Design, Implementation and Construction of a Controller for a 6-DOF Serial Robot

Diplomarbeit

Werner Kollment

Betreuer
Ass.Prof. Dipl.-Ing. Dr.mont. Gerhard Rath
O.Univ.-Prof. Dipl.-Ing. Dr.techn. Paul O'Leary

Montanuniversität Leoben
Institut für Automation

June 2014

Eidesstattliche Erklärung

Abstract

This thesis describes the design of a control unit for a six degree of freedom (6-DOF) industrial robot. A fundamental theoretical introduction into robot kinematics is provided, whereby multiple state-of-the-art approaches are explained and compared. The goal of this thesis is to establish a kinematic framework, which is fully transparent for educational use. Furthermore, interfacing additional algorithms and hardware components such as sensors is simplified; hence, expansion of the system requires less effort compared to conventional controllers. The robot control is based on an industrial personal computer (PC), which is divided into a real-time programmable logic controller (PLC) and a conventional Windows desktop system. The robot is actuated by six frequency-inverter driven servo motors, which are controlled by the PLC. The PLC is equipped with a generic network interface, which enables execution of remote positioning commands. In this thesis, the forward and inverse kinematic computations for the PLC are implemented in MATLAB. Simulink is utilized to provided a real-time environment for the MATLAB functions on a remote PC in order to communicate with the PLC. The network connection is established via the user datagram protocol (UDP), whereby real-time capabilities are ensured. The overall system's safety related functions are controlled by a dedicated safety PLC. The correct functionality of this customized implementation is validated with the existing industrial solution provided by Bernecker & Rainer. As result of this thesis, a fully operational robot control is obtained, which is utilized for educational purposes such as student projects as well as research on robotic related topics.

Zusammenfassung

Diese Diplomarbeit beschreibt die Entwicklung einer Steuerung für einen Industrieroboter mit sechs Freiheitsgraden. Für ein besseres Verständnis der Kinematik werden die notwendigen Grundlagen im Text zusammengefasst und geläufige Methoden erläutert. Die Zielsetzung dieser Diplomarbeit ist es, ein offenes Steuerungssystem für die Ausbildung zu schaffen. Durch das neue Konzept wird eine Einbindung von zusätzlichen Algorithmen und Hardware, wie zum Beispiel Sensoren, erleichtert. Die Steuerung basiert auf einem Industrie PC, die über ein Echtzeitbetriebssystem und ein parallel laufendes Windows System verfügt. Die sechs Servomotoren des Roboters werden von sechs Frequenzumrichtern versorgt, welche vom Industrie PC gesteuert werden. Zur Ansteuerung des Roboters wurde eine Netzwerkschnittstelle geschaffen, welche Positionierungsbefehle entgegen nimmt und ausführt. Die Vorwärts- und Rückwärtskinematik für die Netzwerkschnittstelle wird in MATLAB berechnet. Die Kinematik wurde auf einen externen Computer implementiert und über das UDP-Protokoll über eine Netzwerkschnittstelle kommuniziert. Um die Echtzeitfähigkeit zu gewährleisten wurde in Simulink eine Echtzeitumgebung für die MATLAB-Funktionen der Kinematik geschaffen. Alle sicherheitsrelevanten Funktionen der Steuerung werden von einer Sicherheits- SPS überwacht. Das Ergebnis dieser Diplomarbeit ist eine voll funktionsfähige Robotersteuerung, die für die Ausbildung und Forschung genutzt wird.

Contents

Introduction

The field of robotic conquers more and more parts of daily life. Although, robots are capable of much more than pick and place tasks, they are primary used for such tasks. Most standard robot controllers are therefore optimized for pick and place tasks. This leads to a limitation of the applicability of robots in research. Most robot manufacturers also provide more advanced controllers, but they cost twice as much as a standard robot. Due to this high cost and the fact, that the mechanic system of the robots has hardly advanced in the last ten years, it was decided to build a new controller for an already existing six degree of freedom serial robot. There are three main issues, to be dealt with. The first one is the hardware components and their compatibility with the motors and encoders of the existing robot arm. The second issue is the control software and the different methods to control the six axes. The last issue is the mathematically complex kinematics and the ways to use it. The main components of the hardware were supplied by Bernecker & Rainer (B&R). They consist of six frequency converters to power the motors and a standard industrial PC, which controls the whole system. Also necessary is a transformer to reduce the output voltage of the inverter and make it compatible with the motors. In order to control the robot two different types of control software are used. The first one is a standard robot control software from B&R. The intention for this is to maintain the compatibility with industrial standards. The second control software is a self developed control software. In this particular control software the kinematics is excluded and implemented on a remote pc with more computation power instead. Both programs use the UDP to communicate with each other. The kinematics itself was implemented in MATLAB and Simulink, which allows a rapid prototyping of new kinematic concepts or the easy use of machine vision.

Part I

Theoretical Background

Chapter 1

Basic Concepts of Robot Kinematics

The intention of this chapter is to give an overview of the elementary concepts of robot kinematic. The first section explains how a rotation in three dimensional space can be formulated as a rotation matrix and outlines its special properties. The next sections gives a summary of methods to describe a rotation matrix such as Euler angles, Roll-Pitch-Yaw angles, Axis-Angle concept and quaterions. The last two are a different description of the same kinematic concept. The last two sections introduce homogeneous coordinates and the screw concept.

1.1 Rotation Matrix

Points are simply describable in three-dimensional space. In contrast, the description of a body's orientation in three-dimensional space is a complex task. For this purpose, a frame Σ_{uvw} of three orthogonal unit vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ is attached to the body. The frame and its origin indicates the orientation and the position of the body. This frame acts as a local coordinate system. [15, p24-p25] [16, p38-p47] [3, p21-p23]

This local frame Σ_{uvw} describes the rotation of a body or point from a previous frame Σ_{xyz} into a new frame, when both frames share the same origin.

Position vector $\boldsymbol{p}$ of point P in Σ_{xyz}:

$$\boldsymbol{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}, \tag{1.1}$$

Position vector $\boldsymbol{p}$ of point P in Σ_{uvw}:

$$\boldsymbol{p} = \begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix}. \tag{1.2}$$

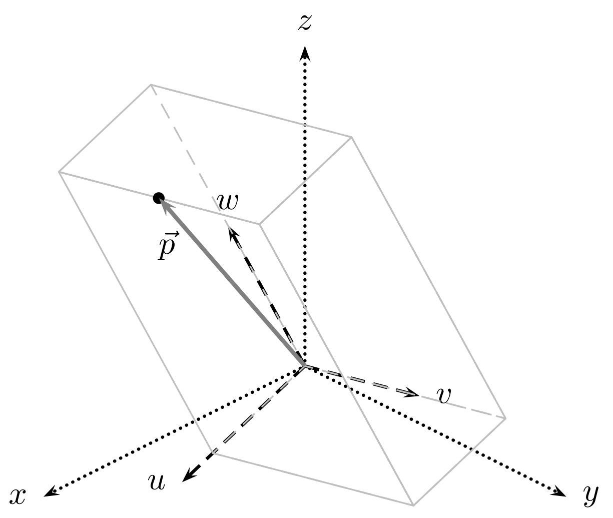


Figure 1.1: The two coordinate frames Σ share the same origin. The length of the position vector $\boldsymbol{p}$ in both frames is the same but to represent the same point P in space in the second frame the vector $\boldsymbol{p}$ needs to be rotated.

This is achieved by choosing a random point P and describing it as linear combination of three unit vectors $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$. If this is done for the global frame Σ_{xyz} as well as the local frame Σ_{uvw}, then two equivalent descriptions for the same point are acquired. This is shown in Figure 1.1 and Equation 1.3 and 1.4.

Σ_{xyz}:

$$\boldsymbol{p_{xyz}} = p_x \boldsymbol{i}_{xyz} + p_y \boldsymbol{j}_{xyz} + p_z \boldsymbol{k}_{xyz}, \tag{1.3}$$

Σ_{uvw}:

$$\boldsymbol{p_{uvw}} = p_u \boldsymbol{i}_{uvw} + p_v \boldsymbol{j}_{uvw} + p_w \boldsymbol{k}_{uvw}. \tag{1.4}$$

The position vector $\boldsymbol{p}$ is multiplied with the direction vectors $\boldsymbol{i}_{xyz}, \boldsymbol{j}_{xyz}, \boldsymbol{k}_{xyz}$ of the global frame Σ_{xyz}. As a result, the components p_x, p_y, p_z of the position vector $\boldsymbol{p}$ are:

$$p_x = \boldsymbol{p}\, \boldsymbol{i}_{xyz}, \tag{1.5}$$
$$p_y = \boldsymbol{p}\, \boldsymbol{j}_{xyz}, \tag{1.6}$$
$$p_z = \boldsymbol{p}\, \boldsymbol{k}_{xyz}. \tag{1.7}$$

$$p_x = (p_u \boldsymbol{i}_{uvw} + p_v \boldsymbol{j}_{uvw} + p_w \boldsymbol{k}_{uvw})\, \boldsymbol{i}_{xyz}, \tag{1.8}$$
$$p_y = (p_u \boldsymbol{i}_{uvw} + p_v \boldsymbol{j}_{uvw} + p_w \boldsymbol{k}_{uvw})\, \boldsymbol{j}_{xyz}, \tag{1.9}$$
$$p_z = (p_u \boldsymbol{i}_{uvw} + p_v \boldsymbol{j}_{uvw} + p_w \boldsymbol{k}_{uvw})\, \boldsymbol{k}_{xyz}. \tag{1.10}$$

In concise matrix form,

$$\underbrace{\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}}_{\boldsymbol{p_{xyz}}} = \underbrace{\begin{bmatrix} \boldsymbol{i}_{uvw}\boldsymbol{i}_{xyz} & \boldsymbol{j}_{uvx}\boldsymbol{i}_{xyz} & \boldsymbol{k}_{uvw}\boldsymbol{i}_{xyz} \\ \boldsymbol{i}_{uvw}\boldsymbol{j}_{xyz} & \boldsymbol{j}_{uvx}\boldsymbol{j}_{xyz} & \boldsymbol{k}_{uvw}\boldsymbol{j}_{xyz} \\ \boldsymbol{i}_{uvw}\boldsymbol{k}_{xyz} & \boldsymbol{j}_{uvx}\boldsymbol{k}_{xyz} & \boldsymbol{k}_{uvw}\boldsymbol{k}_{xyz} \end{bmatrix}}_{\mathsf{R}} \underbrace{\begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix}}_{\boldsymbol{p_{uvw}}} . \tag{1.11}$$

Rearranging the variables as shown in Equation 1.11 simplifies the operation to a simple matrix transformation. The rotation matrix R describes the rotation of the local frame Σ_{uvw} in reference to the global frame Σ_{xyz}. The relation $\boldsymbol{a}\,\boldsymbol{b} = |\boldsymbol{a}||\boldsymbol{b}|\cos(\varphi)$ is used to substitute the scalar product with the cosine function. This is shown in Figure 1.2, for a rotation about the z-axis.

$$\mathsf{R}_z(\varphi) = \begin{bmatrix} \cos(\varphi) & \cos(90+\varphi) & 0 \\ \cos(90-\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} . \tag{1.12}$$

The previous rotation matrix is simplified by using the trigonometric identity $\sin(\varphi) = \cos(90-\varphi)$ and $-\sin(\varphi) = \cos(90+\varphi)$.

$$\mathsf{R}_z(\varphi) = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} , \tag{1.13}$$

$$\mathsf{R}_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\varphi) & -\sin(\varphi) \\ 0 & \sin(\varphi) & \cos(\varphi) \end{bmatrix} , \tag{1.14}$$

$$\mathsf{R}_y(\varphi) = \begin{bmatrix} \cos(\varphi) & 0 & -\sin(\varphi) \\ 0 & 1 & 0 \\ \sin(\varphi) & 0 & \cos(\varphi) \end{bmatrix} . \tag{1.15}$$

By switching the vectors $\boldsymbol{p}, \boldsymbol{i}_{xyz}, \boldsymbol{j}_{xyz}, \boldsymbol{k}_{xyz}$, inversion of the rotation matrix is achieved.

$$\underbrace{\begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix}}_{\boldsymbol{p_{uvw}}} = \underbrace{\begin{bmatrix} \boldsymbol{i}_{uvw}\boldsymbol{i}_{xyz} & \boldsymbol{i}_{uvx}\boldsymbol{j}_{xyz} & \boldsymbol{i}_{uvw}\boldsymbol{k}_{xyz} \\ \boldsymbol{j}_{uvw}\boldsymbol{i}_{xyz} & \boldsymbol{j}_{uvx}\boldsymbol{j}_{xyz} & \boldsymbol{j}_{uvw}\boldsymbol{k}_{xyz} \\ \boldsymbol{k}_{uvw}\boldsymbol{i}_{xyz} & \boldsymbol{k}_{uvx}\boldsymbol{j}_{xyz} & \boldsymbol{k}_{uvw}\boldsymbol{k}_{xyz} \end{bmatrix}}_{\mathsf{R}^{-1}} \underbrace{\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}}_{\boldsymbol{p_{xyz}}} . \tag{1.16}$$

1.1.1 General Properties of the Rotation Matrix

The rotation matrix is an orthogonal matrix $\mathsf{R}^\mathsf{T}\mathsf{R} = \mathsf{I}$ [1] and the properties are displayed in Equation 1.17 and Equation 1.18. Most of this properties are inherited from the cosine function,

[1] in this case also orthonormal

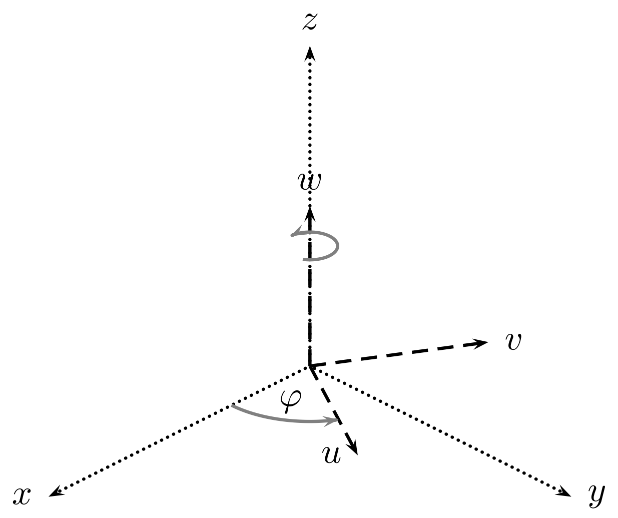


Figure 1.2: Rotation of the coordinate frame Σ about z axis.

$$\mathsf{R}_k(-\varphi) = \mathsf{R}_k(\varphi)^{\mathrm{T}} = \mathsf{R}_k(\varphi)^{-1}, \tag{1.17}$$
$$\mathsf{R}_k(\alpha)\mathsf{R}_k(\beta) = \mathsf{R}_k(\alpha+\beta). \tag{1.18}$$

The rotation matrix can be interpreted as:

1. Rotational transformation from local frame Σ_{uvw} to the global frame Σ_{xyz}.
2. Rotation of a vector about one axis k and a given angle φ.

Practical problems in kinematics require multiple rotations. This can be achieved by rotating a position vector $\boldsymbol{p}$ from frame to frame, as shown in Equations 1.19 and 1.20. This approach is inefficient for multiple rotations. A better way, is to merge all individual rotations into one transformation, like shown in Equation 1.21. This equation exposes, that multiple rotations can be described as product of the single rotation matrices.

Σ_0 to Σ_1:

$$\boldsymbol{p}_1 = \mathsf{R}_1\boldsymbol{p}_0, \tag{1.19}$$

Σ_1 to Σ_2:

$$\boldsymbol{p}_2 = \mathsf{R}_2\boldsymbol{p}_1, \tag{1.20}$$

Σ_0 to Σ_2:

$$\boldsymbol{p}_2 = \mathsf{R}_2 \mathsf{R}_1 \boldsymbol{p}_0. \tag{1.21}$$

Matrix multiplication is not commutative. For this reason, it has to be distinguished between the matrix multiplication from left the pre-multiplication and the matrix multiplication from left the post-multiplication. In many cases these two can only be distinguished by the indices of the matrices or in case of the rotation matrix, by the order of the angles. For the rotation matrix these two types of matrix multiplication have the following meaning: [15, p25-p30] [16, p49-p52]

pre-multiplication: local frame Σ_{uvw} rotates about the fixed global frame Σ_{xyz},

post-multiplication: local frame Σ_{uvw} rotates about its principal axis.

1.1.2 Euler Angles

The Euler angles describe a rotation about the axes of three local frames. The sequence of the rotations is important. Euler angles are commonly used to describe the orientation of a body in space. Euler angles have the advantage, that there is no conversion to the actuating angle of the drives necessary. This is possible because the actuating angle of the drive is measured between stator and rotor, which is also the angle between the joints, where they are fixed. This is similar to the Euler angle, which is measured between two frames.

Euler Angles ZYX Order

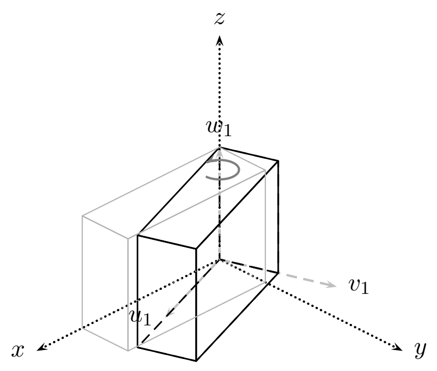
Figure 1.3 shows a rotation in ZYX order. The grey body shows the result of the previous rotation and the black one the result of the actual one. Equation 1.22 displays the order of the single rotations and Equation 1.23 shows the result of the multiplication. In Equation 1.23 the cosine and sine function are substituted by $\cos\alpha \triangleq c\alpha$ and $\sin\alpha \triangleq s\alpha$ to improve the readability.

$$\mathsf{R} = \mathsf{R}_z(\alpha)\, \mathsf{R}_{v_1}(\beta)\, \mathsf{R}_{u_2}(\gamma), \tag{1.22}$$

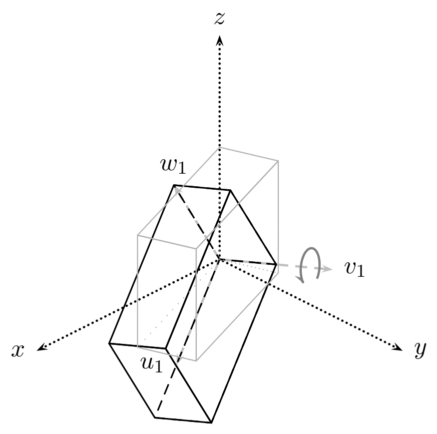
$$\mathsf{R} = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta c\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}. \tag{1.23}$$

Inversion for ZXY Order

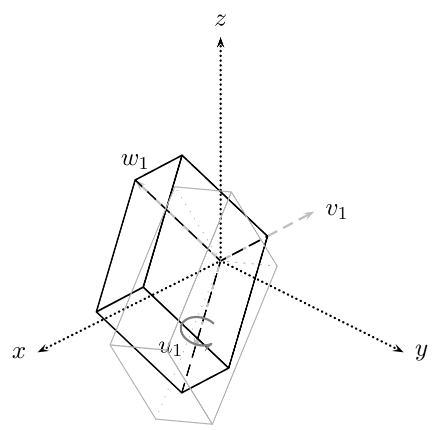
The rotation matrix is known for most orientation problems in kinematics . The rotations parameters are of interest. In case of the Euler-Angles are these the three angles α, β, γ for the known rotation order, here ZYX. These angles are extracted from the rotation matrix, like in Equation 1.23, for the particular rotation order. This is done by searching for suitable entries and calculating



(a) Rotation about Z-axis



(b) Rotation about Y-axis



(c) Rotation about X-axis

Figure 1.3: The figure shows the three single rotations of the Euler angle. It can be seen, that all rotations are relative rotations.

the inverse tangent for each angle with the entries. All entries are suitable where the term with the second angle can be eliminated by division or the trigonometric identity $1 = \sin\alpha^2 + \cos\alpha^2$. Instead of the normal inverse tangent function the inverse tangent-two function is used to avoid problems with the sign. The inverse tangent-two requires two parameters $\alpha = \operatorname{atan2}(x, y)$. This parameters are $x \triangleq \sin\alpha$ and $y \triangleq \ cos\alpha$.

$$\mathsf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} . \tag{1.24}$$

There are two sets of inversion formulas for the Euler-angles. The reason for this are inverse function of the trigonometric functions, which deliver two results for one input value [2].

$\frac{\pi}{2}, \frac{-\pi}{2}$:

$$\beta = \operatorname{atan2}(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}), \tag{1.25}$$
$$\alpha = \operatorname{atan2}(r_{21}, r_{11}), \tag{1.26}$$
$$\gamma = \operatorname{atan2}(r_{32}, r_{33}). \tag{1.27}$$

$\frac{\pi}{2}, \frac{3\pi}{2}$:

$$\beta = \operatorname{atan2}(-r_{31}, -\sqrt{r_{11}^2 + r_{21}^2}), \tag{1.28}$$
$$\alpha = \operatorname{atan2}(-r_{21}, -r_{11}), \tag{1.29}$$
$$\gamma = \operatorname{atan2}(-r_{32}, -r_{33}). \tag{1.30}$$

Euler Angles ZYZ Order

Many robots, which have a spherical wrist, use a ZYZ order for the wrist axes. The purpose for this are design and manufacturing advantages. For this reason the ZYZ order is more important than the previous introduced ZYX order.

$$\mathsf{R}(\phi) = \mathsf{R}_z(\gamma)\,\mathsf{R}_{v_1}(\beta)\,\mathsf{R}_{w_2}(\alpha), \tag{1.31}$$

$$\mathsf{R} = \begin{bmatrix} c\gamma c\beta c\alpha - s\gamma s\alpha & -c\gamma c\beta s\alpha - s\gamma c\alpha & c\gamma s\beta \\ s\gamma c\beta c\alpha - c\gamma s\alpha & -s\gamma c\beta s\alpha + c\gamma c\alpha & s\gamma s\beta \\ -s\beta c\alpha & s\beta s\alpha & c\beta \end{bmatrix} . \tag{1.32}$$

[2]There would be more results but for the solution only the interval $[0, 2\pi]$ is of interest.

Inversion for ZYZ Order

The inversion problem for the ZYZ order is solved as shown for the XYZ order. Only the entries from the rotation matrix change. [16, p53-p56] [3, p48-p50] [15, p53-p56]

$0, \pi$:

$$\beta = \operatorname{atan2}(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}), \tag{1.33}$$
$$\alpha = \operatorname{atan2}(r_{32}, -r_{31}), \tag{1.34}$$
$$\gamma = \operatorname{atan2}(r_{23}, r_{13}). \tag{1.35}$$

$-\pi, 0$:

$$\beta = \operatorname{atan2}(-\sqrt{r_{13}^2 + r_{23}^2}, r_{33}), \tag{1.36}$$
$$\alpha = \operatorname{atan2}(-r_{32}, -r_{31}), \tag{1.37}$$
$$\gamma = \operatorname{atan2}(-r_{23}, r_{13}). \tag{1.38}$$

1.1.3 Roll-Pitch-Yaw Angles

The formulation of the Roll-Pitch-Yaw Angles is quite similar to the Euler angles. Contrary to the Euler angles, which rotate about an axis of the previous frame Σ_{uvw}, the Roll-Pitch-Yaw angles always rotate about the fixed frame Σ_{ZYX}. This becomes visible in the order of the matrix multiplication. In Equation 1.39 can be seen, that the order for Roll-Pitch-Yaw is γ, β, α. In contrast, the order for the Euler angles is α, β, γ. This relation is also displayed in the Figure 1.3 and Figure 1.4, which shows the results for the same value of the angles α, β, γ.

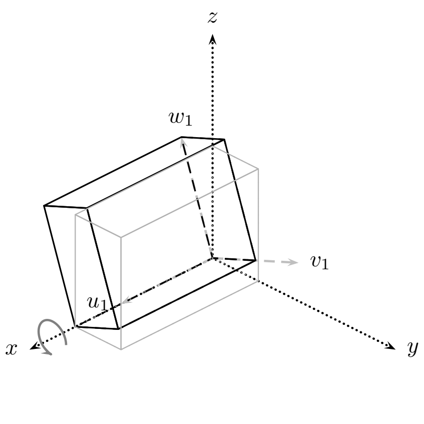
$$\mathsf{R} = \mathsf{R}_z(\gamma)\,\mathsf{R}_y(\beta)\,\mathsf{R}_x(\alpha), \tag{1.39}$$

$$\mathsf{R} = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta c\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}. \tag{1.40}$$

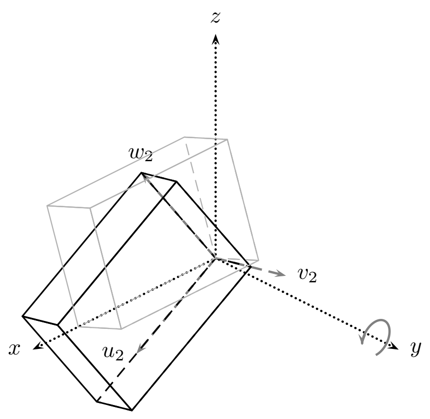
Inversion

The inverse problem for the Roll-Pitch-Yaw angles is equivalent to the inverse problem for the Euler angles. [15, p32-p33] [16, p56-p57] [3, p45-p48]

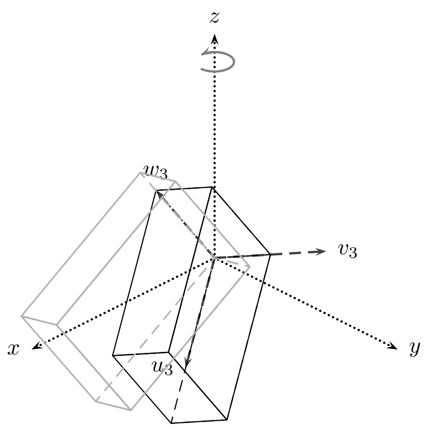
$$\mathsf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{1.41}$$



(a) Rotation about X-axis



(b) Rotation about Y-axis



(c) Rotation about Z-axis

Figure 1.4: This figure shows the three single rotations of Roll-Pitch-Yaw. It can be seen, that all these rotation are absolute rotations (compare with Figure 1.3).

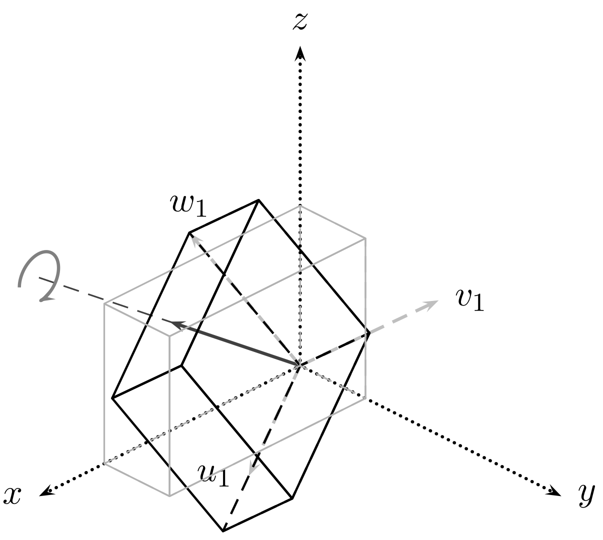


Figure 1.5: The Axis Angle rotation is a rotation about a axis, which is defined by an unit vector $\boldsymbol{r}$.

$\frac{-\pi}{2}, \frac{\pi}{2}$:

$$\beta = \text{atan2}(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}), \tag{1.42}$$

$$\alpha = \text{atan2}(r_{21}, r_{11}), \tag{1.43}$$

$$\gamma = \text{atan2}(r_{32}, r_{33}). \tag{1.44}$$

$\frac{\pi}{2}, \frac{3\pi}{2}$:

$$\beta = \text{atan2}(-r_{31}, -\sqrt{r_{11}^2 + r_{21}^2}), \tag{1.45}$$

$$\alpha = \text{atan2}(-r_{21}, -r_{11}), \tag{1.46}$$

$$\gamma = \text{atan2}(-r_{32}, -r_{33}). \tag{1.47}$$

1.1.4 Axis-Angle

The Axis-Angle concept describe a rotation about an axis, which is defined by unit vector $\boldsymbol{r}$ and the rotation angle ϑ. This is done by following steps:

1. Rotate the unit vector $\boldsymbol{r}$ in such a way, that it is aligned with the z-axis of the global frame;
2. distribute the rotation with the angle ϑ about the z-axis;
3. rotate the unit vector $\boldsymbol{r}$ back into its original configuration.

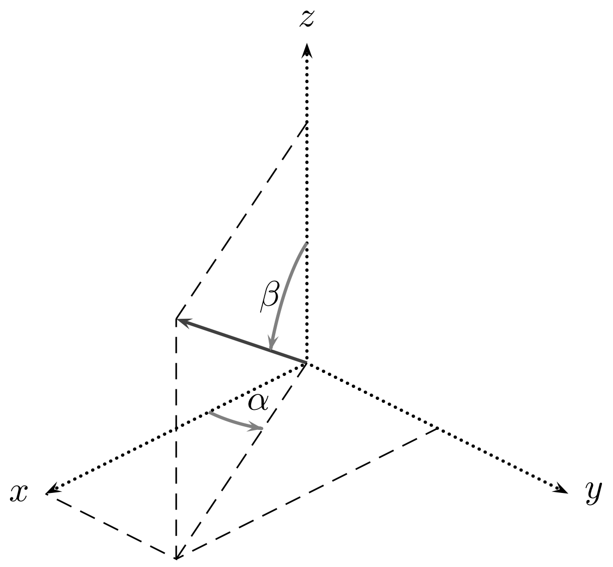


Figure 1.6: Angles of the rotation axis.

All the described steps can be seen in Equation 1.48. The first step is achieved by post multiplying with the inverse rotation matrices $\mathsf{R}_z(-\alpha), \mathsf{R}_y(-\beta)$ of the angle α, β. After that the rotation is done by multiplying $\mathsf{R}_z(\vartheta)$. The last step is obtained by pre-multiplying the rotation matrices $\mathsf{R}_y(\beta), \mathsf{R}_z(\alpha)$.

$$\mathsf{R}_{\boldsymbol{r}}(\vartheta) = \mathsf{R}_z(\alpha)\,\mathsf{R}_y(\beta)\,\mathsf{R}_z(\vartheta)\,\mathsf{R}_z(-\alpha)\,\mathsf{R}_y(-\beta) \tag{1.48}$$

The angles α, β are extracted form the unit vector $\boldsymbol{r}$. This derivation is shown in Figure 1.6 and Equations 1.49 and 1.50.

$$\sin\alpha = \frac{r_y}{\sqrt{r_x^2 + r_y^2}}, \qquad \cos\alpha = \frac{r_x}{\sqrt{r_x^2 + r_y^2}}, \tag{1.49}$$

$$\sin\beta = \sqrt{r_x^2 + r_y^2}, \qquad \cos\beta = r_z. \tag{1.50}$$

In [11, p27-p30] is shown that the matrix exponential function can be transformed into the Rodrigues formula.

$$\exp(\boldsymbol{r}\,\phi) = \mathsf{I} + \boldsymbol{r}\,\sin\phi + \boldsymbol{r}^2\,(1 - \cos\phi). \tag{1.51}$$

The Rodrigues formula is just another representation of the Axis-Angle rotation and leads to the same rotation matrix. This matrix is shown in the following equation.

$$\mathsf{R}_{\boldsymbol{r}(\vartheta)} = \begin{bmatrix} r_x^2(1-c\vartheta)+c\vartheta & r_xr_y(1-c\vartheta)-r_zs\vartheta & r_xr_z(1-c\vartheta)+r_ys\vartheta \\ r_xr_y(1-c\vartheta)+r_zs\vartheta & r_y^2(1-c\vartheta)+c\vartheta & r_yr_z(1-c\vartheta)-r_xs\vartheta \\ r_xr_z(1-c\vartheta)-r_ys\vartheta & r_yr_z(1-c\vartheta)+r_xs\vartheta & r_z^2(1-c\vartheta)+c\vartheta \end{bmatrix}. \tag{1.52}$$

$$\mathsf{R}_{-\boldsymbol{r}}(-\vartheta) = \mathsf{R}_{\boldsymbol{r}}(\vartheta). \tag{1.53}$$

Inversion

The inversion for axis angles uses the known entries of the rotation matrix R to calculate the axis $\boldsymbol{r}$ and the rotation angle ϑ. The equations for axis $\boldsymbol{r}$ and the rotation angle ϑ can be derived by comparing the entries of the rotation matrix R with the entries of the generalized rotation matrix $\mathsf{R}_{\boldsymbol{r}(\vartheta)}$ from Equation 1.52. [3, p51-p55] [15, p33-p35] [16, p52-p59]

$$\mathsf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \tag{1.54}$$

$$\vartheta = \cos^{-1}\left(\frac{r_{11} + r_{22} + r_{33} - 1}{2}\right), \tag{1.55}$$

$$\boldsymbol{r} = \frac{1}{2\sin\vartheta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}. \tag{1.56}$$

1.1.5 Unit Quaterion

Quaterion can be explained as expansion of complex numbers for the three-dimensional space. A quaterion consists of a scalar η and a vector $\boldsymbol{\epsilon}$. The square sum of a quaterion $\eta^2 + \epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 = 1$ equals one, which is the reason why it is called Unit Quaterion. In some applications, such as interpolation or calculation of multiple rotations, the quaterion notation preferred because the quaterions need less computation time than the equivalent matrix notation [17, p512]. Therefore, they are also used to describe robot kinematics [13].

$$\eta = \cos\frac{\vartheta}{2}, \tag{1.57}$$

$$\boldsymbol{\epsilon} = \sin\frac{\vartheta}{2}\boldsymbol{r}. \tag{1.58}$$

The vector $\boldsymbol{r}$ defines the rotation axis and ϑ the rotation angle. Quaterion rotations are equivalent to the Axis-Angle rotations and rotations performed by matrix exponential function.

$$\mathsf{R} = \begin{bmatrix} 2(\eta^2 + \epsilon_x^2) - 1 & 2(\epsilon_x\epsilon_y - \eta\epsilon_z) & 2(\epsilon_x\epsilon_z - \eta\epsilon_y) \\ 2(\epsilon_x\epsilon_y - \eta\epsilon_z) & 2(\eta^2 + \epsilon_y^2) - 1 & 2(\epsilon_y\epsilon_z - \eta\epsilon_x) \\ 2(\epsilon_x\epsilon_z - \eta\epsilon_y) & 2(\epsilon_y\epsilon_z - \eta\epsilon_x) & 2(\eta^2 + \epsilon_z^2) - 1 \end{bmatrix}. \tag{1.59}$$

Inversion

The inversion of the quaterions is similar to the axis-angle inversion and can be derived the same way. [11, 33-34] [3, p55-p56] [15, p35-p37] [4]

$$\eta = \frac{1}{2}\sqrt{r_{11}^2 + r_{22}^2 + r_{33}^2 + 1}, \tag{1.60}$$

$$\boldsymbol{\epsilon} = \frac{1}{2}\begin{bmatrix} \operatorname{sgn}(r_{32} - r_{23})\sqrt{r_{11}^2 - r_{22}^2 - r_{33}^2 + 1} \\ \operatorname{sgn}(r_{13} - r_{31})\sqrt{r_{22}^2 - r_{33}^2 - r_{11}^2 + 1} \\ \operatorname{sgn}(r_{21} - r_{12})\sqrt{r_{33}^2 - r_{11}^2 - r_{22}^2 + 1} \end{bmatrix}. \tag{1.61}$$

1.2 Homogeneous Transformation Matrix

Every movement in three dimensional space can be described with a translation $\boldsymbol{d}$ and a rotation R. Usually this is done by the vector equation 1.62. This form of a transformation is only suitable for small systems or manual calculations. To simplify the calculation, one dimension is added by using homogeneous coordinates, which enables a matrix form of the equation. The resulting 4×4 matrix is called homogeneous transformation matrix A.

$$\boldsymbol{p_1} = \boldsymbol{d} + \mathsf{R}_0^1 \boldsymbol{p_0}, \tag{1.62}$$

$$\begin{bmatrix} 1 \\ p_1 \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{d} & \mathsf{R}_0^1 \end{bmatrix} \begin{bmatrix} 1 \\ p_0 \end{bmatrix}, \tag{1.63}$$

$$\mathsf{A} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{d} & \mathsf{R}_1 \end{bmatrix}. \tag{1.64}$$

The transformation matrix can be separated in following parts:

$$\mathsf{A} = \left[\begin{array}{c|c} \text{scale factor} & \text{perspective transformation} \\ \hline \text{position vector} & \text{rotation matrix} \end{array}\right].$$

The position of the entries varies in literature. In this thesis the European convention is used as shown in the previous equation. In most of the literature the American convention is used, which looks as follows:

$$\mathsf{A} = \left[\begin{array}{c|c} \text{rotation matrix} & \text{position vector} \\ \hline \text{perspective transformation} & \text{scale factor} \end{array}\right].$$

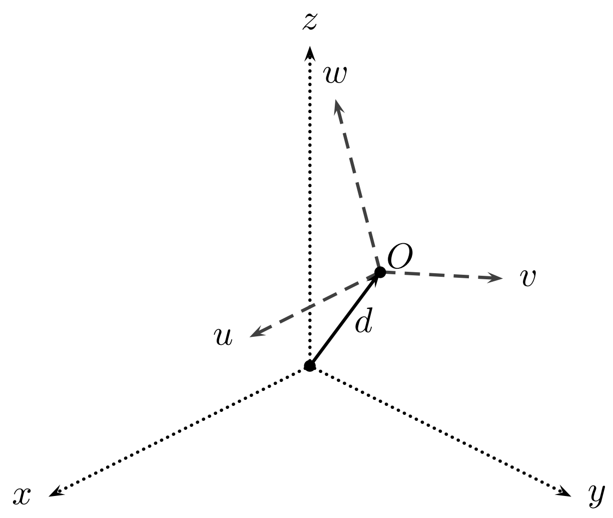


Figure 1.7: The homogeneous coordinates can be interpreted a coordinate frame, where the vector $\boldsymbol{d}$ defines the origin and the unit vectors $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ the coordinate axes.

1.2.1 Properties of the Transformation Matrix

The transformation matrix is quite similar to the rotation matrix, which sometimes leads to a mix-up. Contrary to the Rotation Matrix, the inverse of transformation matrix T can 't be achieved by transposing the matrix.

$$\mathsf{A}^{-1} \neq \mathsf{A}^{\mathrm{T}}. \tag{1.65}$$

A fast way to achieve the inverse of the transformation matrix is to use its structure, like Equation 1.66 shows. This equation can be interpreted as rotation ${\mathsf{R}_0^1}^{\mathrm{T}}$ in the opposite direction and a linear movement $-{\mathsf{R}_0^1}^{\mathrm{T}}\boldsymbol{d}$ also in the opposite direction.

$$\mathsf{A}^{-1} = \begin{bmatrix} 1 & 0 \\ -{\mathsf{R}_0^1}^{\mathrm{T}}\boldsymbol{d} & {\mathsf{R}_0^1}^{\mathrm{T}} \end{bmatrix} \tag{1.66}$$

The indices on the bottom of the transformation matrix A_0^2 indicates the start frame Σ_0 and the indices on the top the end frame Σ_2 of the transformation [15, p37-p39] [12, p45-p54] [3, p29-p31] [16, p60-p63].

$$\mathsf{A}_0^2 = \mathsf{A}_1\,\mathsf{A}_2 = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{p_1} & \mathsf{R}_1 \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{p_2} & \mathsf{R}_2 \end{bmatrix}, \tag{1.67}$$

$$\mathsf{A}_0^2 = \mathsf{A}_1\,\mathsf{A}_2 = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \mathsf{R}_1\boldsymbol{p_1} + \boldsymbol{p_1} & \mathsf{R}_1\mathsf{R}_2 \end{bmatrix}. \tag{1.68}$$

1. The homogeneous transformation matrix A moves and rotates a point from a local frame Σ_{uvw} to the global frame Σ_{ZYX};

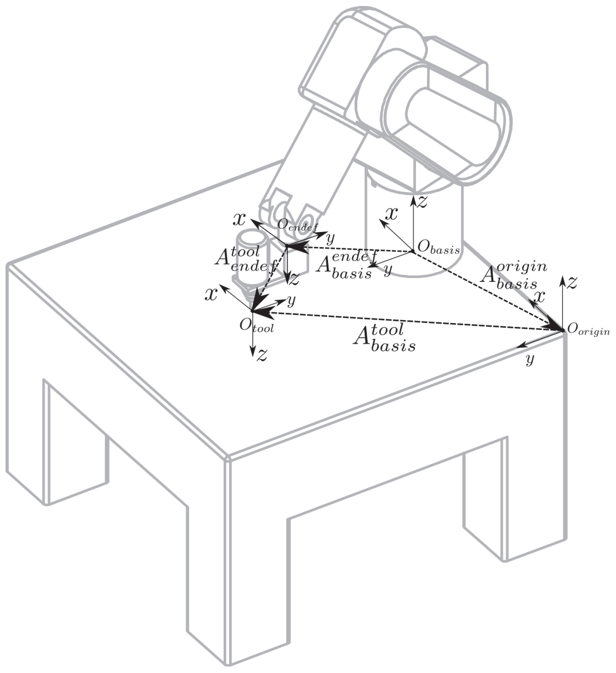


Figure 1.8: The similarity transformation can be represented as vector addition of the position vector of the origins and rotations of the frames.

2. The homogeneous transformation matrix A moves and rotates vectors;
3. The homogeneous transformation matrix A describes position and orientation of an object in three-dimensional space;
4. The homogeneous transformation matrix A represents a frame, which was moved and rotated.

pre-multiplication: Manipulates a local frame Σ_{uvw} about principal axis of the global frame Σ_{xyz}

post-multiplication: Manipulates a local frame Σ_{uvw} about principal axis of the previous local frame

1.2.2 Similarity Transformation

The Similarity Transformation is used to shift between the coordinate systems. This is shown in Figure 1.8 [3, p99-p103].

$$\mathsf{A}^{\mathsf{Tool}}_{\mathsf{Global}} = \mathsf{A}^{\mathsf{Global}^{-1}}_{\mathsf{Basis}}\, \mathsf{A}^{\mathsf{Endef}}_{\mathsf{Basis}}\, \mathsf{A}^{\mathsf{Tool}}_{\mathsf{Endef}} \tag{1.69}$$

It gets applied for following tasks:

1. Describe the tool frame in reference to other frames such as the global frame,

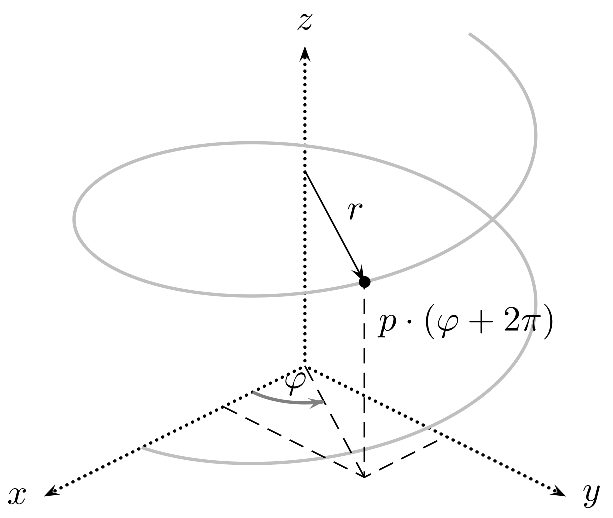


Figure 1.9: Graphical representation of a screw motion of a point.

2. describe local coordinates in the global frame;
3. describe the origin of camera frames in reference to other frames.

1.3 Screws

Another way to describe kinematic transformations in three-dimensional space is the screw [8, p303] [7]. An advantage of the screw is that it is possible to describe rotation and linear movements at the same time. Equation 1.70 shows the equation for a screw in parameter form, where p stands for the pitch and r for the radius.

$$\boldsymbol{x}(t) = \begin{bmatrix} r\cos\varphi(t) \\ r\sin\varphi(t) \\ p\,\varphi(t) \end{bmatrix}, \tag{1.70}$$

$$\boldsymbol{x}(t) = \begin{bmatrix} 0 \\ 0 \\ p\varphi(t) \end{bmatrix} + \begin{bmatrix} \cos\varphi(t) & \sin\varphi(t) & 0 \\ \sin\varphi(t) & \cos\varphi(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix}. \tag{1.71}$$

The screw equation is transformed into a homogeneous transformation matrix, which allows the usage of the homogeneous transformation properties and an easier calculation in matrix form.

$$\mathsf{A}_{\mathsf{screw}} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \frac{\varphi(t)}{2\pi} p\,\boldsymbol{d} & \mathsf{R}_{\boldsymbol{d}}(\varphi(\mathsf{t})) \end{bmatrix} \tag{1.72}$$

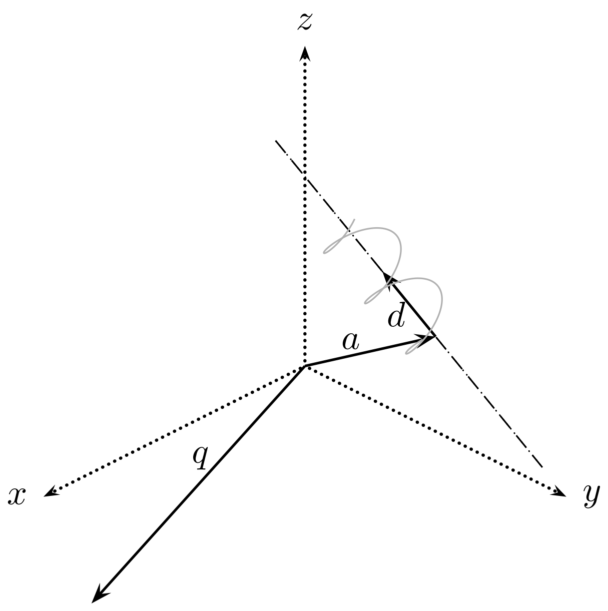


Figure 1.10: This figure shows a graphical interpretation of the Plücker-coordinates of a screw. It can be seen, that the unit vector d represents the screw axis and the vector $\boldsymbol{a}$ the distance of the footpoint.

1.3.1 Plücker Coordinates

Usually the screw axis isn't placed in the origin of the coordinate system nor is it congruent with the principal axis. To overcome this problem, a similarity transformation is performed on the original screw.

$$\mathsf{A}'_{\mathsf{screw}} = \mathsf{N}\,\mathsf{A}\mathsf{N}^{-1}, \tag{1.73}$$

$$\mathsf{A}'_{\mathsf{screw}} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{a} & \mathsf{I} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \frac{\varphi(t)}{2\pi} p\,\boldsymbol{d} & \mathsf{R}_{\boldsymbol{d}}(\varphi(\mathsf{t})) \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ -\boldsymbol{a} & \mathsf{I} \end{bmatrix}, \tag{1.74}$$

$$\mathsf{A}'_{\mathsf{screw}} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \frac{\varphi(t)}{2\pi} p\,\boldsymbol{d} + (\mathsf{I} - \mathsf{R}_{\boldsymbol{d}}(\varphi(\mathsf{t})))\boldsymbol{a} & \mathsf{R}_{\boldsymbol{d}}(\varphi(\mathsf{t})) \end{bmatrix}. \tag{1.75}$$

The vector $\boldsymbol{q}$ is the crossproduct of the position vector $\boldsymbol{a}$ and the direction vector $\boldsymbol{d}$ of the screw axis. The vector $\boldsymbol{q}$ and the direction vector $\boldsymbol{d}$ are combined to the so called Plücker coordinates $[\boldsymbol{d}, \boldsymbol{q}]$. This coordinates frequently appear in calculation of velocities and static torques.

$$\boldsymbol{q} = \boldsymbol{a} \times \boldsymbol{d} \tag{1.76}$$

1.3.2 Inversion of a Screw

In many cases in practical work is the inversion of the screw needed. The inversion starts with a comparison of the entries $\boldsymbol{t}$ and R of A with the entries from the screw matrix $\mathsf{A}'_{\mathsf{screw}}$.The first step is to get the rotation angles φ and the rotation axis $\boldsymbol{d}$ from the rotation matrix $\mathsf{R}_{\boldsymbol{d}}$, the inversion of Axis-Angle rotation is used.

$$\mathsf{A} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{t} & \mathsf{R} \end{bmatrix}. \tag{1.77}$$

$$\boldsymbol{t} = \frac{\varphi}{2\pi} p\, \boldsymbol{d} + (\mathsf{I} - \mathsf{R}_{\boldsymbol{d}})\boldsymbol{a}, \tag{1.78}$$

$$\boldsymbol{d}\,(\mathsf{I} - \mathsf{R}_{\boldsymbol{d}})\,\boldsymbol{a} = \boldsymbol{d}\,(\boldsymbol{t} - \frac{\varphi}{2\pi} p\, \boldsymbol{d}), \tag{1.79}$$

$\mathsf{R}_{\boldsymbol{d}}\,\boldsymbol{d} = \boldsymbol{d}$ this is caused by the fact that $\boldsymbol{d}$ is the rotation axis and rotation axes are never effected by the rotation. To enable a further simplification; the position vector $\boldsymbol{a}$ has to be chosen that way that it is orthogonal to the direction vector $\boldsymbol{d}$, which leads to: $\boldsymbol{d}\,\boldsymbol{a} = 0$. This means also that $\mathsf{R}_{\boldsymbol{d}}\,\boldsymbol{a}$ and $\boldsymbol{d}$ are orthogonal.

$$0 = \boldsymbol{d}\,(\boldsymbol{t} - \frac{\varphi}{2\pi} p\, \boldsymbol{d}), \tag{1.80}$$

$$\boldsymbol{t} = \frac{\varphi}{2\pi} p\, \boldsymbol{d}, \tag{1.81}$$

$$p = \frac{2\pi}{\varphi}(\boldsymbol{d}\,\boldsymbol{t}). \tag{1.82}$$

Chapter 2

Denavit-Hartenberg Convention

This chapter introduces two methods for a systematic description for an open kinematic chain. A kinematic chain is a mechanical structure, where every joint is connected to the next by a variable link. The first method is the Denavit-Hartenberg convention, which can be found in literature. The second method is the screw method, which is based on screws.

2.1 Denavit-Hartenberg Convention

The Denavit-Hartenberg convention is one of the most simple methods to describe a kinematic chain. This method only needs four scalar parameters to describe a joint. One of these parameters varies with the joint motion. This parameter is called joint variable and is usually the parameter d for prismatic joints or for revolute joints the parameter θ. The four parameter have the following meanings:

d_i . . . Coordinate of origin O_i along the z axis z_{i-1} of the previous frame,

a_i . . . Distance between origin O_i and origin O_{i-1},

α_i . . . Angle between axis z_{i-1} and axis z_i about the axis x_i,

θ_i . . . Angle between axis x_{i-1} and axis x_i about the axis z_{i-1}.

These four parameter describe how a coordinate frame Σ has to be rotated and moved to achieve the next frame. Figure 2.1 shows a graphical representation of the parameters. This is done by matrix multiplication and is shown in the following Equation.

$$\begin{aligned}
\mathsf{A} &= \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ 0 & \mathsf{R}_{\boldsymbol{x}}(\alpha) \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{d} & \mathsf{I} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{a} & \mathsf{I} \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ 0 & \mathsf{R}_{\boldsymbol{z}}(\theta) \end{bmatrix}, \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},
\end{aligned}$$

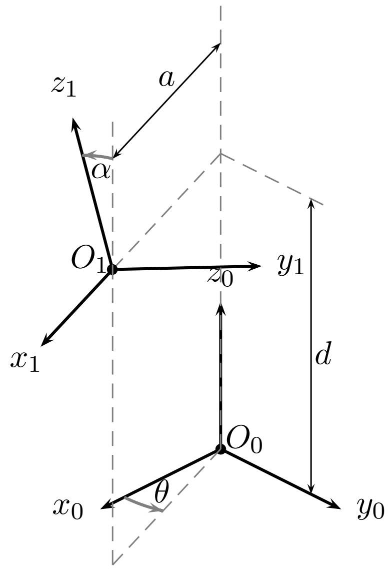


Figure 2.1: Denavit-Hartenberg paramter of a additional coordinate frame.

To achieve a readable form, it is common to write the result of the previous equation as two matrices. The first one contains all static Denavit-Hartenberg parameters and the second one contains the joint variable. [15, p42-p46] [16, p76-p89]

$$\begin{aligned} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ 0 & 0 & \cos(\alpha) & -\sin(\alpha) \\ d & 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}, \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ a\,\cos\theta & \cos\theta & -\cos\alpha\,\sin\theta & \sin\alpha\,\sin\theta \\ a\,\sin\theta & \sin\theta & \cos\alpha\,\cos\theta & -\cos\theta\,\sin\alpha \\ d & 0 & \sin\alpha & \cos\alpha \end{bmatrix}. \end{aligned} \tag{2.1}$$

The Denavid-Hartenberg parameters for a serial kinematic chain can be extracted with the following procedure [15, p43]:

1. Define global coordinate frame;
2. Define reference configuration for example homing position;
3. Choose axis z_i along the joint axis;
4. Locate the origin O_i at the intersection with the common normal to axis z_{i-1} and z_i;

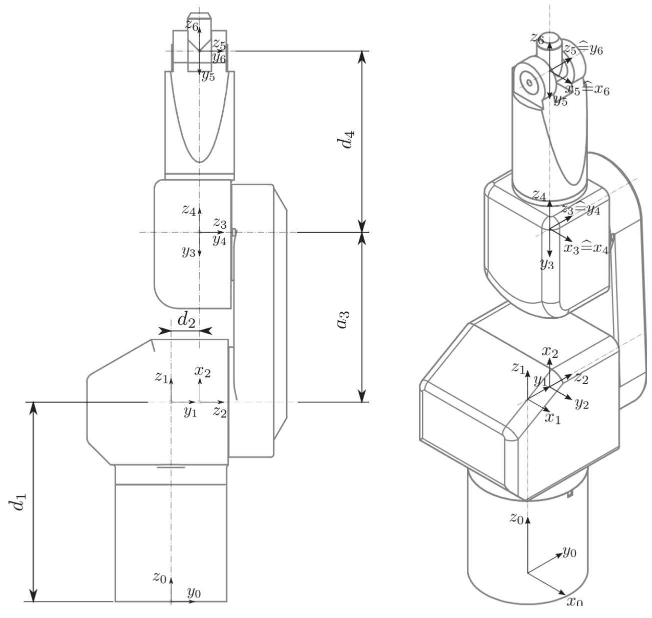
5. Choose x_i along the common normal to axis z_i with direction to joint $i + 1$;
6. Choose y_i to complete the right-handed frame;
7. Determine homogeneous transformation for the joint.

There are some cases, where the rules of this convention aren't clearly defined. This exceptions are shown in the following list:

1. Two consecutive axes are parallel $\longrightarrow$ origin O_i can be freely chosen;
2. Two consecutive axes are intersect $\longrightarrow$ axis x_i is arbitrary;
3. Joint i is prismatic $\longrightarrow$ direction of axis z_{i-1} is arbitrary;
4. In frame 0 only z_0 is specified $\longrightarrow$ origin O_0 and axis x_0 can be freely choose;
5. Last frame n z_n is not uniquely defined while x_n is normal to z_{n-1}.

In order to reduce the complexity of the calculation, it is intended to reduce the number of the non-zero Denavit-Hartenberg parameters. A possible way is to select the free-settable parameter in a way, that most of the following parameters get zero.

2.1.1 Denavit-Hartenberg-Parameters for Stäubli RX60



(a) Front view (b) Isometric view

Figure 2.2: This figure shows the single coordinate frames, which are used to extract the Denavit-Hartenberg parameters for Stäubli RX60 .

The search of the Denavit-Hartenberg parameter is started with the definition of the initial frame Σ_0.

$\Sigma_0 to \Sigma_1$: Frame Σ_1 by shifting the origin of frame Σ_0 along the z-axis to the intersection of the first two axes. The distance d_1, which the frame Σ_0 was shifted, is the first parameter.

$$\mathsf{A}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 & 0 \\ 0 & \sin\theta_1 & \cos\theta_1 & 0 \\ d_1 & 0 & 0 & 1 \end{bmatrix}. \tag{2.2}$$

$\Sigma_1 to \Sigma_2$: In order to enable a transformation according to the Denavit-Hartenberg convention, a rotation of the frame Σ_1 is necessary. The frame Σ_1 is rotated α_1 about the x-axis and θ_1 about the z-axis. Afterwards it is shifted about d_2 along the z-axis to achieve the frame Σ_2. The distance d_2 is free-settable because the axis 2 and axis 3 are parallel.

$$\mathsf{A}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \sin\theta_2 & \cos\theta_2 & 0 \\ d_2 & 0 & 0 & 1 \\ 0 & \cos\theta_2 & -\sin\theta_2 & 0 \end{bmatrix}. \tag{2.3}$$

$\Sigma_2 to \Sigma_3$: Frame Σ_3 is achieved by shifting the frame Σ_2 along the x-axis and rotate it θ_3 about the z-axis.

$$\mathsf{A}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a_3 & -\sin\theta_3 & -\cos\theta_3 & 0 \\ 0 & \cos\theta_3 & -\sin\theta_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.4}$$

$\Sigma_3 to \Sigma_4$: Frame Σ_4 is achieved by rotating frame Σ_3 α_3 about the x-axis and

$$\mathsf{A}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_4 & \sin\theta_4 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & \sin\theta_4 & \cos\theta_4 & 0 \end{bmatrix}. \tag{2.5}$$

$\Sigma_4 to \Sigma_5$: frame Σ_5 is achieved by translating frame Σ_4 d_4 along the z-axis and rotate it α_5 about the x-axis.

$$\mathsf{A}_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_5 & -\sin\theta_5 & 0 \\ 0 & 0 & 0 & 1 \\ d_4 & -\sin\theta_5 & -\cos\theta_5 & 0 \end{bmatrix}. \tag{2.6}$$

$\Sigma_5 to \Sigma_6$: The last frame Σ_6 is calculated by rotating frame Σ_5 α_6 about the x-axis

$$\mathsf{A}_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_6 & \sin\theta_6 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & \sin\theta_6 & \cos\theta_6 & 0 \end{bmatrix}. \tag{2.7}$$

All these parameters for the Stäubli RX60 are given in Table 2.1.

joint	d_i	a_i	α_i	θ_i
1	d_1	0	0	θ_1
2	d_2	0	-90	$\theta_2 - 90$
3	0	a_3	0	$\theta_3 + 90$
4	0	0	90	θ_4
5	d_5	0	-90	θ_5
6	0	0	90	θ_6

Table 2.1: Denavit-Hartenberg-parameters for Stäubli RX60 .

The designer of Stäubli RX60 chose the geometric parameter in a way, that the kinematic equations gets as simple as possible.

2.2 Screw-Method

The screw method is an alternative method to the Denavit-Hartenberg convention. The screw parameters are easier to evaluate than the Denavit-Hartenberg parameters, where a special orientation of the previous coordinate frame is necessary to achieve the next frame.The screw method is also more flexible in its usage [14]. This allows less experienced users to describe the kinematics.

The screw parameters are extracted with the following procedure:

1. Choose fixed coordinate system;
2. Define reference configuration for example homing position;
3. Identify screw axis $\boldsymbol{d}$ and distance $\boldsymbol{a}$ for each joint also identify the joint variable;
4. Determine homogeneous transformation matrix for each joint;

The transformation matrix is calculated by the following formula:

$$\mathsf{A} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \frac{\varphi}{2\pi} t\,\boldsymbol{d} + (\mathsf{I} - \mathsf{R}_{\boldsymbol{d}}(\theta))\boldsymbol{a} & \mathsf{R}_{\boldsymbol{d}}(\theta) \end{bmatrix}. \tag{2.8}$$

Chapter 3

Forward Kinematics

This chapter introduces the forward kinematics and how it can be achieved using the transformation matrix of the Denavit-Hartenberg convention or the screw method. The second topic of this chapter is the forward kinematics for a 6 degree of freedom manipulator with spherical wrist.

The forward kinematic [1] is a mathematical method to calculate the position and orientation of a kinematic chain from the joint variables. For a better separation T is used instead of A for the transformation matrix of a kinematic chain.

$$\mathsf{T}_0^6 = \mathsf{A}_1\,\mathsf{A}_2\,\mathsf{A}_3\,\mathsf{A}_4\,\mathsf{A}_5\,\mathsf{A}_6, \tag{3.1}$$

$$\mathsf{T}_0^6 = \mathsf{A}_1 \begin{bmatrix} 1 & 0 & 0 & 0 \\ p_x & n_x & s_x & a_x \\ p_y & n_y & s_y & a_y \\ p_z & n_z & s_x & a_z \end{bmatrix}. \tag{3.2}$$

position vector: $\boldsymbol{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$, normal vector: $\boldsymbol{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}$,

sliding vector: $\boldsymbol{s} = \begin{bmatrix} s_x \\ s_y \\ s_z \end{bmatrix}$, approach vector: $\boldsymbol{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$.

3.1 Articulated Manipulator with Spherical Wrist

This kind of manipulator has a spherical wrist, which is achieved by an intersection of the last three joint axes. Any movement of the last three joints only changes the orientation of the end effector but not the position. Now the forward kinematic can be separated in two problems, which is used as an important simplification for the inverse problem.

[1]Sometimes the name direct kinematics is used for the forward kinematics.

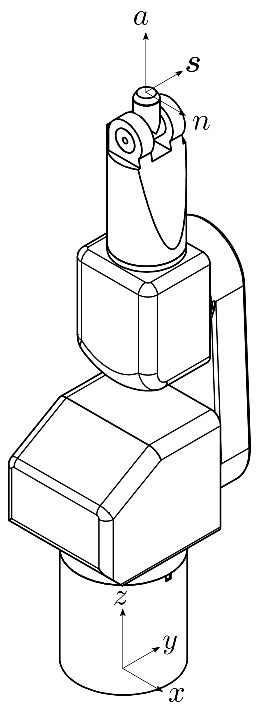


Figure 3.1: Home position of the Stäubli RX60 .

The following equations show the result of the forward kinematic for a Stäubli RX60 robot. The equation for $\boldsymbol{n}$ and $\boldsymbol{s}$ are simplified using the angle sum:

$$\cos(\theta_1 + \theta_2) = \cos\theta_1\,\cos\theta_2 - \sin\theta_1\,\sin\theta_2, \tag{3.3}$$

$$\sin(\theta_1 + \theta_2) = \cos\theta_1\,\sin\theta_2 + \sin\theta_1\,\cos\theta_2. \tag{3.4}$$

In order to improve the readability, the cosine and sine function are substituted by $\cos\theta_i \triangleq c_i$ and $\sin_i \triangleq s_i$.

$$\boldsymbol{p} = \begin{bmatrix} d_4\,(c_1\,c_2\,s_3 + c_1\,c_3\,s_2) - d_2\,s_1 + a_2\,c_1\,s_2 \\ d_4\,(c_2\,s_1\,s_3 + c_3\,s_1\,s_2) + d_2\,c_1 + a_2\,s_1\,s_2 \\ d_1 + d_4\,c_{23} + a_2\,c_2 \end{bmatrix}, \tag{3.5}$$

$$\boldsymbol{n} = \begin{bmatrix} -c_6\,(s_5\,c_1\,s_{23} - c_5\,(c_4\,c_1\,c_{23} - s_1\,s_4)) - s_6\,(s_4\,c_1\,c_{23} + c_4\,s_1) \\ -c_6\,(s_5\,s_1\,s_{23} - c_5\,(c_4\,s_1\,c_{23} + c_1\,s_4)) - s_6\,(s_4\,s_1\,s_{23} - c_1\,c_4) \\ c_6\,(s_5\,(s_2\,s_3 - c_2\,c_3) - c_4\,c_5\,s_{23}) + s_4\,s_6\,s_{23} \end{bmatrix}, \tag{3.6}$$

$$\boldsymbol{s} = \begin{bmatrix} s_6\,(s_5\,c_1\,s_{23} - c_5\,(c_4\,c_1\,c_{23} - s_1\,s_4)) - c_6\,(s_4\,c_1\,c_{23} + c_4\,s_1) \\ s_6\,(s_5\,s_1\,s_{23} - c_5\,(c_4\,s_1\,c_{23} + c_1\,s_4)) - c_6\,(s_4\,s_1\,c_{23} - c_1\,c_4) \\ c_6\,s_4\,s_{23} - s_6\,(s_5\,(s_2\,s_3 - c_2\,c_3) - c_4\,c_5\,s_{23}) \end{bmatrix}, \tag{3.7}$$

$$\boldsymbol{a} = \begin{bmatrix} c_5\,(c_1\,c_2\,s_3 + c_1\,c_3\,s_2) + s_5\,(c_4\,(c_1\,c_2\,c_3 - c_1\,s_2\,s_3) - s_1\,s_4) \\ c_5\,(c_2\,s_1\,s_3 + c_3\,s_1\,s_2) + s_5\,(c_4\,(c_2\,c_3\,s_1 - s_1\,s_2\,s_3) + c_1\,s_4) \\ -c_5\,(s_2\,s_3 - c_2\,c_3) - c_4\,s_5\,(c_2\,s_3 + c_3\,s_2) \end{bmatrix}. \tag{3.8}$$

Chapter 4

Inverse Kinematics

This chapter presents methods for solution of the inverse kinematics.

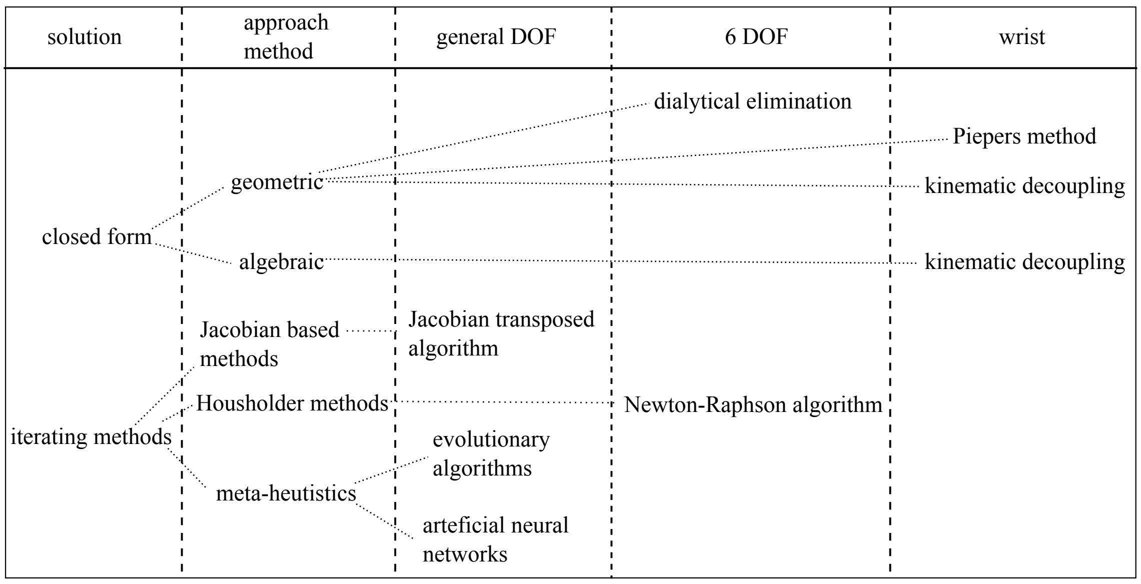


Figure 4.1: Overview of solution methods for the inverse kinematics.

4.1 Iterating Methods

Figure 4.1 shows, that there are many different ways to solve the inverse kinematics. The iterating methods are one of them. One dominant group of the iterating methods are the meta-heuristics. The advantage of the meta-heuristics are, that the solvability of the inverse kinematics doesn't depend on the degree of freedom but they only deliver one solution of all possible. The most common meta-heuristic algorithms for robots are the Artificial Neural Networks [5] [6] [10], the evolutionary algorithm [9]. Another group of the iterating algorithms are the Jacobian based methods. These methods use the inverse of the Jacobian to solve the inverse kinematics. The use of the inverse of the Jacobian is risky, because of the singularity of the Jacobian at special robot configurations, but there are algorithms, which cope with this problem [15, p104-p110].

4.2 Closed Form Solutions

Closed form solutions for the inverse kinematics are preferred in robotic. This solutions have the advantage, that their behaviour can be predicted [16, p95]. Another aspect of the closed form solution is that all possible solutions are calculated and it is possible to choose the most suitable one. For a general six degree of freedom serial robot there exist sixteen possible solutions [1, p288-p289] for the inverse kinematics. The number of solutions can be reduced by special arrangement of the axes. Most of the closed form solutions are only practicable for these small group of 6 degree of freedom robots.

The following examples are based on the articulated manipulator Stäubli RX60. This robot type has in general eight possible solutions for the inverse kinematics but not all of these solutions are physically practicable as Figure 4.2.2 shows.

4.2.1 Kinematic Decoupling

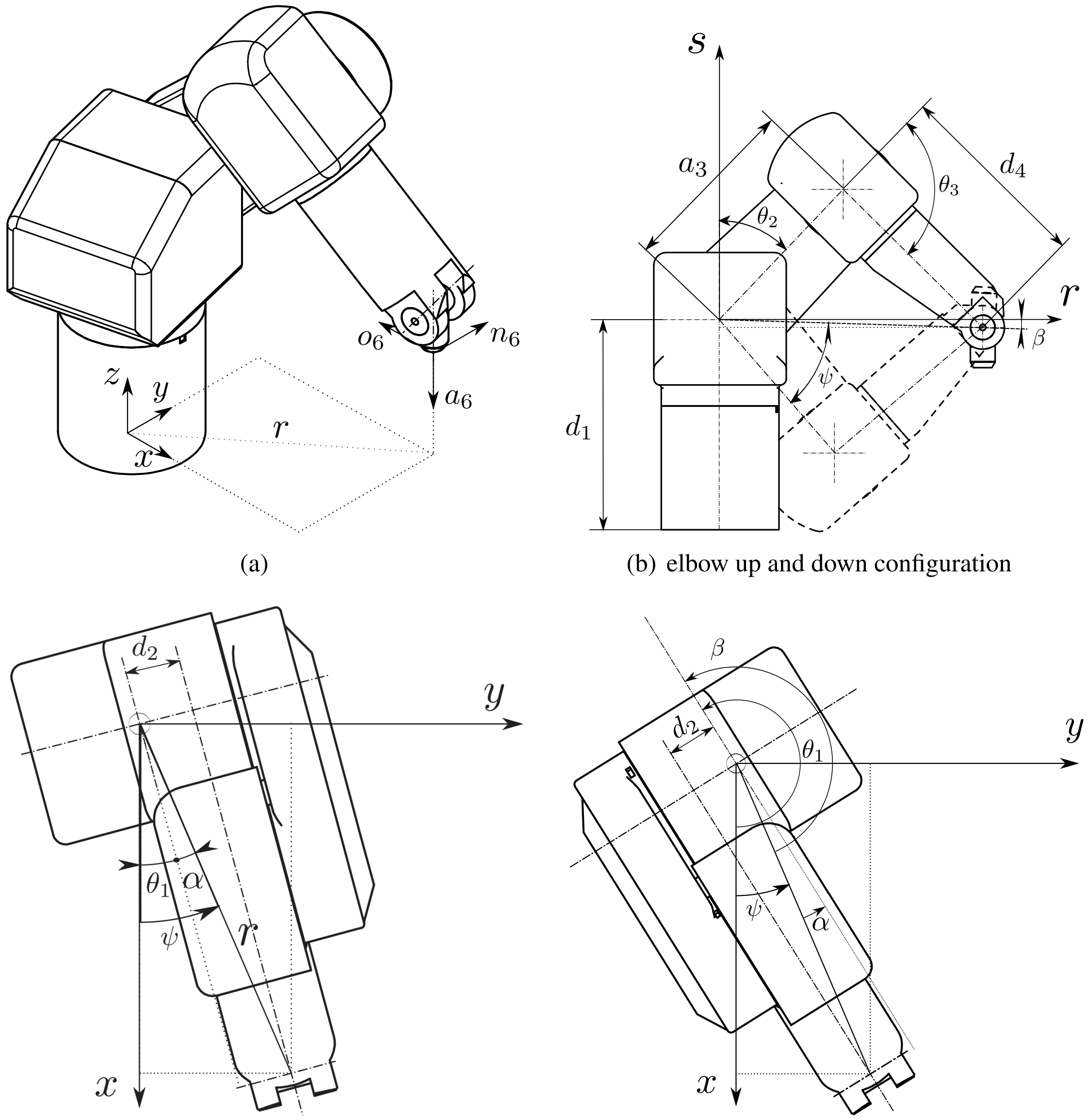
The kinematic decoupling uses the intersection of three axes in one point to divide the inverse kinematics problem into two sub-problems. This two sub-problems are the position and the orientation. There are two approaches to solve the position problem. The geometric approach [3, 126-129] considers the inverse kinematics problem as a geometric problem, which can be solved with geometric concepts. The algebraic approach [3, p122-p126] considers the inverse kinematics as an algebraic problem , which can be solved by equation systems. A clear separation of the two approaches is difficult because of the geometric nature of the problem, which always allows a geometric interpretation. The choice of the approach depends on the user and his knowledge in this areas.

4.2.2 Geometric Approach

The geometric approach uses trigonometric functions, such as the law of cosine, to extract the joint parameters. The robot in the example has an additional shoulder offset d_2. This offset has the purpose to avoid the shoulder singularity. Singularities are arm configurations, where the robot loses at least one of its six possibilities to move. This improvement leads to a more difficult inverse kinematics than it would be without shoulder offset.

Left Arm Configuration

The geometry of the robot allows to reduce the three dimensional problem into a two dimensional problem in the xy-plane. The shoulder offset d_2 causes the demand of additional angles ψ and α to calculate the angle θ_1. The first angle ψ is calculated form p_x and p_y of the position vector. The



(a) (b) elbow up and down configuration

(c) left arm configuration (d) right arm configuration

Figure 4.2: Geometric solution for the inverse kinematics but not all solutions are physically possible.

second angle α is calculated from $r = \sqrt{p_x^2 + p_y^2}$ and the shoulder offset d_2. In Figure 4.2.2 can be seen, that θ_1 can be achieved by subtracting α from ψ.

$$\psi = \text{atan2}(p_x, p_y), \tag{4.1}$$

$$\alpha = \text{atan2}(\sqrt{r^2 - d_2^2}, d_2), \tag{4.2}$$

$$\theta_1 = \psi - \alpha. \tag{4.3}$$

Right Arm Configuration

The solution for the right arm configuration is quite similar to the left arm configuration, except one more additional angle β is needed.

$$\psi = \text{atan2}(x, y), \tag{4.4}$$

$$\alpha = \text{atan2}(\sqrt{r^2 - d_2^2}, d_2), \tag{4.5}$$

$$\beta = \alpha + \pi, \tag{4.6}$$

$$\theta_1 = \beta + \psi. \tag{4.7}$$

Elbow Configurations

For the calculation of the angles θ_2 and θ_3 is the origin coordinate system translated to the intersection of the first two axis. The coordinate system itself is rotated by an angle θ_1 about the z-axis. This new coordinate system allows a simplification to a two dimensional problem. Figure 4.2.2 shows, that the position vector and the connection lines between the axes form a triangle. This triangle can be used to calculate the angle θ_3 by using the cosine law and the inverse of the tangens-two. For the angle θ_2 two additional angles β and ψ are needed. The angle β is calculated from the coordinates r and s. The second angle ψ is calculated by using again the cosine law and the inverse of the tangens-two. Depending on the sign of the coordinate s, θ_2 is calculated by addition or subtraction of β from ψ. [16, p97-p104]

$$D = \cos(180 - \theta_3) = \frac{s^2 + r^2 - a_3^2 - d_4^2}{2\, a_3\, d_4} \tag{4.8}$$

$$\theta_3 = \text{atan2}(\pm\sqrt{1 - D^2}, D) \tag{4.9}$$

$$\beta = \operatorname{atan2}(s, r), \tag{4.10}$$

$$C = \cos\psi = \frac{r^2 + s^2 + a_3^2 - d_4^2}{2\sqrt{r^2 + s^2} a_3}, \tag{4.11}$$

$$\psi = \operatorname{atan2}(\sqrt{1 - C^2}, C), \tag{4.12}$$

$s > 0$:

$$\theta_2 = \psi - \beta. \tag{4.13}$$

$s < 0$:

$$\theta_2 = \psi + \beta. \tag{4.14}$$

Alternative Solution The alternative solution is based on the idea, that every rotating joint describes a circle. This implies, that the link between two joints can only be placed in the intersections of the circles of the two joints.

$$s^2 + r^2 = a_3^2, \tag{4.15}$$
$$(s - p_s)^2 + (r - p_r)^2 = d_4^2, \tag{4.16}$$
$$2\,p_s\,s + 2\,p_r\,r - p_r^2 - p_s^2 = a_3^2 - d_4^2. \tag{4.17}$$

$$\theta_2 = \operatorname{atan2}(s_3, r_3), \tag{4.18}$$
$$\psi = \operatorname{atan2}(p_s - s_3, p_r - r_3), \tag{4.19}$$
$$\theta_3 = \psi - \theta_2. \tag{4.20}$$

4.2.3 Algebraic Approach

The algebraic approach also uses the kinematic decoupling to simplify the inverse kinematics. The decoupling allows a splitting of the transformation matrix T_0^6 into matrices. The first matrix T_0^3 defines the position and the second matrix T_3^6 the orientation of the robots wrist.

$$\mathsf{T}_0^3 = \mathsf{A}_1\,\mathsf{A}_2\,\mathsf{A}_3, \tag{4.21}$$
$$\mathsf{T}_3^6 = \mathsf{A}_4\,\mathsf{A}_5\,\mathsf{A}_6, \tag{4.22}$$
$$\mathsf{A}_{\mathsf{des}} = \mathsf{T}_0^3\,\mathsf{T}_3^6, \tag{4.23}$$
$$\begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{p_{des}} & \mathsf{R}_{\mathsf{des}} \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{p_0^3} & \mathsf{R}_0^3 \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^{\mathrm{T}} \\ \boldsymbol{p_3^6} & \mathsf{R}_3^6 \end{bmatrix}. \tag{4.24}$$

The rotation matrix R_0^3 can be expressed as a matrix multiplication of the rotation matrices of the Denavid-Hartenberg convention. The same relation also works for the position. The resulting equation only has the variable parameters θ_1, θ_2 and θ_3. All parameters, which are related to the first axis, are shifted to the left side of the equation. The left side of the equation describe the arm configuration problem and the right side the elbow configuration problem. The following algorithm is a slight modification of the algorithm described in [8, p428-p436].

$$\begin{aligned} \boldsymbol{p_{des}} &= \boldsymbol{p_0^3} + \mathsf{R}_0^3\, \boldsymbol{p_3^6}, & (4.25)\\ \mathsf{R}_{\mathsf{des}} &= \mathsf{R}_0^3\, \mathsf{R}_3^6, & (4.26)\\ \mathsf{R}_0^3 &= \mathsf{R}_{\boldsymbol{x}}(\alpha_1)\, \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \mathsf{R}_{\boldsymbol{z}}(\theta_2)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_3)\, \mathsf{R}_{\boldsymbol{z}}(\theta_3). & (4.27) \end{aligned}$$

$$\begin{aligned} \boldsymbol{p_{des}} = \mathsf{R}_{\boldsymbol{x}}(\alpha_1)\, \{&\boldsymbol{p_1}\\ &+ \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \boldsymbol{p_2}\\ &+ \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \mathsf{R}_{\boldsymbol{z}}(\theta_2)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_3)\boldsymbol{p_3}\\ &+ \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \mathsf{R}_{\boldsymbol{z}}(\theta_2)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_3)\, \mathsf{R}_{\boldsymbol{z}}(\theta_3)\, \boldsymbol{p_4}\}. \qquad (4.28) \end{aligned}$$

$$\begin{aligned} \mathsf{R}_{\boldsymbol{x}}(\alpha_1)^{\text{-1}}\, \boldsymbol{p_{des}} &= \boldsymbol{p_1}\\ &+ \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \boldsymbol{p_2}\\ &+ \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \mathsf{R}_{\boldsymbol{z}}(\theta_2)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_3)\, \boldsymbol{p_3}\\ &+ \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \mathsf{R}_{\boldsymbol{z}}(\theta_2)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_3)\, \mathsf{R}_{\boldsymbol{z}}(\theta_3)\, \boldsymbol{p_4}, \qquad (4.29) \end{aligned}$$

substitute $\boldsymbol{p_3} + \mathsf{R}_{\boldsymbol{z}}(\theta_3)\, \boldsymbol{p_4}$ with $\tilde{\boldsymbol{p}}$

$$\begin{aligned} \mathsf{R}_{\boldsymbol{x}}(\alpha_1)^{\text{-1}}\, \boldsymbol{p_{des}} &= \boldsymbol{p_1}\\ &+ \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \boldsymbol{p_2}\\ &+ \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \mathsf{R}_{\boldsymbol{z}}(\theta_2)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_3)\, \tilde{\boldsymbol{p}}, \qquad (4.30) \end{aligned}$$

$$\begin{aligned} \mathsf{R}_{\boldsymbol{x}}(\alpha_1)^{\text{-1}}\, \boldsymbol{p_{des}} - \boldsymbol{p_1} &= \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \boldsymbol{p_2}\\ &+ \mathsf{R}_{\boldsymbol{z}}(\theta_1)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \mathsf{R}_{\boldsymbol{z}}(\theta_2)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_3)\, \tilde{\boldsymbol{p}}, \qquad (4.31) \end{aligned}$$

$$\mathsf{R}_{\boldsymbol{z}}(\theta_1)^{\text{-1}} \left(\mathsf{R}_{\boldsymbol{x}}(\alpha_1)^{\text{-1}}\, \boldsymbol{p_{des}} - \boldsymbol{p_1}\right) = \mathsf{R}_{\boldsymbol{x}}(\alpha_2)\, \left(\boldsymbol{p_2} + \mathsf{R}_{\boldsymbol{z}}(\theta_2)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_3)\, \tilde{\boldsymbol{p}}\right). \qquad (4.32)$$

The result of the previous equations with the parameters of the Stäubli RX60 look as follows:

$$\begin{bmatrix} p_x \cos\theta_1 + p_y \sin\theta_1 \\ p_y \cos\theta_1 - p_x \sin\theta_1 \\ p_z - d_1 \end{bmatrix} = \begin{bmatrix} d_4 \sin(\theta_2+\theta_3) + a_3 \sin\theta_2 \\ d_2 \\ d_4 \cos(\theta_2+\theta_3) + a_3 \cos\theta_2 \end{bmatrix} \tag{4.33}$$

The z-entry of the previous equation provides the first equation for the inverse kinematics of the angles θ_2 and θ_3. The Euclidean norm of this equation delivers the second equation.

$$\|\mathsf{R}_{\boldsymbol{z}}(\theta_1)^{\text{-1}} \left(\mathsf{R}_{\boldsymbol{x}}(\alpha_1)^{\text{-1}}\, \boldsymbol{p_{des}} - \boldsymbol{p_1}\right) \|^2 = \|\mathsf{R}_{\boldsymbol{x}}(\alpha_2)\ (\boldsymbol{p_2} + \mathsf{R}_{\boldsymbol{z}}(\theta_2)\, \mathsf{R}_{\boldsymbol{x}}(\alpha_3)\, \tilde{\boldsymbol{p}})\, \|^2 \tag{4.34}$$

$$\sqrt{{d_1}^2 - 2\, d_1\, p_z + {p_x}^2 + {p_y}^2 + {p_z}^2} = \sqrt{a_3^2 + 2\, a_3\, d_4 \cos\theta_3 + d_2^2 + d_4^2} \tag{4.35}$$

For a general robot with wrist, there would be some trigonometric substitutions necessary to calculate the angles θ_2 and θ_3. The structure of Stäubli RX60 simplifies the equations and enables to calculate θ_3. The same fact is also valid for θ_1,which is calculated for the y-entry of Equation 4.33.

$$p_z - d_1 - a3 \cos\theta_2 - d_4 \cos\theta_2 \cos\theta_3 + d_4 \sin\theta_2 \sin\theta_3 = 0 \tag{4.36}$$

$$d_1^2 - 2\, d_1\, p_z + p_x^2 + p_y^2 + p_z^2 - a_3^2 - 2\, a_3\, d_4 \cos\theta_3 - d_2^2 - d_4^2 = 0 \tag{4.37}$$

$$p_y \cos\theta_1 - p_x \sin\theta_1 - d_2 = 0 \tag{4.38}$$

As simplification the half-tangens substitution can be used:

$$t = \tan\frac{\theta}{2}, \tag{4.39}$$

$$\cos\theta = \frac{1-t^2}{1+t^2}, \tag{4.40}$$

$$\sin\theta = \frac{2t}{1+t^2}. \tag{4.41}$$

4.2.4 Orientation

The kinematic decoupling enables the separation of the rotation matrix R_0^6 into two rotation matrices. The first rotation matrix R_0^3 describes the orientation of the wrist joint and is calculated from the results of the inverse kinematics for the position. This implies, that the inverse of the position has been solved. The second matrix R_0^3 is the unknown rotation matrix for the orientation. The orientation of the matrix R_0^6 is known because it has to be equal with the desired orientation of $\mathsf{R}_{\mathsf{des}}$. This fact allows to calculate the unknown rotation matrix R_3^6 by pre multiplying R_0^6 with the inverse of R_0^3. [1]

[1]Remember that the rotation matrix is a orthonormal matrix ($\mathsf{R}^\mathsf{T}\mathsf{R} = \mathsf{I}$).

$$\mathsf{R}_0^6 = \mathsf{R}_0^3\,\mathsf{R}_3^6, \tag{4.42}$$

$$\mathsf{R}_0^{3\text{-}1}\,\mathsf{R}_0^6 = \mathsf{R}_3^6, \tag{4.43}$$

$$\mathsf{R}_0^{3\text{-}1}\,\mathsf{R}_{\mathsf{des}} = \mathsf{R}_3^6. \tag{4.44}$$

The result is the rotation matrix R_3^6 which is equal to the rotation matrix for the Euler-angles. The last three joint angles can be calculated by using the inversion formula of the Euler-angles. The rotation order of the Euler-angles depends on the wrist type but in many cases the ZYZ-order is suitable.

$$\mathsf{R}_3^6 = \mathsf{R}_{\mathsf{Euler}}, \tag{4.45}$$

$$\mathsf{R}_3^6 = \begin{bmatrix} c\theta_6 c\theta_5 c\theta_4 - s\theta_6 s\theta_4 & -c\theta_6 c\theta_5 s\theta_4 - s\theta_6 c\theta_4 & c\theta_6 s\theta_5 \\ s\theta_6 c\theta_5 c\theta_4 - c\theta_6 s\theta_4 & -s\theta_6 c\theta_5 s\theta_4 + c\theta_6 c\theta_4 & s\theta_6 s\theta_5 \\ -s\theta_5 c\theta_4 & s\theta_5 s\theta_4 & c\theta_5 \end{bmatrix}. \tag{4.46}$$

As mentioned in chapter one, the inverse function of sine and cosine delivers two solutions in an interval from 0 to 2π for one input-value. This is lead to two solutions for the inverse orientation [16, p105-p109].

$0, \pi$:

$$\theta_5 = \operatorname{atan2}(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}), \tag{4.47}$$

$$\theta_4 = \operatorname{atan2}(r_{32}, -r_{31}), \tag{4.48}$$

$$\theta_6 = \operatorname{atan2}(r_{23}, r_{13}). \tag{4.49}$$

$-\pi, 0$:

$$\theta_5 = \operatorname{atan2}(-\sqrt{r_{13}^2 + r_{23}^2}, r_{33}), \tag{4.50}$$

$$\theta_4 = \operatorname{atan2}(-r_{32}, -r_{31}), \tag{4.51}$$

$$\theta_6 = \operatorname{atan2}(-r_{23}, r_{13}). \tag{4.52}$$

Part II

Engineering Process

Chapter 5

Development of the Software Concept

The design of a robot controller is an uncommon task in engineering. This reduces the number of experienced people in this topic. In order to compensate this lack of knowledge, a demo version of a robot control was analysed. The demo version was provided by the hardware supplier Bernecker & Rainer (B&R). It is intended to use the robot for teaching and research. Therefore it was necessary to provide an interface between the robot controller and MATLAB as a rapid prototyping platform. Two possible interfaces are the OPC-interface and the UDP-interface.

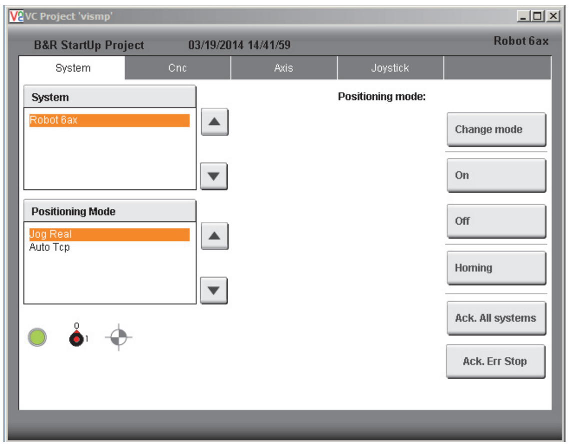
5.1 B&R Robotic Project

For a priori test the B&R AR00 PLC simulation environment was used and the robot was visualized with the B&R robotic visualisation software. The software enabled a priori test of the robot control software and helped to develop new code[1] before the hardware was available. Figure 5.1 shows the simulation environment of the robotic project. The robotic project is divided into two main parts. The first part is the Human-Machine-Interface block and is used to control the graphical user interface. The second block is the Motion-Handling block, which takes care of path planning and the control of the motion. Both blocks are designed to be reusable. Therefore, they are also applied in other B&R applications. The Motion-Handling block can be divided into three tasks. The first task is called the system task and is in charge of controlling states of the system, such as homing or power up all axes. The next task is the CNC task and is used for path planning and to coordinate the axes states. The last task is the Axes task and is applied to control the state of the single axes.

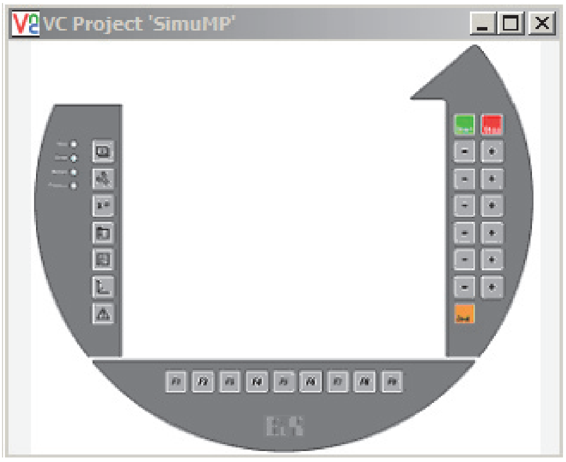
5.1.1 Important Robot Specific NC Commands

The robotic project of B&R uses G-code to control the different robot tasks. G-code is the standard code for programming numerical controlled (NC) machines, such as CNC mills or CNC lathes. Most of the commands are equal to the standardized G-code commands, but there are also some machine specific commands. The most important commands are listed in the following list.

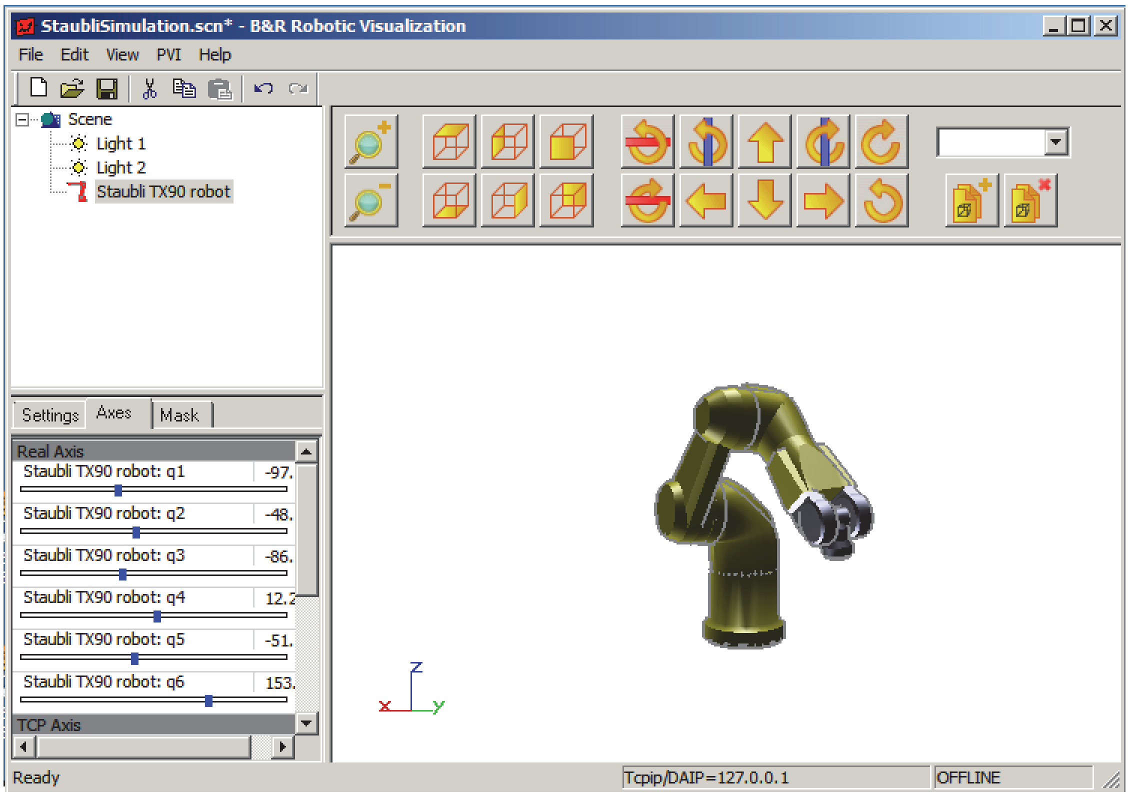
[1]The whole robotic project itself was developed with these simulation tools



(a) Menu visualization



(b) Panel visualization



Staubli TX90 robot: q1 -97.
Staubli TX90 robot: q2 -48.
Staubli TX90 robot: q3 -86.
Staubli TX90 robot: q4 12.2
Staubli TX90 robot: q5 -51.
Staubli TX90 robot: q6 153.

Tcpip/DAIP=127.0.0.1

(c) Robot visualization

Figure 5.1: Simulation environment of the B&R robotic project.

	commands	description
general	G00	Rapid linear interpolation
	G01	linear interpolation
	G02	circle interpolation clockwise
	G03	circle interpolation counter clockwise
	G90	Absolute position coordinates
	G91	Relative position coordinates
	G92	Zero point offset and rotation of coordinate frame
B&R	G100	Rapid joint interpolation
	G101	joint interpolation
	G102	circle interpolation with general orientation
	D_FULL	Tool definition
	WS_MAIN_DEF	Workspace definition
	G801 \G802	Start \End command for a cubic spline

Table 5.1: Frequently used G-code commands

The following section shows examples of use for frequently used commands.

Examples for G01 command

1. Linear movement with defined path orientation:

```
G01 X100 Y300 Z30 A30 B40 C00 SEG=5
```

X,Y,Z	point coordinates
A,B,C	Euler angles for the orientation
SEG=5	Command for segmentation of the path to the desired point. This is necessary for points near a singularities of the robot

2. Movement of the six single axes:

```
G01 Q1=0 Q2=45 Q3=90 Q4=0 Q5=45 Q6=0
```

Q1,...,Q6	Joint angles of the single axis

3. Joint interpolated movement:

```
G101 X100 Y300 Z30 A30 B40 C00
```

X,Y,Z	point coordinates
A,B,C	euler angles for the orientation

Examples for G02

1. Circle interpolation

   ```
   G02 X7.5 Y40 I-30 J-22.5
   ```

X,Y	coordinates of the end point
I,J	coordinates of the circle center

2. Helix

   ```
   G02 I50 H2160 Z100 (6 full rotations)
   ```

X,Y	coordinates of the end point
I,J	coordinates of the circle center
H	Rotation in angle

Splines command

Cubic Spline:

```
G801 CE=0.01 BC1 X-1 Y1
        G1 Y0  Y0       (interpolation point 1)
        G1 X10 Y10      (interpolation point 2)
        G1 X20 Y10      (interpolation point 3)
        G1 X30 Y0       (interpolation point 4)
G802 BC1 X-1 Y0
```

X,Y,Z	interpolation point coordinates
BC1	Boundary condition for the first point defined as first derivative
CE	Chord error of the spline

Example for Workspace and Tool definition

1. Definition of the tool offset and orientation

   ```
   D_FULL DX=20 DY=0 DZ=5 PHI=0 THETA=0 PSI=0
   ```

DX,DY,DZ	Distance to the tool coordinate system
PHI,THETA,PSI	Orientation angles of the tool coordinate system

2. Definition of the workspace border cuboid

```
WS_MAIN_DEF X1=100 Y1=100 Z1=0 X2=500 Y2=500 Z2=400
```

X1,Y1,Z1	First corner of the workspace cuboid
X2,Y2,Z2	Second corner of the workspace cuboid

5.2 B&R MATLAB via OPC

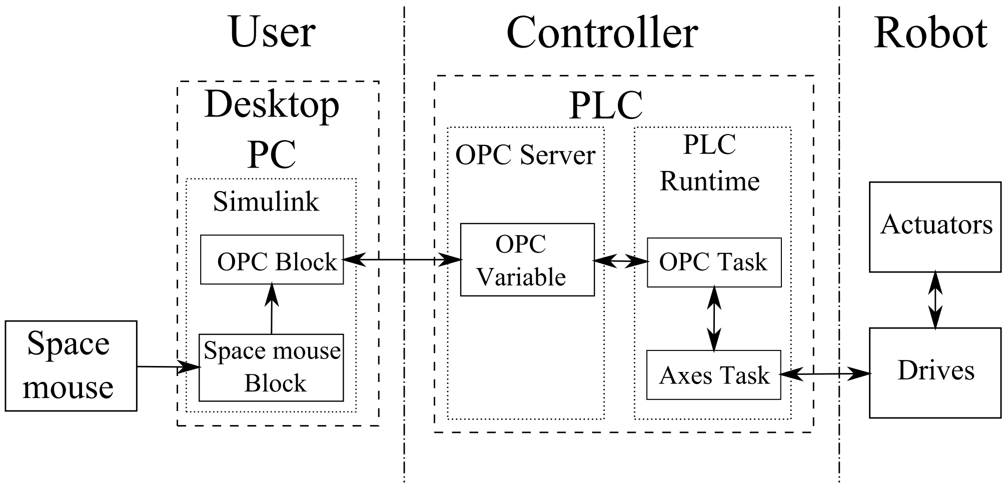


Figure 5.2: Schematic diagram of the OPC-interface.

For the first attempts to enable a real time communication between MATLAB/Simulink and the PLC the OPC-interface was used. This interface uses a server where the OPC variable is declared and stored. This variable can be accessed and also manipulated from MATLAB and the PLC. The advantage of the OPC-interface is, that it is a simple way to communicate between two or more applications without the need of caring about the communication issues. During the tests it was recognized, that the OPC-interface wasn't stable. This behaviour can be explained through the different implementation status of the actual OPC standard in MATLAB and the B&R PLC runtime.

5.3 B&R MATLAB via UDP

An alternative to the OPC-interface was the User Datagram Protocol (UDP), which is used to stream data over a network. Contrary to the TCP/IP protocol, where the reception of the sent data is guaranteed, data losses in the UDP protocol are possible. Vice versa the UDP protocol is much faster than the TCP/IP protocol and in many applications a small degree of data loss is tolerable. Contrary to the OPC-interface, the UDP-interface required some modifications before it was usable. In order to speed up the developing process, a B&R UDP sample program was

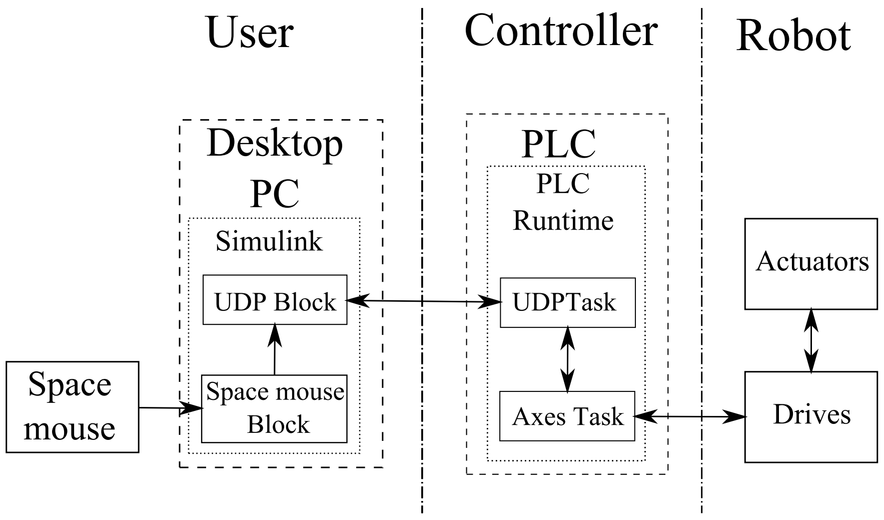


Figure 5.3: Structure diagram of the UDP-interface.

taken as design template. One issue, which caused a little irritation during the developing process, was the byte order of the different platforms. MATLAB uses as default order big endian and the PLC runtime little endian. Another issue was the watchdog, which has to ensure a stable real-time communication. The last issue was the error handling, which was necessary to avoid unintended movements of the robot.

5.4 B&R Motion Sample

B&R provides sample programs for the user, which shall help to simplify the development of their own applications. For the robot controller the motion sample was used. This sample program provides all functions for a basic control of a single axis. In order to enable the control via MATLAB, the functions of the axis have to react on cyclic changes of the set value. Unfortunately these important functions , which are necessary for a control over MATLAB, were missing. Hence, it was decided to expand the Motion sample about the cyclic velocity and the cyclic position function. Both blocks are defined by the PLCopen standard.

5.4.1 Cyclic Velocity Function Block

The cyclic velocity block is a PLCopen function block, which allows a cyclical change of the speed set value for the controlled drive[2]. Figure 5.4 shows the cyclic velocity block and its parameters.

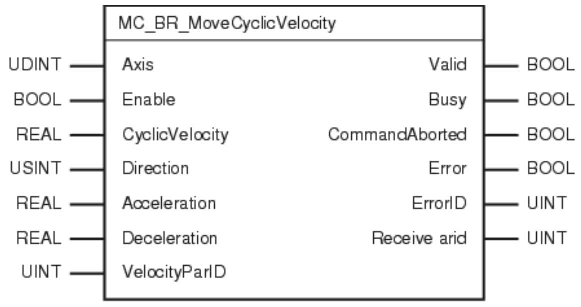


Figure 5.4: PLCopen function block cyclic velocity.

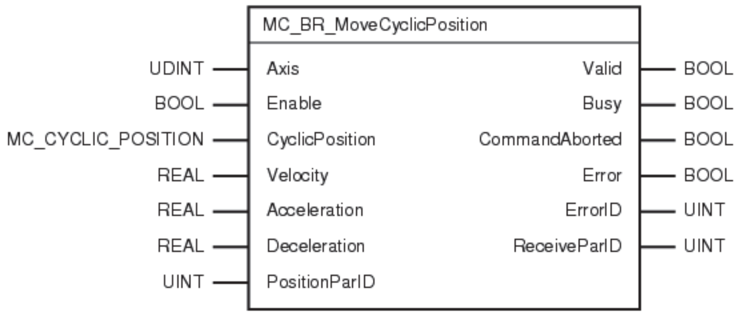


Figure 5.5: PLCopen function block cyclic position.

5.4.2 Cyclic Position Function Block

The cyclic position block is also a PLCopen function block and allows a cyclical change of the set value for the position. One of the special properties of this block is, that it has two inputs for the position. This can be seen in Figure 5.5. The first input accepts integer values as inputs and is intended for a fast positioning task. The second input accepts only real values and is intended for a precise positioning tasks.

[2]During the test period the PLCopen standard was modified, which leads to a modification of the function block itself. This issue caused some problems

Chapter 6

Design and Assembly of the Controller Hardware

6.1 Main Components

The controller consists of following main parts :

APC810 industrial PC B&R standard PLCs provide a storage for the user application the so called USERROM. The maximum size of this USERROM is limited to 3008 kbyte for a PLC with 4MB FLASH. This memory is consumed quiet fast from task such as visualization or motor control. In order to overcome this limitation an industrial PC was used.

ACOPOS variable-frequency drives ACOPOS are variable-frequency drives which can be controlled from PLC by using the Powerlink interface.

Remote I/O The remote I/O block is used to control analogue devices and to read analogue sensor data. It is also controlled from the PLC via the Powerlink interface.

Mobile Panel In order fulfil the recommendation of the robotic standard EN ISO 10218-1, it was necessary to install a mobile panel. This mobile panel acts as single pendant control for the robot.

Transformer The transformer is used to supply the variable-frequency drives. The transformer provides two different voltage levels. The first one is 100 Volt and is the voltage limit for the motors. The second one is 30 Volt and is used to reduce the maximum speed of the robot.

Safety PLC In order to fulfil the recommendation of the robotic standard EN ISO 10218-1, such as enable button, light curtain or interlock switch, it was necessary to build in a Safety PLC to supervise the system.

All of these parts were tested before they were assembled to a controller. The intention of these tests were to exclude hardware errors during the controller software developing. An overview of the whole controller unit and the used parts is presented in Figure 6.1.

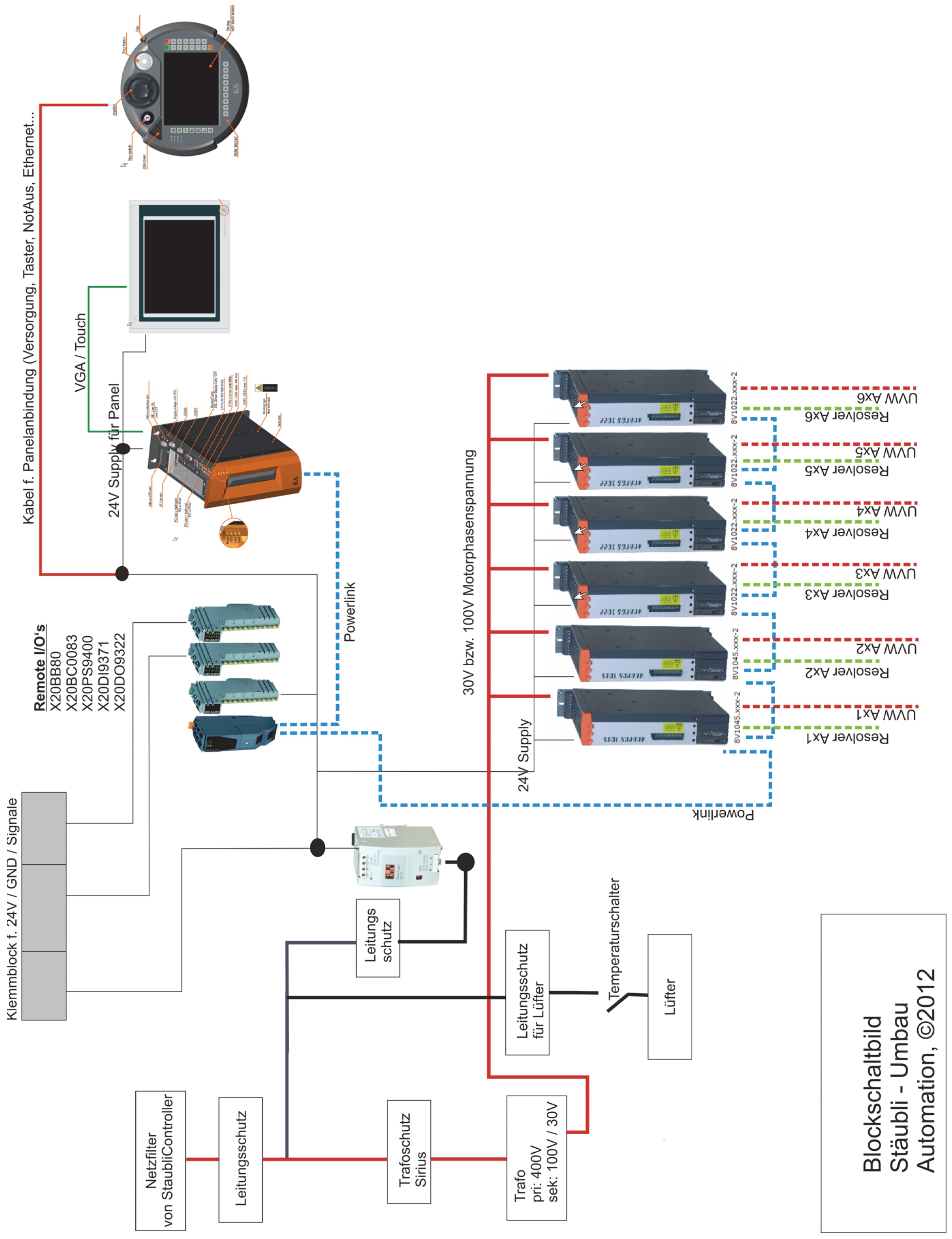


Figure 6.1: Schematic diagram of the Stäubli RX60 controller.

(a) Frontside

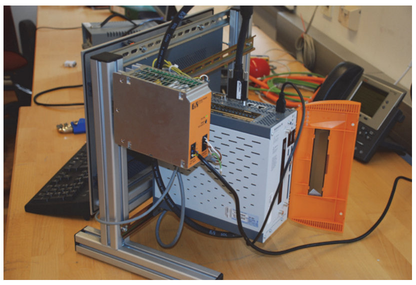

(b) Backside

Figure 6.2: A priori test of the hardware components.

6.2 Industrial PC APC810

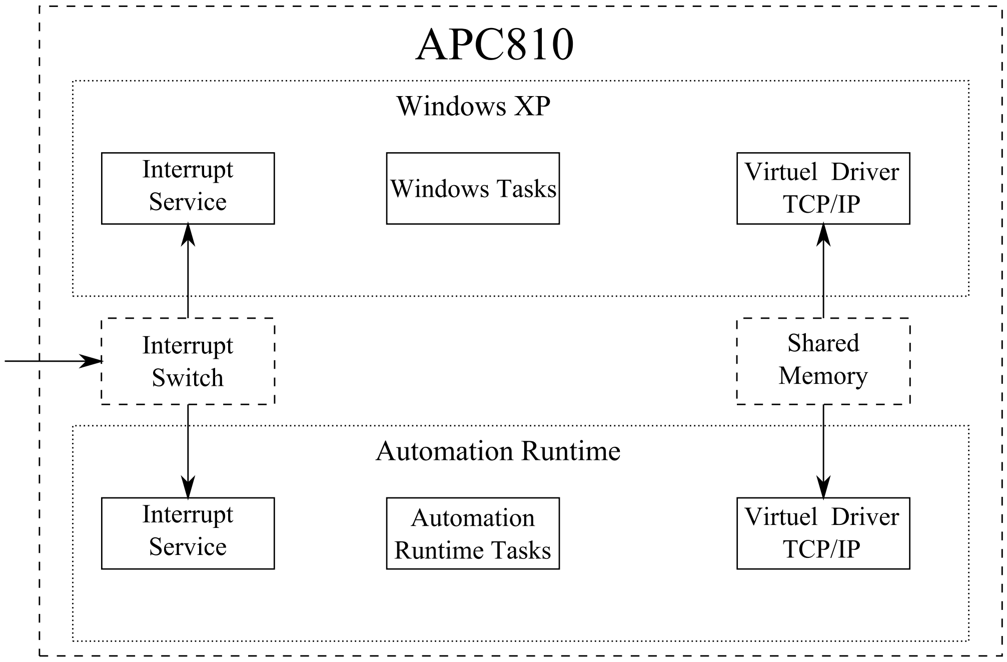


Figure 6.3: Function diagram of the APC810.

The simultaneous control of six axes is a memory consuming task. Standard PLC reach their limit quite fast, especially if they are also used for visualization and other task. For this reason, it was decided to use an industrial PC, which provided more memory and computation power, instead of a normal PLC. It was decide to use the B&R APC810 industrial PC, which can use the two operating systems Windows XP and the real time operating system ARWIN simultaneously. Each of the operating systems uses one core of the dual core processor and a defined part of the RAM-disk. A RAM disk is a virtual storage, which uses a part of the memory of the random access memory (RAM) to store data. Figure 6.3 shows the schematic diagram of the APC810. Originally it was intended to run MATLAB on the industrial PC, but MATLAB was too resource hungry to allow a stable operation of the ARWIN system.

6.3 Safety PLC

The standard EN ISO 10218-1 demands some special safety features from the robot. One of the requirements was, that the robot only can be switched on if a enabling button was pressed. Another requirement is the emergency stop of the robot if the light curtain or alternatively the interlock switch of the robot cage door is activated. The robot Stäubli RX60 has a holding brake, which is used to hold the robot arm in position if the power is lost. This brake is lifted if power is available. In the case of power loss, the brake is activated and stops the system. In order to reduce the braking torque in case of an emergency stop, it was decided to slow down the arm first with the motors and use the holding brake to bring the arm to a complete stop. This behaviour was realized by a switch off delay of 300 ms and a quick stop command which is send to the drives. After the delay time the drive controllers are disabled

6.4 Estimation of the Gear Ratio

The Stäubli RX60 uses synchron servomotors to rotate its joints. Servomotors have a rotary encoder mounted on the motor shaft. This encoder allows a control of the position and the speed of the motor shaft. The robot designer tried to reduce the mass of the moved parts to achieve a good dynamical behaviour. Therefore, they used small motors with a small moment of inertia. This motor are to weak to drive the joint directly and therefore a gear is needed. In order to reduce the cost, the designer the motor used encoder to determine the actual joint angle. This task is simple if the joints are driven directly, where the joint angle is equal with the motor angle. In case of gear driven joint, the joint angle has to be calculated by multiplying the motor angle with the gear ratio.

Unfortunately,the data sheets of the Stäubli RX60 didn't provide the gear ratio. Therefore the gear ratio had to be estimated. For this purpose the single joints were rotated about a known or easy measurable angle, for example 180 degrees. Afterwards the gear ratio was calculated by using the measured value of the encoder.

$$i = \frac{\theta_{out}}{\theta_{in}} \tag{6.1}$$

For an easier handling is θ_{in} substituted by

$$\theta_{in} = c\,\theta'_{in}, \tag{6.2}$$

$$i = \frac{\theta_{out}}{c\,\theta'_{in}}. \tag{6.3}$$

with:

i . . . Gear ratio

θ_{out} . . . Rotation angle of the joint (gear output)

θ_{in} . . . Rotation angle of the motor (gear input)

θ'_{in} . . . Rotation angle of the motor in encoder increments

c . . . Conversation factor

The B&R system uses motor revolution per one joint revolution to describe the gear ratio. For this purpose the gear ratio has to be inverted.

$$motor\,revolution = \frac{1}{i}\,joint\,revolution \tag{6.4}$$

Chapter 7

Programming of the Axis Controller

One of the main issues was the simultaneous control of the six robot axes. The control task for a single axis was developed during the test phase; therefore, by utilizing the already existing code, the development effort for the multi axes control task is reduced. This particular task supervises the individual single axis control tasks. Figure 7.1 gives an overview of the controller concept.

7.1 Single Axis Task

The Single Axis Task controls the basic function of one axis. In order to achieve a clear command structure, four structured variables, that can be used for hierarchical structures, were defined. Structured variables can be explained as objects of a class, which only contains attributes but no functions. This four variables are:

Commands These variables enable the particular drive modes.

Parameters These variables contain the set-values for the particular drive mode.

Status These variables display the actual drive status, such as actual position, actual velocity and also error messages.

Axis state These variables display the actual state of the axis.

The single axis task is based on a finite state machine, which controls the modes of the axis.

7.2 Robot Control Task

The aim of the Robot Control Task is to coordinate the six single axes of the robot. This task also uses structured variables as command structure and is equivalent to the one described in single axis task. Also the structure finite state machine is similar. The idea behind this finite state machine is that a state of the Robot Control task leads the same action as the same state in the Single axis task

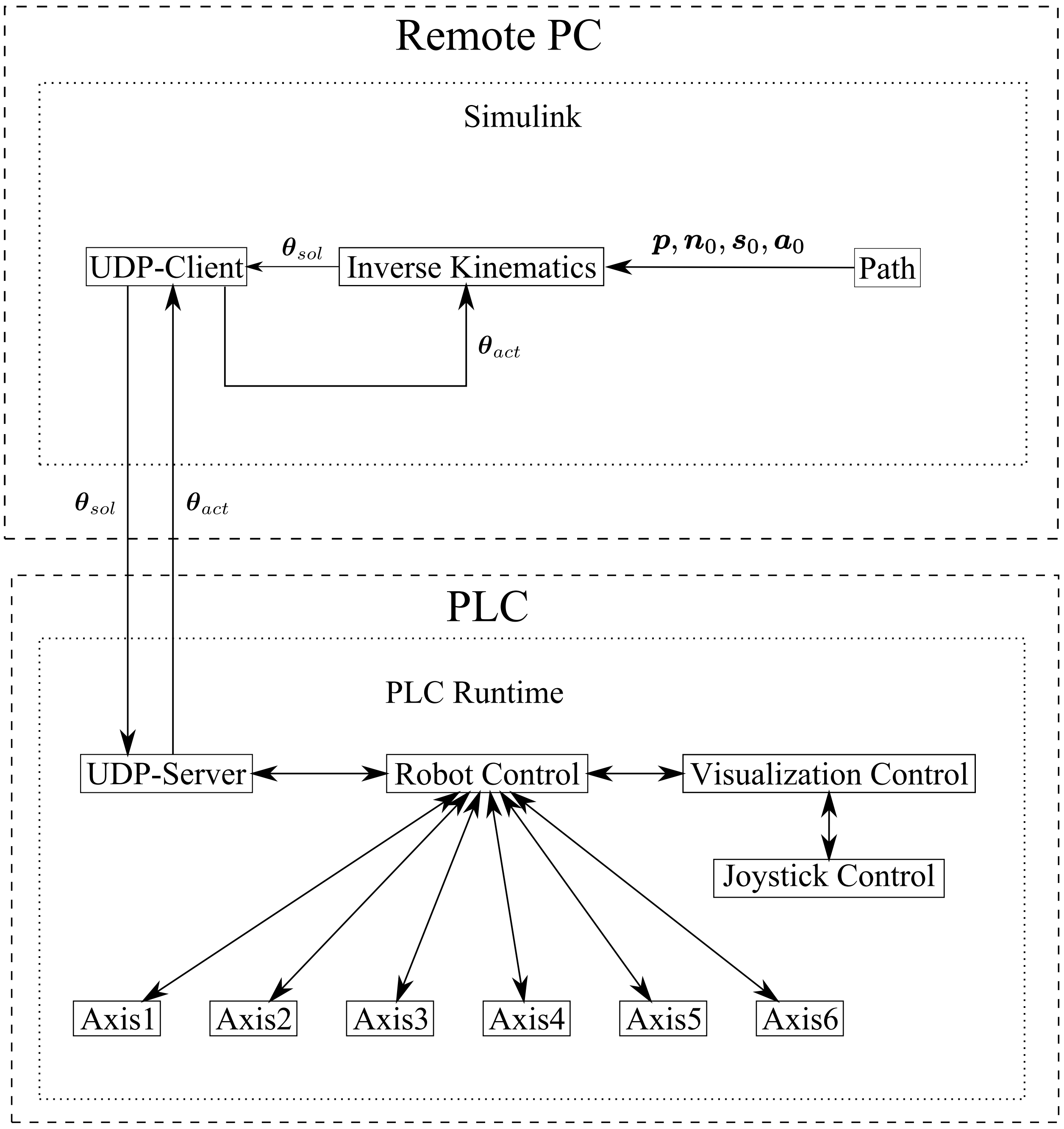

Remote PC
Simulink
UDP-Client
$\boldsymbol{\theta}_{sol}$
Inverse Kinematics
$\boldsymbol{p}, \boldsymbol{n}_0, \boldsymbol{s}_0, \boldsymbol{a}_0$
Path
$\boldsymbol{\theta}_{act}$
$\boldsymbol{\theta}_{sol}$
$\boldsymbol{\theta}_{act}$
PLC
PLC Runtime
UDP-Server
Robot Control
Visualization Control
Joystick Control
Axis1
Axis2
Axis3
Axis4
Axis5
Axis6

Figure 7.1: Overview of the controller concept.

but only with six axes. The state diagram of this finite state machine can be found in the appendix. The following examples code shows the power on state and how all six axes are handled.

Listing 7.1: First solution for conjoining the status variables

```

POWERON:
        ActualState := 'POWERON';
        //Send global command only ONCE

        // Switch on of the controller of the single axes
        IF (cycle = 0) THEN
                FOR axes_index := 0 TO (max_loopcount) DO
                        GlobalControl[axes_index].Command.Power := TRUE
                           ;
                        PowerCheck[axes_index] := 0;
                END_FOR
                cycle := 1;
        END_IF

        // Check if all axes controller are switched on
        FOR axes_index := 0 TO (max_loopcount) DO
                IF (axes_index = 0) THEN
                        Merker := TRUE;
                END_IF
                // Conjoin all status variables by using Merker
                Merker := GlobalControl[axes_index].Status.DriveStatus.
                   ControllerStatus AND Merker;
        END_FOR


        // Exit state
        IF (Merker = TRUE) AND (axes_index = max_index) THEN
                RobStatus.VisuDrivesEnabled := TRUE;
                STATE := READY;
        END_IF
```

In this example code the for-loop was used to reduce the code and combine the status variables of the axis to one variable. Therefore the variable `Merker`is conjoined with the status variable by `AND`. The result is stored again in the variable `Merker`. If all status variables of the axis are `TRUE`, then the variable `Merker` is also `TRUE`. If one of the status variables is `FALSE` then the variable `Merker` is also `FALSE`. Alternatively the same result could be achieved by conjoining all status variables with `AND`. This is shown in the following example code.

Listing 7.2: Second solution for conjoining the status variables

```
//Check if all Axes are homed...
RobStatus.Homed:= GlobalControl[0].Status.DriveStatus.HomingOk
                         AND GlobalControl[1].Status.DriveStatus.
                            HomingOk
                            AND GlobalControl[2].Status.DriveStatus.
                               HomingOk
                            AND GlobalControl[3].Status.DriveStatus.
                               HomingOk
                            AND GlobalControl[4].Status.DriveStatus.
                               HomingOk
                            AND GlobalControl[5].Status.DriveStatus.
                               HomingOk;
```

7.3 Interpolation Concept

In the first attempt, it was tried to move along a path by using the cyclic position function block. This block didn't allow a smooth movement. This behaviour can be explained by the acceleration and deceleration ramp, which is applied after and before the desired point. If the distance between the points is too short, there is a permanent acceleration and decelerating motion, which leads to vibrations. Therefore it was decided to use the cyclic velocity function block instead. This block has the disadvantage, that an interpolation controller is necessary to reach the desired position.

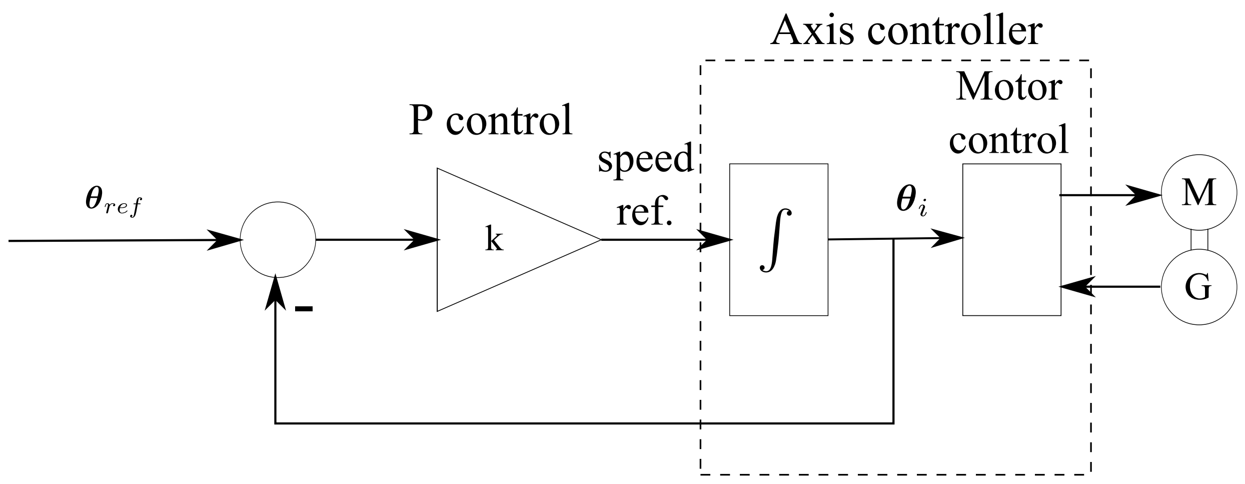


Figure 7.2: Interpolation concept for the Stäubli RX60 .

The following equations show, that the behaviour of the interpolation concept in Figure 7.2 is equivalent to the behaviour of a first order system.

$$\theta_i = \frac{1}{s}(\theta_{ref} - \theta_i)\,k, \tag{7.1}$$

$$s\,\theta_i = k\,\theta_{ref} - k\,\theta_i, \tag{7.2}$$

$$s\,\theta_i + k\,\theta_i = k\,\theta_{ref}, \tag{7.3}$$

$$\theta_i\,(s+k) = k\,\theta_{ref}, \tag{7.4}$$

$$\frac{\theta_i}{\theta_{ref}} = \frac{k}{k+s} = \frac{1}{\frac{s}{k}+1}. \tag{7.5}$$

7.4 Programming of the Safety PLC

The safety PLC has a simple programming environment, which is diagram based. The programming environment always uploads the actual program for the storage of the safety PLC. This feature was very useful to deal with the version problem. The safety PLC has to take care of the following tasks:

1. Enable all drive controllers.
2. Force robot to stop, if the light curtain of the robot cage or the interlock switch of the cage door is activated.
3. Control the emergency stop buttons and force the robot to stop. This stop is performed by an electrical brake. The final stop is achieved by the holding brake, which has a release delay of 300 ms to allow an electrical braking before.

Figure 7.3 shows the program for the Stäubli RX60 .

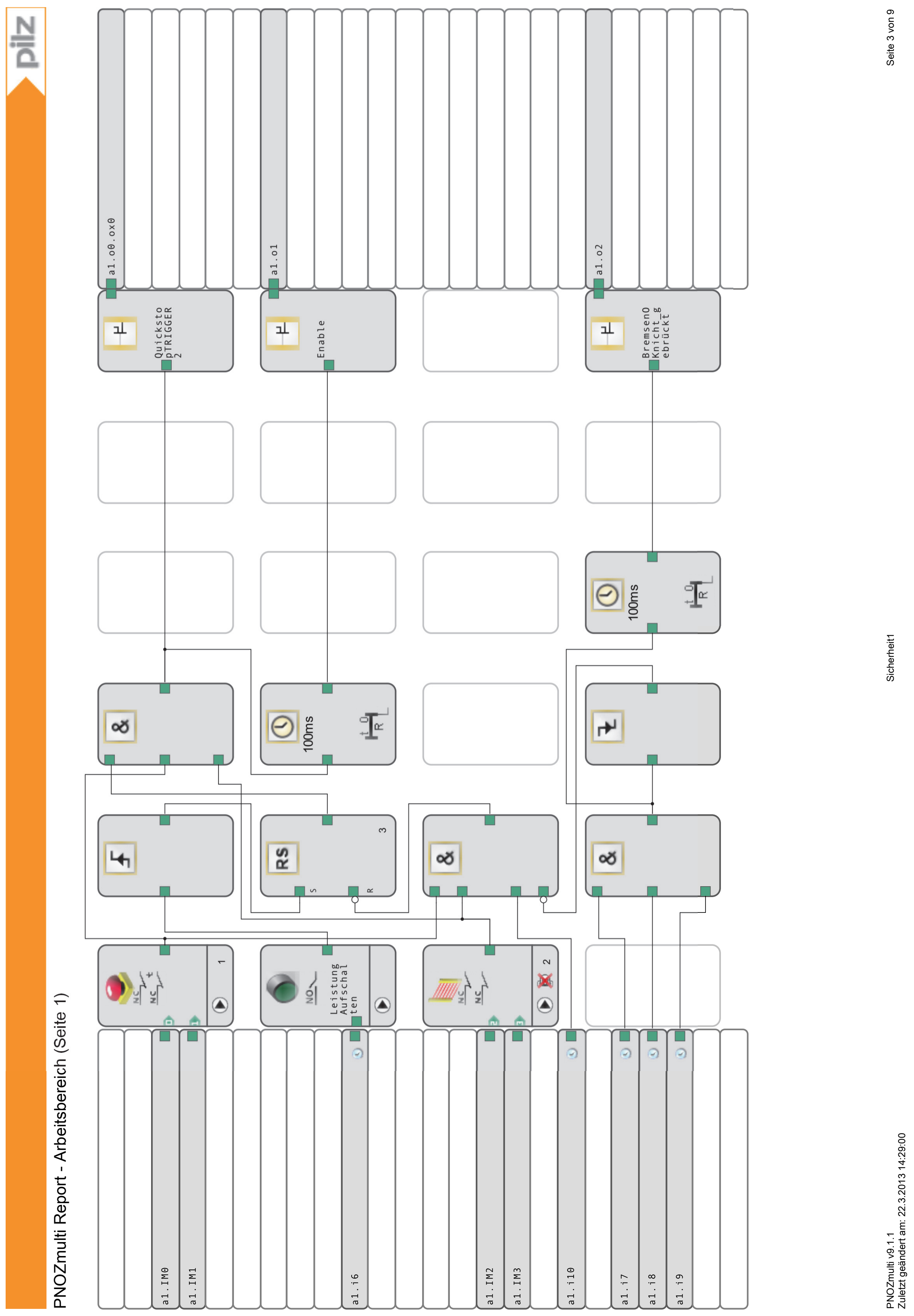


Figure 7.3: This figure shows the programming environment of the safety PLC. On the left section shows the input channel, the middle part the sequential logic and the right section the output channels.

Chapter 8

Implementation of Robot Kinematics

During the design process, it was decided to separate the control of the robot and the kinematics. The kinematics and especially the inverse kinematics is a mathematical complex field. This is the reason, why it is a popular test field for optimization algorithms, such as the artificial neural network [10] [5] [6], evolutionary algorithm [9] or genetic algorithm [2]. MATLAB is optimized for matrix and numerical calculation, which makes it a obvious tool for developing new algorithm and software prototypes. Therefore, it was decided to implement the kinematics in MATLAB\Simulink.

8.1 Implementation

The following section show the inverse kinematics. In order to improve the readability of the code, the important mathematical relations are written in mathematical notation. Additionally, the same line numbers as in the original code are used to simplify the comparison of the code segments. The original code can be found in the appendix.

8.1.1 Inverse Kinematics

Position

The following section shows the inverse kinematics for the angles θ_2 and θ_3. The calculation of the angle θ_1 can be found in the border handling section.

Listing 8.1: border handling

```
200 % computing the solutions for u2, u3 in case of o beeing inside the
       boundary sphere
201
202 numberSol2 = numberSol/2;
203
204 v2 = zeros(3, numberSol2);
205
```

```
for i=1:numberSol2

          $m = p_x * cos(v(i)) + p_y * sin(v(i))$
          $n = p_z - d_1$

          %sol := solve($2*(m*x+n*z) - a_2^2 + d_4^2 - m^2 - n^2, x^2 + z^2 - a_2^2$, $x, z$):

          $aa = 2*(p_x*\cos(v(i)) + p_y*\sin(v(i)))$
          $bb = 2*(p_z - d_1)$
          $cc = -a_2^2 + d_4^2 - m^2 - n^2$

          if (abs(cc) < obj.eps)
             $t(1) = \mathrm{atan2}(-aa, bb)$
             $t(2) = t + \pi$
          else
             $\alpha = \mathrm{atan2}(-bb/cc, -aa/cc)$
             $\beta = \mathrm{atan2}(\frac{\sqrt{a_2^2*(aa^2+bb^2)-cc^2}}{|cc|}, 1)$
             $t(1) = \alpha - \beta$
             $t(2) = \alpha + \beta$
          end

          for j=1:2

             %o3 := <subs(sol[j], x), subs(sol[j], z)>:

             $\boldsymbol{o}_3 = a_2 * [\cos(t(j)); sin(t(j))]$
             $v2(i,j) = \mathrm{atan2}(o_3(1), o_3(2))$
             $\phi = \mathrm{atan2}(m - o3(1), n - o3(2))$
             $v3(i,j) = \phi - v2(i,j)$
          end
end
```

Orientation

In order to optimize the computing time, some properties of the vector calculus are used. For an easier reading of the source code some of the important relations are summarized.

Dot product:

$$\boldsymbol{a}\,\boldsymbol{b} = |\boldsymbol{a}|\,|\boldsymbol{b}|\cos(\varphi) \tag{8.1}$$

If $\boldsymbol{a}$ and $\boldsymbol{b}$ are unit vector:

$$\boldsymbol{a}\,\boldsymbol{b} = \cos(\varphi) \tag{8.2}$$

Cross product:

$$|\boldsymbol{a} \times \boldsymbol{b}| = |\boldsymbol{a}|\,|\boldsymbol{b}|\,\sin\varphi \tag{8.3}$$

If $\boldsymbol{a}$ and $\boldsymbol{b}$ are unit vector:

$$|\boldsymbol{a} \times \boldsymbol{b}| = \sin\varphi \tag{8.4}$$

normalizing of vectors:

$$\boldsymbol{a}_{norm} = \frac{\boldsymbol{a}}{|\boldsymbol{a}|} \tag{8.5}$$

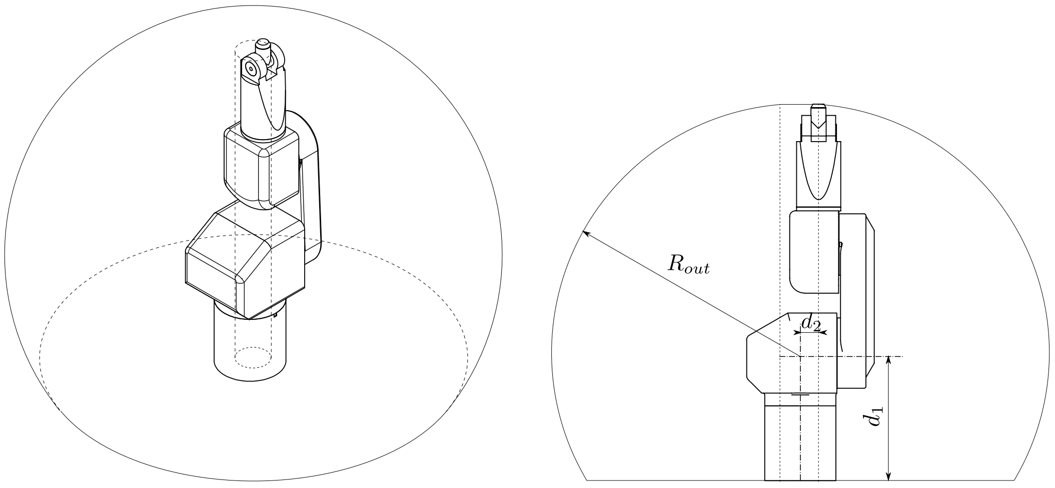
Listing 8.2: orienation

```
nos = 2*nos123
U = zeros(6,nos)

% Normalizing of the input vectors and calculation of the three
    orientation vectors.

$\boldsymbol{n}_6 = \boldsymbol{f1}/|\boldsymbol{f1}|$
$\boldsymbol{a}_6 = \boldsymbol{f1} \times \boldsymbol{f2}$
$\boldsymbol{a}_6 = \boldsymbol{a}_6/|\boldsymbol{a}_6|$
$\boldsymbol{s}_6 = \boldsymbol{a}_6 \times \boldsymbol{n}_6$

for j=1:nos123
  j2 = 2*j
  j1 = j2-1

  for i=1:3
   U(i,j1) = U123(i,j)
   U(i,j2) = U123(i,j)
  end

  % Calculation of the orientation vector of the third joint

  sn1 = sin(U123(1,j))
  cs1 = cos(U123(1,j))
  sn23 = sin(U123(2,j)+U123(3,j))
  cs23 = cos(U123(2,j)+U123(3,j))

 $\boldsymbol{a}_3 = [sn1; cs1; 0]$


 $\boldsymbol{a}_4 = [cs1\, sn23; sn1\, sn23; cs23]$

  % Calculation of $\cos\theta_5$
  $cs5 = \boldsymbol{a}_4, \boldsymbol{a}_6$

  % Exactly two different solution triples for u4, u5, u6 for every
     solution triple of u1, u2, u3

  if (abs(abs(cs5) - 1.0) > obj.eps)

        $\boldsymbol{a}_5 = \boldsymbol{a}_4 \times \boldsymbol{a}_6$
        $denom = |\boldsymbol{a}_5|$
        $fak = 1/denom$
        $\boldsymbol{a}_5 = fak\,\boldsymbol{a}_5$
        $sn5 = fak * (1.0 - cs5^2)$

        % Calculation of the two solutions for the angle $\theta_4$
        $cs4 = \boldsymbol{a}_3\,\boldsymbol{a}_5$
        $sn4 = fak * (\boldsymbol{a}_3\,\boldsymbol{a}_6)$
        $U(4, j1) = \text{atan2}(sn4, cs4)$
        $U(4, j2) = \text{atan2}(-sn4, -cs4)$

        % Calculation of the two solutions for the angle $\theta_6$
        $cs6 = \boldsymbol{a}_5\,\boldsymbol{s}_6$
        $sn6 = fak\,(\boldsymbol{a}_4\,\boldsymbol{s}_6)$
        $U(6, j1) = \text{atan2}(sn6, cs6)$
        $U(6, j2) = \text{atan2}(-sn6, -cs6)$
  else
  .
  .
  .
  end
 % Calculation of the two solutions for the angle $\theta_4$

 U(5,j1) = atan2(sn5, cs5);
 U(5,j2) = - U(5,j1);
```

8.1.2 Border Handling

The points in space, which are reachable with the robot, are limited by the physical properties of the robot, such as arm length. This is an important issue which has to be dealt with inside the kinematics because the boundaries are singularities of the robot. As Figure 8.1 shows, the Stäubli RX60 has a spherical outer workspace boundary and a cylindrical inner workspace boundary. As mentioned in chapter two, the design engineer tries to avoid singularities by design of the robot. This design leads to an inner cylindrical border of the workspace.

Listing 8.3: border handling



(a) Isometric view (b) Front view

Figure 8.1: Theoretical workspace of the Stäubli RX60 (axis limits are not reconsidered).

```
$dist1^2 = p_x^2 + p_y^2$
$dist2^2 = p_x^2 + p_y^2 + (p_z - d_1)^2$

% Point is in workspace
if($dist1^2 > r_{in}^2$)&&($dist2^2 < R_{out}^2$)
% Point lies on the boundary cylinder
 if(($dist1^2 < r_{in}^2$)
% Point lies on lower or upper boundary circle
   if($dist2^2 > R_{out}^2$)
    numberSol=1
    $\theta_1 = \text{atan2}(-p_x, p_y)$
% Point lies on the upper boundary circle
      if($|p_z - d_1 - a_3 - d_4|$<tolerance)
       $\theta_2 = 0$
% Point lies on the lower boundary circle
      else
       $\theta_2 = \pi$
      end
    $\theta_3 = 0$
% Point lies on the boundary cylinder and inside the boundary sphere
   else
     numberSol=2
     $v(1) = \text{atan2}(-p_x, p_y)$
     $\theta_1(1) = v(1)$
     $\theta_1(2) = \theta_1(1)$
  end
% Point lies outside the boundary cylinder
 else
   $\alpha = \text{atan2}(-p_x, p_y)$
   $\beta = \text{atan2}(\sqrt{dist1^2 - d_2^2}, d_2)$
```

```

    $v(1) = \alpha - \beta$
    $v(2) = \alpha + \beta$
     for j=1:2
       if($v(j) > \pi$)
         $v(j) = v(j) - 2\pi$
       end
       if($v(j) < -\pi$)
         $v(j) = v(j) + 2\pi$
       end
     end
% Point lies on the boundary sphere
     if($dist2^2 > R^2$)
       numberSol=2
      $\theta_1(1) = v(1)$
      $\theta_1(2) = v(2)$
       for j=1:2
         $\theta_2(j) = \mathrm{atan2}(p_x \cos v(j) + p_y \sin v(j), p_z - d_1)$
         $\theta_3(j) = 0$
       end
    end
% Point lies inside the boundary sphere and out side the boundary
    cylinder
   else
    numberSol=4
    $\theta_1(1) = v(1)$
    $\theta_1(2) = v(1)$
    $\theta_1(3) = v(2)$
    $\theta_1(4) = v(2)$
   end
```

8.1.3 Implementation in MATLAB

The first version of the kinematic was written as MATLAB class. This class contains two functions beside the robot parameters. The first function is called DK and calculates the transformation matrix of the forward kinematics. The second function is called IK and calculates the possible solutions and evaluates the closest solution to the actual configuration. Just as the B&RRobotic Project this kinematic was developed with a simulation tool, which leads to problems with the rotation direction. Not all assumed rotation directions of simulation did fit with rotation directions of the robot axes.

Closest Solution

In the general case the inverse kinematic of Stäubli RX60 has eight solutions, but the robot can execute only one of them. Therefore an algorithm is needed to find the most suitable of the solutions.

The algorithm for the Stäubli RX60 uses the euclidean norm of the difference between the actual joint angles vector and the solution vector. The solution vector, whose norm has the smallest value, is the closest solution to the actual robot position.

Listing 8.4: Choose solution

```
dist=1000;
sol=1;

 for i=1:numSolutions

         dist1=$\|\boldsymbol{\theta}(i) - \boldsymbol{\theta}_{act}\|_2^2$

         if (dist<dist1)
                 sol=i
                 dist1=dist
         end
 end
```

Wrist Singularity Handling

The wrist singularity appears, if the axis of joint 4 and 6 are collinear and their motions are redundant.

Listing 8.5: Singularity Handling

```
cs5=$\boldsymbol{a}_4\,\boldsymbol{a}_6$
sn5=0.0
csw=$\boldsymbol{a}_3\,\boldsymbol{s}_3$
snw=$(\boldsymbol{a}_3 \times \boldsymbol{s}_3)\,\boldsymbol{a}_4$

w=$\mathrm{atan2}(snw, csw)$

 if(cs5>0)
  $\theta_4(i) = \frac{\theta_{4act}-\theta_{6act}+w}{2}$
  $\theta_6(i) = w - \theta_4(i)$
 else
  $\theta_4(i) = \frac{\theta_{4act}-\theta_{6act}+w}{2}$
  $\theta_6(i) = -w + \theta_4(i)$
 end
```

8.1.4 Implementation in Simulink

Simulink is compiled to C and therefore much faster than MATLAB. Therefore, it was decided to implement the kinematics in Simulink. Simulink provides a function block, which allows to use MATLAB script functions in Simulink. The disadvantage of this function block is, that it only supports the basic data types of MATLAB.

Real-time in Simulink

MATLAB\Simulink doesn't operate in real time.[1] A communication with a real time system requires a synchronisation of MATLAB\Simulink with the real time. This is achieved with the MATLAB function waitfortreal, which keeps MATLAB in busy wait state till the set time is over.

8.1.5 Test of the Kinematics

In order to test the functionality of the kinematics two different test programs were written. The first program was written in MATLAB and the second one in Simulink.

MATLAB

The original kinematics was written as MATLAB class and therefore it was obvious to write a test script in MATLAB. The MATLAB script is suitable to define a path and let it be executed from the robot. During the test the following difficulties appeared:

1. In case, that the script is aborted, the UDP-connecting has to be closed manually via command line, which is sometimes forgotten. Therefore an error handling routine is necessary.
2. The calculation in MATLAB aren't executed in real-time. In case, that MATLAB sends the points of the path too fast for the PLC, data gets lost. Therefore a synchronization with the real-time is necessary.
3. The standard user interaction are limited. Therefore a controlled program flow would be necessary.

The test showed, that it is possible to control the robot via MATLAB but it is more effort necessary to deal with the difficulties than in Simulink.

Simulink

The kinematics was also tested in Simulink. Simulink provides for many common input devices, such as joysticks or cameras, function block to read their values. Therefore, a standard 3D-mouse was used to generate the path and perform the test. In order to achieve a real-time behaviour some changes in the solver parameters were necessary. Instead of the default variable-steps time of the solver were a fixed-steps time of 0.01 s was used and the simulation time was set to infinity.

[1]Depending on the task, it is faster or slower than real-time.

Chapter 9

Conclusion and Future Work

This thesis presented the development and engineering of a control unit for a 6 DOF industrial robot. The new controller concept provides a transparent framework and enables a separation of the robot kinematics from the control hardware. For this purpose a generic network interface for the PLC has been developed, which enables the real-time execution of remote positioning commands. The network interface allows the use of more powerful PC and the scientific computation platform MATLAB. Therefore, MATLAB is used as platform for the robot kinematics computations. Due to the use of standard industrial components the maintenance is simplified. The transparent framework simplifies the embedding and parametrizing of new sensors.

For standard application, the robot interprets G-codes stored in CNC files, that are generated with several tools. A direct control from MATLAB/Simulink was implemented successfully and opens new facilities for future applications. Research on methods for robot calibration and registration may be done with the actual equipment.

It would be interesting to test, whether real-time control would be possible with the use of WLAN instead of LAN. This implies the application of mobile devices, such as phones and tablets, for controlling the robot. In future cyber-physical systems, this techniques will become more important.

For educational purposes it is possible to integrate the control of the robot into student projects. For example, during writing this thesis a project was carried out successfully, where students controlled the robot with a Kinect^{TM}motion sensor. Integration of such sensors is possible without the need of downloading software to the robot controller.

A further new idea is to use the robot for 3D printing on surfaces of arbitrary geometry. A dedicated printing head for this purpose is already under construction.

Appendix A

Source Code

A.1 Kinematics Test in MATLAB

Listing A.1: Test of the kinematics in MATLAB

```
close all; clc;
%% Create path for robot
z=327:1:476;

winkel=linspace(0,2*pi,600)

xcenter=374
ycenter=49
zcenter=320
r=50

x=xcenter+r*cos(winkel);
y=ycenter+r*sin(winkel);
z=linspace(zcenter,zcenter,600);

% Break for the user to control data
disp('ENTER, um Bahnmatrix zu erstellen...')
pause;

% Figure for a visualization of the path
figure(1)
plot3(x,y,z,'*')
grid on

% Merge the single vectors of the path  coordinates to one Matrix [x,y,
   z'];
Bahnmatrix=[x;y;z]';

[zeile,spalte]=size(Bahnmatrix);
```

```

zaehler=0;

%% Setup a connetion to PLC via UDP-Object

u = udp('10.43.70.10', 25000, 'LocalPort',30000, 'ByteOrder', '
    littleEndian');

fopen(u);

pause(0.2)

% Create object of the class which deals with the kinematic
my_robot = StaeubliRX60(290.0, 341.0, 49.0, 310.0);

actAnglegrd(1)=0;

% Create data to enable the comunictation with the PLC. The PLC waits
    till
% it recives from the remote PC, therefore it is necesssary to send
    data
% first
data(1) = 11;

% Convert value to Integer
senddata = int32(data);

fwrite(u,senddata, 'int32');

pause(0.2);

actAnglegrd = fread(u,6,'single');

sendactAnglegrd = int32(actAnglegrd);

disp('ENTER, um Initialposition zu übergeben... (RICHTIGKEIT PRÜFEN!!!'
    )
pause;

fwrite(u,sendactAnglegrd, 'int32');

%%

disp('ENTER, um Bahn abzuarbeiten...')
pause;

for zahl=1:zeile

```

```
 % Reading of the actual axes angles
 actAnglegrd = fread(u,6,'single');

 % Convertation from deg to rad
 actAngle=(pi/180000)*actAnglegrd;

 % Invertation of values for axis 5 to be equiavalent with the positiv
 % direction of the robot
 actAngle(5)=-actAngle(5)
 % Correction of movement of axis 6, which is caused by the gears
 actAngle(6) = actAngle(6)-actAngle(5);
%
% Calculation of the actual position of the robot arm by using the
   forward
% kinematc.
my_Pos = DK(my_robot,actAngle);

% Calculation of the endefector orientation
hv = my_Pos{6}*[0; 1; 0; 0];
f1 = hv(2:4); % Convert 4x1 vetor to 3x1 vector
hv = my_Pos{6}*[0; 0; 1; 0];
f2 = hv(2:4);
f3 = cross(f1, f2);

% Calculation of the inverse kinematic
[j0, U] = my_robot.IK(o6, f1, f2,actAngle);

% Convertation from rad to deg
 solution=(180/pi)*U(:,j0);
% Invertation of values for axis 5
  solution(5) = -solution(5);
% Correction of movement of axis 6
  solution(6) = solution(6) - solution(5);

% Scaling of the solution angles to fit with the PLC representation
 solution = 1000*solution;
 sendsolution = int32(solution);

fwrite(u,sendsolution, 'int32');

pause(0.02);

end

%% Closing the UDP-Connection and deleting of the udp-object
fclose(u);
delete(u);
```

A.2 Kinematics Test in Simulink

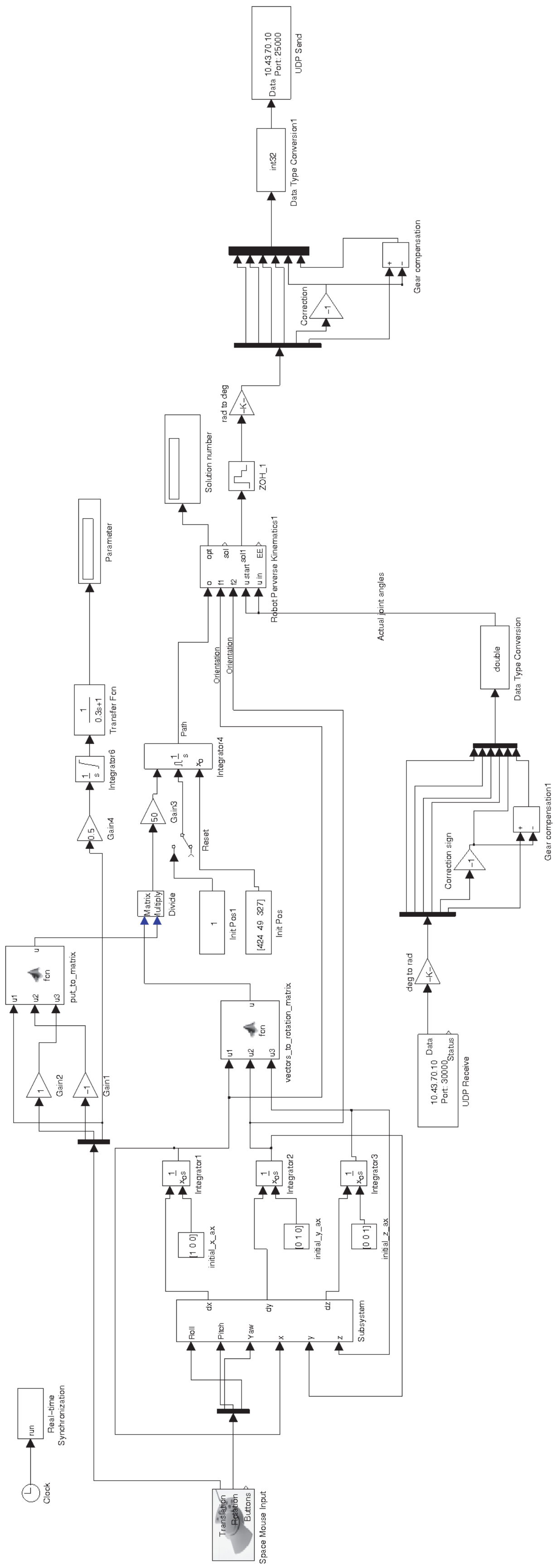


Figure A.1: Simulink model for the test of the kinematics.

A.3 Kinematics of the Stäubli RX60

Listing A.2: Source code of the class Stäubli RX60

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% class 'StaeubliRX60':
% this implements the direct kinematics (DK) and the inverse kinematics
    (IK)
% of the Staeubli RX60 robot

classdef StaeubliRX60

  properties
    % the relevant DH-parameters of the robot:
    a2;
    d1;
    d2;
    d4;

    eps;          % a small value for determining the numerical
       accuracy

    a2_sq;
    d2_sq;
    d4_sq;
    R;            %R ... radius of the allowed sphere (workspace)
    rMinusEps_sq; % r = d[2] ... radius of the forbidden cylinder (
       workspace)
    rPlusEps_sq;
    RPlusEps_sq;
    RMinusEps_sq;

  end

  methods
    % constructor:
    function obj = StaeubliRX60(a2, d1, d2, d4)
      obj.a2 = a2;
      obj.d1 = d1;
      obj.d2 = d2;
      obj.d4 = d4;

      obj.eps = 0.0001;

      obj.R = sqrt((obj.a2+obj.d4)^2 + obj.d2^2);
      obj.a2_sq = obj.a2^2;
      obj.d2_sq = obj.d2^2;
      obj.d4_sq = obj.d4^2;
      obj.rMinusEps_sq = (obj.d2-obj.eps)^2;
      obj.rPlusEps_sq = (obj.d2+obj.eps)^2;
      obj.RPlusEps_sq = (obj.R+obj.eps)^2;
      obj.RMinusEps_sq = (obj.R-obj.eps)^2;
    end

 %%
    % function 'DK':
    % computes the transformation matrices B(i)
    % belonging to the input angles (u1, u2, u3, u4, u5, u6)
    % INPUT:  u ... vector of the 6 rotation angles:
    %         u = (u1, u2, u3, u4, u5, u6);
    % OUTPUT: the matrices B
    function B = DK(obj, u)
      %local cs, sn, Rz, B;

      B = cell(6, 1);

      % transformation matrix B{1}
      % for the rotation around axis number 1
      % (i.e., the motion Sigma1/Sigma0):
      cs = cos(u(1));
      sn = sin(u(1));
      B{1} = [   1.0  0.0 0.0 0.0;
                 0.0   cs -sn 0.0;
                 0.0   sn  cs 0.0;
              obj.d1  0.0 0.0 1.0];


      % transformation matrix B{2}
      % for the combination of the first two rotations (axis number 1
         and 2);
      % (i.e., the motion Sigma2/Sigma0):
      cs = cos(u(2));
      sn = sin(u(2));
      B{2} = B{1}*[   1.0  0.0 0.0 0.0;
                      0.0   sn  cs 0.0;
                   obj.d2  0.0 0.0 1.0;
                      0.0   cs -sn 0.0];

      % transformation matrix B{3}
      % for the combination of the first three rotations (axis number
         1, 2 and 3);
      % (i.e., the motion Sigma3/Sigma0):
      cs = cos(u(3));
      sn = sin(u(3));
      B{3} = B{2}*[   1.0  0.0 0.0 0.0;
                   obj.a2  -sn -cs 0.0;
```

```
                      0.0   cs -sn 0.0;
                      0.0  0.0 0.0 1.0];

        % transformation matrix B{4}
        % for the combination of the first four rotations (axis number
           1,...,4);
        % (i.e., the motion Sigma4/Sigma0):
        cs = cos(u(4));
        sn = sin(u(4));
        B{4} = B{3}*[1.0  0.0 0.0  0.0;
                     0.0   cs -sn  0.0;
                     0.0  0.0 0.0 -1.0;
                     0.0   sn  cs  0.0];

        % transformation matrix B{5}
        % for the combination of the first five rotations (axis number
           1,...,5);
        % (i.e., the motion Sigma5/Sigma0):
        cs = cos(u(5));
        sn = sin(u(5));
        B{5} = B{4}*[   1.0  0.0 0.0  0.0;
                        0.0   cs -sn  0.0;
                        0.0  0.0 0.0  1.0;
                     obj.d4  -sn -cs  0.0];

        % transformation matrix B{6}
        % for the combination of all six rotations (axis number 1,...,6);
        % this is the end-effector motion Sigma6/Sigma0:
        cs = cos(u(6));
        sn = sin(u(6));
        B{6} = B{5}*[1.0  0.0 0.0  0.0;
                     0.0   cs -sn  0.0;
                     0.0  0.0 0.0 -1.0;
                     0.0   sn  cs  0.0];
    end

  %%

    % function 'IK':
    % inverse kinematics of the Staeubli RX60 robot:
    % INPUT: o ... position vector of the origin of the endeffectors'
        coordinate system;
    %        f1... direction vector of the x-axis of the endeffectors'
        coordinate system;
    %        f2... another direction vector parallel to the xy-plane
    %              of the endeffectors' coordinate system;
    %        u ... a vector of 6 start values for the angles u1, ...,
        u6 to be found
    % OUTPUT: 6x(nos)-matrix U (nos = number of solutions),
    %         each of its columns contains a solution
    %         6-tupel u_1, ..., u_6 for the rotation angles";
    %         integer value j0, index of the solution 6-tupel closest to u
    function [j0, U] = IK(obj, o, f1, f2, u)

%       local U123, U, nos123, nos1232, nos, dist1_sq, dist2_sq, indic,
%           i, j, j0, j1, j2, insideSphere, m, n, phi, o3, sol, v1, v2, v3,
%           sn1, cs1, sn23, cs23, sn4, cs4, sn5, cs5, sn6, cs6,
%           e, denom, fak, csw, snw, w, dist, dist1, res;

      dist1_sq = o(1)^2 + o(2)^2;
      dist2_sq = dist1_sq + (o(3)-obj.d1)^2;

      if (dist1_sq > obj.rMinusEps_sq) && (dist2_sq < obj.RPlusEps_sq) % point o is in workspace
        insideSphere = false;
        if (dist1_sq < obj.rPlusEps_sq) % o lies on the boundary cylinder
          if (dist2_sq > obj.RMinusEps_sq) % o lies on the upper or lower boundary circle
            nos123 = 1;
            U123 = zeros(3, 1);
            U123(1,1) = atan2(-o(1), o(2));
            if (abs(o(3) - obj.d1 - obj.a2 - obj.d4) < 0.001) % o lies on the upper boundary circle
              U123(2,1) = 0.0;
            else                                              % o lies on the lower boundary circle
              U123(2,1) = pi;
            end
            U123(3,1) = 0.0;
          else          % o lies on the boundary cylinder AND inside the boundary sphere
            v1(1) = atan2(-o(1), o(2));
            nos123 = 2;
            U123 = zeros(3, nos123);
            U123(1,1) = v1(1);
            U123(1,2) = U123(1,1);
            insideSphere = true;
          end
        else % o lies outside the boundary cylinder
          %v1 = solve(o(1)*sin(u1) - o(2)*cos(u1) + d(2), u1);
          alph = atan2(-o(1), o(2));
          bet = atan2(sqrt(dist1_sq - obj.d2_sq), obj.d2);
          v1(1) = alph - bet;
          v1(2) = alph + bet;
          for j=1:2
            if (v1(j) > pi)
              v1(j) = v1(j) - 2*pi;
            end
            if (v1(j) < -pi)
              v1(j) = v1(j) + 2*pi;
            end
          end
          if (dist2_sq > obj.RMinusEps_sq)       % o lies on the
             boundary sphere
            nos123 = 2;
            U123 = zeros(3,nos123);
            U123(1,1) = v1(1);
            U123(1,2) = v1(2);
            for j=1:2
              U123(2,j) = atan2(o(1)*cos(v1(j)) + o(2)*sin(v1(j)), o(3)
                 -obj.d1);
              U123(3,j) = 0.0;
            end
          else      % o lies outside the boundary cylinder AND inside
             the boundary sphere
            nos123 = 4;
            U123 = zeros(3,nos123);
            U123(1,1) = v1(1);
            U123(1,2) = v1(1);
            U123(1,3) = v1(2);
            U123(1,4) = v1(2);
            insideSphere = true;
          end
        end

        if (insideSphere == true) % computing the solutions for u2, u3
           in case of o beeing inside the boundary sphere
          nos1232 = nos123/2;
          v2 = zeros(3, nos1232);
          for i=1:nos1232
            m = o(1)*cos(v1(i)) + o(2)*sin(v1(i));
            n = o(3) - obj.d1;
            %sol := solve({2*(m*x + n*z) - a2_sq + d[4]^2 - m^2 - n^2,
               x^2 + z^2 - a2_sq}, {x, z}):
            aa = 2*(o(1)*cos(v1(i)) + o(2)*sin(v1(i)));
            bb = 2*(o(3) - obj.d1);
            cc = - obj.a2_sq + obj.d4_sq - m^2 - n^2;
            if (abs(cc) < obj.eps)
              t(1) = atan2(-aa, bb);
              t(2) = t + pi;
            else
              alph = atan2(-bb/cc, -aa/cc);
              bet = atan2(sqrt(obj.a2_sq*(aa^2 + bb^2) - cc^2)/abs(cc), 1);
              t(1) = alph-bet;
              t(2) = alph+bet;
            end

            for j=1:2
              %o3 := <subs(sol[j], x), subs(sol[j], z)>:
              o3 = obj.a2*[cos(t(j)); sin(t(j))];
              v2(i,j) = atan2(o3(1), o3(2));
              phi = atan2(m-o3(1), n-o3(2));
              v3(i,j) = phi - v2(i,j);
            end
          end
          U123(2,1) = v2(1,1);
          U123(2,2) = v2(1,2);
          U123(3,1) = v3(1,1);
          U123(3,2) = v3(1,2);
          if (nos123 == 4)
            U123(2,3) = v2(2,1);
            U123(2,4) = v2(2,2);
            U123(3,3) = v3(2,1);
            U123(3,4) = v3(2,2);
          end
        end


        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%% Berechnung der Loesungen fuer u_4, u_5, u_6;
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        nos = 2*nos123;
        U = zeros(6,nos);

        e61 = f1/norm(f1);
        e63 = cross(f1, f2);
        e63 = e63/norm(e63);
        e62 = cross(e63, e61);
        for j=1:nos123
          j2 = 2*j;
          j1 = j2-1;
          for i=1:3
           U(i,j1) = U123(i,j);
           U(i,j2) = U123(i,j);
          end
          sn1 = sin(U123(1,j));
          cs1 = cos(U123(1,j));
```

```
          sn23 = sin(U123(2,j)+U123(3,j));
          cs23 = cos(U123(2,j)+U123(3,j));
          e33 = [-sn1; cs1; 0];
          e43 = [cs1*sn23; sn1*sn23; cs23];

          cs5 = dot(e43, e63);
          if (abs(abs(cs5) - 1.0) > obj.eps)   % exactly two different solution triples for u4, u5, u6
                                           % for every solution triple of u1, u2, u3

            e53 = cross(e43, e63);
            denom = norm(e53);
            fak = (1.0/denom);
            e53 = fak*e53;
            sn5 =   fak*(1.0 - cs5^2);
            cs4 = dot(e33, e53);
            sn4 = fak*dot(e33, e63);
            U(4,j1) = atan2(sn4, cs4);
            U(4,j2) = atan2(-sn4, -cs4);
            cs6 = dot(e53, e62);
            sn6 = fak*dot(e43, e62);
            U(6,j1) = atan2(sn6, cs6);
            U(6,j2) = atan2(-sn6, -cs6);
          else               % special case: axes g4 and g6 are the same
            sn5 = 0.0;
            csw = dot(e33, e62);
            snw = dot(cross(e33, e62), e43);
            w = atan2(snw, csw);
            if (cs5 > 0)
              U(4,j1) = 0.5*(u(4)-u(6)+w);
              U(6,j1) =  w - U(4, j);
            else
              U(4, j) = 0.5*(u(4)+u(6)+w);
              U(6, j) =  -w + U(4, j);
            end
            U(4,j2) = U(4,j1);
            U(6,j2) = U(6,j2);
          end
          U(5,j1) = atan2(sn5, cs5);
          U(5,j2) = - U(5,j1);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % determine the optimal angle 6-tupel w.r.t. u among the nos solution 6-tupels:
        dist1 = 10000.0;
```

```
    for j=1:nos
      dist = norm(U(:,j) - u);
      if (dist < dist1)
        j0 = j;
        dist1 = dist;
      end
    end


  else
    j0 = 0;
    U = 0;
  end
end









% function 'pedalPoint':
% INPUT:  a ... column vector; point a whose pedal point "p" has to be found
%         acc ... accuracy of the procedure (a small number)
%         tStart ... optional input; a start value for the parameter t
% OUTPUT: pedalPointFound ... boolean variable which is returned as "1"
%         in case of success and as "0" else;
%         p ... pedal point (column vector)
%         t ... parameter value of the pedal point of the curve (number)
function [pedalPointFound, p, t] = pedalPoint(obj, a, acc, tStart)
  if (nargin==3)
    % determination of a proper intitial parameter value tP:
    t0 = obj.t0;
    t1 = obj.t1;
    anz = 30;
    dist = 1000000;
    t_inc = (t1 - t0)/(anz - 1);
    for i = 0:anz-1
        t = t0 + i*t_inc;
        x = obj.point(t);
        dist1 = norm(a - x);
        if dist1 < dist
```

```
              tP = t;
              dist = dist1;
          end
      end
    else
      % using the intial parameter value "tStart" prescribed by the
         user:
      tP = tStart;
    end

    t = tP;
    x = obj.point(t);
    c = a - x;

    delta = 0.0001;
    xt = obj.tangentVector(t, delta);

    cs = dot(c, xt);

    eps = abs(cs);

    zaehler = 1;
    maxSteps = 20;
    while (eps > acc) && (zaehler < maxSteps)
      % 2nd derivative vector
      xtt = obj.derivativeVector(t, 2, delta);

      % 1st derivative of the function cs(t) = <a - x(t), xt(t)>
      % to find the root of this function:
      cst = - dot(xt, xt) + dot(c, xtt);

      epsilon = 0.0001;
      if abs(cst) < epsilon
        if cst < 0
          cst = -epsilon;
          else
          cst = epsilon;
        end
      end

      % new t
      t = t - cs/cst;
      x = obj.point(t);
      c = a - x;
      xt = obj.tangentVector(t, delta);
      cs = dot(c, xt);

      eps = abs(cs);
```

```

        p = x;
        zaehler = zaehler + 1;
      end
      if zaehler == maxSteps
        pedalPointFound = false;
      else
        pedalPointFound = true;
      end
    end

  end
end
```

A.4 Statediagram of the RobotControl Task

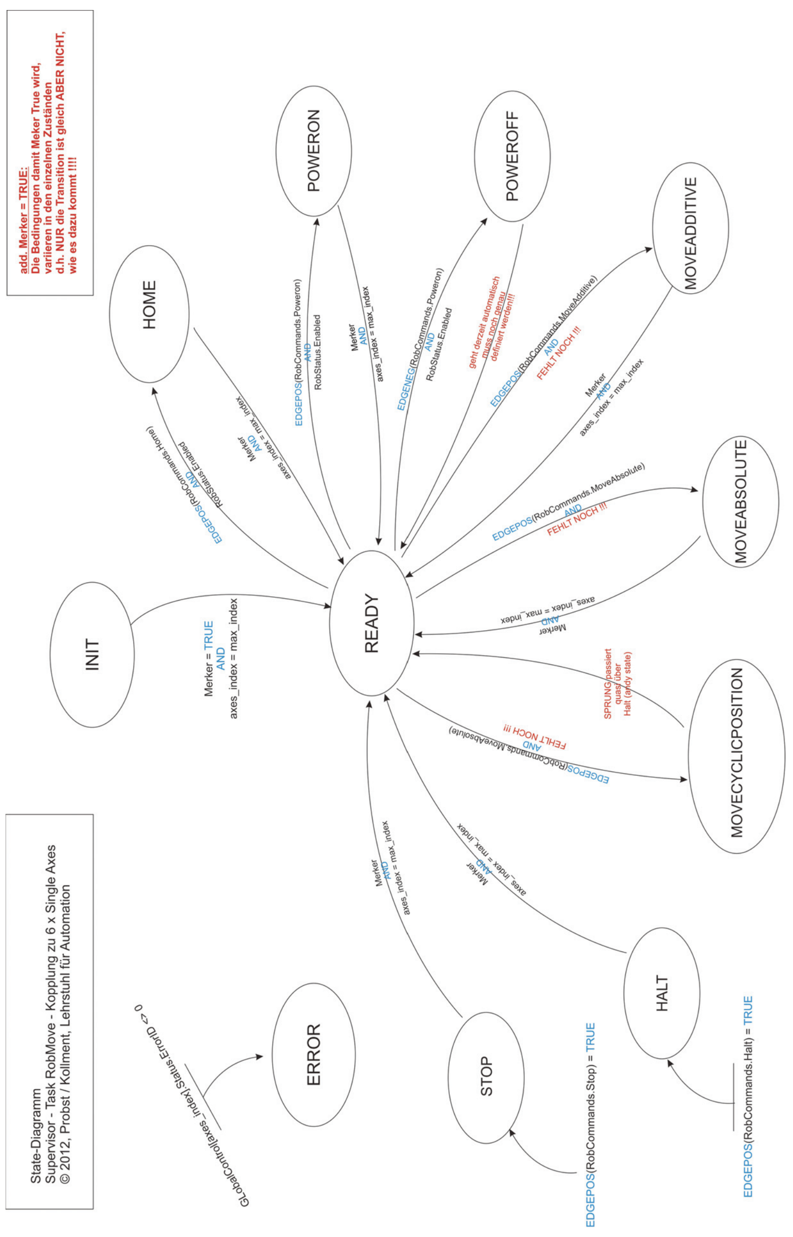

add. Merker = TRUE:
Die Bedingungen damit Meker True wird, variieren in den einzelnen Zuständen
d.h. NUR die Transition ist gleich ABER NICHT, wie es dazu kommt !!!!
State-Diagramm
Supervisor - Task RobMove - Kopplung zu 6 x Single Axes
© 2012, Probst / Kollment, Lehrstuhl für Automation
INIT
READY
HOME
POWERON
POWEROFF
MOVEADDITIVE
MOVEABSOLUTE
MOVECYCLICPOSITION
HALT
STOP
ERROR
Merker = TRUE AND axes_index = max_index
EDGEPOS(RobCommands.Home) AND RobStatus.Enabled
EDGEPOS(RobCommands.Poweron) AND RobStatus.Enabled
EDGENEG(RobCommands.Poweron) AND RobStatus.Enabled
geht derzeit automatisch muss noch genau definiert werden!!!
EDGEPOS(RobCommands.MoveAdditive) AND FEHLT NOCH !!!
EDGEPOS(RobCommands.MoveAbsolute) AND FEHLT NOCH !!!
SPRUNG passiert quasi über Halt (andy state)
Merker AND axes_index = max_index
EDGEPOS(RobCommands.Stop) = TRUE
EDGEPOS(RobCommands.Halt) = TRUE
GlobalControl[axes_index].Status.ErrorID <> 0

Figure A.2: Statediagram for the Robot Control Task

List of Figures

List of Tables

Bibliography

[1] J. Angeles. *Fundaments of Robotic Mechanical Systems*. Springer-Verlag, Berlin, 1997.

[2] F. Chappelle and P. Bidaud. Closed form solution for inverse kinematics approximation of general 6R manipulators. *Mechanism and Machine Theory*, 39(1):323–338, 2004.

[3] J. Craig. *Intruduction to Robotics*. Addison-Wesely Publishing Company, second edition, 1989.

[4] R. Goldman. Understanding quaterions. *Graphical Models*, 73(1):21–49, 2011.

[5] A.T. Hasan, A.M.S. Hamouda, N. Ismail, and H.M.A.A. Al-Assadi. An adaptiv-learning algorithm to solve the inverse kinematics problem of a 6 d.o.f serial robot manipulator. *Advances in Engineering Software*, 37(1):432–438, 2006.

[6] A.T. Hasan, N. Ismail, A.M.S. Hamouda, I. Aris, M.H. Marhaban, and H.M.A.A Al-Assadi. Artificial neural network-based kinematics Jacobian solution for serial manipulatar passing through singular configurations. *Advances in Engineering Software*, 41(1):359–367, 2010.

[7] K.H. Hunt. Don't cross-thread the screw. *Journal of Robotic Systems*, 20(7):317–339, 2003.

[8] M. Husty, A. Karger, H. Sachs, and W. Steinhilper. *Kinematik und Robotik*. Springer-Verlag, Berlin, 1997.

[9] P. Kalra, P.B. Mahapatra, and D.K. Aggarwal. An evolutionary approach for solving the multimodal inverse kinematics problem of industrial robots. *Mechanism and Machine Theory*, 41(1):1213–1229, 2006.

[10] B. Karlik and S. Aydin. An improved approach to the solution of inverse kinematics problem for robot manipulators. *Engineering Applications or Artificial Intelligence*, 13(1):159–164, 2000.

[11] R. Murray. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, second edition, 1994.

[12] S.B. Niku. *Intruduction to Robotics:Analysis,Control,Applications*. John Wiley and Sons, second edition, 2010.

[13] S. Qiao, Q. Liao, S. Wei, and H. Su. Inverse kinematic analysis of the general serial manipulators based on douple quaternions. *Mechanism and Machine Theory*, 45(1):193–199, 2010.

[14] C.R. Rocha, C.P. Tonetto, and A. Dias. A comparison between the Denavit-Hartenber and the screw-based methods used in kinematic modeling of robot manipulators. *Robotics and Computer-Integrated Manifacturing*, 27(1):723–728, 2011.

[15] L. Scilavicco and B. Siciliano. *Modelling and control of robot manipulators*. Springer-Verlag, London, second edition, 2000.

[16] M.W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley and Sons,Inc, New Jersey,Hoboken, 2005.

[17] James M Van Verth and Lars M Bishop. *Essential mathematics for games and interactive applications: a programmer's Guide*. CRC Press, 2008.