Modeling and Control of a Bending Backwards Industrial Robot

Erik Wernholt¹, Måns Östring²

¹Division of Automatic Control
Department of Electrical Engineering
Linköpings universitet, SE-581 83 Linköping, Sweden
WWW: http://www.control.isy.liu.se

²Department of Science and Technology
Linköpings universitet, SE-601 74 Norrköping, Sweden
E-mail: erikw@isy.liu.se, manos@itn.liu.se

19th May 2003

Report no.: LiTH-ISY-R-2522

Technical reports from the Control & Communication group in Linköping are available at http://www.control.isy.liu.se/publications.
Abstract

In this work we have looked at various parts of modeling of robots. First the rigid body motion is studied, spanning from kinematics to dynamics and path and trajectory generation. We have also looked into how to extend the rigid body model with flexible gear-boxes and how this could be incorporated with Robotics Toolbox [Corke, 1996]. A very simple feedforward control based on the rigid model is applied in addition to PID control and in the simulations the overshoot is halved compared to only PID control. We have also tried LQ control and in our simulations the effect of torque disturbances on the arm has been lowered by a factor five, compared to diagonal PID controllers.

Keywords: modeling, control, robotics, bend-over-backwards capability, mechanical flexibilities
Contents

1 Introduction ................................................. 1
  1.1 Outline .............................................. 1

2 Rigid Motions and Homogeneous Transformation .......... 2
  2.1 Rigid Motion ........................................ 2
  2.2 Homogeneous Transformation .......................... 2
  2.3 Composition of Rotations .............................. 3
  2.4 Rotation About Fixed vs. Current Frame .............. 3
  2.5 Different Representations ............................. 3
  2.6 Robotics Toolbox .................................... 3

3 Forward Kinematics ....................................... 5
  3.1 Denavit-Hartenberg Representation ...................... 5
  3.2 Robotics Toolbox ..................................... 6
  3.3 Example IRB 7600 ..................................... 6

4 Velocity Dynamics, Jacobian .............................. 11
  4.1 Calculation of the Jacobian ............................ 12
  4.2 Singularities ........................................ 13
  4.3 Inverse Velocity and Acceleration ..................... 13

5 Dynamics .................................................. 14
  5.1 Euler-Lagrange Method ................................ 14
  5.2 Newton-Euler Method .................................. 15
  5.3 Example IRB 7600 cont. ................................ 17

6 Path and Trajectory Generation .......................... 19
  6.1 General .............................................. 19
  6.2 Joint Space Motion ................................... 20
  6.3 Cartesian Space Schemes (Operational Space) ......... 22
  6.4 Collision-Free Path Planning .......................... 23
  6.5 Programming Language, Rapid ........................ 23
  6.6 Trajectory Planning using MATLAB™ .................. 24

7 Simulations ................................................ 25
  7.1 Model Equations of Flexible Robot ..................... 25
  7.2 Preliminaries ......................................... 27
  7.3 Comparison With and Without Feedforward Control .... 27
  7.4 Comparison PID Control and LQ Control ............... 29

8 Conclusions and Remarks .................................. 33
1 Introduction

This document is the result of the graduate course Robot Modeling and Control, given at the Department of Electrical Engineering, Linköping University, Sweden. The aim is to give an overview of the procedure from modeling to path generation and control of robots. Throughout the work the Robotics Toolbox [Corke, 1996] and MATLAB™ is used and several examples show how to use the toolbox.

In the examples in this document we will use an asymmetrical robot with bend-over-backwards capability, similar to the ABB IRB 7600 [ABB, 2002]. Bend-over-backwards means that there’s no need to rotate the robot to reach for things behind it. Instead you simply swing the arm backwards, which will make working cells more compact and greatly extend the working range of the robot.

The robot model includes flexibilities in the gear-box and the control structures that are used for the robot are

- Diagonal PID controllers.
- LQ using a linearized model in one working point for the design.
- Feedforward of the non-flexible part of the robot.

The controllers are evaluated regarding path following and disturbance rejection (disturbance torque on motor and arm).

1.1 Outline

The report is organized as follows:

- Section 2 describes rigid motions and homogeneous transformations.
- Section 3 deals with the kinematics of the robot.
- Section 4 discusses velocity kinematics, the Jacobian.
- Section 5 presents the dynamics.
- Section 6 covers path and trajectory generation.
- Section 7 displays the simulations. This is the most important section in the report. If you choose to only read parts of this report, this part is the one that you must read.
- Section 8 gives some conclusions and reflections of the work.

Sections 2-6 is general in nature and can be applied to any industrial robot. In Section 7 and 8 the focus is on the ABB IRB 7600, but principles are applicable to other robots as well.
2 Rigid Motions and Homogeneous Transformation

To be able to describe the robot kinematics in a convenient way, various coordinate systems are needed. This section aims to describe the coordinate systems and the transformations between these systems. The material mainly comes from [Spong and Vidyasagar, 1989, Chapters 2-3], with some additional examples using the Robotics Toolbox [Corke, 1996] in MATLAB™.

2.1 Rigid Motion

Consider two coordinate systems (frames), \( o_0x_0y_0z_0 \) and \( o_1x_1y_1z_1 \) in Fig. 1. Their relation can be described by a combination of a pure rotation and a pure translation, called a rigid motion.

![Figure 1: Two coordinate systems.](image)

A point \( p \) in \( \mathbb{R}^3 \) can then be expressed as \( p_0 \) in frame 0 and \( p_1 \) in frame 1, and their relation as

\[
p_0 = R^1_0 p_1 + d^1_0
\]

where \( R^1_0 \) is the (orthogonal, \( R^{-1} = R^T \)) rotation matrix describing the orientation of \( o_1x_1y_1z_1 \) with respect to \( o_0x_0y_0z_0 \) and \( d^1_0 \) is the translation of frame 1 relative to frame 0.

2.2 Homogeneous Transformation

For easy notation, a rigid motion is sometimes expressed as

\[
\begin{bmatrix} p_0 \\ 1 \end{bmatrix} = H_0^1 \begin{bmatrix} p_1 \\ 1 \end{bmatrix}, \quad H_0^1 = \begin{bmatrix} R^1_0 & d^1_0 \\ 0 & 1 \end{bmatrix}
\]

where \( H_0^1 \) is called the homogeneous transformation.

The inverse transformation \( H_1^0 \) is given by

\[
H_1^0 = \begin{bmatrix} R^0_1 & -R^0_1 d^1_0 \\ 0 & 1 \end{bmatrix}
\]
2.3 Composition of Rotations

Now consider three frames, $ox_0y_0z_0$, $ox_1y_1z_1$ and $ox_2y_2z_2$, where their origins coincide. Let $R_j^i$ be the rotation matrix describing the orientation of frame $j$ relative to frame $i$. Their relation then is

$$R_0^2 = R_0^1 R_1^2$$

(4)

This can be interpreted as follows: “First let all frames coincide. Then rotate frame 1 relative to frame 0 according to $R_0^1$. With frame 2 coincident with frame 1, rotate frame 2 relative to frame 1 according to $R_1^2$.”

2.4 Rotation About Fixed vs. Current Frame

Let $R_{z,\theta}$ be a basic rotation of $\theta$ degrees around the $z$-axis, and let $R_{y,\phi}$ and $R_{x,\psi}$ be defined similarly.

Suppose $R_0^1$ represents $R_{z,\theta}$ followed by $R_{y,\phi}$.

Rotation about current frame gives that we first rotate about $z_0$ and then about $y_1$, which gives $R_0^1 = R_{z,\theta} R_{y,\phi}$.

On the other hand, rotating about the fixed frame $ox_0y_0z_0$, gives that we first rotate about $z_0$ and then about $y_0$ (not $y_1$!), which gives $R_0^1 = R_{y,\phi} R_{z,\theta}$ (reverse order!).

2.5 Different Representations

Since the rotation matrix $R$ is orthogonal with $\det R = +1$, it can be represented using only 3 parameters. Many representations occur, for example axis/angle, Euler angles, roll-pitch-yaw angles, and quaternions. See [Spong and Vidyasagar, 1989] for details.

2.6 Robotics Toolbox

In Robotics Toolbox [Corke, 1996] there are a number of functions for homogeneous transformations, see Table 2.6. (There are also some additional functions for quaternions.)

T_Rotx=rotx(pi/4)

T_Rotx =

$$\begin{bmatrix}
1.0000 & 0 & 0 & 0 \\
0 & 0.7071 & -0.7071 & 0 \\
0 & 0.7071 & 0.7071 & 0 \\
0 & 0 & 0 & 1.0000
\end{bmatrix}$$

T_Transy=transl([0 1 0])

T_Transy =

$$\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1
\end{bmatrix}$$
Table 1: Functions in Robotics Toolbox [Corke, 1996] for homogeneous transformations.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eul2tr</td>
<td>Euler angle to homogeneous transform</td>
</tr>
<tr>
<td>oa2tr</td>
<td>Orientation and approach vector to homogeneous transform</td>
</tr>
<tr>
<td>rotvec</td>
<td>Homogeneous transform for rotation about arbitrary vector</td>
</tr>
<tr>
<td>rotx</td>
<td>Homogeneous transform for rotation about X-axis</td>
</tr>
<tr>
<td>roty</td>
<td>Homogeneous transform for rotation about Y-axis</td>
</tr>
<tr>
<td>rotz</td>
<td>Homogeneous transform for rotation about Z-axis</td>
</tr>
<tr>
<td>rpy2tr</td>
<td>Roll/pitch/yaw angles to homogeneous transform</td>
</tr>
<tr>
<td>tr2eul</td>
<td>Homogeneous transform to Euler angles</td>
</tr>
<tr>
<td>tr2rot</td>
<td>Homogeneous transform to rotation sub-matrix</td>
</tr>
<tr>
<td>tr2rpy</td>
<td>Homogeneous transform to roll/pitch/yaw angles</td>
</tr>
<tr>
<td>transl</td>
<td>Set or extract the translational component of a homogeneous transform</td>
</tr>
<tr>
<td>trnorm</td>
<td>Normalize a homogeneous transform</td>
</tr>
</tbody>
</table>

T1 = \( T_{\text{Transy}} \cdot T_{\text{Rotx}} \)

\[
T1 =
\begin{bmatrix}
1.0000 & 0 & 0 & 0 \\
0 & 0.7071 & -0.7071 & 1.0000 \\
0 & 0.7071 & 0.7071 & 0 \\
0 & 0 & 0 & 1.0000
\end{bmatrix}
\]

T2 = \( T_{\text{Rotx}} \cdot T_{\text{Transy}} \)

\[
T2 =
\begin{bmatrix}
1.0000 & 0 & 0 & 0 \\
0 & 0.7071 & -0.7071 & 0.7071 \\
0 & 0.7071 & 0.7071 & 0.7071 \\
0 & 0 & 0 & 1.0000
\end{bmatrix}
\]
3 Forward Kinematics

A definition of forward kinematics could be: “Given the joint variables of the robot, determine the position and orientation of the end-effector.”

The robot is seen as a set of rigid links connected together at various joints. It is enough to consider two kinds of joints, revolute and prismatic joints:

- Both 1 degree-of-motion: angle of rotation and amount of linear displacement respectively.
- Other joints can be seen as a combination of these two.

Suppose a robot has:
- \( n + 1 \) links, numbered from 0 to \( n \)
- \( n \) joints, numbered from 1 to \( n \), where joint \( i \) connects links \( i - 1 \) and \( i \), and each joint having a joint variable \( q_i \) (angle or displacement)
- \( n + 1 \) frames, each frame \( i \) rigidly attached to link \( i \)

Let \( A_i \) be the homogeneous matrix that transforms the coordinates of a point from frame \( i \) to frame \( i - 1 \). \( A_i \) will then be a function of \( q_i \).

The homogeneous matrix from frame \( j \) to \( i \) is called, by convention, a transformation matrix:

\[
T^i_j = \begin{cases} 
    A_{i+1}A_{i+2} \cdots A_j & \text{if } i < j \\
    I & \text{if } i = j \\
    (T^j_i)^{-1} & \text{if } i > j
\end{cases}
\]  

(5)

3.1 Denavit-Hartenberg Representation

To do the kinematic analysis in a systematic manner, some conventions need to be introduced. Here, Denavit-Hartenberg representation is used.

According to the D-H convention, \( A_i \) is represented as:

\[
A_i = \text{Rot}_{x,\theta_i} \text{Trans}_{z, d_i} \text{Trans}_{x, a_i} \text{Rot}_{z, \alpha_i}
\]  

(6)

with the four quantities

- **angle** \( \theta_i \) = the angle between \( x_{i-1} \) and \( x_i \) measured about \( z_{i-1} \).
- **offset** \( d_i \) = distance along \( z_{i-1} \) from \( o_{i-1} \) to the intersection of the \( x_i \) and \( z_{i-1} \) axes.
- **length** \( a_i \) = distance along \( x_i \) from the intersection of the \( x_i \) and \( z_{i-1} \) axes to \( o_i \).
- **twist** \( \alpha_i \) = the angle between \( z_{i-1} \) and \( z_i \) measured about \( x_i \).

For a given link, three parameters are constant, while the forth, \( \theta_i \) for a revolute joint and \( d_i \) for a prismatic joint, is the joint variable \( q_i \).

Only four parameters are needed (not six!), but that limits the freedom in choosing origin and coordinate axes.

The frames and parameters are derived according to the DH-algorithm in [Spong and Vidyasagar, 1989, pages 71-72].
3.2 Robotics Toolbox

In Robotics Toolbox we have the following functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>construct a robot link object</td>
</tr>
<tr>
<td>robot</td>
<td>construct a robot object</td>
</tr>
<tr>
<td>fkine</td>
<td>compute forward kinematics</td>
</tr>
</tbody>
</table>

3.3 Example IRB 7600

As mentioned in the introduction, we will consider the ABB industrial robot IRB 7600 [ABB, 2002], see Fig. 2. For the DH-algorithm we will need some measurements of the different links. These are shown in the blueprint in Fig. 3.

Figure 2: The ABB IRB 7600 manipulator [ABB, 2002].

Steps 1 to 6 in the DH-algorithm give the coordinate systems in Fig. 4. The DH parameters are then derived according to Step 7 (see description of DH parameters in Section 3.1), giving the values shown in Table 2.

<table>
<thead>
<tr>
<th>Joint/Link</th>
<th>$\alpha_i$ [rad]</th>
<th>$a_i$ [m]</th>
<th>$\theta_i$ [rad]</th>
<th>$d_i$ [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-\pi/2$</td>
<td>0.41</td>
<td>0</td>
<td>0.78</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1.075</td>
<td>$-\pi/2$</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>$-\pi/2$</td>
<td>0.165</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$\pi/2$</td>
<td>0</td>
<td>0</td>
<td>1.056</td>
</tr>
<tr>
<td>5</td>
<td>$-\pi/2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>$\pi$</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 2: DH parameters for the IRB 7600 manipulator.

Using the Robotics Toolbox, the IRB 7600 manipulator can now be represented as follows:
Figure 3: A blueprint of the ABB IRB 7600 manipulator [ABB, 2002].

Figure 4: Frames for the IRB 7600 manipulator, according to the DH representation.
% Create links
% L =LINK([alpha A theta D sigma offset], CONVENTION)
clear L
L{1} = link([-pi/2 0.41 0 0.78 0 0 ],'standard');
L{2} = link([0 1.075 -pi/2 0 0 -pi/2],'standard');
L{3} = link([-pi/2 0.165 0 0 0 0 ],'standard');
L{4} = link([pi/2 0 0 1.056 0 0 ],'standard');
L{5} = link([-pi/2 0 0 0 0 0 ],'standard');
L{6} = link([0 0 pi 0.25 0 pi ],'standard');

% Joint limits
L{1}.qlim = pi/180*[-180 180];
L{2}.qlim = pi/180*[-60 85];
L{3}.qlim = pi/180*[-180 60];
L{4}.qlim = pi/180*[-300 300];
L{5}.qlim = pi/180*[-100 100];
L{6}.qlim = pi/180*[-300 300];

% Create robot
irb7600 = robot(L, 'IRB7600');

Using the robot/plot function with \( q = q_x = [0 \ 0 \ 0]^T \), Fig. 5 is generated.

% Plot robot
figure
qz = [0 0 0];
plot(irb7600,qz);

The \textit{fkine} function gives the transformation matrix \( T^6_0 \) from frame 6 to frame 0, which (of course) is the same as the matrix \( t = A_1 A_2 \ldots A_6 \):

% Get Transformation matrix T_0^6
T = fkine(irb7600,qz)

T =
\[
\begin{bmatrix}
-0.5736 & -0.0000 & -0.8192 & 0.5057 \\
-0.0000 & 1.0000 & 0.0000 & 0.0000 \\
0.8192 & 0.0000 & -0.5736 & -0.0106 \\
0 & 0 & 0 & 1.0000
\end{bmatrix}
\]

% Get A_i
q=qz;
L=irb7600.link;
t=diag([1 1 1 1]);
for i=1:irb7600.n,
A=L{i}(q(i));
t=t*A;
end

% t same as T above:
Figure 5: Figure generated by the robot/plot function.

\[
t =
\begin{pmatrix}
-0.5736 & -0.0000 & -0.8192 & 0.5057 \\
-0.0000 & 1.0000 & 0.0000 & 0.0000 \\
0.8192 & 0.0000 & -0.5736 & -0.0106 \\
0 & 0 & 0 & 1.0000
\end{pmatrix}
\]

To get some validation of the model, the manipulator workspace for the wrist center point (WCP) is computed, when varying \( q_2 \) and \( q_3 \) in their range of movement:

\[
\begin{align*}
&\text{t} = \\
&\begin{pmatrix}
-0.5736 & -0.0000 & -0.8192 & 0.5057 \\
-0.0000 & 1.0000 & 0.0000 & 0.0000 \\
0.8192 & 0.0000 & -0.5736 & -0.0106 \\
0 & 0 & 0 & 1.0000
\end{pmatrix}
\end{align*}
\]

The function \texttt{plotrobotws} is written by the authors. The result is shown in Fig. 6, which is similar to the one shown in the product specification of the IRB 7600 robot [ABB, 2002].
Figure 6: Workspace for the IRB 7600 manipulator WCP for variations of $q_2$ and $q_3$ ($q_2$ fixed for each curve in the figure).
4 Velocity Dynamics, Jacobian

The material in this section is mainly based on [Spong and Vidyasagar, 1989].

The Jacobian is a very essential part of robot modeling and control. It is used in

- Planning and execution of smooth trajectories.
- Determination of singular configurations.
- Execution of coordinated (anthropomorphic) motion.
- Derivation of dynamic equations of motion.
- Transformation of forces and torques from the end-effector to the joints.

In case we have an $n$-link robot arm the Jacobian becomes a $6 \times n$ matrix. It describes the transformation of an $n$-vector of joint velocities to a 6-vector of linear and angular velocities of the end effector. It can also be used to describe the velocity of any point of the manipulator.

We start with the forward kinematics

$$ T^n_0(q) = \begin{bmatrix} R^n_0(q) & d^n_0(q) \\ 0 & 1 \end{bmatrix} $$

(7)

where $q$ denotes joint position. For further calculation we treat the angular velocity and the linear velocity separately. The angular velocity vector $\omega^n_0$ of the end effector is defined by the skew symmetric matrix

$$ S(\omega^n_0) = R^n_0(R^n_0)^T $$

(8)

where

$$ S(\omega) = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} $$

(9)

and the linear velocity is

$$ v^n_0 = d^n_0 $$

(10)

What we would like to find is the matrices $J_v$ and $J_\omega$ such that

$$ v^n_0 = J_v \dot{q} $$

(11)

$$ \omega^n_0 = J_\omega \dot{q} $$

(12)

this can also be written

$$ \begin{bmatrix} v^n_0 \\ \omega^n_0 \end{bmatrix} = J^n_0 \dot{q} $$

(13)

where $J^n_0$ is the sought manipulator Jacobian or just Jacobian. This equation can be solved symbolically or numerically.
4.1 Calculation of the Jacobian

We start with looking at the angular velocity. We will calculate the angular velocity of the end effector by looking at the angular velocity of each joint and then sum them up. For a revolute joint the angular velocity of frame $i$ seen from frame $i-1$ is

$$\omega_{i-1}^i = \dot{q}_i z_{i-1}$$  \hspace{1cm} (14)

where

$$z_{i-1} = R_0^{i-1} k$$ \hspace{1cm} (15)

and $k = [0 \hspace{0.2cm} 0 \hspace{0.2cm} 1]^T$. If the joint is prismatic there is no rotation and

$$\omega_{i-1}^i = 0$$ \hspace{1cm} (16)

The total rotation is then calculated as

$$\omega_0^n = \sum_{i=1}^n \rho_i \dot{q}_i z_{i-1}$$ \hspace{1cm} (17)

where $\rho = 0$ for prismatic joint and 1 otherwise.

Now in a second step we will look at the linear velocity. From the chain rule we have

$$\hat{d}_0^n = \sum_{i=1}^n \frac{\partial d_0^n}{\partial \dot{q}_i} \dot{q}_i$$ \hspace{1cm} (18)

We compare this equation with Eq. (11). From this comparison we can see that the $i$-th column of $J_v$ simply is $\frac{\partial \omega_0^n}{\partial \dot{q}_i}$. This means that $J_v,i$ can be calculated by setting $\dot{q}_i = 1$ and all other joint velocities to zero.

Once more we divide the calculation into prismatic or revolute joints. For a prismatic joint we have for the $i$-th joint

$$\hat{d}_0^n = d_0^{i-1} + R_0^{i-1} \hat{d}_{i-1}^n$$ \hspace{1cm} (19)

After differentiation and using the fact that a change of joint $i$ does not change $d_0^{i-1}$ or $R_0^{i-1}$ such that these derivatives becomes zero, we have

$$\dot{d}_0^n = R_0^{i-1} \dot{d}_{i-1}^n = \dot{d}_i z_{i-1}$$ \hspace{1cm} (20)

and this gives

$$J_{v,i} = \frac{\partial d_0^n}{\partial \dot{q}_i} = z_{i-1}$$ \hspace{1cm} (21)

For a revolute joint using the same calculations as above gives us

$$\hat{d}_0^n = R_0^{i-1} \hat{d}_{i-1}^n$$ \hspace{1cm} (22)
using
\[ \dot{d}_{i-1}^n = \dot{q}_i \mathbf{k} \times d_{i-1}^n \quad (v = \omega \times r) \] (23)
gives us
\[ \dot{d}_0^n = \dot{q}_i R_0^{i-1} (\mathbf{k} \times d_{i-1}^n) \] (24)
and
\[ J_{v,i} = \frac{\partial \dot{d}_0^n}{\partial q_i} = R_0^{i-1} (\mathbf{k} \times d_{i-1}^n) \] (25)

4.2 Singularities

Singularities are points in space where two or more of the columns of the Jacobian are linearly dependent of each other. This can be tested as \( \det(J) = 0 \) (if \( n = 6 \)). The reasons why these points are interesting are:

- In a singular point certain directions of motion may be unattainable.
- In a singular point a bounded gripper velocity may lead to unbounded joint velocities.
- In the surrounding of a singularity there will not exist a unique solution to the inverse kinematics problem. It may be no solution or infinitely many solutions.

A typical situation of a singular point is a configuration where it is possible to cancel the rotation of one joint by rotating another joint in the opposite direction.

4.3 Inverse Velocity and Acceleration

The inverse velocity and acceleration is calculated using the Jacobian. First recall that
\[ \dot{x} = J(q)\dot{q} \] (26)
thus the inverse velocity is given by
\[ \dot{q} = J(q)^{-1} \dot{x} \] (27)

Differentiation of Eq. (26) gives
\[ \ddot{x} = J(q)\ddot{q} + \left( \frac{d}{dt}J(q) \right)\dot{q} \] (28)
and the acceleration can be solved as
\[ \ddot{q} = J(q)^{-1} (\ddot{x} - \left( \frac{d}{dt}J(q) \right)J(q)^{-1} \dot{x}) \] (29)
5 Dynamics

Here, the objective is to introduce two methods for deriving the dynamical equations describing the time evolution for a rigid body robot. The material is mainly based on [Spong and Vidyasagar, 1989].

The Euler-Lagrange method is a tool from analytical mechanics where the whole system is analyzed at the same time using the Lagrangian (the difference between kinetic and potential energy). In the Newton-Euler method each link is considered in turn, and the equations describing linear and angular motions are derived. Since the links are coupled, the Newton-Euler method leads to a recursion from the base link to the tool link, and back again.

5.1 Euler-Lagrange Method

Mechanical systems can be described using the Euler-Lagrange method, given that the following two assumptions are met:

- The system must be subject to holonomic constraints.
- The constraint forces must satisfy the principle of virtual work.

A constraint on the $k$ coordinates $r_1, \ldots, r_k$ is called holonomic if it is of the form $g(r_1, \ldots, r_k) = 0$, and non-holonomic otherwise. If a system is subject to $l$ holonomic constraints, the constraint system has $l$ fewer degrees of freedom than the unconstrained system. Then a new set of, so called, generalized coordinates $q_1, \ldots, q_n$ are introduced where $r_i = r_i(q_1, \ldots, q_n)$ and all $q_i$ are independent.

For the principle of virtual work to make sense, virtual displacements must be introduced. Virtual displacements are any set of $\delta_1, \ldots, \delta_k$ of infinitesimal displacements that are consistent with the constraints. The principle of virtual work then requires that the constraint forces do no work during a virtual displacement. If this is the case, only external forces need to be considered, which is the case for the rigid bodies that are considered here.

The Euler-Lagrange method then gives the following equations

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_j} - \frac{\partial L}{\partial q_j} = \tau_j, \quad j = 1, \ldots, n \quad (30)$$

where $L = K - V$ is the Lagrangian, $K$ is the potential energy, and $V$ is the potential energy of the system. These equations are called the Lagrangian equations or Euler-Lagrange equations of motion.

Consider a body with translational velocity $v_c$, and angular velocity $\omega$. The kinetic energy then is

$$K = \frac{1}{2} m v_c^T v_c + \frac{1}{2} \omega^T I \omega \quad (31)$$

where $m$ is the body mass, and $I$ is the inertia matrix. It is important to note that $I$ and $\omega$ must be expressed in the same coordinate system. Usually $I$ is expressed in a body fixed coordinate system which makes it constant in time.

From the velocity kinematics section, we have

$$v_{cij} = J_{cij} \dot{q}, \quad \omega_j = (R_0^T J_{\omega j} \dot{q})$$

14
where \((R_j^0)^T\) is used to transform the angular velocity from the inertial frame to the frame attached to link \(j\). Note that both the Jacobians, \(J_{v_j}\) and \(J_{\omega_j}\), and the transformation matrix, \((R_j^0)^T\), are actually functions of \(q\).

The kinetic energy can now be expressed as

\[ K = \frac{1}{2} q^T \sum_{j=1}^{n} \left[ m_j J_{v_j}^T J_{v_j} + J_{\omega_j}^T R_j^0 \left( R_j^0 \right)^T J_{\omega_j} \right] \dot{q} \]

\[ = \frac{1}{2} q^T D(q) \dot{q} \]

where the “inertia matrix” \(D(q)\) is symmetric and positive definite. The potential energy \(V = V(q)\) is independent of \(\dot{q}\).

Using the Lagrangian equations we finally end up in

\[ M(q) \ddot{q} + C(q, \dot{q}) \dot{q} + g(q) = \tau \]

where the elements of the matrix \(C(q, \dot{q})\) are defined as \((m_{ij} \text{ is the } i, j \text{ element of } M(q))\)

\[ c_{kj} = \sum_{i=1}^{n} \frac{1}{2} \left( \frac{\partial m_{kj}}{\partial q_i} + \frac{\partial m_{ki}}{\partial q_j} - \frac{\partial m_{ij}}{\partial q_k} \right) \dot{q}_i \]

and

\[ g(q) = -\frac{\partial V}{\partial q} \]

This method works fine even for large systems, but is not used for the experiments carried out in this project. Instead the Robotics Toolbox is used, which is based on the Newton-Euler method.

5.2 Newton-Euler Method

This method leads to exactly the same dynamical equations, Eq. (35), as the Euler-Lagrange method, but is rather different. The idea of the Newton-Euler method is to treat each link of the robot in turn, and write down the equations for linear and angular motion. Since each link is coupled to other links, this method leads to a so-called forward-backward recursion.

Since both the Newton-Euler method and the Euler-Lagrange method in the end give the same equations, a fair question is which method is the best. The answer is that it depends; if we are only looking for closed-form equations like Eq. (35), the Lagrangian method might be the best one. However, if are looking for the value of some forces in the structure, then the Newton-Euler method might be the best one.

First, we choose frames 0, \(\ldots\), \(n\) where frame 0 is an inertial frame and frame \(i \geq 1\) is rigidly attached to link \(i\). The following vectors are all expressed in
frame $i$:

\[ \begin{align*}
\dot{a}_{c,i} &= \text{the acceleration of the center of mass of link } i. \\
\dot{a}_{e,i} &= \text{the acceleration of the end of link } i \text{ (i.e. joint } i+1). \\
\omega_i &= \text{the angular velocity of frame } i \text{ w.r.t. frame 0.} \\
\alpha_i &= \text{the angular acceleration of frame } i \text{ w.r.t. frame 0.} \\
g_i &= \text{the acceleration due to gravity.} \\
f_i &= \text{the force exerted by link } i-1 \text{ on link } i. \\
\tau_i &= \text{the torque exerted by link } i-1 \text{ on link } i. \\
R_{i+1}^i &= \text{the rotation matrix from frame } i+1 \text{ to frame } i.
\end{align*} \]

The following vectors are also expressed in frame $i$, but are constant as a function of $q$.

\[ \begin{align*}
m_i &= \text{the mass of link } i. \\
I_i &= \text{the inertia matrix of link } i \text{ about a frame parallel to frame } i \text{ whose origin is at the center of mass of link } i. \\
r_{i,ci} &= \text{the vector from joint } i \text{ to the center of mass of link } i. \\
r_{i,+1,ci} &= \text{the vector from joint } i+1 \text{ to the center of mass of link } i. \\
r_{i,i+1} &= \text{the vector from joint } i \text{ to joint } i+1.
\end{align*} \]

By a recursive procedure in the direction of increasing $i$, values for $\omega_i, \alpha_i$ and $a_{c,i}$ can be found. Using the force balance and moment balance equations, one can derive a recursive procedure in the direction of decreasing $i$ for computing $f_i$ and $\tau_i$. This can be summarized as follows.

1. Start with the initial conditions

\[ \omega_0 = 0, \alpha_0 = 0, a_{c,0} = 0, a_{c,0} = 0 \] (38)

and solve

\[ \begin{align*}
\omega_i &= (R_{i-1}^i)^T \omega_{i-1} + b_i \dot{q}_i \\
\alpha_i &= (R_{i-1}^i)^T \alpha_{i-1} + b_i \dot{\omega}_i + \omega_i \times b_i \dot{q}_i \\
a_{c,i} &= (R_{i-1}^i)^T a_{c,i-1} + \dot{\omega}_i \times r_{i,i+1} + \omega_i \times (\omega_i \times r_{i,i+1}) \quad (41) \\
a_{c,i} &= (R_{i-1}^i)^T a_{c,i-1} + \dot{\omega}_i \times r_{i,ci} + \omega_i \times (\omega_i \times r_{i,ci}) \quad (42)
\end{align*} \]

to compute $\omega_i, \alpha_i$ and $a_{c,i}$ for $i$ increasing from 1 to $n$.

2. Start with the terminal conditions

\[ f_{n+1} = 0, \tau_{n+1} = 0 \] (43)

and use

\[ \begin{align*}
f_i &= R_{i+1}^i f_{i+1} + m_i a_{c,i} - m_i g_i \\
\tau_i &= R_{i+1}^i \tau_{i+1} - f_i \times r_{i,ci} + (R_{i+1}^i f_{i+1}) \times r_{i+1,ci} \\
&\quad + \alpha_i + \omega_i \times (I_i \omega_i) \\
&\quad + \alpha_i + \omega_i \times (I_i \omega_i)
\end{align*} \quad (45)

(46)

to compute $f_i$ and $\tau_i$ for $i$ decreasing from $n$ to 1.
5.3 Example IRB 7600 cont.

Here, we will continue using the ABB industrial robot IRB 7600 [ABB, 2002], introduced in Section 3.3 as an example. However, since the dynamical data are not given in [ABB, 2002], values for the inertia matrix $I$ and the mass center position $r$ for each link corresponds to a robot of the same size as the IRB 7600. To simplify the dynamics, we will also consider only three links, and therefore change the definition of the mass, length and inertia for link 3 to incorporate the properties of links 4-6 as well. See Fig. 7 for a plot of the robot with frames.

Figure 7: The modified IRB 7600 robot (only 3 axes).

```matlab
% Create links
% L = LINK([alpha A theta D sigma offset], CONVENTION)
clear L
L{1} = link([-pi/2 0.41 0 0.78 0 0],'standard');
L{2} = link([0 1.075 -pi/2 0 0 -pi/2],'standard');
L{3} = link([pi/2 1.056 pi/2 0 0 pi/2],'standard');

% Joint limits
L{1}.qlim = pi/180*[-180 180];
L{2}.qlim = pi/180*[-60 85];
L{3}.qlim = pi/180*[-180 60];

% Arm inertia matrices
L{1}.I = diag([56.3 89 58.4]);
L{2}.I = diag([1.4 12 12]);
L{3}.I = [4.84 0 4.4
         0 268.5 0
         4.4 0 267.9];
```

17
% Arm masses
L{1}.m = 480;
L{2}.m = 264;
L{3}.m = 426.7;

% Vector to mass center
L{1}.r = [-0.4011 0.4774 -0.0354]’;
L{2}.r = [-0.5390 0.0140 -0.2350]’;
L{3}.r = [-0.1477 0 0.3026]’;

In Robotics toolbox, there are quite a few functions for dynamics, see Table 5.3, with the most important one being the rne function, where rne stands for recursive Newton-Euler. All functions except fdyn and friction use the rne function.

RNE Compute inverse dynamics via recursive Newton-Euler formulation

TAU = RNE(ROBOT, Q, QD, QDD)
TAU = RNE(ROBOT, [Q QD QDD])

Returns the joint torque required to achieve the specified joint position, velocity and acceleration state.

Gravity vector is an attribute of the robot object but this may be overridden by providing a gravity acceleration vector [gx gy gz].

TAU = RNE(ROBOT, Q, QD, QDD, GRAV)
TAU = RNE(ROBOT, [Q QD QDD], GRAV)

An external force/moment acting on the end of the manipulator may also be specified by a 6-element vector [Fx Fy Fz Mx My Mz].

TAU = RNE(ROBOT, Q, QD, QDD, GRAV, FEXT)
TAU = RNE(ROBOT, [Q QD QDD], GRAV, FEXT)

where Q, QD and QDD are row vectors of the manipulator state; pos, vel, and accel.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accel</td>
<td>compute forward dynamics</td>
</tr>
<tr>
<td>cinertia</td>
<td>compute Cartesian manipulator inertia matrix</td>
</tr>
<tr>
<td>coriolis</td>
<td>compute centripetal/coriolis torque</td>
</tr>
<tr>
<td>fdyn</td>
<td>integrate forward dynamics (motion for given torques)</td>
</tr>
<tr>
<td>friction</td>
<td>joint friction</td>
</tr>
<tr>
<td>gravload</td>
<td>compute gravity loading on joints</td>
</tr>
<tr>
<td>inertia</td>
<td>compute manipulator inertia matrix (M)</td>
</tr>
<tr>
<td>itorque</td>
<td>compute inertia torque</td>
</tr>
<tr>
<td>rne</td>
<td>inverse dynamics (torques for given motion)</td>
</tr>
</tbody>
</table>

Table 3: Functions in Robotics Toolbox [Corke, 1996] for dynamics.
6 Path and Trajectory Generation

In this section we will explain different ways of generating the path and trajectory for the reference.

The chapter tries to cover the material presented in [Craig, 1989, Chapter 7] and [Sciavicco and Siciliano, 2002, Chapter 5]. A suggestion for further reading is the Ph.D. thesis by Ola Dahl [Dahl, 1992].

6.1 General

First we need to define some notations or concepts.

Path: When we talk about path we are only considering the geometric path in space including position and orientation.

Trajectory: This is the same as path with the addition of a time law. The subject of Trajectory generation includes the problem of finding the time history of position, velocity, and acceleration for each degree of freedom.

There are a lot of different aspects that must be considered. The most important ones are:

• How should one specify a trajectory? An example can be that only goal position and orientation is specified by the user and the path and velocity profile is figured out by the system itself.

• How is the trajectory represented internally in the computer?

• How should we compute the trajectory from the internal representation?

We will try to give some answers to these questions in the following pages.

Typically first the user has certain demands on the system. Things which are often given by the user are,

• Goal position and orientation.

• Via points, intermediate points between start and goal position.

• Sometimes both spatial and temporal information. The spatial information defines a path in space. This is often specified using coordinates. The temporal information adds a time dependence. It is often specified by giving velocities and accelerations.

Another requirement that is preferable for a control point of view is that the motion is smooth. What we mean by this is:

• The trajectory has a continuous position. This is always a requirement because the robot can not move from one position to another in zero seconds.

• The trajectory has a continuous derivative. This is almost always the case since a step in the velocity requires the acceleration to be infinite.

• Continuous second derivative, acceleration. This is preferable and therefore sometimes a requirement. The reason is because jerky motions tend to cause increased wear in the robot and also cause vibrations.
6.2 Joint Space Motion

The first type of motion we will look at is joint space motion. We assume that the user has specified the path points, which include initial and final positions and optionally some sequence of via points or intermediate points. Each path point includes position and orientation of the tool frame and is specified in the Cartesian coordinate system. Therefore, as a first step, path points in the Cartesian coordinate system is transformed to joint angles using inverse kinematics.

Now the path is the specified in joint space. This have the advantages that it is easy to compute and there are no problems with singularities. In the next section we will look at how to compute the trajectory between the points.

6.2.1 Point-To-Point Motion

Let the first position be represented by $q_0$ and the end position be represented by $q_f$ ($f$ for final). By moving from one point to another the constraints on the motion become

\begin{align*}
q(0) &= q_0 \\
q(t_f) &= q_f \\
\dot{q}(0) &= 0 \\
\dot{q}(t_f) &= 0
\end{align*}  

(47)

Let the motion be defined by a cubic polynomial

\[ q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \]  

(48)

Then solving for Eq. (47) and Eq. (48) gives

\begin{align*}
a_0 &= q_0 \\
a_1 &= 0 \\
a_2 &= \frac{3}{t_f^2} (q_f - q_0) \\
a_3 &= -\frac{2}{t_f^3} (q_f - q_0)
\end{align*}  

(49)

An example of a motion using a cubic polynomial can be seen in Fig. 8.

6.2.2 Suitable Trajectory with Cubic Polynomial?

This cubic polynomial originates from the solution to an optimization problem (see [Sciavicco and Siciliano, 2002]). First we assume that the robot can be modeled as a rigid body. The dynamics is then, using $I$ to denote the inertia and $\tau$ for the torque,

\[ I\ddot{q}(t) = \tau \]  

(50)

Assuming the same distance as in Eq. (47) gives

\[ \int_0^{t_f} \dot{q}(t) dt = q_f - q_0 \]  

(51)
Minimizing the criterion
\[
\min_{\tau} \int_{t_0}^{t_f} \tau^2(t)dt
\]  
subject to \[\int_{t_0}^{t_f} \dot{q}(t)dt = q_f - q_0\]
\[I\ddot{q}(t) = \tau\]
gives a solution of the form
\[\dot{q}(t) = at^2 + bt + c\]
This is exactly the cubic polynomial in Eq. (48).

### 6.2.3 Points in Between
As a next step we want to be able to pass through points without stopping. (Otherwise point-to-point motion can be used.)
Replace Eq. (47) with \[\dot{q}(0) = \dot{q}_0, \dot{q}(t_f) = \dot{q}_f\]. Then the solution is similar to Eq. (49). The remaining problem is how to find \(\dot{q}_0\) and \(\dot{q}_f\)? The velocities are usually found from one of the following ways:

1. The velocities are specified by the user in Cartesian coordinates. Then the inverse Jacobian is used to find joint velocities.

2. The velocities are specified by the system using some heuristic way. A typical example is that the mean velocity is chosen if passing the point and zero velocity is chosen if the velocity change direction.

3. The velocities are chosen by the system such that the acceleration is continuous, e.g. using cubic splines. One drawback is that the more points in the path the bigger the equation system for solving the coefficients becomes. Therefore, the whole trajectory must be known before the motion take place and this also means that it may be computationally expensive.

It is also possible to use higher order polynomials. Then, for example, the position, velocity and acceleration at the beginning are specified. To be able to solve this a 5th order polynomial must be used. The path generation in the Robotics toolbox uses a 7th order polynomial [Corke, 1996], see Fig. 9 for an example using \texttt{jtraj} from the toolbox.
6.2.4 Linear Function with Parabolic Blends

Another way of approaching the subject is to separate the motion in different regions. In this section the motion is divided in an acceleration phase, a constant velocity phase, and a deceleration phase. The region between two regions with constant velocity is called a blend region. A constant acceleration in the blend region is concatenated together with linear path with constant velocity such that it is continuous in position and velocity. This makes the position a parabolic function of time. An example of this type of motion can be seen in Fig. 10.

If extended to many points, the path does not pass through the points. This can be fixed with pseudo via points. That is, placing points between the original points so that the constant velocity path passes throw the original points. Details can be found in [Craig, 1989].

6.3 Cartesian Space Schemes (Operational Space)

So far we have only discussed motions in joint space. In this section we look at motions in Cartesian space, which is the same as operational space in this context. We will look at Cartesian straight line motion (spline of linear functions with parabolic blends).

The rotation matrix cannot be interpolated in a linear way because then it will not be a correct rotation matrix. We will therefore use angle-axis represen-
tation instead of rotation matrix. The angle-axis representation consists of a 6 × 1 vector called χ. This angle-axis representation can then be used for linear interpolation. The solution can be summarized as:

First χ is selected for each via point and then the same methods as for joint space is used for the interpolation between points.

Then, for every sample of the Cartesian space trajectory, inverse kinematics is used to give the joint angles. The inverse kinematics is very expensive computationally. Therefore, if a very fine grid is needed further points can be interpolated in joint space.

The inverse Jacobian can be used for velocity and it and its derivative can be used for acceleration. Often numerical differentiation is used for the velocity and acceleration.

Note that the same blend time must be used for each degree of freedom if the path should be linear in Cartesian space.

6.3.1 Geometric Problems

There are several additional problems that can be encountered when planning paths in Cartesian space.

1. Intermediate points may be unreachable, because of the physical structure of the robot. (This problem doesn’t exist in joint space, if we do not take into account collisions.)


3. Start and goal reachable in different solutions of the inverse kinematics.

6.4 Collision-Free Path Planning

This is a subject open to active research. One can see two major ways in the research community.

The first approach is based on a graph representation of space, dividing it into small volumes which are connected to other volumes. This leads to an exponential complexity in the number of joints and is therefore quite computationally expensive.

The second approach is based on artificial potential fields. Near an object the potential field approaches infinity. The trajectory is then found using optimization. The potential fields prevent the trajectory from going through an object.

But this is still an active research field and it will not be discussed more in detail in this document.

6.5 Programming Language, Rapid

In this section we present an example of how the trajectory is specified by the user. In this case the robot programming language Rapid by ABB Robotics is used.

First an example where joint motion is used from the current position, p0, to position, p1 with maximum velocity, vmax.

MoveJ p1, vmax, fine, tool1;
The word **fine** means that the trajectory must pass exactly through that point and **tool1** just specifies which tool is used for the moment. Using a language such as Rapid it is possible to define symbolic positions that automatically compensates the tool position if another tool is used.

Other commands in Rapid are: **MoveL** for linear motion in operational space, **MoveC** for circular motion and many other control structures, counters, i/o communication and so on.

### 6.6 Trajectory Planning using Matlab™

This section lists the different functions related to path planning using the Robotics Toolbox in Matlab™. There are mainly three commands for trajectory generation: **jtraj**, **ctraj**, and **interp**.

<table>
<thead>
<tr>
<th>jtraj</th>
<th>Computes a joint space trajectory between two points.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctraj</td>
<td>Computes a Cartesian trajectory between two points.</td>
</tr>
<tr>
<td>interp</td>
<td>Interpolation of homogeneous transformations.</td>
</tr>
</tbody>
</table>
7 Simulations

In this section, simulations of the modified ABB IRB 7600 robot are shown. The robot model is based on the rigid model from Section 5.3 and then extended with gear-box flexibilities. This has the advantage that the Robotics Toolbox can be used for the rigid part of the robot.

We will see simulations comparing different control strategies, namely diagonal PID and LQ based control. We will also see how a simple feedforward controller, based on the rigid model, can be used together with the diagonal PID controller for improved performance, see Fig. 11.

The performance of the controllers depends very much on how they are tuned. Therefore it is impossible to compare the results from different controllers with each other by only looking at the simulations. Here we will only try to show some typical behaviors.

![Figure 11: An example of a robot controlled by feedforward and feedback controllers.](image)

### 7.1 Model Equations of Flexible Robot

If Eq. (35) is linearized at $q = q_0$, $\dot{q} = 0$, $\tau = \tau_0$, the following equation is derived

$$M(q_0)\ddot{q} = \tau$$  \hspace{1cm} (54)

where, for easy notation, $\tau$ now denotes offset torque from $\tau_0$. The linearized model is used when calculating the LQ controller. In the simulations the nonlinear model is used.

Let’s introduce a gear-box with flexibility according to Fig. 12, where the flexibility is on the arm side (after the gear-box). The linearized system can then be described as

$$M_m\ddot{q}_m + N^{-1}K(N^{-1}q_m - q_a) + N^{-1}D(N^{-1}q_m - \dot{q}_a) + F_m\dot{q}_m = \tau_m$$  \hspace{1cm} (55a)

$$M_a\ddot{q}_a - K(N^{-1}q_m - q_a) - D(N^{-1}q_m - \dot{q}_a) = 0$$  \hspace{1cm} (55b)

where $M_m$ is the motor inertia, $N$ is the gear ratio, $K$ is the spring constant, $D$ is the damper constant, $F_m$ is the viscous friction constant, and $M_a$ is the
Figure 12: Two-mass flexible model of the robot arm.

arm inertia. This description can easily be extended to more than one axis if we allow \( q_m \) and \( q_a \) to be vectors, let \( M_a = M(q_a) \) from Eq. (54), and extend all other parameters to diagonal matrices.

The following values have been used, which are values for a robot of the same size as the ABB IRB 7600.

\[
\begin{align*}
M_m &= \text{diag}(5.7 \cdot 10^{-3}, \ 4.3 \cdot 10^{-3}, \ 4.4 \cdot 10^{-3}) \\
N &= \text{diag}(200, \ 210, \ 201) \\
K &= \text{diag}(3.62 \cdot 10^6, \ 2.22 \cdot 10^6, \ 1.13 \cdot 10^6) \\
D &= \text{diag}(400, \ 400, \ 400) \\
F_m &= \text{diag}(10^{-4}, \ 10^{-4}, \ 10^{-4})
\end{align*}
\]

In the position \( q_a = [0 \ 0 \ 0]^T \), the inertia matrix \( M_a \) is

\[
\begin{align*}
\text{ans} &= \text{inertia(irm7600, [0 \ 0 \ 0])} \\
&= \begin{bmatrix}
1.0e+03 * \\
1.1625 & 0.0333 & 0.0000 \\
0.0333 & 1.5182 & 0.7984 \\
0.0000 & 0.7984 & 0.6596
\end{bmatrix}
\end{align*}
\]

In a straightforward way, Eq. (55) can be converted into a state space form

\[
\begin{align*}
\dot{x} &= Ax + Bu \\
y &= Cx
\end{align*}
\]

with state vector \( x = \begin{bmatrix} q_a^T & q_m^T & q_a^T & q_m^T \end{bmatrix}^T \), input \( u = \tau_m \), output \( y = q_a \), and matrices \( A, B, \) and \( C \) defined as follows.

\[
A = \begin{bmatrix}
0 & 0 & I & 0 \\
0 & 0 & 0 & I \\
-M_m^{-1}K & M_m^{-1}KN^{-1} & -M_m^{-1}D & M_m^{-1}DN^{-1} \\
M_m^{-1}N^{-1}K & -M_m^{-1}N^{-1}KN^{-1} & M_m^{-1}N^{-1}D & -M_m^{-1}(N^{-1}DN^{-1} + F_m)
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
0 \\
0 \\
0 \\
M_m^{-1}
\end{bmatrix}
C = \begin{bmatrix} I & 0 & 0 \\
\end{bmatrix}
\]

Using this state space form, the Bode diagram in Fig. 13 is derived. We can see that there is large coupling between axes two and three. Since the robot model is asymmetric, there is also a coupling effect between axis 1 and axes 2 and 3.
7.2 Preliminaries

The main goal of control is to get the arm to follow a specified trajectory. It is relatively easy to find a reference trajectory for the arm. However, standard procedure in robotics today is to measure only the motor position (to increase performance, additional sensors are sometimes added). Due to the gear-box flexibilities, there will be some dynamics between motor and arm positions. In our application we assume that we know all states, which include both motor and arm position. How should we now construct a reference signal for the motor position?

If only the motor positions are measured, there are at least two options. One way is to try to come up with appropriate reference signals for the motor positions which will give a good behavior for the arm positions. Another way would be to use an observer to estimate the arm positions and then apply a controller.

This is a pretty big issue and is not covered in this document. Instead the same reference is used for arm and motor position, which will give reduced performance. The reference signal is generated using a linear function with parabolic blends as explained in Section 6.2.4. It is a 5cm motion in 0.25s. One can also argue that a reference signal for the velocity should be created, but we will not go into details on how the reference signal should be created. The reason is that this is only important for the LQ controller and for evaluation of the LQ controller, only disturbance rejection is considered and therefore a zero reference is used in order to avoid the problem of reference signals.

7.3 Comparison With and Without Feedforward Control

Here we will look at a feedforward addition to the diagonal PID control. The feedforward part is only based on the rigid model, but the simulations are per-
formed using the addition of the flexible gear-boxes.

The feedforward part is calculated using the inertia, $I$, at the current position and the acceleration to get the feedforward torque as

$$\tau_{ff} = I\ddot{q}_{ref}$$

(58)

where $\ddot{q}_{ref}$ is the acceleration reference. Although the inertia is a function of the position $q$ of the robot, we will only calculate it for the initial position of movement. This is a good approximation for the reference used here. If the robot would do a greater movement the inertia must be recalculated fairly often.

Both the PID controller and the LQ controller in the next section are tuned for reference performance. The PID controller was given the following values

$$P_1 = 10, \quad I_1 = 40, \quad D_1 = 1$$

(59a)

$$P_2 = 20, \quad I_2 = 40, \quad D_2 = 1$$

(59b)

$$P_3 = 10, \quad I_3 = 40, \quad D_3 = 1$$

(59c)

PID control of the robot without feedforward control can be seen in Fig. 14.

![Figure 14: PID control of the robot without feedforward.](image)

If this is compared with PID and feedforward control in Fig. 15 we can see that the overshot is less than half of that acquired using only PID control. The idea behind the feedforward is that the feedforward part should contribute to the main part of the control signal. The feedforward part only affects the transfer function from reference signal to output. Theoretically it is possible to make it the inverse of the system and thus have perfect tracking. In practice this is not possible due to model errors and disturbances. This must be handled by the controller. Here we have only used a model of the robot without flexibilities to show how the feedforward and the feedback parts of the controller work together.
7.4 Comparison PID Control and LQ Control

An LQ controller (see e.g. [Glad and Ljung, 2001] for details) is computed using the augmented system

\[ \dot{x} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix} x + \begin{bmatrix} B \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \]  
\[ y = \begin{bmatrix} C \\ 0 \end{bmatrix} x \]  
(60a)

with \( u = -Lx + Lr r \). The reason for using the augmented system is to achieve an integral part in the controller to remove static errors in the arm position due to disturbances (can be seen as PID instead of PD).

\( L \) is calculated using the MATLAB \texttt{lqr} function with \( Q_1 \) (penalty for states) and \( Q_2 \) (penalty for control signal) being

\[ Q_1 = \text{\texttt{diag}}(10^{5}, 10^{5}, 10^{5}, 0, 0, 0, 0, 0, 0, 0.1, 0.1, 0.1, 10^{11}, 10^{11}, 10^{11}) \]  
\[ Q_2 = \text{\texttt{diag}}(1, 1, 1) \]  
(61a)

\[ Q_1 \) and \( Q_2 \) have been chosen after a tuning procedure where we have tested different choices and simulated the system to get a performance comparative to the PID controller in Section 7.3. We have chosen to have big penalty terms for the arm positions and also some for the motor velocity. If we skip the penalty on the motor velocity, we will have very large oscillations on the motor side. Then there is a penalty on the integral states, to remove static errors due to disturbances.

\( L_r \) is calculated as

\[ L_r = L_{1:3} + L_{4:6} N \]
which means that the arm reference signal is scaled and used as motor reference as well. This is not optimal and any conclusion of the reference performance should not be drawn. Instead the LQ controller is evaluated using disturbances. In that case the reference signal is zero and the way the reference signal enters the system has no importance.

![Figure 16](image)

Figure 16: LQ control of the robot without feedforward.

![Figure 17](image)

Figure 17: Disturbance torque applied to axis 1 (solid), axis 2 (dotted) and axis 3 (dash-dotted) respectively.

Torque disturbances according to Fig. 17 are used to evaluate the LQ control. The reason is that better behavior of the closed loop system from reference to output can be achieved using prefiltering or feedforward (in theory at least). For the interested reader the closed loop system using the PID controller can be seen in Fig. 18 and the closed loop system using the LQ controller can be seen in Fig. 19. As said before, both the LQ controller and the PID controller are tuned for reference performance (simulations using the reference signal are shown in Fig. 14 and Fig. 16 respectively).
Figure 18: Bode diagram for the closed loop system using the PID controller in Eq. (59).

Figure 19: Bode diagram for the closed loop system using the LQ controller.

To evaluate the LQ controller, a torque disturbance is applied according to Fig. 17. A simulation of disturbances using LQ control of the robot can be seen in Fig. 20. This can be compared to the simulation of disturbances using PID control in Fig. 21. From the comparison we can see that the effect of the disturbance is a factor 5 lower for the LQ controller.
Figure 20: LQ control of the robot with disturbances.

Figure 21: PID control of the robot with disturbances.
8 Conclusions and Remarks

In this work we have looked at various parts of modeling of robots. First the rigid body motion was studied, spanning from kinematics to dynamics, and secondly, path and trajectory generation was covered.

For evaluation of some control strategies, a more realistic robot model was needed. This robot model was implemented with a rather simple addition of gear-box flexibilities to the rigid body dynamics from the Robotics Toolbox [Corke, 1996].

In simulations, we have seen how the addition of very simple feedforward improves the control despite that we have neglected the flexibilities in the feedforward calculations. The overshoot is halved compared to only using PID control.

In the comparison of LQ control versus PID control it is much harder to draw conclusions. The reason is that it depends very much on the tuning of the controllers. However, in our simulations the effect of torque disturbances on the arm has been lowered by a factor five, compared to diagonal PID controllers.

LQ versus PID, as stated in this document, is actually a competition between complex and simple controllers. The PID controller is diagonal, which means that for the control of a certain axis, it only uses information about the position and velocity of that particular axis. In addition, it only uses motor side information. The LQ controller, on the other hand, has access to arm side measurements and uses information from all axes for the control of each axis. Since the LQ controller have more degrees of freedom (actually 45 compared to 9), it will be superior if it is well-tuned. The LQ controller will be able to suppress coupling effects between the axes, since information about other axes is used for each controller. In addition, the measurements of the arm position will improve disturbance rejection and the ability to follow a given trajectory. The price we must pay for the improved performance mainly is additional hardware such as arm sensors and a more complex control structure.
References


