causing a robot to move from A to B in a smooth, controlled fashion is to cause each joint to move as specified by a smooth function of time. Commonly, each joint starts and ends ist motion at the same time, so that the robot motion appears coordinated. Exactly how to compute these motion functions is the problem of **trajectory generation**.

A **robot programming language** serves as an interface between the human user and the industrial robot. Central questions arise such as:
How are motions through space described easily by a programmer ?
How are multiple robots programmed so that they can work in parallel ?
How are sensor-based actions described in a language ?

The sophistication of the user interface is becoming extremely important as robots and other programmable devices are applied to more and more demanding industrial applications.

An **off-line programming system** is a robot programming environment which has been sufficiently extended, generally by means of computer graphics, so that the development of robot programs can take place without access to the robot itself. A common argument raised in the favor is that an off-line programming system will not cause production equipment (i.e. the robot) to be tied up when it needs to be reprogrammed; hence, automated factories can stay in production mode a greater percentage of time.

A robot constists of three main systems:

- Programming system:

A robot user needs to „teach“ the robot the specific task which is to be performed. This can be done with the so called teach in method. The programmer uses the teach box of the robot control and drives the robot to the diesired positions and stores them and other values like travel speed, corner smoothing parameters, process parameters, etc. For the programming period which can take a significant time for complex tasks the production is idle. Another possibility is the use of Off-line programming systems mentioned before.

- Robot control:

The robot control interprets the robot's application program and generates a series of joint values and joint velocities (and sometimes accellerations) in an appropriate way for the feedback control. Today mainly an independant joint control approach is taken, but more and more sophisticated control approaches, like adaptive control, nonlinear control, etc are involved in robotics. The following figure shows a possible control structure.

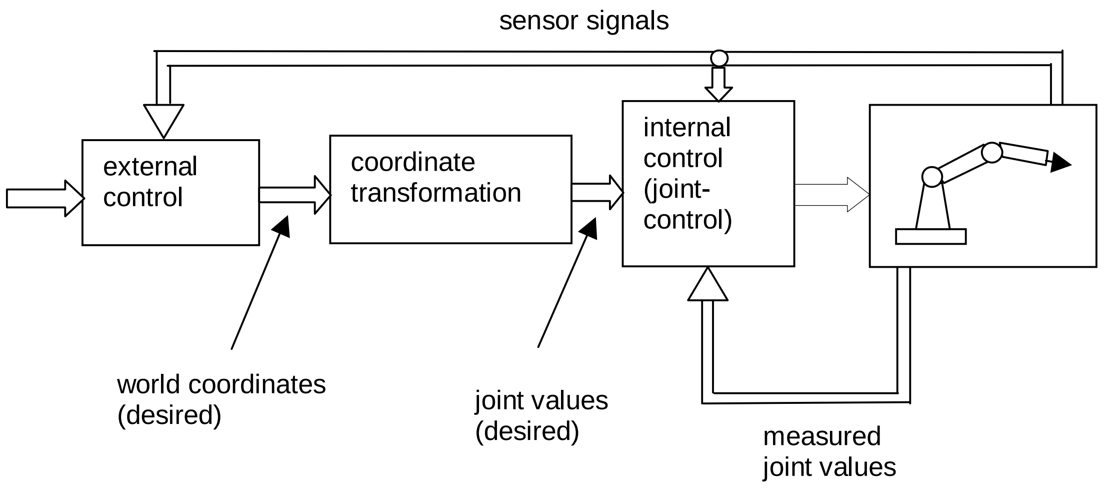

sensor signals
external control
coordinate transformation
internal control (joint-control)
world coordinates (desired)
joint values (desired)
measured joint values

Figure 1-1: *structure of a robot control*

- Robot mechanics:

The robot mechanics will transform the joint torques applied by the servo drives into an appropriate motion. Nowadays AC motors are used to drive the axis.
Resolvers serve as a position sensor and are normally located on the shaft of the actuator. Resolvers are devices which output two analog signals – one is the sine of the shaft angle and the other the cosine. The shaft angle is determined from the relative magnitude of the two signals.

1.3. Types of robot motion

PTP(Point to Point)-motion

A motion which specifies the configuration of the robot at the start and the end point. The motion between these points is not determined in the sense that the TCP follows a desired (cartesian) path. This motion type is mainly relevant for pick and place type tasks, where the position and orientation along the path is not important. Since this motion is very hard to predict in cartesian space, the programmer has to be careful regarding collisions between the robot and ist environment. On the other hand this motion is easy (and fast) in calculation, because no transformation of inverse kinematics needs to be computed.

Figure 1-1 describes two examples:

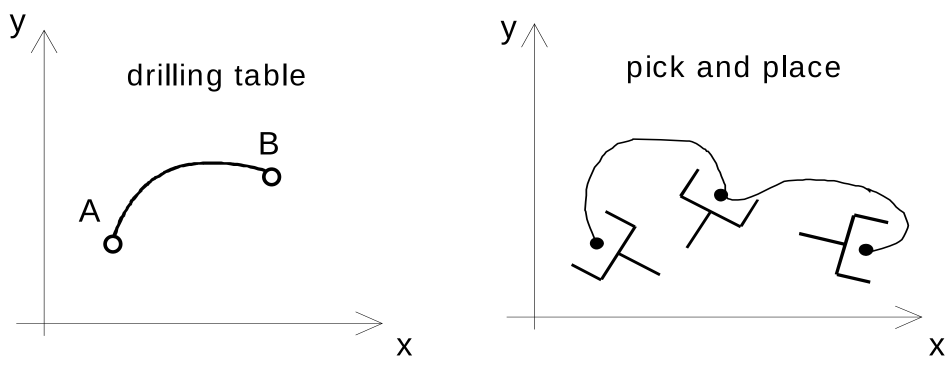


Figure 1-1: *PTP-motions in world coordinate system, left: path bewteen two drilling holes, right: robot end-effector and three configurations for pick & place*

Applications for PTP motions are spot welding, handling, pick and place, (partly) machine handling and machine transfer

CP (Continuous Path) motion

A motion type which specifies the whole path regarding position and orientation of the TCP. That means that the motions of each axes are mutually depending. These motions are more computationally expensive to execute at run time compared to PTP motions, since inverse kinematics must be solved for each time step (usually around 10 milliseconds in todays robot controller). Possible cartesian motion types include linear interpolation (specified by start and end point) and circular interpolation (specified by start point, via point and end point).
Figure 1-2 describes two examples

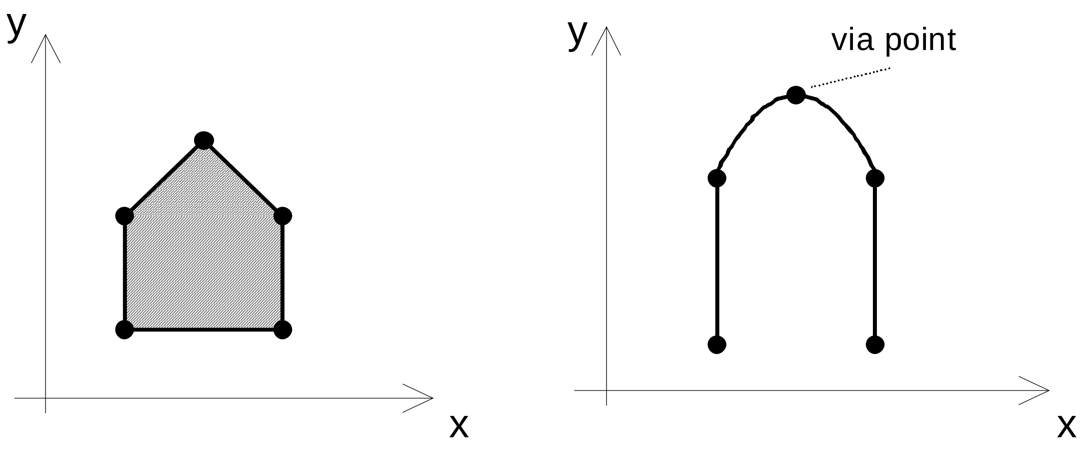


Figure 1-2: *example for interpolated motions between programmed positions, left: a sequence consisting of 5 linear motion, right: linear – circular – linear motion*

Applications for CP motions are arc welding, deburring, clueing, etc.

2. Spatial descriptions and transformations

The following section summarizes the main mathematical methods in order to describe positions and orientations with respect to a particular coordinate system.

2.1. Descriptions: positions, orientations and frames

To describe the position and orientation of the TCP, gripper, robot links, objects in the environment, etc. coordinate systems are used.

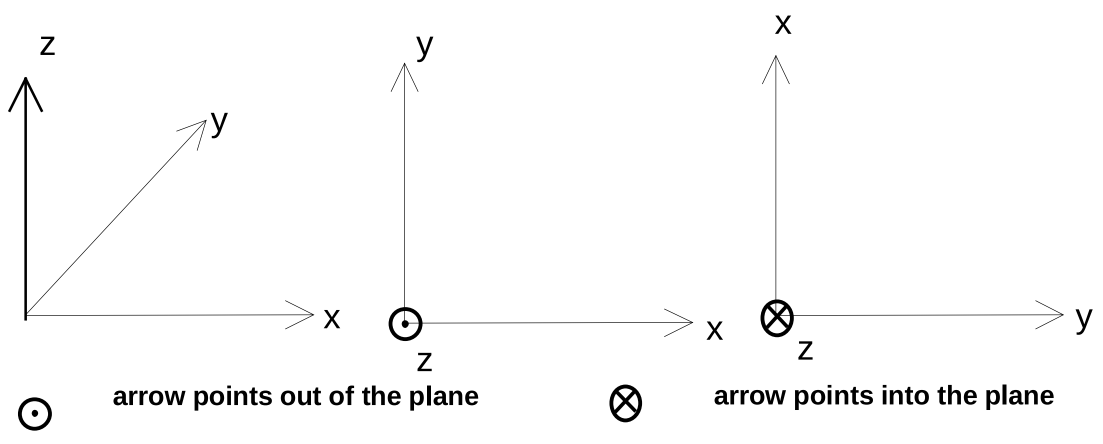


Figure 2-1: *Coordinate system*

For coordinate systems the following rule can be used:

Three-finger-rule of the right hand:
Thumb points in (positive) x-direction, pointing finger in (positive) y-direction, middle finger point then in (positive) z-direction.

Once a coordinate system is established we can locate any point in the world with 3x1 position vector. Sometimes more than one coordinate system are defined. So vectors must be tagged with information identifying which coordinate system they are defined within. In this lecture vectors are written with a leading subscript indicating the coordinate system to which they are referenced (unless it is clear from context, this subscript is omitted).

Geometrically a vector can be interpreted as an arrow describing its length and direction.

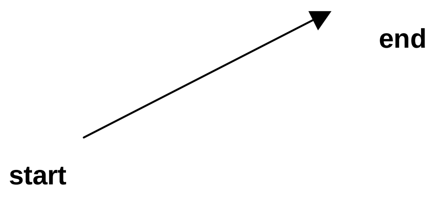


Figure 2-2: *geometrical description of a vector*

A vector can be translated without change (two vectors are identical, if they equal in length and direction). In the context of a coordinate system a (three-dimensional) vector is a three-tupel of numerical values indicating the distances along the axes of the coordinate system. Each of these distances along an axis can be thought of as a result of projecting the vector onto the corresponding axis.

$$\underline{\mathbf{v}} = \begin{pmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \end{pmatrix}$$

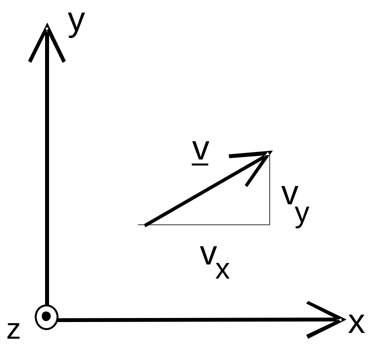


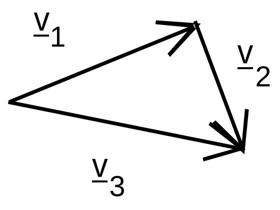
Figure 2-3: *description of a vector in a coordinate system*

<u>modulus (length) of a vectors:</u> $|\underline{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$ (2.1)

<u>(unit) direction of a vector:</u> $$\underline{e}_v = \frac{\underline{v}}{|\underline{v}|} = \begin{pmatrix} v_x / |\underline{v}| \\ v_y / |\underline{v}| \\ v_z / |\underline{v}| \end{pmatrix} \quad (2.2)$$

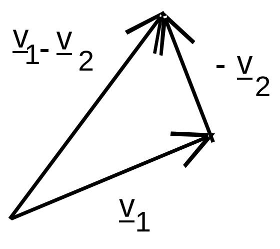
<u>addition of vectors:</u>

$$\underline{v}_3 = \underline{v}_1 + \underline{v}_2 = \begin{pmatrix} v_{1x} + v_{2x} \\ v_{1y} + v_{2y} \\ v_{1z} + v_{2z} \end{pmatrix} \quad (2.3)$$



<u>subtraction of vectors:</u>

$$\underline{v}_1 - \underline{v}_2 = \begin{pmatrix} v_{1x} - v_{2x} \\ v_{1y} - v_{2y} \\ v_{1z} - v_{2z} \end{pmatrix} \quad (2.4)$$



<u>cross product of two vectors:</u> $\underline{v}_3 = \underline{v}_1 \times \underline{v}_2$

$$\underline{v}_3 = \begin{pmatrix} v_{3x} \\ v_{3y} \\ v_{3z} \end{pmatrix} = \begin{pmatrix} v_{1y} \cdot v_{2z} - v_{1z} \cdot v_{2y} \\ v_{1z} \cdot v_{2x} - v_{1x} \cdot v_{2z} \\ v_{1x} \cdot v_{2y} - v_{1y} \cdot v_{2x} \end{pmatrix} \tag{2.5}$$

$\underline{v}_3$ is orthogonal to the plane, generated by $\underline{v}_1$ and $\underline{v}_2$. The three-finger-rule derives the direction of $\underline{v}_3$ (see Figure 2-4).

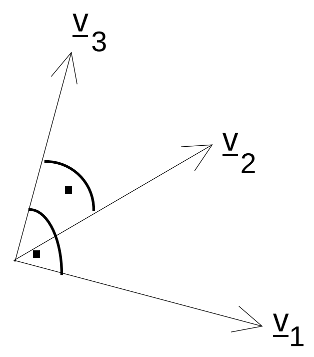


Figure 2-4: *cross product*

Dot product of two vectors:

The dot product of two vectors is defined by (2.6). The result is a scalar value which is related to the opening angle between the vectors. If one of the vectors is a unit vector (a vector with unit length) the result is the length of projection onto the unit vector. (see Figure 2-5).

$$\underline{a} \cdot \underline{b} = |\underline{a}| \cdot |\underline{b}| \cdot \cos\varphi \ , \quad \underline{a} \cdot \underline{b} = a_x \cdot b_x + a_y \cdot b_y + a_z \cdot b_z \tag{2.6}$$

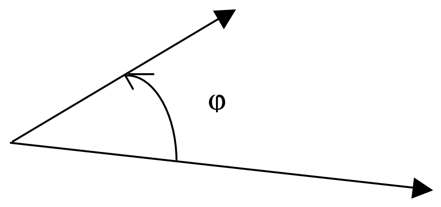


Figure 2-5: *dot product of two vectors*

Base vectors $\underline{x}_i$, $\underline{y}_i$, $\underline{z}_i$ of the coordinate system K_i . A cartesian coordinate system is fully described by ist base vectors $\underline{x}_i$, $\underline{y}_i$, $\underline{z}_i$, which are orthogonal to each other (Figure 2-6).

$$\underline{x}_i = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \underline{y}_i = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \underline{z}_i = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, |\underline{x}_i| = |\underline{y}_i| = |\underline{z}_i| = 1 \ (unit - vectors) \tag{2.7}$$

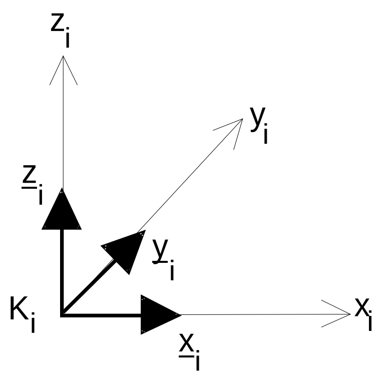


Figure 2-6: *base vectors of a coordinate system*

Rotation matrix:

Rotation matrices are used to describe the base vectors of a coordinate system relative to another reference system. A vector $\underline{a}$ can be expressed with such a rotation matrix in different coordinate systems.

${}^{I}\underline{a}$: vector $\underline{a}$, represented in the i-th coordinate system

Given the base vectors of two coordinate systems i and k:

$$ {}^{I}_{K}\underline{A} = \begin{pmatrix} {}^{I}\underline{x}_{K} & {}^{I}\underline{y}_{K} & {}^{I}\underline{z}_{K} \end{pmatrix} = \begin{pmatrix} \underline{x}_k \cdot \underline{x}_i & \underline{y}_k \cdot \underline{x}_i & \underline{z}_k \cdot \underline{x}_i \\ \underline{x}_k \cdot \underline{y}_i & \underline{y}_k \cdot \underline{y}_i & \underline{z}_k \cdot \underline{y}_i \\ \underline{x}_k \cdot \underline{z}_i & \underline{y}_k \cdot \underline{z}_i & \underline{z}_k \cdot \underline{z}_i \end{pmatrix} . \tag{2.8} $$

Each component of the (3x3) rotation matrix ${}^{I}_{K}\underline{A}$ can be written as the dot product of a pair of unit vectors. Note, that the components of any vector are simply the projections of that vector onto the unit direction of its reference frame. Since the dot product of two unit vectors yields the cosine of the angle between them, these components are oftens referred to as direction cosines.

As explained before and further inspection of (2.8) shows that the rows of the matrix are the unit vectors of system i expressed in system k. Hence, ${}^{K}_{I}\underline{A}$, the description of frame i relative to k is given by the transpose of (2.8). This suggests that the inverse of a rotation matrix is equal to ist transpose.

Exercise: proof ${}^{K}_{I}\underline{A}^{T}\,{}^{K}_{I}\underline{A} = I$

An arbitrary vector $\underline{a}$ represented in the coordinate system k can be expressed in the coordinate system i using the rotation matrix (and vice versa):

$$ {}^{I}\underline{a} = {}^{I}_{K}\underline{A} \cdot {}^{K}\underline{a} \quad or \quad {}^{K}\underline{a} = {}^{K}_{I}\underline{A} \cdot {}^{I}\underline{a} \quad with \quad {}^{K}_{I}\underline{A} = {}^{I}_{K}\underline{A}^{T} \tag{2.9} $$

Properties of rotations:
- det(A) = 1

- columns are mutually orthogonal
- possible to describe rotations with fewer than 9 numbers
- result from linear algebra: Cayley's formula for orthogonal matrices

$$R = (I_3 - S)^{-1}(I_3 + S), \text{ S is scew symmetric} (S = -S^T), S = \begin{pmatrix} 0 & -s_z & s_y \\ s_z & 0 & -s_x \\ -s_y & s_x & 0 \end{pmatrix}$$

- three parameters to describe rotation

- same result by investigation of R
- six constraints and nine matrix elements

  $|x| = |y| = |z| = 1$

  $x^T y = y^T z = x^T z = 0$

- rotations do not generally commute: $R_1 R_2 \neq R_2 R_1$

Exercise:

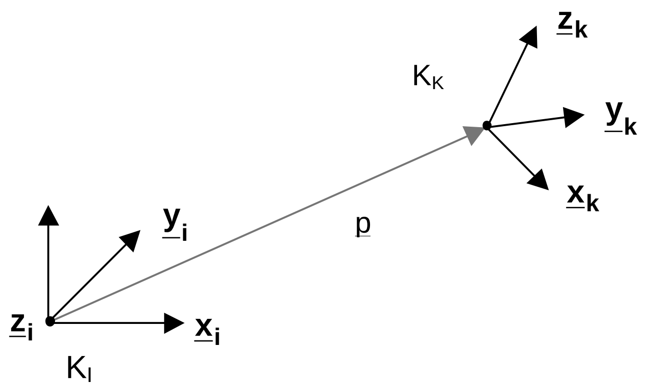
Calculate $Rot_X(\frac{\pi}{2}) Rot_Z(\frac{\pi}{2})$ and $Rot_Z(\frac{\pi}{2}) Rot_X(\frac{\pi}{2})$ and solve both compound transformations geometrically by drawing a frame diagram.

<u>Homogeneous coordinates (frames)</u>

4x4 homogeneous transforms are used to cast the rotation and translation of a general transform into a single matrix form. In other fields of study it can be used to compute perspective and scaling operations. Hence, they describe the spatial relationship between two coordinate systems (see Figure 2-7).

$$\begin{pmatrix} {}^I\underline{a} \\ 1 \end{pmatrix} = \begin{pmatrix} {}^I_K\underline{A} & {}^I\underline{p} \\ 0^T & 1 \end{pmatrix} \cdot \begin{pmatrix} {}^K\underline{a} \\ 1 \end{pmatrix}$$

- 1 added as the last element of the 4x1 vector
- row (0 0 0 1) is added as last row of 4x4 matrix



$${}^I\underline{x}_K,\ {}^I\underline{y}_K,\ {}^I\underline{z}_K,\ {}^I\underline{p}$$

Figure 2-7: *spatial relationship between two coordinate systems*

The homogeneous transform ${}^{I}_{K}\underline{D}$ can be viewed as the representation of the vectors $\underline{x}_k$, $\underline{y}_k$, $\underline{z}_k$, $\underline{p}$ in the i-th coordinate system:

$$ {}^{I}_{K}\underline{D} = \begin{pmatrix} {}^{I}\underline{x}_K & {}^{I}\underline{y}_K & {}^{I}\underline{z}_K & {}^{I}\underline{p}_K \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (2.10) $$

Homogeneous transforms of several coordinate systems (compound transformations):

Combining two transformations is performed by multiplication of two homogenous transforms (see Figure 2-8):

$$ {}^{I}_{I+2}\underline{D} = {}^{I}_{I+1}\underline{D} \cdot {}^{I+1}_{I+2}\underline{D} = \begin{pmatrix} {}^{I}\underline{x}_{I+2} & {}^{I}\underline{y}_{I+2} & {}^{I}\underline{z}_{I+2} & {}^{I}\underline{p}_{I+2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (2.11) $$

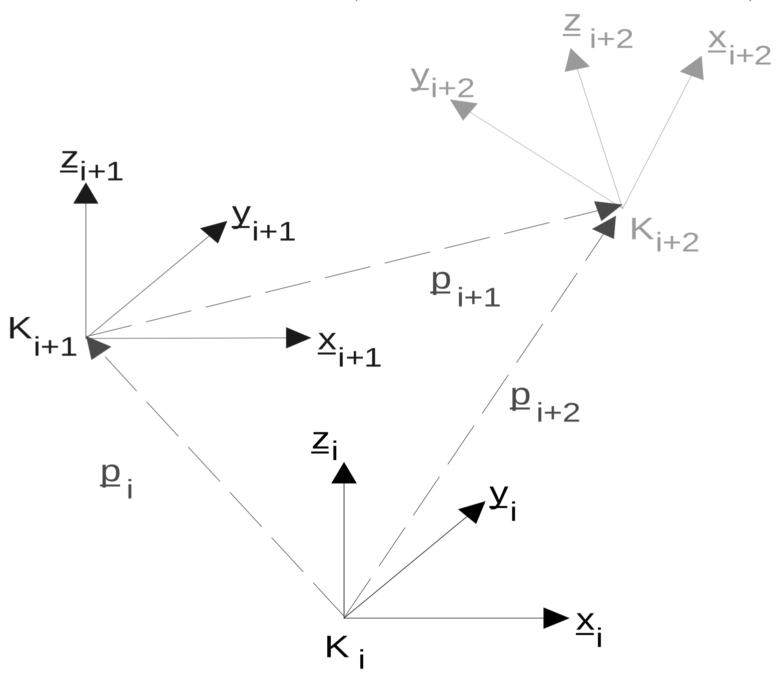


Figure 2-8: *Compound transformations*

Exercise:

- Describe rotational and translational part of ${}^{I}_{I+2}\underline{D}$ in terms of the rotational and translational parts of ${}^{I}_{I+1}\underline{D}$ and ${}^{I+1}_{I+2}\underline{D}$.
- Invert a homogenous transform $D = \begin{pmatrix} R & p \\ 0^T & 1 \end{pmatrix}$

Interpretations:

- A homogeneous transform is a description of a frame. ${}^{I}_{K}\underline{D}$ describes the frame K relative to the frame I. Specifically, the columns of ${}^{I}_{K}\underline{R}$ are unit vectors defining the principal axes of frame K and p locates the position of the origin of K relativ to frame I.

- A homogeneous transform is a mapping. ${}^{I}_{K}\underline{D}$ maps ${}^{K}P \mapsto {}^{I}P$.

2.2. Considerations

Consideration of robots only with open kinematic chain

Example:

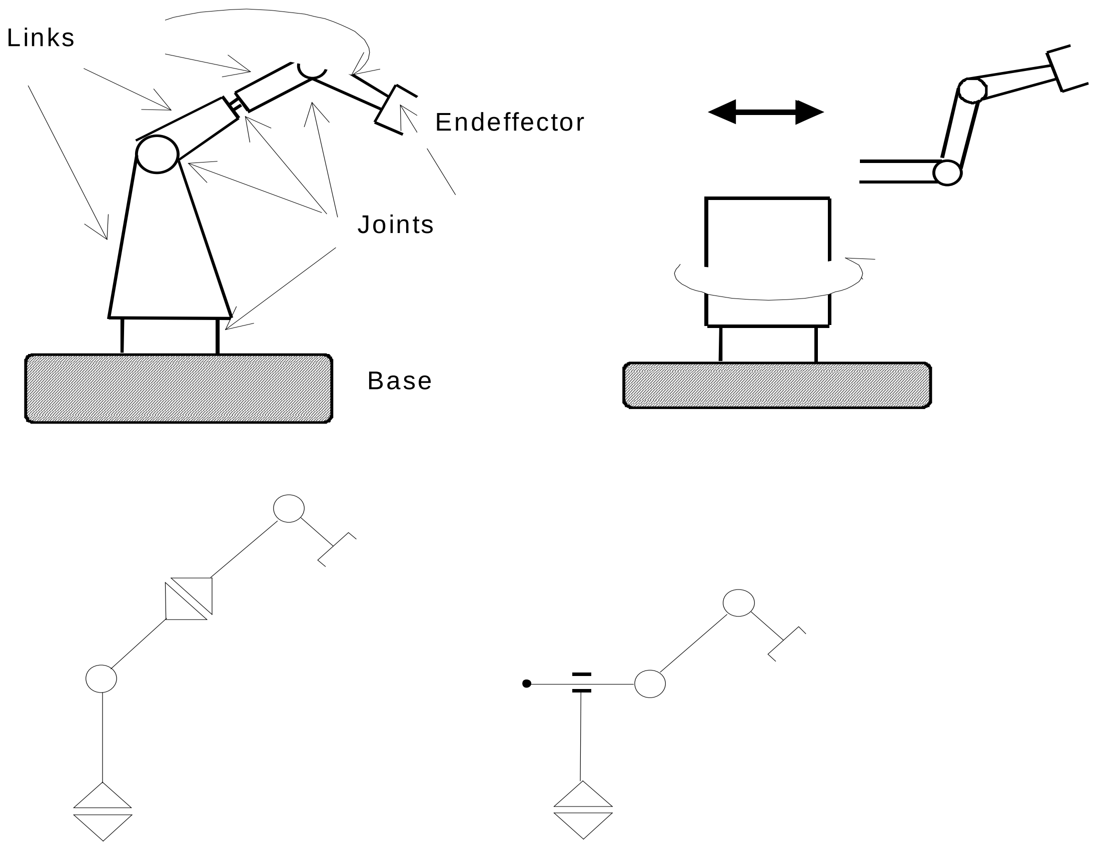


Figure 2-1: *possible design of robots and their schematic sketch*

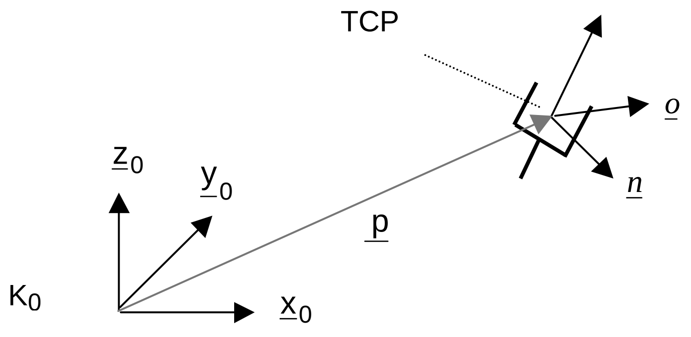


Figure 2-2: *Description of the position and orientation of the tool center point (TCP) with respect to the base (world) frame K_0*

Attached to the TCP is the tool frame described by

Translation $\underline{p}$ with respect to K_0	(Position)
$\underline{n}$, $\underline{o}$, $\underline{a}$ with respect to K_0	(Orientation)

Rotation matrix representing the orientation:

$\left({}^0\underline{n},\ {}^0\underline{o},\ {}^0\underline{a}\right)$: **n**ormal, **o**rthogonal, **a**pproach

homogeneous transformation integrating position and orientation with respect to base frame
K_0:

$$\underline{T} = {}^0_N\underline{D} = \begin{pmatrix} {}^0\underline{n} & {}^0\underline{o} & {}^0\underline{a} & {}^0\underline{p} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (2.12)$$

Index n denotes the degree of freedom of the robot

2.3. Orientation defined by Euler-Angles

Avoid redundant information by introduction of angles

translation $\underline{p}$ with respect to K_0	(Position)
three angles a, b, c	(Orientierung)

Fixed angles: rotation takes place around fixed frame
Euler angles: rotation takes place around moving frame

Definition: X-Y-Z fixed angles

a:	rotation around the $\underline{z}_0$-axis
b:	rotation around the $\underline{y}_0$-axis
c:	rotation around the $\underline{x}_0$-axis

Each of the three rotations takes place about an axis of the fixed reference frame.

$$R_{XYZ}(c,b,a) = R_Z(a)R_Y(b)R_X(c) = \begin{pmatrix} ca & -sa & 0 \\ sa & ca & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} cb & 0 & sb \\ 0 & 1 & 0 \\ -sb & 0 & cb \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & cc & -sc \\ 0 & sc & cc \end{pmatrix}$$

$$= \begin{pmatrix} cacb & casbsc - sacc & casbcc + sasc \\ sacb & sasbsc + cacc & sasbcc - casc \\ -sb & cbsc & cbcc \end{pmatrix} \text{with } sa = \sin a, ca = \cos a, ... \qquad (2.13)$$

Inverse problem is of interest.

$$R_{XYZ}(c,b,a) = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

Exercise: Calculate a,b,c depending on matrix values. What happens, if B=+/- $\pi/2$

Definition A: (Z-Y-Z Euler angles)

α:	rotation around the $\underline{z}_0$-axis
β:	rotation around the (new) $\underline{y}$-axis
γ:	rotation around the (new) $\underline{z}$-axis

The following relation holds between α, β, γ and $\underline{n}$, $\underline{o}$, $\underline{a}$:

$$\underline{n} = \begin{pmatrix} \cos\alpha\cdot\cos\beta\cdot\cos\gamma - \sin\alpha\cdot\sin\gamma \\ \sin\alpha\cdot\cos\beta\cdot\cos\gamma + \cos\alpha\cdot\sin\gamma \\ -\sin\beta\cdot\cos\gamma \end{pmatrix} \quad \underline{o} = \begin{pmatrix} -\cos\alpha\cdot\cos\beta\cdot\sin\gamma - \sin\alpha\cdot\cos\gamma \\ -\sin\alpha\cdot\cos\beta\cdot\sin\gamma + \cos\alpha\cdot\cos\gamma \\ \sin\beta\cdot\sin\gamma \end{pmatrix} \quad \underline{a} = \begin{pmatrix} \cos\alpha\cdot\sin\beta \\ \sin\alpha\cdot\sin\beta \\ \cos\beta \end{pmatrix}$$

$$\beta = \arccos(a_z) \quad with: 0 < \beta < \pi$$
$$\alpha = \arctan2(a_y, a_x)$$
$$\gamma = \arctan2(o_z, -n_z)$$

(2.14)

if β=0: $\alpha + \gamma = \arctan2(n_y, n_x)$

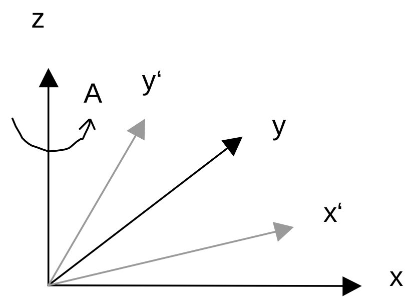
Arctan calculates values between -π/2 and π/2
Arctan2 calculates values between -π and π

$$\theta = \arctan2(y,x) = \begin{cases} \theta = \arctan(\frac{y}{x}),\ 0 \le \theta \le \pi/2,\ \text{für } +x,+y \\ \theta = \pi + \arctan(\frac{y}{x}),\ \pi/2 \le \theta \le \pi,\ \text{für } -x,+y \\ \theta = -\pi + \arctan(\frac{y}{x}),\ -\pi \le \theta \le -\pi/2,\ \text{für } -x,-y \\ \theta = \arctan(\frac{y}{x}),\ -\pi/2 \le \theta \le 0,\ \text{für } +x,-y \end{cases}$$

(2.15)

Definition B (Z-Y-X Euler-Angles)

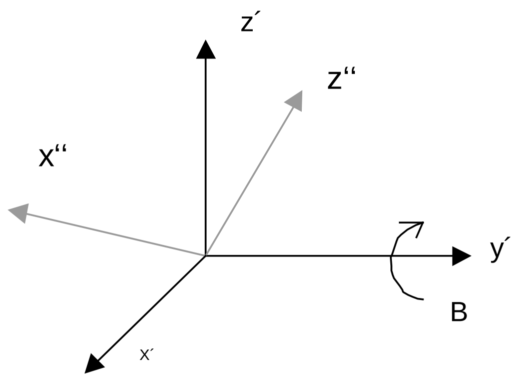
Angle A rotates around the z-axis:



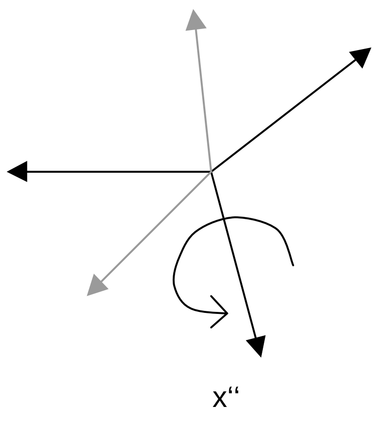
$$ {}^{'}_{0}\underline{A} = \begin{pmatrix} Cos\,A & -Sin\,A & 0 \\ Sin\,A & Cos\,A & 0 \\ 0 & 0 & 1 \end{pmatrix} $$

Angle B rotates around the new y-axis

$$ {}^{''}_{'}\underline{A} = \begin{pmatrix} Cos\,B & 0 & Sin\,B \\ 0 & 1 & 0 \\ -Sin\,B & 0 & Cos\,B \end{pmatrix} $$



Angle C rotates around the new x-axis (x“):



$$ {}^{*}_{''}\underline{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & Cos\,C & -Sin\,C \\ 0 & Sin\,C & Cos\,C \end{pmatrix} $$

$${}^{*}\underline{A} = {}^{,}\underline{A} \cdot {}^{''}_{,}\underline{A} \cdot {}^{*}_{''}\underline{A} = \begin{pmatrix} C_A \cdot C_B & -S_A \cdot C_C + C_A \cdot S_B \cdot S_C & S_A \cdot S_C + C_A \cdot S_B \cdot S_C \\ S_A \cdot C_B & C_A \cdot C_C + S_A \cdot S_B \cdot S_C & -C_A \cdot S_C + S_A \cdot S_B \cdot C_C \\ -S_B & C_B \cdot S_C & C_B \cdot C_C \end{pmatrix} \tag{2.16}$$

with $C_A = Cos\,A,\ S_A = Sin\,A \quad etc$

relationships:

$$\underline{n} = \begin{pmatrix} C_A \cdot C_B \\ S_A \cdot C_B \\ S_B \end{pmatrix} \quad \underline{o} = \begin{pmatrix} -S_A \cdot C_C - C_A \cdot S_B \cdot S_C \\ C_A \cdot C_C - S_A \cdot S_B \cdot S_C \\ C_B \cdot S_C \end{pmatrix} \quad \underline{a} = \begin{pmatrix} S_A \cdot S_C - C_A \cdot S_B \cdot S_C \\ -C_A \cdot S_C - S_A \cdot S_B \cdot C_C \\ C_B \cdot C_C \end{pmatrix}$$

A=arctan2(ℓ_y, ℓ_x)
B=arcsin(ℓ_z)
(2.17)
C=arctan2(m_z, n_z)

Definition: Equivalent angle-axis

Rotation about the vector K by an angle a according to the right hand rule

$$R_K(a) = \begin{pmatrix} k_x k_x va + ca & k_x k_y va - k_z sa & k_x k_z va + k_y sa \\ k_x k_y va + k_z sa & k_y k_y va + ca & k_y k_z va - k_x sa \\ k_x k_z va + k_y sa & k_y k_z va + k_x sa & k_z k_z va + ca \end{pmatrix} \text{with } va = 1 - \cos a \tag{2.18}$$

3. Robot kinematics

Definitions:

- A robot may be thought of as a set of bodies connected in a chain by joints.
- These bodies are called links.
- Joints form a connection between a neighboring pair of links

Properties:

- Normally robots consist of joints with one degree of freedom (1 DOF).
- Revolute/prismatic joints
- n-DOF joints can be modeled as n joints with 1 DOF connected with n-1 links of zero length
- Positioning a robot in 3-space a minimum of six joints is required
- Typical robot consist of 6 joints
- Joint axis are defined by lines in space
- A link can be specified by two numbers which define the relative location of the two joint axes in space: link length and link twist
- Link length: measured along the line which is mutually perpendicular to both axes
- Link twist: measured in the plane defined by the perpendicular axis

3.1. Link connection description by Denavit-Hartenberg notation

Neighboring links have a common joint axis between them

Parameters:

- Distance along common axis from one link to the other (link offset)
- Amount of rotation about this common axis between one link and its neighbour (joint angle)

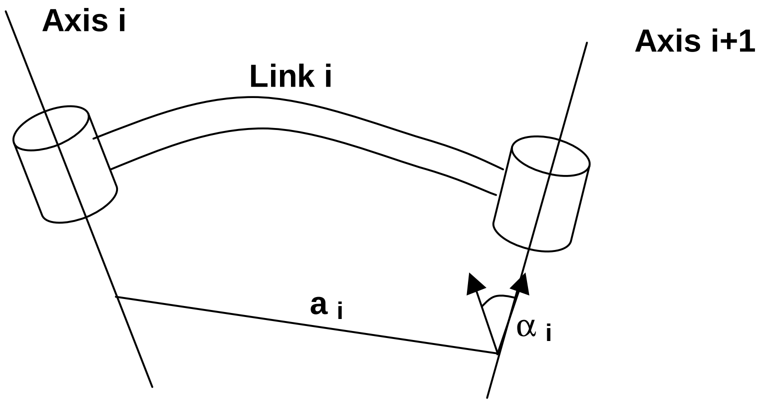


Figure 3-1: *The length a and twist α of a link*

A serial link robot consists of a sequence of links connected together by actuated joints. For an n DOF robot, there will be n joints and n links. The base of the robot is link 0 and is not considered one of the (n=6) links. Link 1 is connected to the base link by joint 1. There is no joint at the end of the final link (TCP). The only significance of links is that they maintain a fixed relationship between the robot joints at each end of the link. Any link can be characterized by two dimensions: the common normal distance a_i (called link length) and the angle α_i (called link twist) between the axes in a plane perpendicular to a_i (see Figure 3-1). Generally, two links are connected at each joint axis (see Figure 3-2).

The axis will have two normals to it, one for each link. The relative position of two such connected links is given by d_i, the distance between the normals along the joint i axis, and θ_i the angle between the normals measured in a plane normal to the axis. d_i and θ_i are called the distance and the angle between the links, respectively.
In order to describe the relationship between links, we will assign coordinate systems (frames) to each link. We will first consider revolute joints in which θ_i is the joint variable. The origin of the frame of link i is set to be at the intersection of the common normal between the axes of joints i and i+1 and the axis of joint i+1. In case of intersecting joint axes, the origin is at the point of intersection of the joint axes. If the axes are parallel, the origin is chosen to make the joint distance zero for the next link whose coordinate origin is defined. The z axis for link i will be aligned with the axis of joint i+1. The x axis will be aligned with any common normal which exists and is directed along the normal from joint i to joint i+1. In case of intersecting joints, the direction of the x axis is parallel or antiparallel to the vector cross product z_{i-1} x z_i. Notice that this condition is also satisfied for the x axis directed along the normal between joints i and i+1. θ_i is zero for the i-th revolute joint when x_{i-1} and x_i are parallel and have the same direction.
In case of prismatic joint, the distance d_i is the joint variable. The direction of the joint axis is the direction in which the joint moves. The direction of the axis is defined but, unlike a revolute joint, the position in space is not defined. In the case of a prismatic joint, the length a_i has no meaning and is set to zero. The origin of the frame for a prismatic joint is coincident with the next defined link origin. The z axis of the prismatic joint is aligned with the axis of joint i+1. The x_i axis is parallel or antiparallel to the vector cross product of the direction of the prismatic joint and z_i. For a prismatic joint we will define the zero position when $d_i = 0$.
With the robot in its zero position, the positive sense of rotation for revolute joints or displacement for prismatic joints can be decided and the sense of the direction of the z axis determined. The origin of the base link (zero) will be coincident with the origin of link 1. If it is desired to define a different reference frame, then the relationship between the reference and base frames can be described by a fixed homogeneous transformation. At the end of the robot, the final displacement d_6 or rotation θ_6 occurs with respect to z_5. The origin of the frame for link 6 is chosen to be coincident with that of the link 5 frame. If a tool (or end effector) is used whose origin and axes do not coincide with the frame of link 6, the tool can be related by a fixed homogeneous transformation to link 6.
Having assigned frames to all links according to the preceding scheme, we can establish the relationship between successive frames i-1,i by the following rotations and translations:

- θ_i = the angle between X_{i-1} and X_i measured about Z_{i-1}
- d_i = the distance from X_{i-1} to X_i measured along Z_{i-1}
- a_i = the distance from Z_{i-1} to Z_i measured along X_i
- α_i = the angle between Z_{i-1} and Z_i measured about X_i

Due to the authors of this method attaching frames to links, these four parameters are called the Denavit Hartenberg parameters (DH parameters).

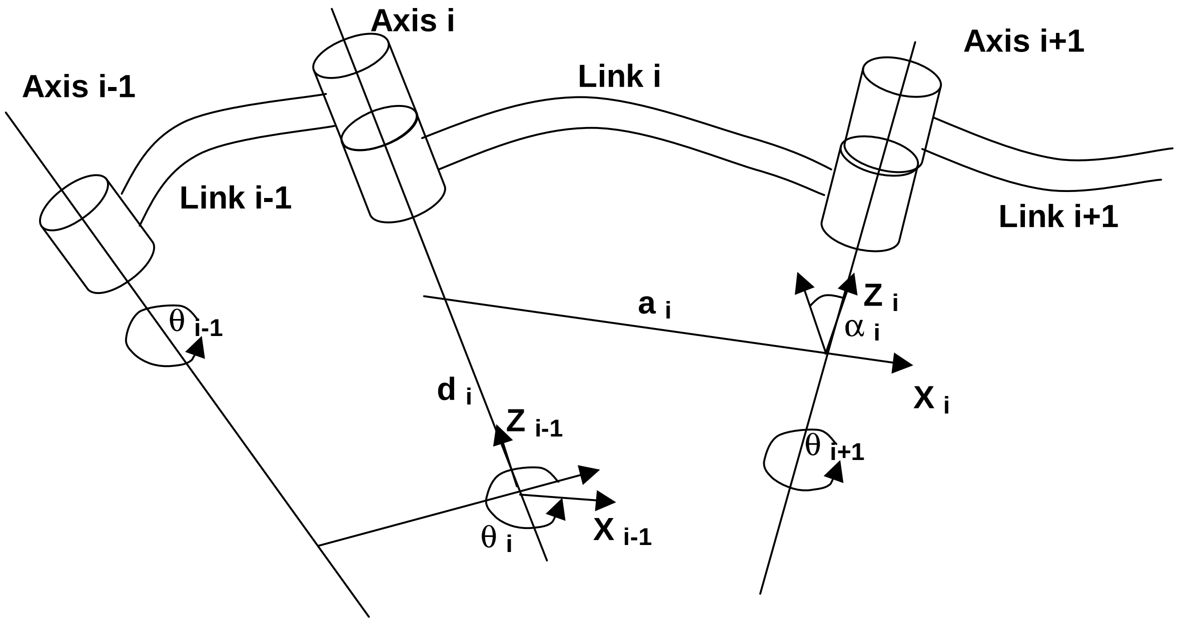


Figure 3-2: *Link parameters θ, d, a and α*

The transformation from one link frame to the next can now be defined by homogeneous transformations using this notation:

$$ {}^{I-1}_{I}\underline{D} = Rot_{Z_{i-1}}(\theta_i) Trans_{Z_{i-1}}(d_i) Trans_{X_i}(a_i) Rot_{X_i}(\alpha_i) $$
$$ = \begin{pmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3.1) $$

The homogeneous matrices ${}^{I-1}_{I}\underline{D}$ are called Denavit-Hartenberg matrices.

Concenating link transformations to calculate the single transformation that relates frame N (usually the TCP frame) to frame 0 (usually the base frame):

$$ \underline{T} = {}^{0}_{N}\underline{D} = {}^{0}_{1}\underline{D} \cdot {}^{1}_{2}\underline{D} \cdots {}^{N-2}_{N-1}\underline{D} \cdot {}^{N-1}_{N}\underline{D} \quad (3.2) $$

n defines the number of joints.

In case of rotational joints the components of the DH-matrices ${}^{I-1}_{I}\underline{D}$ only depend on the joint angles θ_i , since the parameters a_i , d_i , α_i are constants. For that case the transformation

$$ \underline{T} = \underline{T}(\theta_1, \theta_2, \cdots, \theta_n) = \underline{T}(\underline{\theta}) $$

is a function of all joint angles. The joint angles can be formed to a vector: $\underline{\theta} = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{pmatrix}$.

Examples2-dof robot with parallel axes

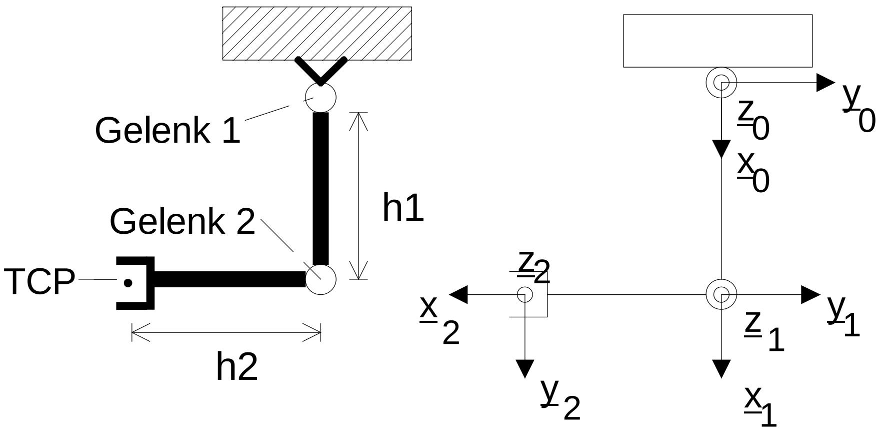


Figure 3-3: *Planar 2-dof robot including coordinate system*

The following table shows the set of DH parameters: $q_1$and q_2 are variables (joint angles)

Joint	θ_i **/ degree**	d_i	A_i	α_i **/ degree**
1	0	0	h1	0
2	-90	0	h2	0

The DH-transformations are:

$${}^0_1\underline{D} = \begin{pmatrix} \cos q_1 & -\sin q_1 & 0 & h_1 \cdot \cos q_1 \\ \sin q_1 & \cos q_1 & 0 & h_1 \cdot \sin q_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad {}^1_2\underline{D} = \begin{pmatrix} -\sin q_2 & -\cos q_2 & 0 & -h_2 \cdot \sin q_2 \\ \cos q_2 & -\sin q_2 & 0 & h_2 \cdot \cos q_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$${}^0_2\underline{D} = \underline{T}(\underline{q}) = \begin{pmatrix} -C_1 S2 - S1C2 & -C_1 C2 + S1S2 & 0 & h2(-C_1 S2 - S1C2) + h1C1 \\ -S_1 S2 + C1C2 & -S_1 C2 - C1S2 & 0 & h2(-S_1 S2 + C1C2) + h1S1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{matrix} C_i = \cos q_i \\ S_i = \sin q_i \end{matrix}$$

2-dof robot with perpendicular axes

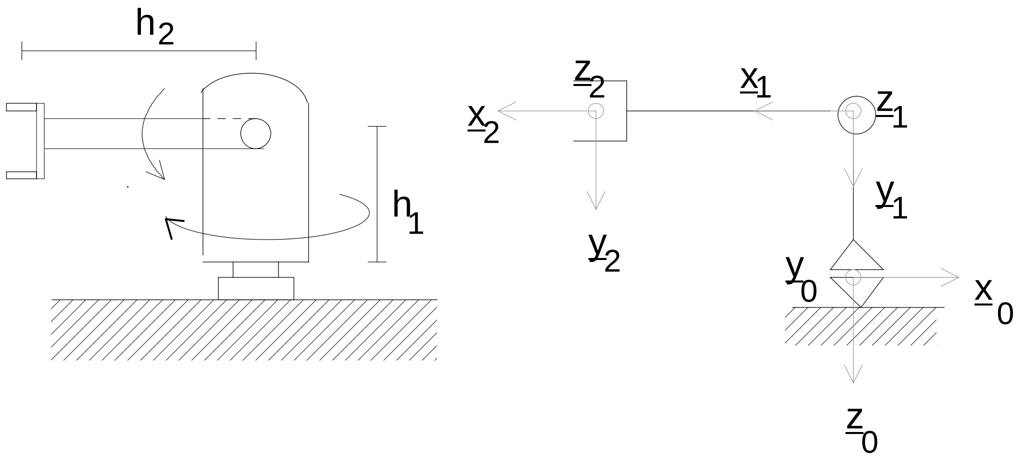


Figure 3-4: *2-dof robot with perpendicular axes*

DH parameter:

joint	q_i / degree	D_i	a_i	α_i / degree
1	180	-h1	0	90
2	0	0	h2	0

q_1, q_2 are variables, a_1, a_2, d_1, d_2, α_1, α_2 are constants

3-dof robot with 2 rotational joints and 1 prismatic joint

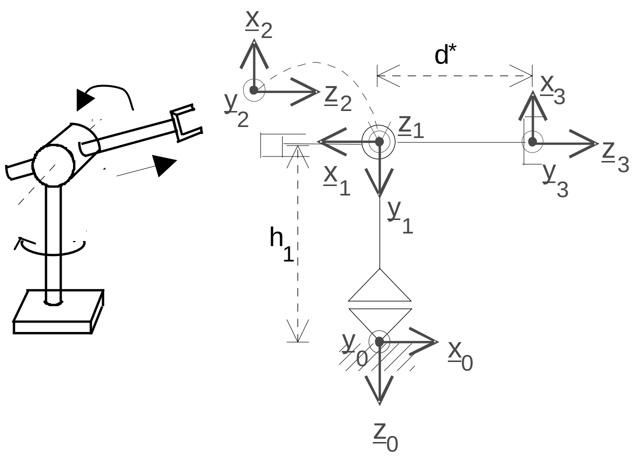


Figure 3-5: *3-dof robot with 2 rotational joints and 1 prismatic joint*

DH parameter

joint	q_i / degree	D_i	a_i	α_ι / degree
1	180	-h1	0	90
2	90	0	0	90
3	0	d*	0	0

q_1, q_2 ,d_3 : variables, a_1, a_2,a_3 d_1, d_2, α_1, α_2, α_3 constants

6-dof robot with 6 rotational joints (frequently used kinematic structure)

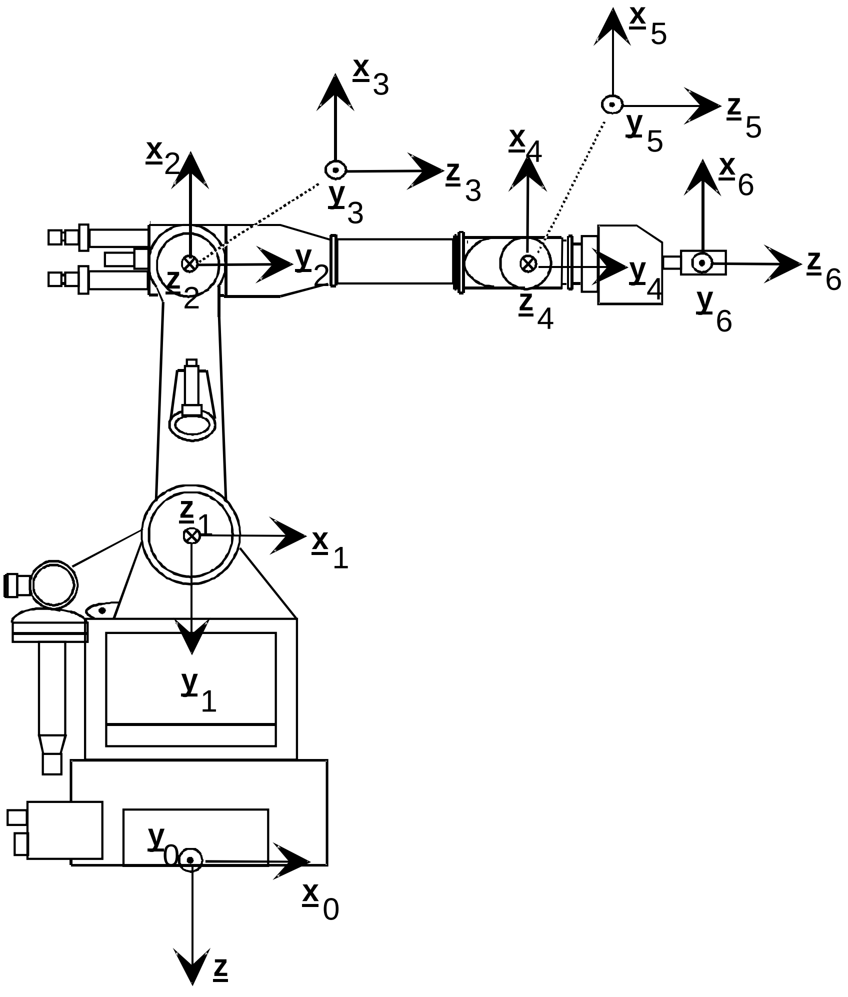


Figure 3-6: *6-dof robot with 6 rotational joints (frequently used kinematic structure)*

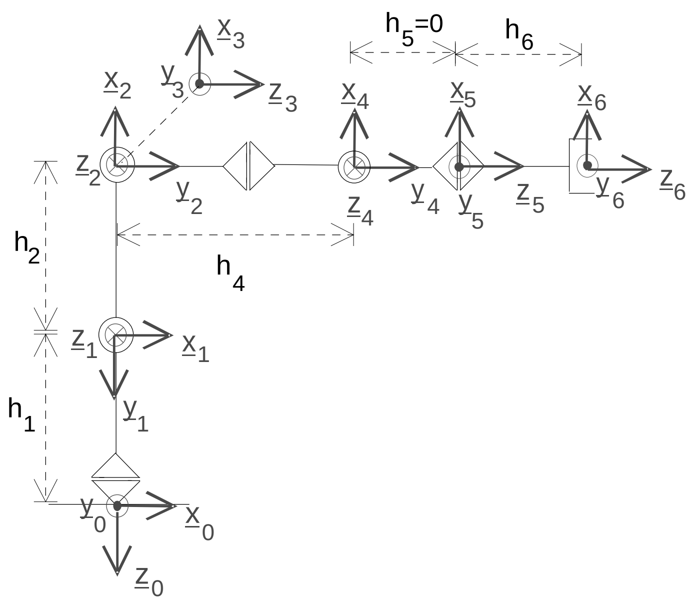


Figure 3-7: *sketch of 6-dof robot*

DH parameter:

Joint	**q_i / degree**	**d_i**	**A_i**	**α_i / degree**
1	0	-h1(-0.78m)	0	90
2	-90	0	h2(0.8m)	0
3	0	0	0	-90
4	0	h4 (0.8m)	0	90
5	0	0	0	-90
6	0	h6 (0.334m)	0	0

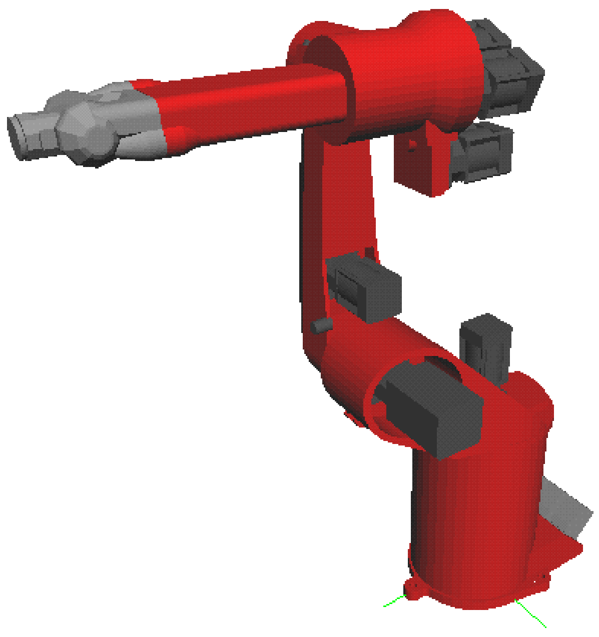


Figure 3-8: *Reis robot RV6 (payload 6 kg)*

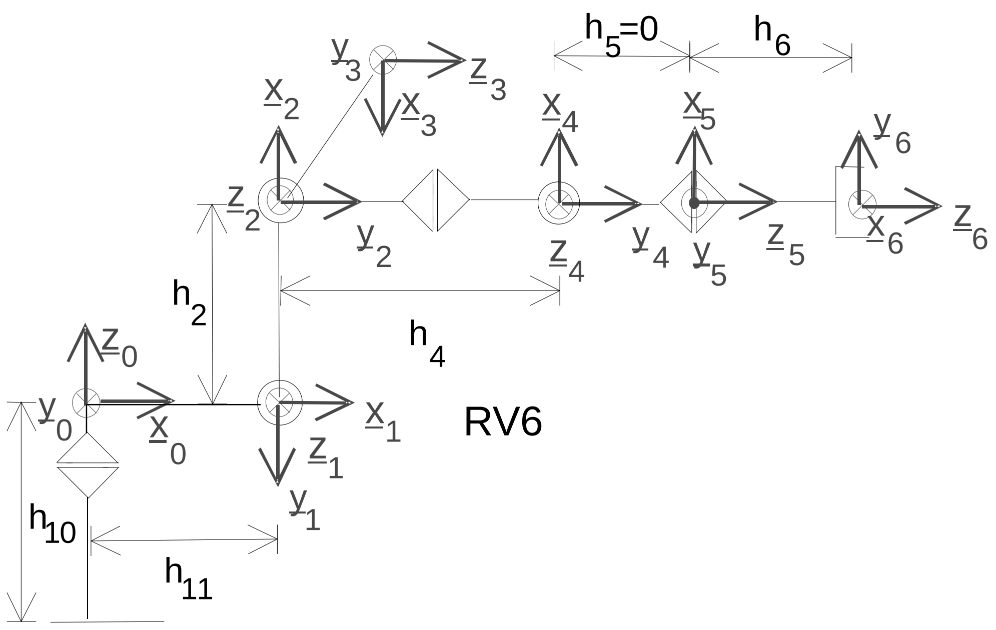


Figure 3-9: Reis robot *RV6, sketch*

DH parameter RV6:

Joint	q_i / rad	d_i	A_i	α_i / rad
1	0	0	h11	-π/2
2	-π/2	0	h2	0
3	π	0	0	π/2
4	π	h4	0	π/2
5	0	0	0	-π/2
6	-π/2	h6	0	0

h11=0.28m
h2=0.615m
h4=0.54m
h6= depending on tool

3.2. Forward kinematics

Given the joint values of the robot with n degrees of freedom, the homogenous matrix defining the position and orientation of the TCP is:

$$\underline{T}(\underline{q}) = \begin{pmatrix} \underline{n}(\underline{q}) & \underline{o}(\underline{q}) & \underline{a}(\underline{q}) & \underline{p}(\underline{q}) \\ 0 & 0 & 0 & 1 \end{pmatrix} = {}^0_1\underline{D} \cdot {}^1_2\underline{D} \cdot \ldots \cdot {}^{N-2}_{N-1}\underline{D} \cdot {}^{N-1}_{N}\underline{D}$$

The vectors p, n, o, a are described in the base (world) frame. This mapping from robot coordinates to world coordinates is unique.

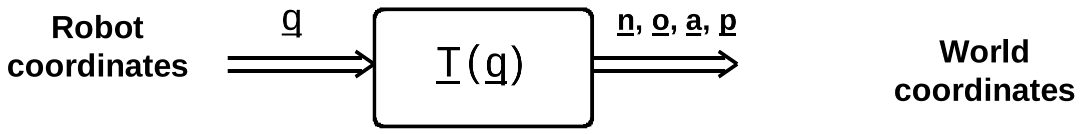


Figure 3-1: *forward kinematics: mapping from robot coordinates to world coordinates*

3.3. Inverse kinematics

Given the position and orientationg of the TCP with respect to the base frame calculate the joint coordinates of the robot which correspond it.

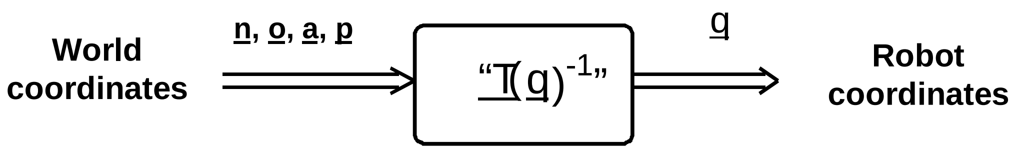


Figure 3-1: *inverse kinematics: mapping from world coordinates to robot coordinates*

In the equation $\begin{pmatrix} {}^0\underline{n} & {}^0\underline{o} & {}^0\underline{a} & {}^0\underline{p} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \underline{T}(\underline{q})$ the left side is given.

Problem:
Given T(q) as 16 numerical values (four of which are trivial), solve (2.24) for the 6 joint angles $q_1,..,q_6$.
Among the 9 equations arising from the rotation matrix portion of T, only three equations are independant. These added with 3 equations from the position vector portion of T give 6 equations with 6 unknowns. These equations are nonlinear, transcendental equations which can be quite difficult to solve.

Problems related to inverse kinematics:

- Existence of solutions
- Multiple solutions
- Method of solution

Existence of solutions

The existence of solutions is closely linked to the workspace of the robot which is the volume of space which the TCP of the robot can reach. For the solution to exist, the specified goal point must be in the workspace.

Dexterous workspace: volume of space which the robot's TCP can reach with all orientations
Reachable workspace: volume of space which the robot's TCP can reach with at least one orientation

Examples:

2 DOF robot with $l_1 = l_2$

Dexterous workspace:	the origin
Reachable workspace:	disc of radius $l_1 + l_2$

2 DOF robot with $l_1 \neq l_2$

Dexterous workspace:	empty
Reachable workspace:	ring of outer radius $l_1 + l_2$ and inner radius $\lvert l_1 - l_2 \rvert$

Since the 6 parameters are necessary to describe an arbitrary position and orientation of the TCP, the degree of freedom of the robot must be at least 6, in order a dexterous workspace exists at all.

Multiple solutions:

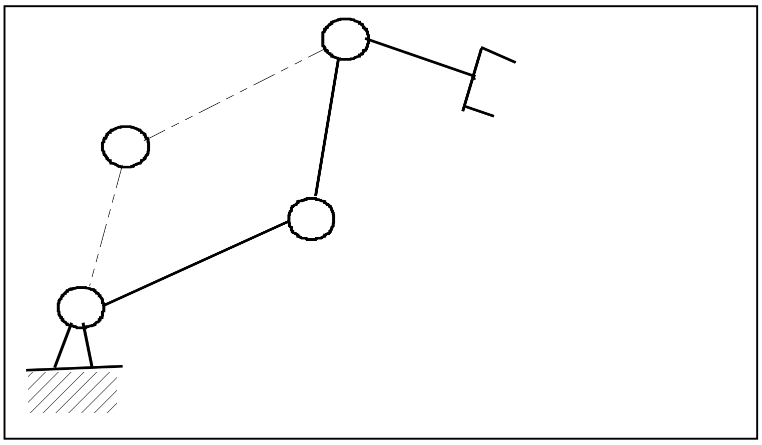


Figure 3-2: *Example for multiple solutions for a 2 DOF robot*

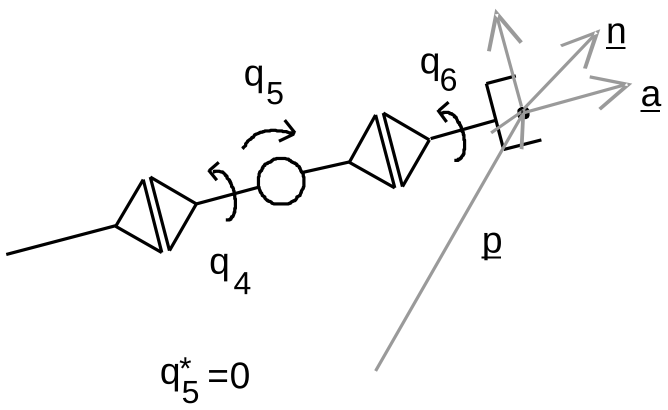


Figure 3-3: *example for infinite solutions*

Suppose in example of

Figure 3-3, that the given vectors p and n are realized by the joint angles $q_1^*,\ q_2^*,\ q_3^*,\ q_5^* = 0$. The orientation can be realized by an infinite numbrer of pairs q_4 and q_6 (normally by $q_4+q_6 = 0$). In practical implementations of the inverse kinematics an additional condition will generate the solution, like

$$c_1(q_{4,i-1} - q_{4,i})^2 + c_2(q_{6,i-1} - q_{6,i})^2 \overset{!}{=} MIN$$

This sum minimizes the travel range of q_4 and q_6 .

Method of solution:

There are two classes:

- Closed form solutions (analytical, geometrical)
- Numerical solutions

We concentrate in this lecture on closed form solutions.

Example : planar 2-dof robot:

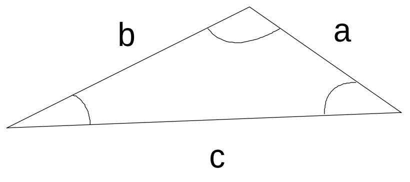
Given $p_x^{(0)}$ *and* $p_y^{(0)}$ as cartesian position. With

$$p_x^{(0)} = h_2(\cos q_1 \cos q_2 - \sin q_1 \sin q_2) + h_1 \cos q_1$$
$$p_y^{(0)} = h_2(\sin q_1 \cos q_2 + \cos q_1 \sin q_2) + h_1 \sin q_1$$

the joint values q_1 and q_2 can be calculated. It depends on the kinematic structure whether an analytical solution is possible.

Geometrical solution:

Law of cosine:



$$a^2 = b^2 + c^2 - 2 \cdot b \cdot c \cdot \cos\alpha$$

Example: planar 2-dof robot (Figure 3-4)

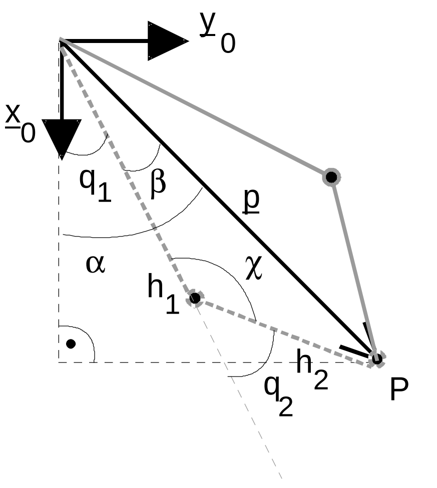


Figure 3-4: *Geometrical solution of 2 dof robot*

Relationships:

$$\alpha = \arctan 2(p_y^{(0)}, p_x^{(0)})$$

For β the cosine rule applies:

$$h_2^2 = h_1^2 + |\underline{p}|^2 - 2 \cdot h_1 \cdot |\underline{p}| \cdot \cos\beta, \quad \beta = \arccos\left(\frac{h_1^2 + |\underline{p}|^2 - h_2^2}{2 \cdot h_1 \cdot |\underline{p}|}\right)$$

Therefore $\boxed{q_1 = \alpha - \beta}$

For χ the cosine rule applies:

$$|\underline{p}|^2 = h_2^2 + h_1^2 - 2 \cdot h_1 \cdot h_2 \cdot \cos\chi, \quad \chi = \arccos\left(\frac{h_1^2 + h_2^2 - |\underline{p}|^2}{2 \cdot h_1 \cdot h_2}\right)$$

Therefore $\boxed{q_2 = \pi - \chi}$

The arccos function is not unique. For the vector p̲ the solution indicated by the dotted line is given by the positiv sign of β and χ and the straight line solution by the negative sign. Obviously the solution is not unique.

Solution for a typical 6 DOF robot with intersecting wrist axis

The general solution simplifies, if the robot has a „wrist“ with intersecting axes (axes 4-6). A partial decoupling of position and orientation is possible.

For this case the intersection point $\underline{p}_3$ can be calculated by

$${}^0\underline{p}_3 = {}^0\underline{p} - {}^0\underline{a} \cdot d_6$$

The position of the $\underline{p}_3$ only depends on the first 3 joint values q_1, q_2 and q_3. These 3 joint values are calculated depending on $\underline{p}_3$.

Calculation of q_3 :

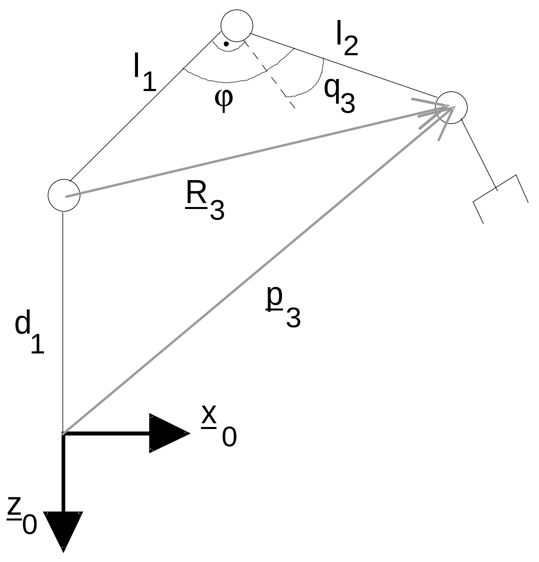


Figure 3-5: *Calculation of q_3*

$$\underline{R}_3 = \underline{p}_3 + \begin{pmatrix} 0 \\ 0 \\ d_1 \end{pmatrix}, \quad |\underline{R}_3| = \sqrt{p_{3x}^2 + p_{3y}^2 + (p_{3z} + d_1)^2}$$

law of cosine for φ : $|\underline{R}_3|^2 = l_1^2 + l_2^2 - 2 \cdot l_1 \cdot l_2 \cdot \cos\varphi$

$$\varphi = \arccos \frac{l_1^2 + l_2^2 - |\underline{R}_3|^2}{2 \cdot l_1 \cdot l_2} \quad \text{therefore } q_3 \text{ is known : } q_3 = \frac{\pi}{2} - \varphi$$

Calculation of q_2 :

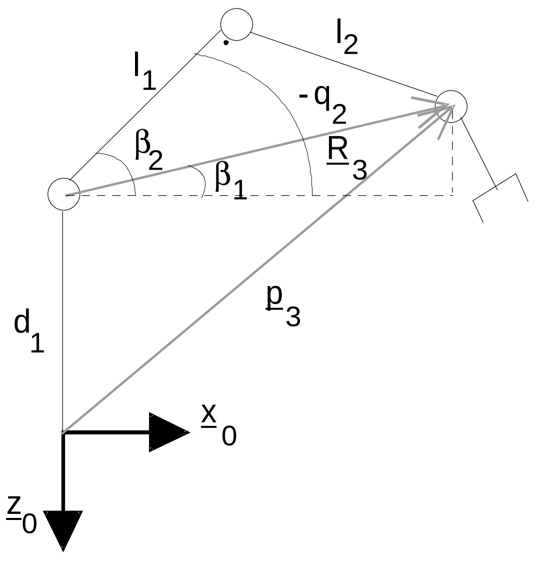


Figure 3-6: *Calculation of q_2*

$$\sin\beta_1 = \frac{|p_{3z}| - d_1}{|\underline{R}_3|}\beta_1 = \arcsin\frac{|p_{3z}| - d_1}{|\underline{R}_3|}$$

$$\text{law of cosine for } \beta_2 : l_2^2 = l_1^2 + |\underline{R}_3|^2 - 2\cdot l_1 \cdot |\underline{R}_3| \cdot \cos\beta_2$$

$$\beta_2 = \arccos\frac{l_1^2 + |\underline{R}_3|^2 - l_2^2}{2\cdot l_1 \cdot |\underline{R}_3|} \quad \text{therefore } q_2 \text{ is known: } q_2 = -(\beta_1 + \beta_2)$$

Calculation of q_1 :

Figure 3-7: *Calculation of q_1*

$$q_1 = \arctan 2(p_{3y}, p_{3x})$$

Calculation of q_5 :

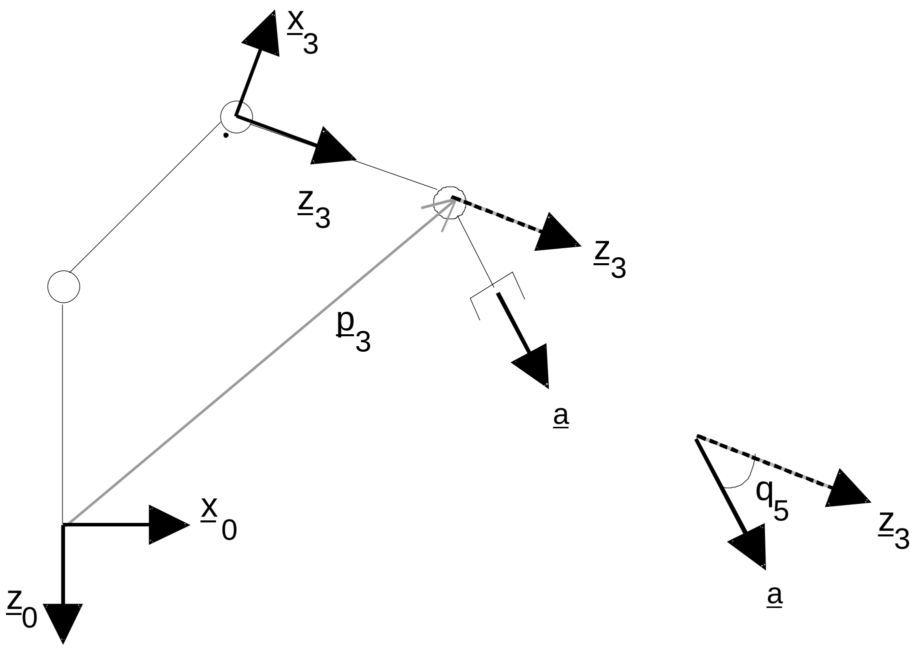


Figure 3-8: *Calculation of q_5*

$$q_5 = \arccos(\underline{z}_3 \bullet \underline{a})$$

Calculation of q_4:

$$\underline{T} = {}^0_1\underline{D}\cdot{}^1_2\underline{D}\cdot{}^2_3\underline{D}\cdot{}^3_4\underline{D}\cdot{}^4_5\underline{D}\cdot{}^5_6\underline{D}$$

$$\underline{T} = {}^0_3\underline{D}\cdot{}^3_6\underline{D} \Rightarrow {}^3_6\underline{D} = \left[{}^0_3\underline{D}\right]^{-1}\cdot\underline{T} \Rightarrow$$

$${}^3_4\underline{D}\cdot{}^4_5\underline{D}\cdot{}^5_6\underline{D} = \left[{}^0_1\underline{D}\cdot{}^1_2\underline{D}\cdot{}^2_3\underline{D}\right]^{-1}\cdot\underline{T}$$

The right side of the last equation is known, since $\underline{T}$ is given and the Denavit-Hartenberg matrices at the right side only depend on q_1, q_2, q_3 , which now available. The left side depends only q_4, q_5 and q_6. The modulus of q_5 is also available. It is possible to eliminate q_6 and to calculate q_4 .

Calculation of q$_6$:

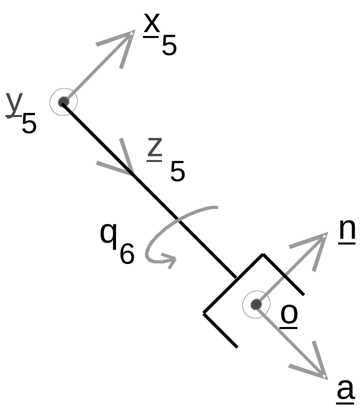


Figure 3-9: *Calculation of q$_6$*

$$q_6 = \arccos({}^0\underline{y}_5 \bullet \underline{o})$$

$${}^0\underline{y}_5 \text{ is known, since } {}^0_5\underline{\mathrm{D}} \text{ can be calculated by the given } \mathrm{q}_1,\ldots,q_5$$

$${}^0_5\underline{\mathrm{D}} = \begin{pmatrix} {}^0\underline{x}_5 & {}^0\underline{y}_5 & {}^0\underline{z}_5 & {}^0\underline{p}_5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Trajectory Generation

Scope: methods of computing a trajectory in multidimensional space

Definition: trajectory refers to a time history of position, velocity and acceleration for each degree of freeom

Issues:

- Specification of trajectories
- Motion description easy for robot operators
- start and end point
- geometrical properties
- Representation of trajectories in the robot control
- Computing trajectories on-line (at run time)

General considerations

- Motion of robot is described as motion of TCP (tool frame)
- Supports robot operator's imagination
- Decoupling the motion description from any robot, end-effector (modularity)
- Exchangeability with other robots
- Supports the idea of moving frames (conveyer belt)

<u>Basic problem:</u> move the robot from the start position, given by the tool frame $T_{initial}$ to the end position given by the tool frame T_{final}.

Spatial motion constraints:

- Specification of motion might include socalled via points
- Via points: intermediate points between start and end points

Temporal motion contraints:

- Specification of motion might include elapsed time between via points

Requirements:

- Execution of smooth motions
- Smooth (motion) function: function and its first derivative is continous
- Jerky motions increase wear on the mechanism (gears) and cause vibrations by exciting resonances of the robot.

There are two methods of path generation:

- Joint space schemes: path shapes in space and time are described in terms of functions of joint angles. This motion type is named Point-to-Point motion (PTP)

- Cartesian space schemes: path shapes in space and time are described in terms of functions of cartesian coordinates. This motion type is named Continous Path motion (CP)

4.1. Trajectory generation in joint space (PTP motions)

- Path shape (in space and time) described in terms of functions of joint angles
- Description of path points (via points plus start and end point) in terms of tool frames
- Each path point is converted into joint angles by application of inverse kinematics
- Identifying a smooth function for each of the n joints passing through the via points and ends at the target point
- Synchronisation of motion (each joint starts and ends at the same time)
- Joint which travels the longest distance defines the travel time (assuming same maximal acceleration for each joint)

- In between via points the shape of the path is complex if described in cartesian space
- Joint space trajectory generation schemes are easy to compute
- Each joint motion is calculated independantly from other joints

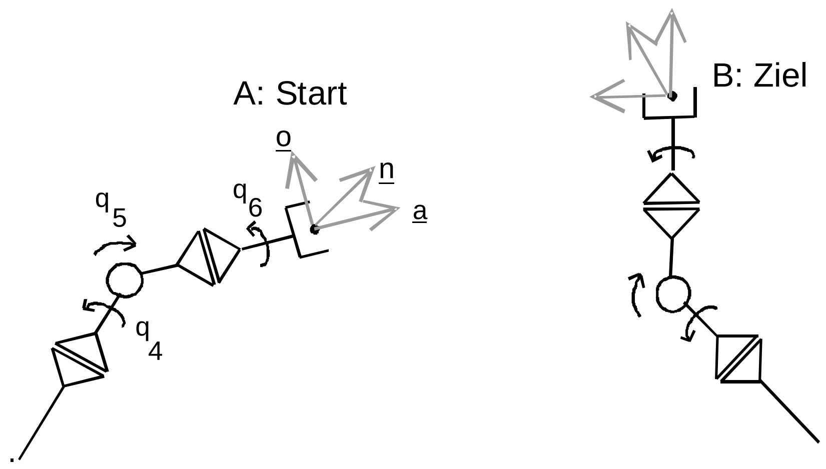


Figure 4-1: *PTP motion between two points*

Many, actually infinite, smooth functions exist for such motions

Four contraints on the (single) joint function q(t) are evident:

- Start configuration $q(0) = q_A$, end configuration $q(t_{end}) = q_B$
- Velocities $\dot{q}(0) = 0$, $\dot{q}(t_{end}) = 0$

Four constraints can be satisfied by a polynomial of degree 3 or higher.

In case of via points the velocity is not zero.

- Start configuration $q(0) = q_A$, end configuration $q(t_{end}) = q_B$
- Velocities $\dot{q}(0) = \dot{q}_0$, $\dot{q}(t_{end}) = \dot{q}_{end}$

Choosing velocities by

- robot user
- automatically chosen by the robot control

Using these type of polynomials does not in general generate time optimal motions

Linear function with parabolic blends:

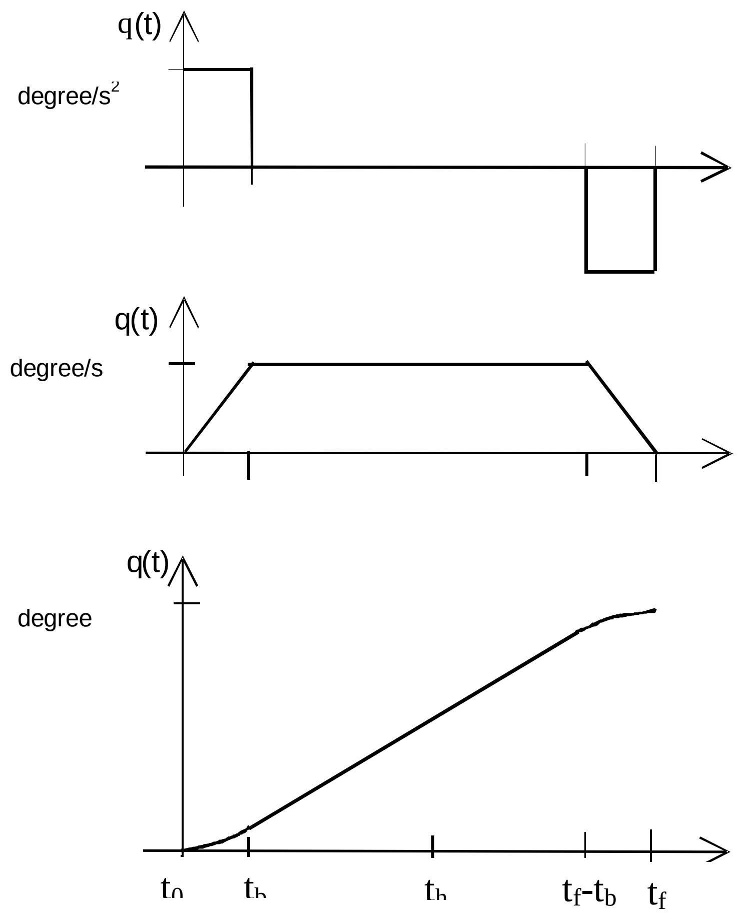


Figure 4-2: *path with trapezoidal velocity profile*

smooth motion constructed by adding parabolic blends to the linear function

Assumptions:

- Constant acceleration during blend
- Same duration for the (two) parabolic blends
- Therefore symmetry about the halfway point in time t_h

Velocity at the end of the blend is equal to velocity during the linear section:

$\ddot{q}t_b = \dfrac{q_h - q_b}{t_h - t_b}$ and

$q_b = q_0 + \dfrac{1}{2}\ddot{q}t_b^2$ leads to

$$\ddot{q}t_b^2 - \ddot{q}tt_b + (q_f - q_0) = 0$$

Free parameters $\ddot{q}$ and t_b. Usually acceleration is chosen and the equation is solved for t_b.

- Acceleration must be sufficiently high (otherwise a solution does not exist)
- Acceleration is not continous, which excites vibrations

Alternative: sinoidal profile

Acceleration defined by: $$\ddot q(t) = \ddot q_{\max}\cdot\sin^2\left(\frac{\pi}{t_b}\cdot t\right) \tag{4.1}$$

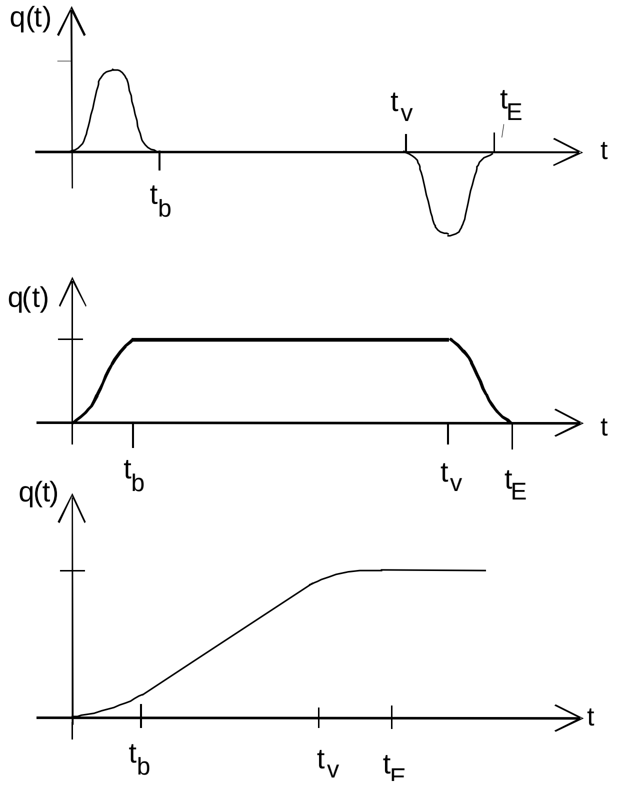


Figure 4-3: *profile for a sinusoidal path*

The following relationships are valid:

$$\dot q_{\max} = \frac{t_f\cdot\ddot q_{\max}}{4} - \sqrt{\frac{t_f^2\cdot\ddot q_{\max}^2}{16} - \frac{q_f\cdot\ddot q_{\max}}{2}},\ \ddot q_{\max} = \frac{2\cdot\dot q_{\max}^2}{\dot q_{\max}\cdot t_f - q_f}, t_b = t_f - \frac{q_f}{\dot q_{\max}}$$

$$t_b = \frac{2\cdot\dot q_{\max}}{\ddot q_{\max}}, t_v = t_f - t_b \tag{4.2}$$

The constraint on the acceleration is

$$t_f^2\cdot\ddot q_{\max} \ge 8\cdot q_f \tag{4.3}$$

Calculation of q(t):

$$0 \le t \le t_b : q(t) = \ddot{q}_{\max}\left[\frac{1}{4}\cdot t^2 + \frac{t_b^2}{8\pi^2}\left(\cos\frac{2\pi}{t_b}\cdot t - 1\right)\right]$$

$$t_b \le t \le t_v : q(t) = \dot{q}_{\max}\cdot\left(t - \frac{1}{2}\cdot t_b\right)$$

$$t_v \le t \le t_f : q(t) = \frac{\ddot{q}_{\max}}{2}\cdot\left[t_f\cdot t - \frac{t^2}{2} + t_f\cdot t_b - \frac{t_f^2}{2} - t_b^2 + \frac{t_b^2}{4\pi^2}\left(1 - \cos(\frac{2\pi}{t_b}\cdot(t - t_v))\right)\right]$$

(4.4)

4.1.1. Asynchrone PTP

independant calculation of joint angles along each path
different travel time for each axis in general (misbehaviour)

- for each joint the values $q_0, q_f, \ddot{q}_{\max}$ are given. The travel time t_i for each joint is calculated

4.1.2. Synchrone PTP:

each joint ends its motion at the same time

- for each joint the values $q_0, q_f, \ddot{q}_{\max}$ are given. The travel time t_i for each joint is calculated
- calculate the maximum travel time $t_{f,\max} = \max\limits_{i=1,..,dof}\{t_{f,i}\}$
- for each joint set $t_{f,i} = t_{f,\max}$
- adjust joint speeds $\dot{q}_{\max,i}$

$$t_{f,i} = t_{f,\max} = \frac{q_{f,i}}{q_{\max,i}} + \frac{q_{\max,i}}{q_{\max,i}} \Rightarrow q^2_{\max,i} - \tfrac{1}{2} q_{\max,i}\cdot q_{\max,i}\cdot t_{f,\max} + \tfrac{1}{2} q_{f,i}\cdot q_{\max,i} = 0$$

and therefore

$$\dot{q}_{\max,i} = \frac{\ddot{q}_{\max,i}\cdot t_{f,\max}}{4} - \sqrt{\frac{\ddot{q}^2_{\max,i}\cdot t^2_{f,\max} - 8q_{f,i}\cdot q_{\max,i}}{16}}$$

(4.5)

The negative sign of the square root has to be chosen in order to fulfill *$2t_b < t_{f,max}$*

4.2. Trajectory generation in cartesian space

robot user would like to control the motion between start and end point
several possibilities:

- straight line motion (TCP follows a straight line)
- circular motion (TCP follows a circle segment)
- spline motion

- inverse kinematics needs to be calculated at run time
- computational expensive (depending on the robot)

issues:

- interpolation of TCP position (linear change of coordinates)
- interpolation of orientation (linear change of matrix elements would fail)

4.2.1. Linear Interpolation

Start point and end point are given in world frame:

start point: $\underline{x}_A = \begin{pmatrix} p_{x,A} \\ p_{y,A} \\ p_{z,A} \\ \hline \alpha_A \\ \beta_A \\ \chi_A \end{pmatrix} = \begin{pmatrix} \underline{p}_A \\ \underline{\Theta}_A \end{pmatrix}$ target point $\underline{x}_B = \begin{pmatrix} p_{x,B} \\ p_{y,B} \\ p_{z,B} \\ \hline \alpha_B \\ \beta_B \\ \chi_B \end{pmatrix} = \begin{pmatrix} \underline{p}_B \\ \underline{\Theta}_B \end{pmatrix}$

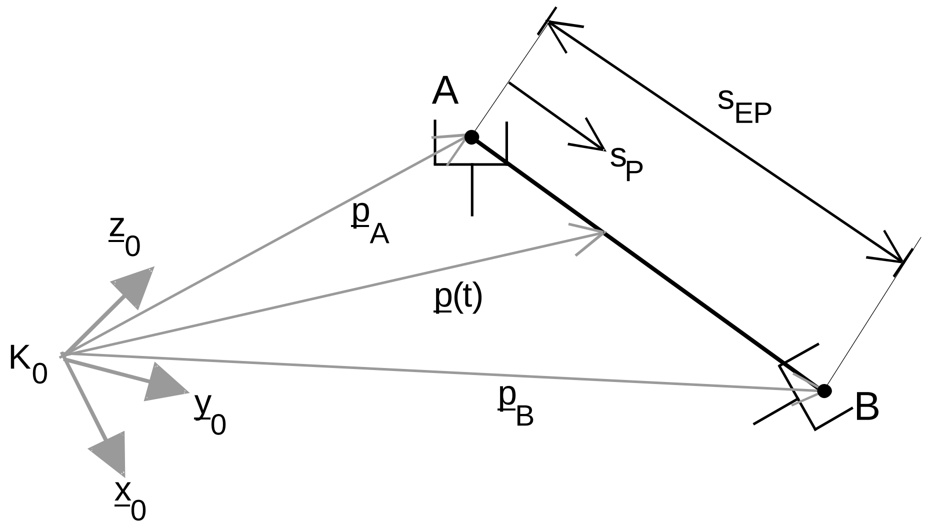


Figure 4-1: *linear interpolation for the positioning of the TCP in world frame*

- TCP is moving between the points A and B along a straight line
- Robot user sets motion speed and motion acceleration
- Orientation changes from (α_A, β_A, γ_A) to (α_B, β_B, γ_B)
- Orientation defined by Euler angles
- Calculation of intermediate positions $\underline{p}$ and orientations $\underline{\Theta}$:

$$\underline{p}(t) = \underline{p}_A + \frac{s_p(t)}{s_f} \cdot (\underline{p}_B - \underline{p}_A) \qquad (4.6)$$

s_P denotes the path parameter, s_{EP} denotes the path length:

$$s_{fP} = \left|\underline{p}_B - \underline{p}_A\right| = \sqrt{(p_{x,B} - p_{x,A})^2 + (p_{y,B} - p_{y,A})^2 + (p_{z,B} - p_{z,A})^2}$$
(4.7)

The motion starts at time t=0 and ends in the target at time t=t_{EP}.

$\dot{s}_P(0) = \dot{s}_P(t_{fP}) = 0$.

Orientation:

$$\underline{\Theta}(t) = \underline{\Theta}_A + \frac{s_\Theta(t)}{s_{E\Theta}} \cdot (\underline{\Theta}_B - \underline{\Theta}_A)$$
$$s_{f\Theta} = \left|\underline{\Theta}_B - \underline{\Theta}_A\right| = \sqrt{(\alpha_B - \alpha_A)^2 + (\beta_B - \beta_A)^2 + (\chi_B - \chi_A)^2}$$
(4.8)

$$\dot{s}_\Theta(0) = \dot{s}_\Theta(t_{E\Theta}) = 0$$

For the positioning of the TCP velocity profiles like in the PTP case are used

- Trapezoidal profile
- Sinusoidal profile

Robot user sets speed and acceleration for position and orientation
The duration of the motion (in case of sinusoidal profile) is:

$$t_{fP} = \frac{s_{fP}}{v_P} + \frac{v_P}{b_P} \text{ and } t_{f\Theta} = \frac{s_{f\Theta}}{v_\Theta} + \frac{v_\Theta}{b_\Theta}$$
(4.9)

Synchronisation of motion:

$$t_f = Max(t_{fP}, t_{f\Theta}) \ .$$

Case 1: $t_f = t_{fP}$

$$v_\Theta = \frac{b_\Theta \cdot t_f}{2} - \sqrt{\frac{b_\Theta^2 \cdot t_f^2}{4} - s_\Theta \cdot b_\Theta} \quad , \quad t_{b\Theta} = \frac{v_\Theta}{b_\Theta} \ .$$
(4.10)

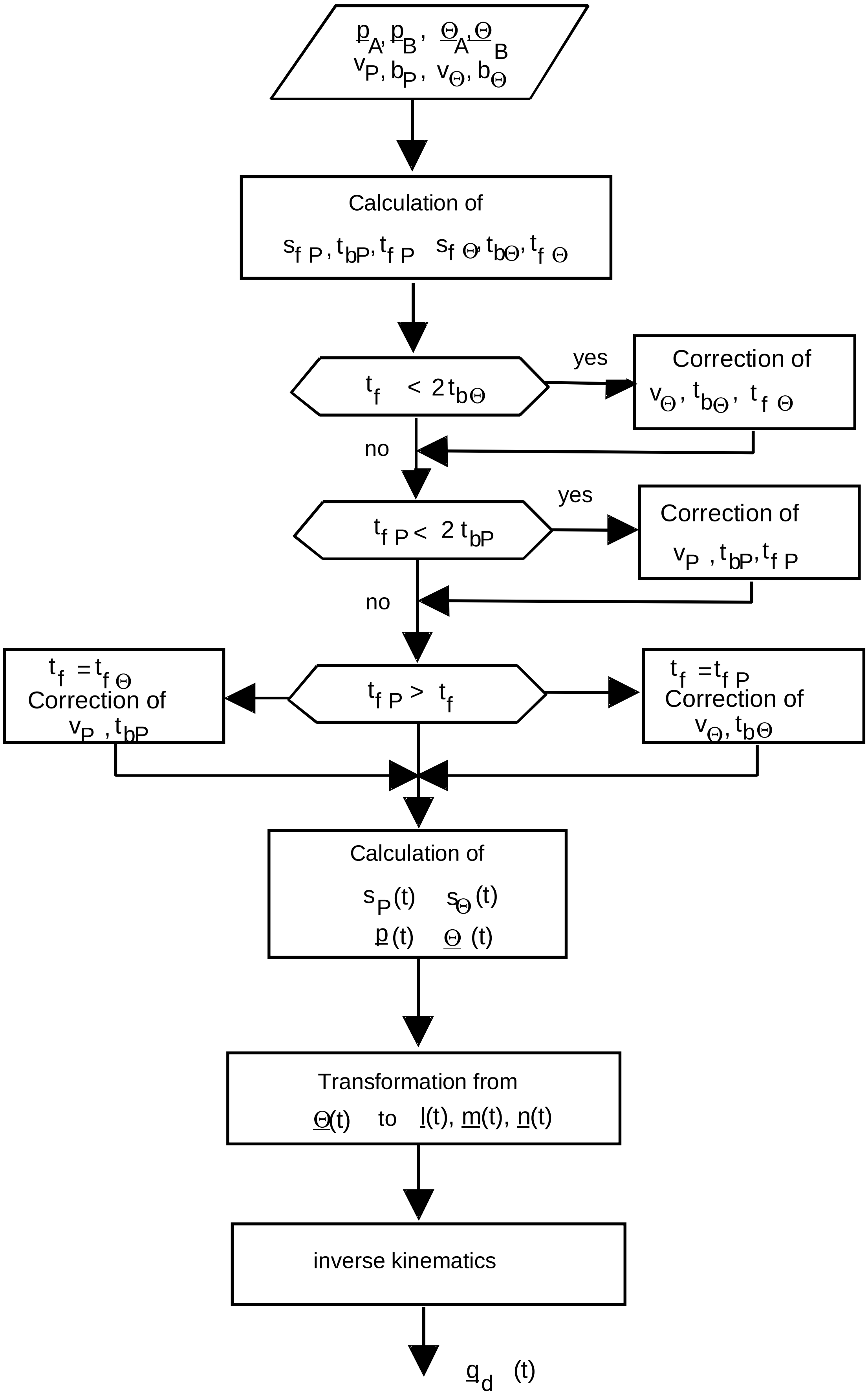
Case 2: $t_f = t_{f\Theta}$

$$v_P = \frac{b_P \cdot t_f}{2} - \sqrt{\frac{b_P^2 \cdot t_f^2}{4} - s_P \cdot b_P} \quad , \quad t_{bP} = \frac{v_P}{b_P} \ .$$
(4.11)

Reference to the PTP motion:

$$\begin{array}{lcl}
t_b, t_V \rightarrow t_{bP}, t_{VP} & & t_b, t_V \rightarrow t_{b\Theta}, t_{V\Theta} \\
q(t) \rightarrow s_P(t) & & \mathrm{s(t)} \rightarrow \mathrm{s}_\Theta(t) \\
\dot{q}_{\max}(t) \rightarrow v_P(t) & \text{or} & v_m(t) \rightarrow v_\Theta(t) \\
\ddot{q}_{\max}(t) \rightarrow b_P(t) & & b_m(t) \rightarrow b_\Theta(t)
\end{array}$$

The following scheme presents the algorithm for straight line motions


$\underline{p}_A, \underline{p}_B, \underline{\Theta}_A, \underline{\Theta}_B$
$v_P, b_P, v_\Theta, b_\Theta$
Calculation of
$s_{f\,P}, t_{bP}, t_{f\,P}$ $s_{f\,\Theta}, t_{b\Theta}, t_{f\,\Theta}$
$t_f < 2 t_{b\Theta}$
yes
Correction of
$v_\Theta, t_{b\Theta}, t_{f\,\Theta}$
no
$t_{f\,P} < 2 t_{bP}$
yes
Correction of
$v_P, t_{bP}, t_{f\,P}$
no
$t_f = t_{f\,\Theta}$
Correction of
v_P, t_{bP}
$t_{f\,P} > t_f$
$t_f = t_{f\,P}$
Correction of
$v_\Theta, t_{b\Theta}$
Calculation of
$s_P(t)$ $s_\Theta(t)$
$\underline{p}(t)$ $\underline{\Theta}(t)$
Transformation from
$\underline{\Theta}(t)$ to $\underline{l}(t), \underline{m}(t), \underline{n}(t)$
inverse kinematics
$\underline{q}_d(t)$

Problems related to cartesian motions:

- Intermediate points unreachable (out of work space)
- High joint rates near singularities
- Start and target position reachable in different solutions

4.2.2. Circular interpolation

Motion of the TCP along a circular segment from point P_1 via P_2 to point P_3.
P_2 is sometimes called auxiliary point.
The points P_1,P_2,P_3 and the velocity v_C and acceleration b_C are defined by the robot user.

- from points P_1,P_2,P_3 calculate center point M and radius r by

$$\vec{n}_K = (P_2 - P_1) \times (P_3 - P_2)\,.$$

$E_i = \{\, X \in P^3 \mid \vec{x} \cdot \vec{n}_i = d_i \,,\, \vec{x} := \overrightarrow{OX} \,\}$ for *i*=1,2 with $\begin{array}{l} \vec{n}_1 := (P_2 - P_1), d_1 := \frac{1}{2}(P_1 + P_2) \cdot \vec{n}_1 \\ \vec{n}_2 := (P_3 - P_2), d_2 := \frac{1}{2}(P_2 + P_3) \cdot \vec{n}_2 \end{array}$

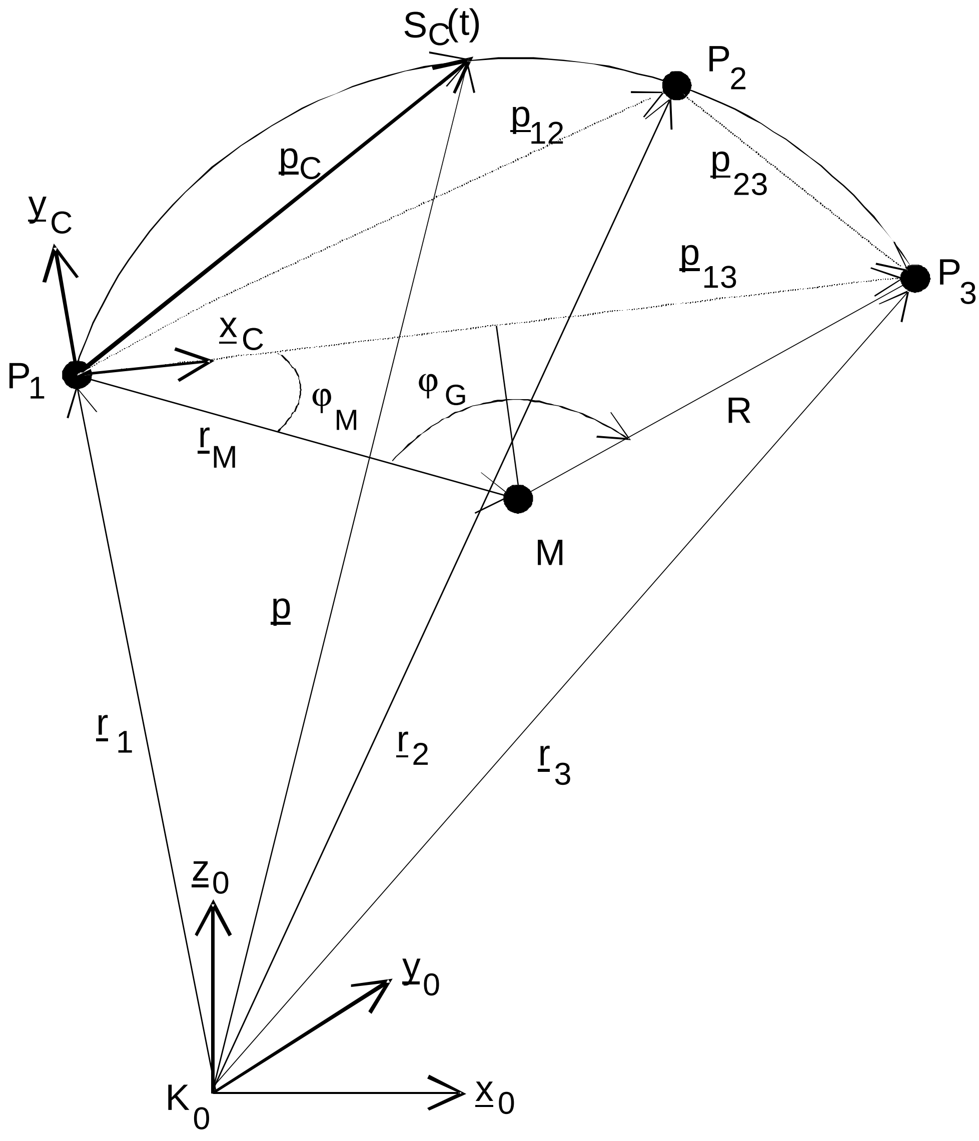
The vector $\vec{m}$ of the center point *M* is the solution of the linear system

$$\begin{pmatrix} \vec{n}_1^T \\ \vec{n}_2^T \\ \vec{n}_K^T \end{pmatrix} \vec{m} = \begin{pmatrix} d_1 \\ d_2 \\ d_K \end{pmatrix},$$

- introduction of path parameter $s_C(t) = r*\varphi(t)$, where $\varphi(t)$ is opening angle up to actual position. $s_{CE} = r*\varphi_G$ denotes the total path length.
- analog procedure to linear interpolation

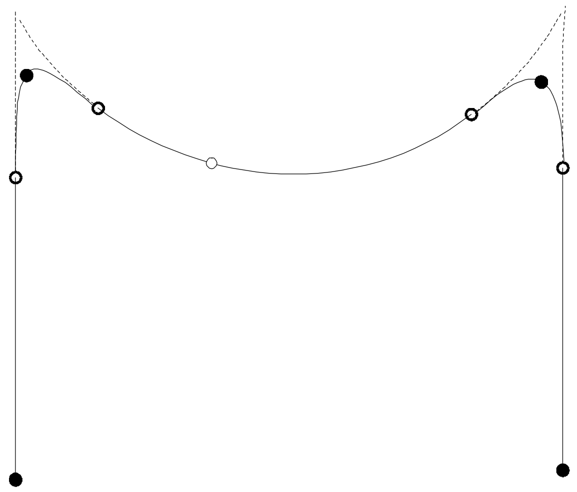
Exercise:

Given the points P_1,P_2,P_3 defining a circular segment, with
$P_1 = [0\ 1\ 1]^T$; $P_2 = [1\ 1\ 2]^T$; $P_3 = [2\ 1\ 1]^T$ Calculate center point M and radius r.



4.2.3. Corner smoothing

The above mentioned interpolation methods let the motion stop at the end point
Corner smoothing let the robot move from one segment to another without reducing speed



4.2.4. Spline interpolation based on Bézier curves

- Bezier curves are useful to design trajectories with certain „roundness“ properties
- Defined by a series of control points with associated weighting factors
- Weighting factors „pull“ the curve towards the control points
- Bezier curves are based on Bernstein polynomials (we consider here cubic polynomials, n=3)

$$B_k^n(u) := \binom{n}{k}(1-u)^{n-k}u^k$$

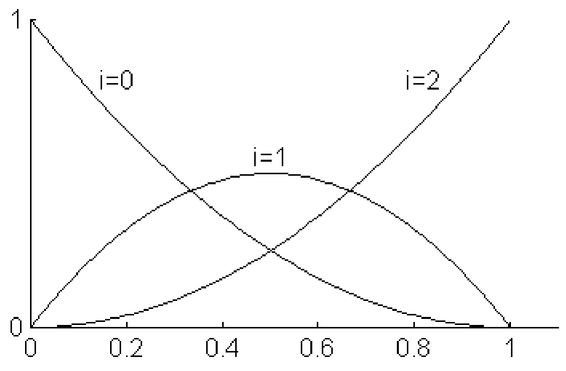
with normalized parameter interval.

Bernstein polynomials can be derived from formula:

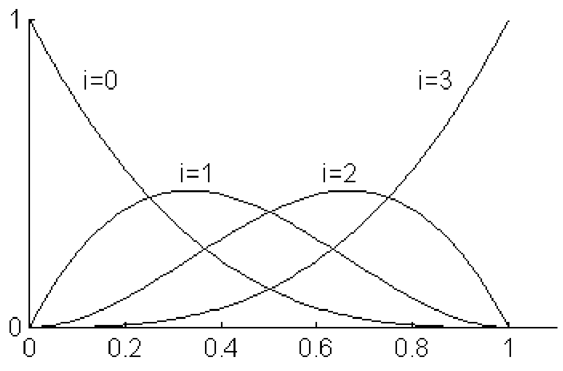
$$1 = ((1-u)+u)^n = \sum_{k=0}^{n}\binom{n}{k}(1-u)^{n-k}u^k$$

Bernstein polynomials with n=2 (quadratic) and n=3 (cubic)

$$B_i^2(u), B_i^3(u)$$



(a)



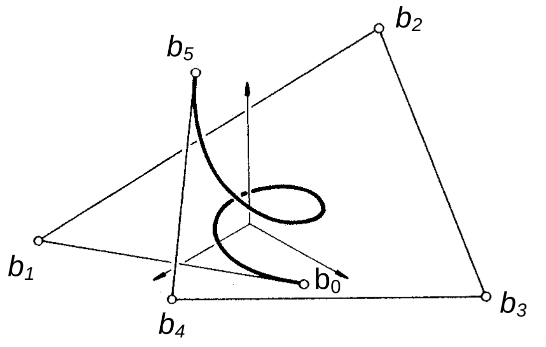
(b)

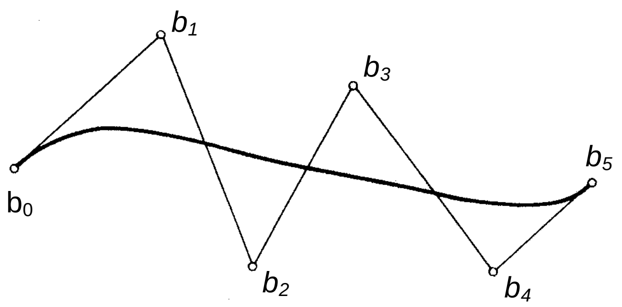
Definition of Bezier curve is based on these polynomials

$$X(u) = \sum_{i=0}^{n} b_i B_i^n(u)$$

b_i denotes the Bézier-points or control points. The polygon connecting the Bézier-points are called Bézier-polygon or control polygon.

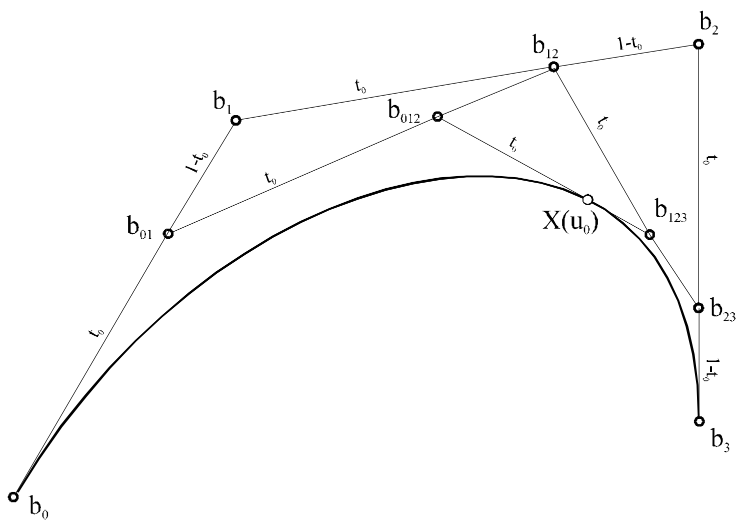
Two examples for Bezier curves with associated control polygon





de Casteljau-Algorithm is used to calculate points on the Bezier curve: $X_n(u_0)$

Geometrical interpretation: Consecutive division of control polygon $(1-u_0):u_0$



Exercise:
Given the (two-dimensional) control points B$_0$, B$_1$, B$_2$, and B$_3$ for a cubic Bezier curve by B$_0$ =$(0,0)^T$, B$_1$ =$(1,0)^T$, B$_2$ =$(1,2)^T$, B$_3$ =$(0,2)^T$

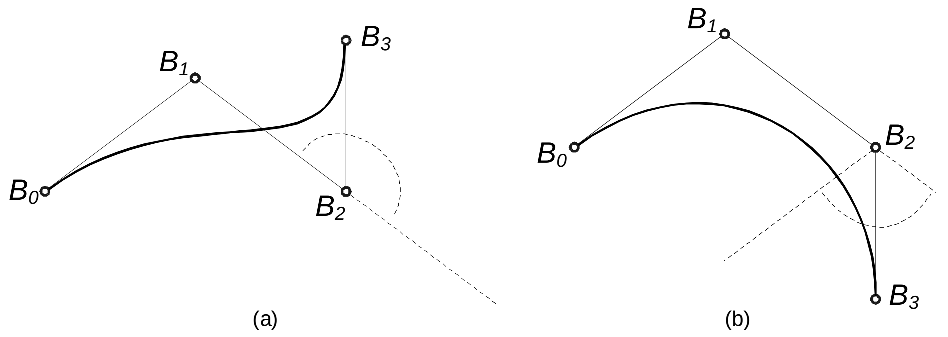
Calculate $X(0{,}5)$ geometrically and by computation of $X(u)=\sum_{i=0}^{n} b_i B_i^n(u)$

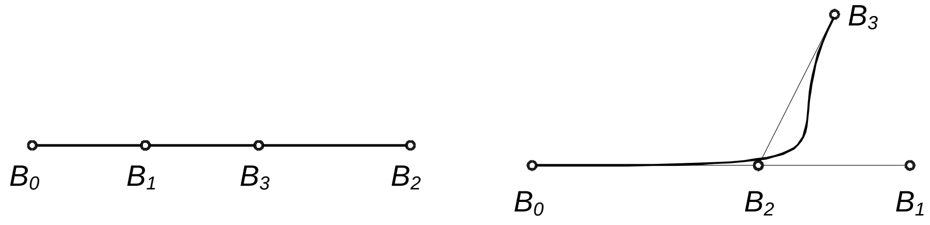
Transformation of parameter interval [0 , 1] to $[a,b]$ by $u \mapsto a+(b-a)u$:

Generalized definition of Bernstein polynomials:

$$B_k^n(u) := \frac{1}{(b-a)^n}\binom{n}{k}(b-u)^{n-k}(u-a)^k$$

Further examples:

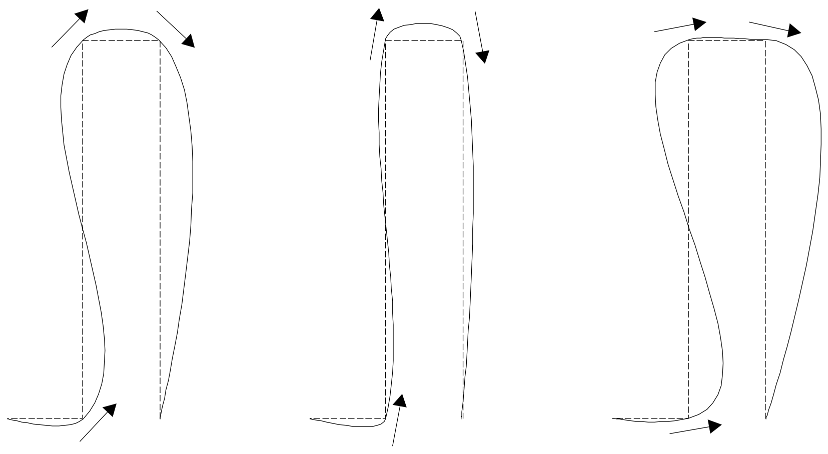




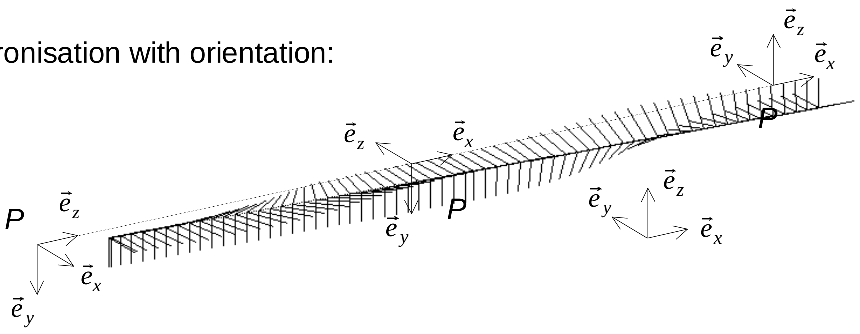
Problems:

- Path length can not be calculated in closed form (like for linear or circular segments)
- Numerical approximation possible

- Integration in robot controllers:
- Teach points programmed by robot user
- System „estimates“ tangents



Synchronisation with orientation:



Generic trajectory generation (example linear interpolation):

$B_0 := P_A$

$B_1 := P_A + \frac{1}{3}(P_E - P_A)$

$B_2 := P_E - \frac{1}{3}(P_E - P_A)$

$B_3 := P_E$

5. Robot programming languages

Robot programming systems are the interface between the robot and the human user. The sophistication of such an user interface is becoming very important as robots are applied to more and more demanding industrial applications.
In considering the programming of manipulators, it is important to remember that they are typically only a minor part of an automated process. The term workcell is used to describe a local collection of equipment which may include one or more robots, conveyor systems, part feeders and fixtures. At the next higher level, workcells might be interconnected in factorywide networks so that a central control computer can control the overall factory flow.

5.1. Levels of robot programming

Three levels of robot programming exist:

- Teach In programming
- Explicit robot languages
- Task level programming languages

Teach In programming

The robot will be moved by the human user through interaction with a teach pendant (sometimes called teach box). Teach pendant are hand-held button boxes which allow control of each robot joint or of each cartesian degree of freedom. Todays controllers allow alphanumeric input, testing and branching so that simple programs involving logic can be entered.

Explicit robot languages

With the arrival of inexpensive and powerful computers, the trend has been increasingly toward programming robots via programs written in computer programming languages. Usually these computer programming languages have special features which apply to the problems of programming robots. An international standard has been established with the programming language IRL (Industrial Robot Language, DIN 66312)

Task level programming

The third level of robot programming methodology is embodied in task-level programming languages. These are languages which allow the user to command desired subgoals of the task directly, rather than to specify the details of every action the robot is to take. In such a system, the user is able to include instructions in the application program at a significantly higher level than in an explicit programming language. A task-level programming system must have the ability to perform many planning tasks automatically. For example, if an instruction to *grasp the bolt* is issued, the system must plan a path of the manipulator which avoids collision with any surrounding obstacles, automatically choose a good grasp location on the bolt, and grasp it. In contrast, in an explicit robot language, all these choices must be made by the programmer.

The border between explicit robot programming languages is quite distinct. Incremental advances are beeing made to explicit robot programming languages which help to ease programming, but these enhancements cannot be counted as components of a task-level programming system. True task-level programming of robots does not exist yet in industrial controllers but is an active topic of research.

5.2. Requirements of a robot programming language

Important requirements of a robot programming language are:

- World modeling
- Motion specification
- Flow of execution
- Sensor inegration

World modeling:

- Existence of geometric types to present
  - Joint angle sets
  - Cartesian positions
  - Orientations
  - Representation of frames
- Ability to to do math on on structured types like frames, vectors and rotation matrices
- Ability to describe geometric entities like frames in several different convenient representations with the ability to convert between representations

Motion execution:

- Description of desired motion (motion type, velocity, acceleration)
- Specifications of via points, goal points, corner smoothing parameters
- Ability to specify goals relative to various frames, including frames defined by the user and frames in motion (on a conveyor for example)

Flow of execution:

- Support of concepts like testing and branching
- Looping, calls to subroutines
- Parallel processing, signal and wait primitives
- Interrupt handling

Sensor integration:

- Interaction with sensors
- Integration with vision systems
- Sensor to track the conveyor belt motion
- Force torque sensor for force control strategies