Simulation of the inverted pendulum

Interdisciplinary Project in
Computer Science and Mathematics

Christian Wachinger
Computer Science

Michael Pock
Computer Science

Project Supervisor:
Prof. Dr. Peter Rentrop
TU München

17.12.2004

Contents

Abstract

In this paper, the problem of the inverted n-bar pendulum in the plane is discussed. The descriptor form and the state space form, which corresponds to an ordinary differential equation system (ODE), are deduced. The simulating systems for solving the ODE are described and controllers for the one-bar and the two-bar case are developped. Also, a neural network for the task of controlling the inverted pendulum was trained.

1 Task of the interdisciplinary project

The project was done by two computer science students and the project was supervised by Professor Rentrop, chair of numerical mathematics at TU München. The task was split up into three parts:

1. We had to model the inverse n-bar pendulum in the plane with the help of the descriptor and state space form. The descriptor form bases on redundant coordinates and results in a differential algebraic equation of index 3. It is possible to solve the constraints explicitly as the inverse n-bar pendulum has a tree structure. This is the transfer from the descriptor to the state space form which is characterised by a minimal set of local coordinates. The state space form is a system of ordinary differential equations.
   The pendulum should be simulated with the help of the mathematical development environment Matlab.

2. The modelled system should be controlled by a classical PD-controller, which can be deduced from the pendulum equations. With help of this controller a neural network should be developed, which can also regulate the inverted pendulum.

3. The simulation of the pendulum should be visualized with the help of the graphical tools in Matlab.

2 Mechanical model

2.1 General problem

The problem we have to deal with can be described with the help of the equations of motion, which lead to a large system of differential algebraic equations. [3] These are the Lagrange equations of the first kind, which are in descriptor form. They describe a mechanical system of bodies with massless connections. The equations of motion according to Newton without constraints are

$$\begin{aligned} \dot{p} &= v \\ M(p,t)\dot{v} &= f(p,v,t) \end{aligned} \tag{1}$$

with $p \in \mathbb{R}$ the vector of position coordinates, $v \in \mathbb{R}^n$ the vector of the velocity coordinates, $M(p,t) \in \mathbb{R}^{n,n}$ the symmetric and positive definite mass matrix and $f(p,v,t) \in \mathbb{R}^n$ the vector for the applied external forces. The connections cause the following constraints which are also denoted as the geometry of the system

$$0 = b(p,t), \quad b(p,t) \in \mathbb{R}^q.$$

By using Lagrange multipliers $\lambda(t) \in \mathbb{R}^q$, equation (1) can be transformed to the descriptor form

$$\begin{aligned} \dot{p} &= v \\ M(p,t)\dot{v} &= f(p,v,\lambda,t) - (\frac{\partial}{\partial p})b(p,t))^T\lambda \\ 0 &= b(p,t). \end{aligned} \tag{2}$$

The equation (2) is a *differential-algebraic system* (DAE) of index three. In the following sections we will apply these equations to the inverted pendulum problem. So far it is not possible to derive a controller directly from the descriptor form, so it is necessary to transfer them into the state space form. It corresponds to an *ordinary differential equation system* (ODE)

$$\tilde{M}(\tilde{p},t)\ddot{\tilde{p}} = \tilde{f}(\tilde{p},\tilde{v},t) \tag{3}$$

System (3) is also known as the *Lagrange equations of the second kind.*

2.2 Inverted one-bar pendulum

In this part we will concentrate on modelling an inverted one-bar pendulum. The model of the mechanical system of this pendulum can be seen in figure 1. It consists of a cart with mass m concentrated in the joint and one bar with length $2l_1$ and mass m_1 concentrated in its centre. The car can move in horizontal direction. The inertia I_1 is under the influence of the gravity. Coordinates p are:

- the position x of the cart
- the position x_1, y_1 of the bar centre
- the angle φ_1.

Introducing constraint forces $F_{z1}, \cdots, F_{z4}$ the Newton-Euler formulation gives

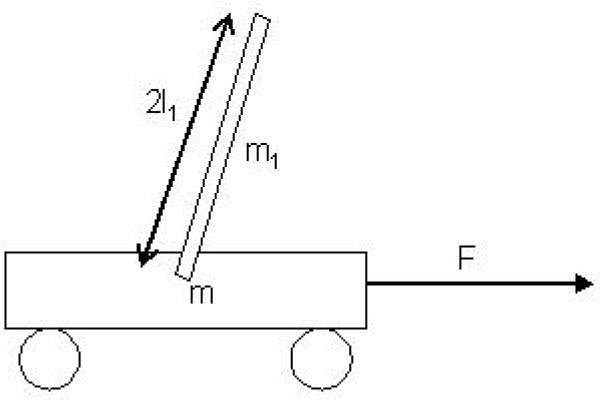


Figure 1: Cart with inverted one-bar pendulum

$$\begin{aligned}
m\ddot{x} &= F + F_{z1}\\
m_1\ddot{x}_1 &= F_{z2}\\
m_1\ddot{y}_1 &= -m_1 g + F_{z3}\\
I_1\ddot{\varphi}_1 &= F_{z4}\\
\\
0 &= x - x_1 + l_1 sin(\varphi_1)\\
0 &= y_1 - l_1 cos(\varphi_1)
\end{aligned}$$

The last two equations describe the geometry

$$b(p) = b(x, x_1, y_1, \varphi_1) = 0$$

of the system. The inertia of a bar with length $2l_1$ is according to Steiner

$$I_1 = \frac{4}{3} m_1 l_1^2.$$

With the help of D'Alembert, see [3], we get the descriptor form:

$$\begin{aligned}
m\ddot{x} &= F + \lambda_1\\
m_1\ddot{x}_1 &= -\lambda_1\\
m_1\ddot{y}_1 &= -m_1 g - \lambda_2\\
I_1\ddot{\varphi}_1 &= \lambda_1 l_1 cos\varphi_1 - \lambda_2 l_1 sin\varphi_1\\
0 &= x - x_1 + l_1 sin(\varphi_1)\\
0 &= y_1 - l_1 cos(\varphi_1)
\end{aligned} \tag{4}$$

The state space form can be deduced from (4) by solving for x and φ_1.

$$\begin{pmatrix} m + m_1 & m_1 l_1 cos\varphi_1 \\ m_1 l_1 cos\varphi_1 & I_1 + m_1 l_1^2 \end{pmatrix} \begin{pmatrix} \ddot{x} \\ \ddot{\varphi}_1 \end{pmatrix} = \begin{pmatrix} F + m_1 l_1 sin\varphi_1 \dot{\varphi}_1{}^2 \\ m_1 l_1 sin\varphi_1 g \end{pmatrix} \tag{5}$$

(5) is a classical state space formulation consisting of two second order ODEs with a symmetric mass-matrix.

2.3 Inverted n-bar pendulum

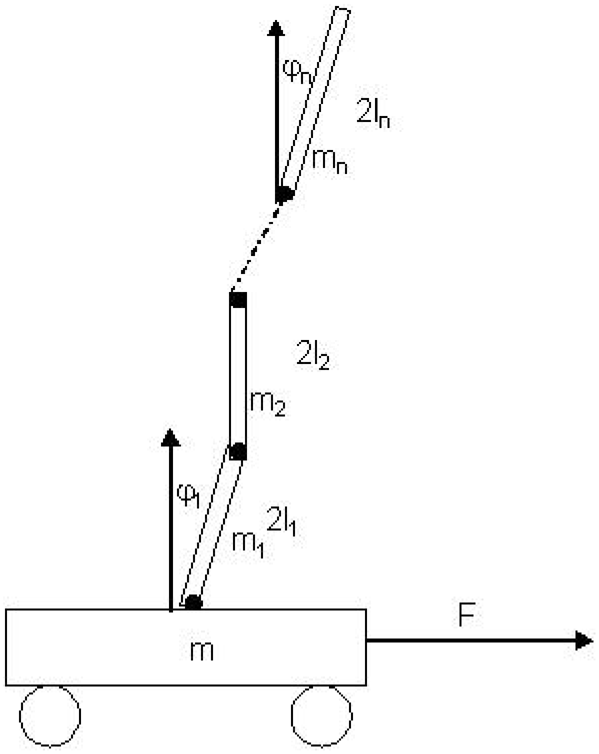


Figure 2: Cart with inverted n-bar pendulum

The layout of the cart with the n-bars can be seen in the figure 2. The equations of motion can be written analogously to the one-bar problem. The state space form is listed below, the deduction can be seen in [3]. We define:

$$c_{ji} := c_{ji}(\varphi) = cos(\varphi_j - \varphi_i), \quad s_{ji} := s_{ji}(\varphi) = sin(\varphi_j - \varphi_i)$$

$$\mathbf{m}_i = \sum_{k=i+1}^{n} m_k, \quad i, j = 1, \cdots, n$$

The following matrix M is symmetric:

$$M = \begin{pmatrix} m + \mathbf{m}_0 & (m_1 + 2\mathbf{m}_1)l_1 \cos\varphi_1 & \cdots & (m_j + 2\mathbf{m}_j)l_j \cos\varphi_j & \cdots & \cdots & \cdots & m_n l_n \cos\varphi_n \\ \cdots & I_1 + (m_1 + 4\mathbf{m}_1)l_1^2 & \cdots & 2(m_j + 2\mathbf{m}_j)l_i l_j & \cdots & \cdots & \cdots & 2m_n l_1 l_n c_{n1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & I_i + (m_i + 4\mathbf{m}_i)l_i^2 & \cdots & 2(m_j + 2\mathbf{m}_j)l_i l_j c_{ji} & \cdots & 2m_n l_i l_n c_{ni} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & I_n + m_n l_n^2 \end{pmatrix}$$

$$B = \begin{pmatrix} F \\ (m_1 + 2\mathbf{m}_1)g l_1 \sin\varphi_1 \\ \vdots \\ (m_i + 2\mathbf{m}_i)g l_i \sin\varphi_i \\ \vdots \\ m_n g l_n \sin\varphi_n \end{pmatrix} + \begin{pmatrix} (m_1 + 2\mathbf{m}_1)l_1 \sin\varphi_1 \\ 2(m_1 + 2\mathbf{m}_1)l_1^2 s_{11} \\ \vdots \\ 2(m_i + 2\mathbf{m}_i)l_1 l_i s_{1i} \\ \vdots \\ 2m_n l_1 l_n s_{1n} \end{pmatrix} \dot{\varphi}_1^2 + \cdots + \begin{pmatrix} (m_j + 2\mathbf{m}_j)l_j \sin\varphi_j \\ 2(m_j + 2\mathbf{m}_1)l_j l_1 s_{j1} \\ \vdots \\ 2(m_j + 2\mathbf{m}_i)l_j l_i s_{ji} \\ \vdots \\ 2m_n l_j l_n s_{jn} \end{pmatrix} \dot{\varphi}_j^2 +$$

$$+ \cdots + \begin{pmatrix} m_n l_n \sin\varphi_n \\ 2m_n l_1 l_n s_{n1} \\ \vdots \\ 2m_n l_i l_n s_{ni} \\ \vdots \\ 2m_n l_n^2 s_{nn} \end{pmatrix} \dot{\varphi}_n^2$$

The general compact state space form is:

$$M \begin{pmatrix} \ddot{x} \\ \ddot{\varphi}_1 \\ \vdots \\ \ddot{\varphi}_i \\ \vdots \\ \ddot{\varphi}_n \end{pmatrix} = B \tag{6}$$

3 Simulation systems

For the simulation of the inverted pendulum problem, we have used the mathematical development environment Matlab. Matlab was chosen, as it is widely used in the field of numerical mathematics and supports solving ordinary differential equations. Moreover, it is possible to visualize the simulation results. In our program we used the ode45, a standard solver included in Matlab, to solve the ordinary differential equation (ODE). The ode45 implements the method of Dormand-Prince, which is a member of the class of Runge-Kutta-Fehlberg methods. The reason why we need such a solver is that it is not possible to solve the ODE analytically.

3.1 One-step solver

For solving an initial value problem

$$y' = f(x,y), \quad y(x_0) = y_0 \tag{7}$$

a numerical method is needed. One step solver are defined by a function $\Phi(x,y,h;f)$ which gives approximated values $y_i := y(x_i)$ for the exact solution $y(x)$:

$$\begin{aligned} y_{i+1} &:= y_i + h\Phi(x_i, y_i, h; f) \\ x_{i+1} &:= x_i + h \end{aligned}$$

where h denotes the step size. In the following be x and y arbitrary but fixed, and $z(t)$ is the exact solution of the initial value problem

$$z'(t) = f(t, z(t)), \quad z(x) = y$$

with the initial values x and y. Then the function

$$\Delta(x,y,h;f) := \begin{cases} \frac{z(x+h)-y}{h} & h \neq 0 \\ f(x,y) & h = 0 \end{cases}$$

describes the differential quotient of the exact solution $z(t)$ with step size h, whereas $\Phi(x,y,h;f)$ is the differential quotient of the approximated solution with step size h.
The difference $\tau = \Delta - \Phi$ is the measure of quality of the approximation method and is denoted as local discretisation error.

In the following, $F_N(a,b)$ is defined as the set of all functions f, for which exist all partial derivations of order N on the area

$$S = \{x, y | a \leq x \leq b, y \in \mathbb{R}^n\}, \quad \text{a,b finite,}$$

where they are continuous and limited.

One step solvers have to fulfill

$$\lim_{h \to 0} \tau(x,y,h;f) = 0.$$

This is equivalent to

$$\lim_{h \to 0} \Phi(x,y,h;f) = f(x,y).$$

If this condition holds for all $x \in [a,b]$, $y \in F_1(a,b)$, then Φ and the corresponding one step method are called *consistent*.

The one step method is of *order* p, if

$$\tau(x,y,h;f) = O(h^p)$$

holds for all $x \in [a,b], y \in \mathbb{R}, f \in F_p(a,b)$.

The *global discretisation error*

$$e_n(X) := y(X) - y_n \quad X = x_n \text{ fix }, n \text{ variable}$$

is the difference between exact solution and the approximated solution.

The one step method is denoted as *convergent*, if:

$$\lim_{n\to\infty} \|e_n(X)\| = 0.$$

Theorem: Methods of order $p > 0$ are convergent and it holds

$$e_n(X) = O(h^p).$$

This means that the order of the global discretisation error is equal to the order of the local discretisation error.

The crucial problem concerning one step methods is the choice of the step size h. If the step size is too small, the computational effort of the method is unnecessary high, but if the step size is too large, the global discretisation error increases. For initial values x_0, y_0 a step size as large as possible would be chosen,so that the global discretisation error is below a boundary ϵ after each step. Therefore a step size control is necessary.

3.2 Explicit Euler

The most elementar method of solving initial value problems is the explicit Euler. The value of y_{i+1} can be calculated the following way:

$$y_{i+1} = y_i + h \cdot f(x_i, y_i) \tag{8}$$

The explicit Euler calculates the new value y_{i+1} by following the tangent at the old value y_i for a distance of h. The slope of the tagent is given by the value of $f(x_i, y_i)$. The explicit Euler uses

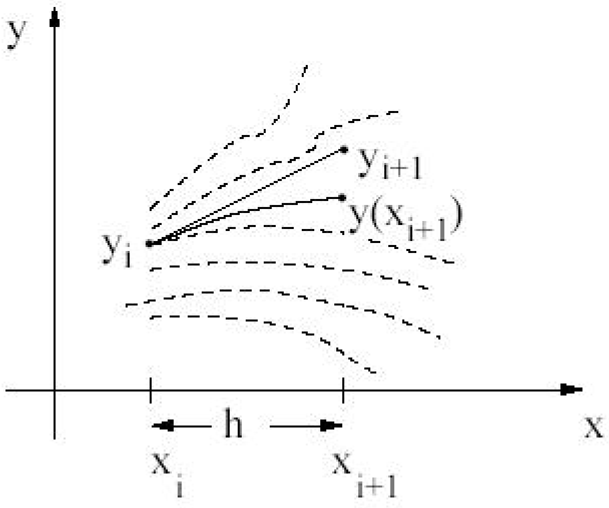


Figure 3: Explicit Euler

no step size control, the step size h is fix. So it is only useful in special cases, where the function to integrate is pretty flat. But it is very easy to implement and calculates very fast, so it can be a good choice.

3.3 Runge-Kutta methods

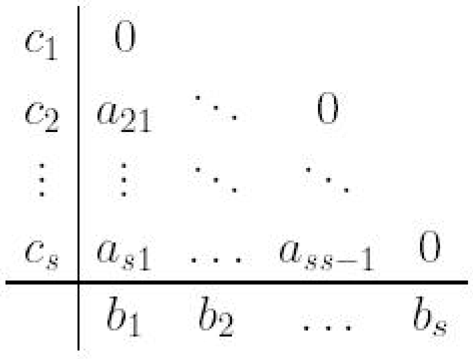
The Runge-Kutta methods are a special kind of one step solvers, which evaluate the right side in each step several times. The intermediate results are combined linearly.
The general discretisation schema for one step of a Runge-Kutta method is

$$y_1 = y_0 + h(b_1K_1 + b_2K_2 + \cdots + b_sK_s)$$

with corrections

$$K_i = f(x_0 + c_ih, y_0 + h\sum_{j=1}^{i-1} a_{ij}K_j), \quad i = 1, \cdots, s.$$

The coefficients are summarized in a tableau, the so called *Butcher-tableau*, see figure 4.



$$\begin{array}{c|ccccc} c_1 & 0 & & & & \\ c_2 & a_{21} & \ddots & 0 & & \\ \vdots & \vdots & \ddots & \ddots & & \\ c_s & a_{s1} & \ldots & a_{ss-1} & 0 \\ \hline & b_1 & b_2 & \ldots & b_s \end{array}$$

Figure 4: Butchertableau

3.4 Step size control

The Runge-Kutta methods use an equidistant grid, but this is for most applications inefficient. A better solution is to use an adaptive step size control. The grid has to be chosen so that

- a given accuracy of the numerical solution is reached
- the needed computational effort is minimized.

As the characteristics of the solution are a priori unknown, a good grid structure can not be chosen previous to the numerical integration. Instead, the grid points have to be adapted during the computation of the solution.
Trying to apply this to Runge-Kutta methods lead to the following technique:
To create a method of order p (for y_{i+1}), it is combined with a method of order p+1 (for $\hat{y}_{k+1}$). This method for y_{i+1} is called the *embedded method.* The idea of embedding was developed by Fehlberg and methods using this technique therefore are called *Runge-Kutta-Fehlberg* methods. This leads to a modified Butchertableau. (see figure 5)

The new step size is calculated with

$$h_{new} = h \sqrt[p+1]{\frac{\epsilon}{\|y - \hat{y}\|}}$$

where ϵ denotes the tolerance.

$$
\begin{array}{c|c}
c & A \\
\hline
 & b^T \\
\hline
 & \hat{b}^T
\end{array}
\qquad
\begin{array}{l}
\\
\longrightarrow \; y_1 = y_0 + h \sum_{i=1}^{s} b_i K_i \\
\longrightarrow \; \hat{y}_1 = y_0 + h \sum_{i=1}^{\hat{s}} \hat{b}_i K_i
\end{array}
$$

Figure 5: Modified Butchertableau for embedded Runge-Kutta-methods

3.5 Dormand-Prince method

The Dormand-Prince method is a member of the Runge-Kutta-Fehlberg class with order 4(5). It means that the method has order 5 and the embedded method has order 4. This is described by the following equations:

$$
\begin{aligned}
y(x_0 + h) &= y_0 + h \sum_{k=0}^{4} b_k f_k(x_0, y_0; h) \\
\hat{y}(x_0 + h) &= y_0 + h \sum_{k=0}^{5} \hat{b_k} f_k(x_0, y_0; h) \\
f_k &= f(x_0 + c_k h, y_0 + h \sum_{t=0}^{k-1} a_{kl} f_l)
\end{aligned}
$$

In Matlab this ODE solver is implemented in the function ode45. The coefficients from Dormand and Prince can be seen in figure 6.

4 Controller design

4.1 Basics of controller design

There are several basic requirements to a controller:

- Stability
- Stationary accuracy
- Promptness

The basic principle for the control of a system is a complete description of the system by equations. For these inverted pendulum problem the equations are stated in section 2.
A linear dynamic system is described by

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & \frac{-56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & \frac{-25360}{2187} & \frac{64448}{6561} & \frac{-212}{729} & & & \\
1 & \frac{9017}{3168} & \frac{-355}{33} & \frac{46732}{5247} & \frac{49}{176} & \frac{-5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & \frac{-2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & \frac{-2187}{6784} & \frac{11}{84} & 0 \\
\hline
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & \frac{-92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

Figure 6: Butchertableau for Dormand-Prince-method

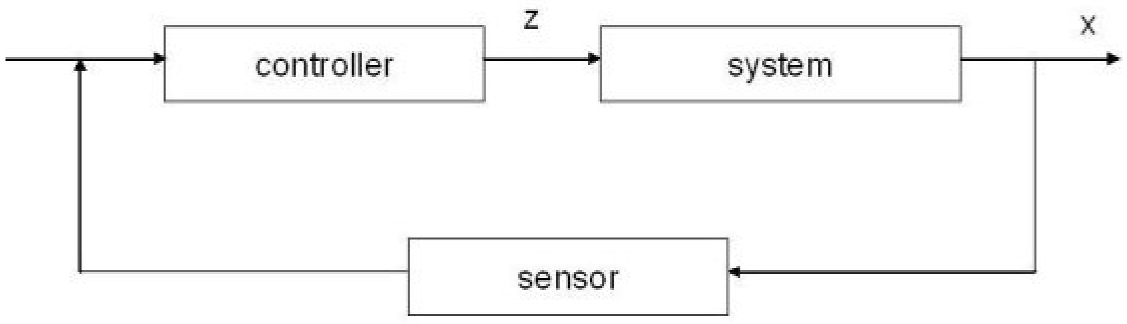


Figure 7: Schema of the functionality of the controller

$$
\begin{aligned}
\dot{x}(t) &= Ax(t) + Bu(t) \\
y(t) &= Cx(t) + Du(t)
\end{aligned}
\tag{9}
$$

with $A, B, C, D \in \mathbb{R}^{n,n}$ constant. Assuming that there is no disturbance, the vector $u(t)$ represents the control variables. The vector $y(t)$ denotes the measurement values. For the inverted pendulum problem, this means, that C is the unity matrix I and D is zero. The initial state $x_0 = x(t_0)$ is in general unknown. There are several characteristics of a controller:

Controllability: The system (9) is called *controllable*, if the state space vector x can be moved to the finite state 0 within a finite time frame and an arbitrary initial state x_0 by the correct choice of the variable control vector u. The finite state 0 is not a limitation, as the coordinate system can be translated to the appropriate values.

Observability: The system (9) is called *observable*, if the initial state x_0 can be uniquely calculated with a known $u(t)$ and the measurement of $y(t)$.

Stability: The system (9) is called *stable* if the solution $x(t)$ of the homogeneous state space differential equation $\dot{x} = Ax$ tends to 0 for $t \to \infty$. This holds for any initial state x_0.

A linear system hast to be controllable and observable, so that it can be controlled. The stability of the control path is necessary, as otherwise one of the basic requirements is not fulfilled.
There are two different possibilities for the design of a controller. It can be designed in state space or in frequency domain. In the following we will concentrate on the development in state space where the linear system (9) can be handled with the KALMAN-criteria:

- The system (9) with $u \in \mathbb{R}^p$ is controllable if and only if the $n \times np$ controllability matrix

  $$Q_S = (B, AB, A^2B, \cdots, A^{n-1}B)$$

  has maximum rank n.

- The system (9) with $y \in \mathbb{R}^q$ is observable if and only if the $nq \times n$ observability matrix Q_B

$$\begin{pmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{pmatrix}$$

  has maximum rank n.

For the caracterisation of the stability, there exists the following theorem:

Theorem: The system (9) is stable if and only if all eigenvalues of A have a negative real part.

4.2 Controlling with pole location presetting

In the following a controller is designed by pole location presetting. This is done by presetting the eigenvalues $\lambda_1, \cdots, \lambda_n$ of the state space control to ensure that the controller r has these eigenvalues. This leads to:

$$\begin{aligned} det(sI - (A - br^T)) &= \prod_{\nu=1}^{n}(s - \lambda_\nu) = s^n + p_{n-1}s^{n-1} + \cdots + p_1 s + p_0 \\ \Longrightarrow s^n + a_{n-1}(r)s^{n-1} + \cdots + a_0(r) &= s^n + p_{n-1}s^{n-1} + \cdots + p_1 s + p_0 \end{aligned}$$

r can be calculated by a comparison of the coefficients. But this approach has the disadvantage, that the effort of the evaluation of the determinante is too high and therefore the formula of Ackerman is used.

Theorem of Ackermann: If the control path $\dot{x} = Ax + bu$ is controllable and the state space control has the characteristic polynomial $p(s) = s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0$, then the control vector is chosen as

$$r^T = p_0 t_1^T + p_1 t_1^T A + \cdots + p_{n-1} t_1^T A^{n-1} + t_1^T A^n = t_1^T[p_0 I + p_1 A + \cdots + p_{n-1}A^{n-1} + A^n] = t_1^T p(A).$$

with t_1^T as the last row of the inverse controllability matrix Q_S^{-1} and it is calculated by the system of equations

$$\begin{aligned} t_1^T b &= 0 \\ t_1^T Ab &= 0 \\ &\vdots \\ t_1^T A^{n-2} b &= 0 \\ t_1^T A^{n-1} b &= 1 \end{aligned}$$

4.3 Controller design for the inverted pendulum

In this section, the theory of control design of the last section shall be applied to the inverted pendulum problem. There are many articles concerning the problem of broomstick balancing, e.g. K. Furuta, [4] who approached the problem from a technical point of view and P. J. Larcombe, [5] who focused on the mathematical point of view.
The inherent dificulty of controlling the inverted pendulum is, that is not a linear system. Unfortunately, there are only linear controllers, which cannot control non-linear systems. But as the inverted pendulum behaves nearly linear, while it is balanced, it is possible to linearise the equations of the pendulum for small angles. So the linear controlling theory can be applied.

4.3.1 Inverse one-bar pendulum

To develop a controller, the equation (5) has to solved for $\ddot{x}$ and $\ddot{\varphi}_1$ and linearised by setting $cos(\varphi_1) = 1$, $sin(\varphi_1) = \varphi_1$ and $\dot{\varphi_1}^2 = 0$ for small φ_1:

$$\begin{aligned}\ddot{x} &= -\frac{3m_1 g}{4m_1 + 7m} \cdot \varphi_1 + \frac{7}{4m_1 + 7m} \cdot F \\ \ddot{\varphi}_1 &= -\frac{3(m + m_1)g}{l(4m_1 + 7m)} \cdot \varphi_1 - \frac{3}{l(4m_1 + 7m)} \cdot F\end{aligned}$$

By setting the state variables $x_1 := x$, $x_2 := \dot{x}$, $x_3 := \varphi_1$, $x_4 := \dot{\varphi}_1$ ($x^T = (x_1, \cdots, x_4)$) and the input value $u := F$ the control system is:

$$\dot{x} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{3m_1 g}{4m_1+7m} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{3(m+m_1)g}{l(4m_1+7m)} & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ \frac{7}{4m_1+7m} \\ 0 \\ -\frac{3}{l(4m_1+7m)} \end{pmatrix} u =: Ax + bu$$

The control path is controllable as the determinate of the controllability matrix

$$Q_S = (B, AB, A^2B, \cdots, A^{n-1}B)$$

is not zero. Moreover it is observable as the observability matrix Q_B

$$\begin{pmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{pmatrix}$$

has maximum rank 4, see [6]. Therefore the first two criteria are fulfilled and to fulfill also the third criteria, the stability, all the eigenvalues of A must have a negative real part. This is not true as the eigenvalues are:

$$s_{1/2} = 0, \quad s_{3/4} = \pm\sqrt{\frac{3(m + m_1)g}{l(4m_1 + 7m)}}$$

Applying all these steps to the inverted pendulum problem we get the following results:

$$t_1^T = [\frac{l(4m_1 + 7m)}{3g}, 0, \frac{7l^2(4m_1 + 7m)}{9g}, 0]$$

leading to

$$\begin{aligned}
r_1 &= -p_0 \cdot \frac{l(4m_1+7m)}{3g} \\
r_2 &= -p_1 \cdot \frac{l(4m_1+7m)}{3g} \\
r_3 &= -p_0 \cdot \frac{7l^2(4m_1+7m)}{9g} - p_2 \cdot \frac{l(4m_1+7m)}{3g} - (m+m_1)g \\
r_4 &= -p_1 \cdot \frac{7l^2(4m_1+7m)}{9g} - p_3 \cdot \frac{l(4m_1+7m)}{3g}
\end{aligned}$$

In the next step the variables $p_0, \cdots, p_3$ have to be calculated. This works with the Theorem of Ackermann. Denoting the eigenvalues as $\lambda_1, \cdots, \lambda_4$, this leads to

$$p(s) = (s-\lambda_1)(s-\lambda_2)(s-\lambda_3)(s-\lambda_4) = s^4 + p_3 s^3 + p_2 s^2 + p_1 s + p_0 \qquad (10)$$

By using the quad eigenvalue $\lambda = -1$ the coefficients p_i are

$$p_0 = 1,\ p_1 = 4,\ p_2 = 6,\ p_3 = 4.$$

Eventually the force F which controls the system is made up of

$$F = -r_1 x - r_2 \dot{x} - r_3 \varphi - r_4 \dot{\varphi}.$$

4.3.2 Inverted double pendulum

In this section a controller for the inverted double pendulum shall be developed. An approach like in the one-bar case is chosen. First of all the state space form of the double pendulum problem is linearised to reduce the complexity of the system. This is achieved by setting $cos(\varphi_i) = 1$, $sin(\varphi_i) = \varphi_i$ and $\dot{\varphi_i^2} = \dot{\varphi_2^2} = 0$ for small φ_i with i = 1,2. Moreover it is assumed that $m_1 = m_2$ and $l_1 = l_2$. Concerning this and solving for $\ddot{x}$, $\ddot{\varphi_1}$ and $\ddot{\varphi_2}$ leads to

$$\begin{pmatrix} \ddot{x} \\ \ddot{\varphi_1} \\ \ddot{\varphi_2} \end{pmatrix} = \frac{3}{l_1(56m_1+97m} \begin{pmatrix} l_1 \cdot (\frac{97}{3}F - 45m_1\varphi_1 g - m_1\varphi_2 g) \\ 3\cdot(-5F + (7m+11m_1)\varphi_1 g - (2m+m_1)\varphi_2 g) \\ -F - 9(2m+m_1)\varphi_1 g + (19m+11m_1)\varphi_2 g \end{pmatrix}$$

Similar to the one-bar pendulum the state variables are set to $x_1 := x$, $x_2 := \dot{x}$, $x_3 := \varphi_1$, $x_4 := \dot{\varphi_1}$, $x_5 := \varphi_2$, $x_6 := \dot{\varphi_2}$ ($x^T = (x_1, \cdots, x_6)$) and the input value $u := F$. This leads to the control system:

$$\dot{x} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{135m_1 g}{56m_1+97m} & 0 & -\frac{135m_1 g}{56m_1+97m} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{9(7m+11m_1)g}{l_1(56m_1+97m)} & 0 & -\frac{9(2m+m_1)g}{l_1(56m_1+97m)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{27(2m+m_1)g}{l_1(56m_1+97m)} & 0 & -\frac{3(19m+11m_1)g}{l_1(56m_1+97m)} & 0 \end{pmatrix} x + \begin{pmatrix} 0 \\ \frac{97}{56m_1+97m} \\ 0 \\ -\frac{45}{l_1(56m_1+97m)} \\ 0 \\ -\frac{3}{l_1(56m_1+97m)} \end{pmatrix} u =: Ax + bu$$

The control path is also controllable and observable, as the determinants of the controllability matrix Q_S and the observability matrix Q_B are not zero. But like in the one-bar case it is not stable.
The Theorem of Ackermann is applied to the system and we receive the vector t_1, with which the controller can be developed. A sixfold eigenvalue $\lambda = -3$ is used and the coefficients p_i are:

$$p_0 = 729,\ p_1 = 1458,\ p_2 = 1215,\ p_3 = 540,\ p_4 = 135,\ p_5 = 18.$$

The force F is set to

$$F = -r_1 x - r_2 \dot{x} - r_3 \varphi_1 - r_4 \dot{\varphi}_1 - r_5 \varphi_2 - r_6 \dot{\varphi}_2.$$

5 Neural network

In this section an approach to control the inverted pendulum by using neural networks is described. Neural networks are modelled similar to the human brain. They are often used to solve hard problems, where it is difficult to write exact algorithms, or where the exact algorithms are to slow.

5.1 Basics of neural networks

The aim of neural networks is to *simulate* a function. But before the function can be simulated, the network has to *learn* its behaviour. For this purpose exist several *learning algorithms*. What kind of functions the network is able to simulate, depends on its topology.

A neural network consists of a set of so called *neurons* and connections inbetween, the so called *synapses*. The neural network can also be seen as a graph $G = (V, E)$ where the nodes of the graph V are the neurons and the edges of the graph G are the synapses. Each synapse $(i, j) \in E$ has a certain weight $w_{ji} \in \mathbb{R}$.

5.2 Feed-forward networks

A special kind of neural networks are the *feed-forward networks* (figure 8). The neurons of this networks are allocated of several disjunct layers, where the first layer is the *input layer* and the last one is the *output layer*. Layers between the first and last layer are the so called *hidden layers*. There are no connections between neurons of the same layer. The information flows just in one direction from the input to the output layer.
How many layers are used in a neural network depends on the problem. By changing the number of layers in a network, the separation possibility of it changes. With a one-layered network linear separation can be achieved. With a two layered network convex areas can be separated.
The activation of a neuron o_k in layer r can be computed as

$$net_k = \sum_{\text{l} \in \text{ layer } < \text{ r}} w_{kl} o_l, \quad o_k = f(net_k)$$

with an activation function f. The activation function should be non-linear. One frequently used function is the logistic function

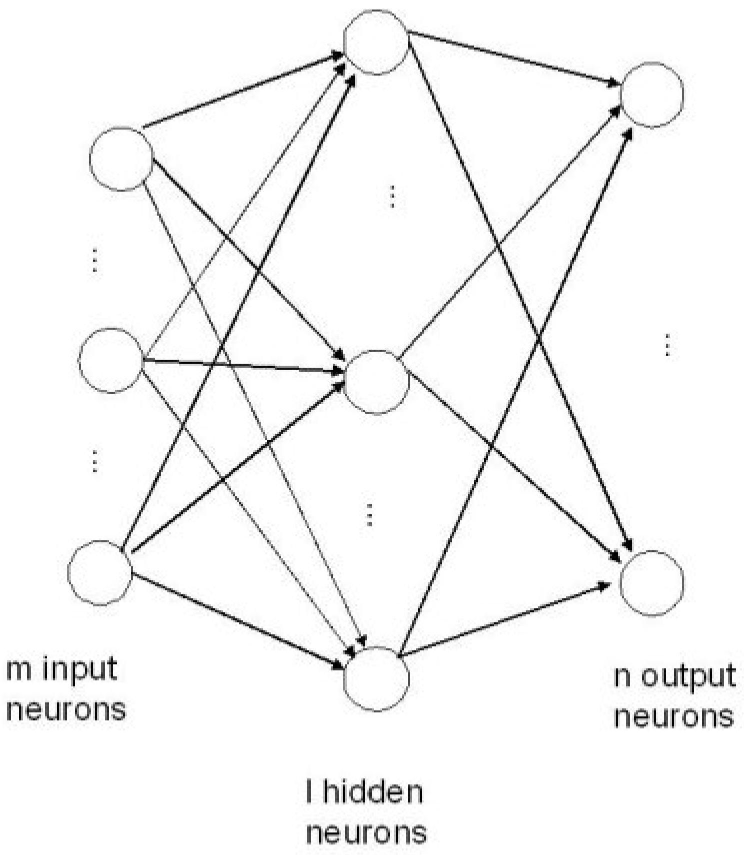


Figure 8: Two-layered feed-forward network

$$f(x) = \frac{1}{1+e^{-x}}$$

with the derivation

$$f'(x) = f(x)(1 - f(x)).$$

5.3 Learning through backpropagation

The most frequently used learning algorithm for both supervised and reinforcement learning is the backpropagation algorithm.
In the first part a simple forward propagation is used to evaluate the network and to determine the output error

$$E = \frac{1}{2}\sum_{\text{i in output layer}}(o_i - d_i)^2$$

with the desired output d_i of the i-th output node. In the next part a technique called gradient descent is used to change the weights

$$w_{ij} := w_{ij} + \Delta w_{ij} = w_{ij} - \alpha\frac{\partial E}{\partial w_{ij}}$$

with the learn rate α. The gradient is

$$-\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}} = \delta_i o_j$$

with

$$\delta_i = -\frac{\partial E}{\partial net_i} = -\frac{\partial E}{\partial o_i} f'(net_i).$$

If i is an output node then

$$\frac{\partial E}{\partial o_i} = (o_i - d_i).$$

Otherwise if i is not an output node and in the r-th layer then by using the chain rule

$$\frac{\partial E}{\partial o_i} = \sum_{\text{k} \in \text{layer} > \text{r}} \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial o_i} = \sum_{\text{k} \in \text{layer} > \text{r}} \delta_k w_{ki}.$$

Now an error propagation algorithm is used which is symmetric to the forward propagation algorithm:

$$net_k = \sum_{\text{l} \in \text{layer} < \text{r}} w_{kl} f(net_l).$$

When using the error propagation algorithm, first of all the error $(d_i - o_i)f'(net_i)$ must be assigned to the output nodes. Then the error is backpropagated through the network

$$\delta_i = f'(net_i) \sum_{\text{k} \in \text{layer} > \text{r}} w_{ki}\delta_k.$$

This means, that the error of one node is calculated by summing up the errors of all successive nodes, multiplying each error with the corresponding weight and multiplying this sum with the activation of the node applied to the derivation of f. Eventually all the weights of the network are changed

$$w_{ij} := w_{ij} + \alpha\delta_i o_j.$$

Like stated above a gradient descent method is used within the backpropagation algorithm. The problem of this method is, that it can get stuck in local minimum and therefore, the global minimum can not be found. It depends on the initial choice of the weights, which minimum is found. This means that with a good choice of the initial weights, within some training steps, a good minimum can be found, whereas otherwise, if there is a bad initial set of weights, even after a lot of time spent in training of the network, it will not work properly.

5.4 Reinforcement networks

A reinforcement network learns without knowing exact results given by a teacher. It only knows, when it has failed, and it tries to minimize these failures. The failure signals are backprogated through the network, and so the network's weights are changed.
The problem of this method is, that a failure can encouter after many succesfull steps, so that it is unkown, which step has triggered this failure. This problem can be solved by using two functions: an *action function*, which maps the current state into control actions, and an *evaluation function*, which evaluates the current state. The evaluation function is used to assign credit of the current action.
Obviously the evaluation function depends only on the current state. Therefore a reinforcement network learning the evaluation function should not have the problems like with the action function. So it is possible, to build two different networks (see figure 9), which learn the evaluation and the action function, instead of build just one network for the action function and to implement an evaluation function, which has been chosen before.

5.4.1 Controlling the inverted pendulum with a reinforcement network

To control the pendulum, the cart is pushed either left or right with constant force F after time Δt. The aim is, that the network is able to make the right choice, in which direction it should push the cart.

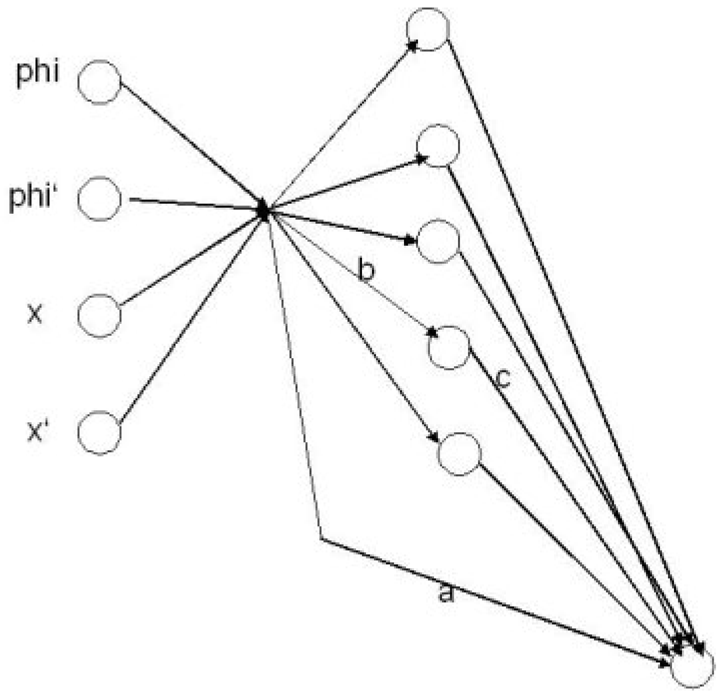


Figure 9: Topology of action and evaluation network

One possible solution (see also [2]) can be seen in figure 10. Both networks have four input, five hidden and one output node and get $\varphi, \dot{\varphi}, x$ and $\dot{x}$ as input. For learning, the evaluation network gets also the failure signal of the inverted pendulum as further input and it transmits his output to the action network. The failure signal always occurs, if $|x| > 3.4$ or the angle of the bar is getting greater than 12 degree. The output of the action network is the probability of pushing the cart to one direction, and it is directly transmitted to the pendulum.
To train this network, the weights are set initially to random values. Then x and φ are randomly set to values in the valid range. The case for this is that so the effect of *overfitting*, that means, that the network can only control pendulums with angles it has learnt, can be avoided. After every failure signal, the cart is reseted to a random but valid position, with a random angle of the pendulum.

5.4.2 Implementation of the reinforcement network

To implement the reinforcement network, the following problems have to be solved:

- Which method should be used to integrate the ODE?
- Which learning parameters should be used, that the network can learn fast, but also precisely enough?

As we have an equidistant timegrid, it is not possible to use the Dormand-Prince-method or the other in Matlab implemented integrators. So it is necessary to use a much simpler method.
The most simple integration-method is the explicit Euler. For a fine grid and functions with small derivations, it integrates the function fast and precisly. But it does not work well with a rough grid or functions with large derivations, as it becomes very unprecisly.
But dealing with the inverted pendulum, this problems can be avoided. The grid can be chosen small enough, and as the calculation is breaked, if there are to large values, our function, which is integrated, is pretty flat. So it is a good choice, as it works very well for this case and it is also very easy to implement.
The right choice of the learning parameters is much harder. We have succesfully used the following ones (see [1]):

$$\begin{aligned} \gamma &= 0.9 \\ \beta_e &= 0.2 \\ \beta_a &= 0.2 \\ \rho_e &= 1 \\ \rho_a &= 0.05 \end{aligned}$$

Be t the current time. Define the input vector v as follow:

$$v = \begin{pmatrix} \frac{x+2,4}{4,8} \\ \frac{\dot{x}+1,5}{3} \\ \frac{\varphi+0,2094}{0,4186} \\ \frac{\dot{\varphi}+2,01}{4,02} \\ 0,5 \end{pmatrix}$$

Define the following activation functions:

- y_e^t for the hidden layer in the evaluation network e at time t
- z_e^t for the output node in the evaluation network e at time t
- y_a^t for the hidden layer in the action network at a time t
- z_a^t for the output node in the action network at a time t

If t is not superscripted explicitly, the current time is meant.
Define the weights for the networks as:

- a_e for the weights of the input to the output layer in the evaluation network e
- b_e for the weights of the input to the hidden layer in the evaluation network e
- c_e for the weights of the hidden to the output layer in the evaluation network e
- a_a for the weights of the input to the output layer in the action network a
- b_a for the weights of the input to the hidden layer in the action network a
- c_a for the weights of the hidden to the output layer in the action network a

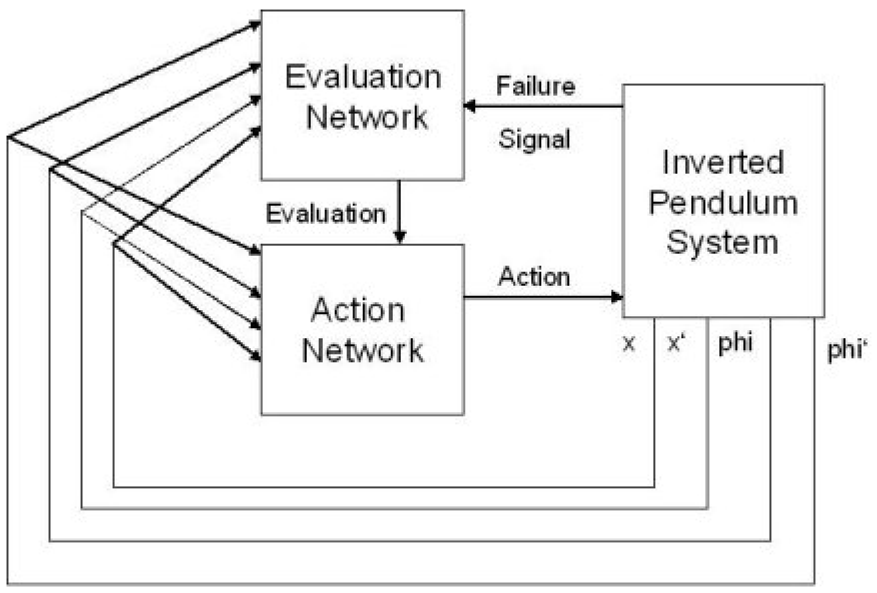


Figure 10: The complete system with action and evaluation network

Define $\hat{r}$:

$$\hat{r} := \begin{cases} -1 - z_e^{t-1} & \text{if failure occured} \\ \gamma z_e^t - z_e^{t-1} & \text{else} \end{cases}$$

The influence of the failure signal on the pendulum depends of the choice of $\hat{r}$. Define p as following:

$$p := \begin{cases} 1 - z_a & \text{if cart was pushed to the right} \\ -z_a & \text{else} \end{cases}$$

Be f the logistical function

$$f(x) = \frac{1}{1-e^{-x}}$$

Be $1 \leq i \leq 5$, then the weights are changed on the following way:

$$\begin{aligned} a_e(i) &= a_e(i) + \beta_e \cdot \hat{r} \cdot f'((y_e)(i)) \cdot sgn(c_e(i)) \\ a_a(i) &= a_a(i) + \beta_a \cdot \hat{r} \cdot f'((y_a)(i)) \cdot sgn(c_a(i)) \cdot p \\ b_e(i) &= b_e(i) + \rho_e \cdot \hat{r} \cdot v(i) \\ b_a(i) &= b_a(i) + \rho_a \cdot \hat{r} \cdot p \cdot v(i) \\ c_e(i) &= c_e(i) + \rho_e \cdot \hat{r} \cdot y_e(i) \\ c_a(i) &= c_a(i) + \rho_a \cdot \hat{r} \cdot p \cdot y_a(i) \end{aligned}$$

6 Simulation results

6.1 One-bar Pendulum

6.1.1 Simulation with the PD-Controller

The inverted pendulum system was implemented in Matlab. For controlling the pendulum the controller, designed in section 5, was used. The behaviour of the controller changes when using different eigenvalues λ in equation (10). The maximum interval of controllable values was from -59.01 degree to 59.01 degree. This was obtained with $\lambda = -1.1$. The plots for $\varphi = 59.01$ can be seen in figure 11. Further increasing of φ leads to a chaotic system (see figure 12).

The change of λ influences not only the range of controllable angles but also the speed, in

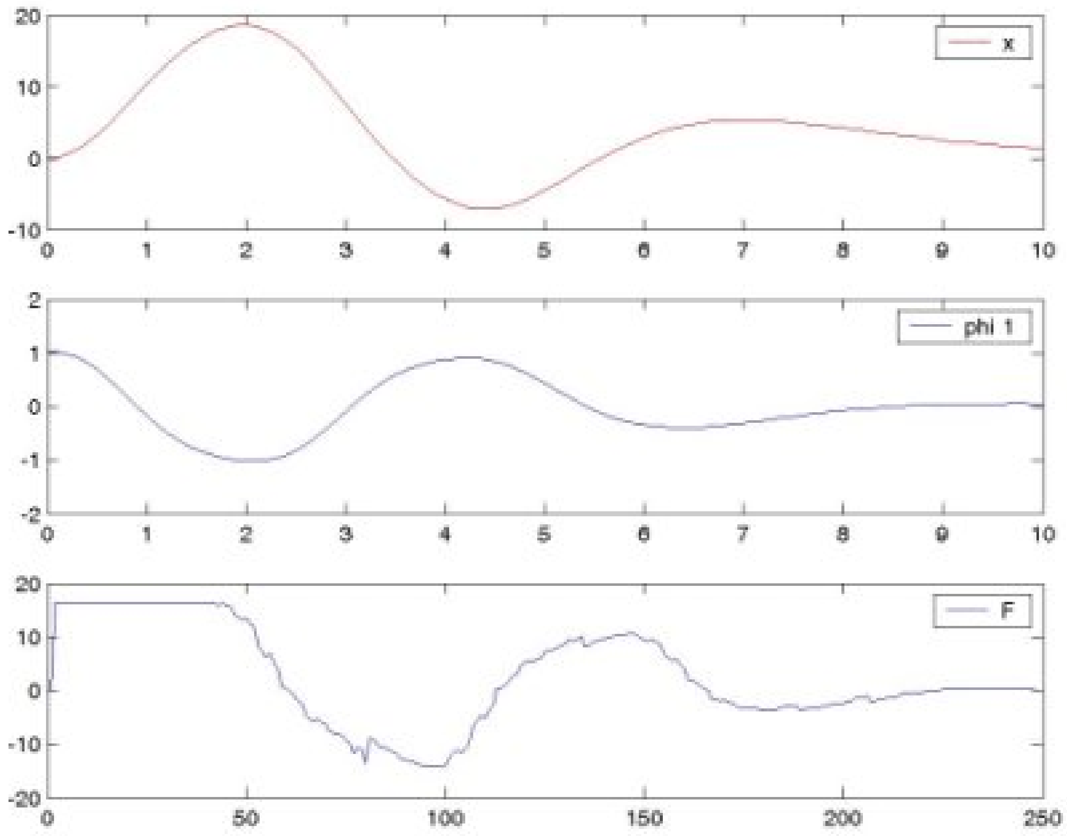


Figure 11: Plot of x, phi and F with initial deflection of 59° (PD-controller)

which the pendulum is controlled. In figure 13 to 15 there are plots with an initial deflection of 23 degree using $\lambda = -0.5$, $\lambda = -1$ and $\lambda = -4$. The images show, that lower values of λ lead to a faster control of the pendulum. A screenshot of the animation is shown in figure 16.

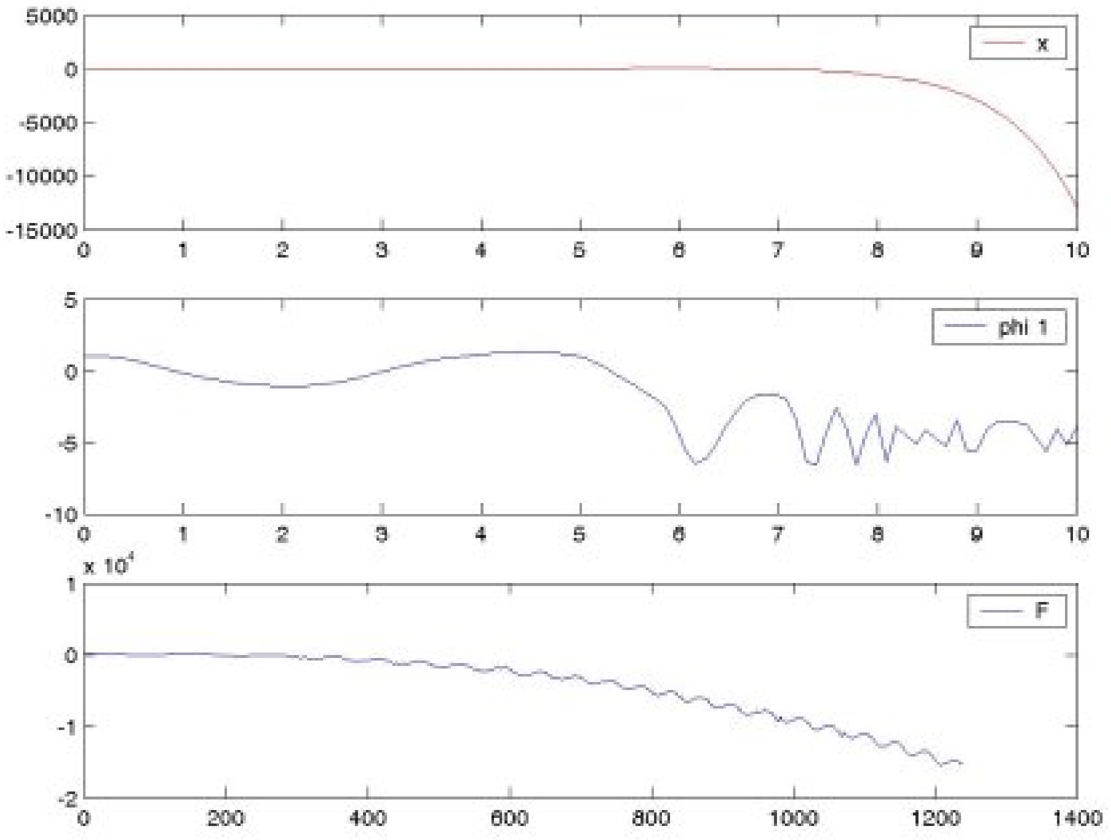


Figure 12: Plot of x, phi and F with initial deflection greater than 59° (PD-controller)

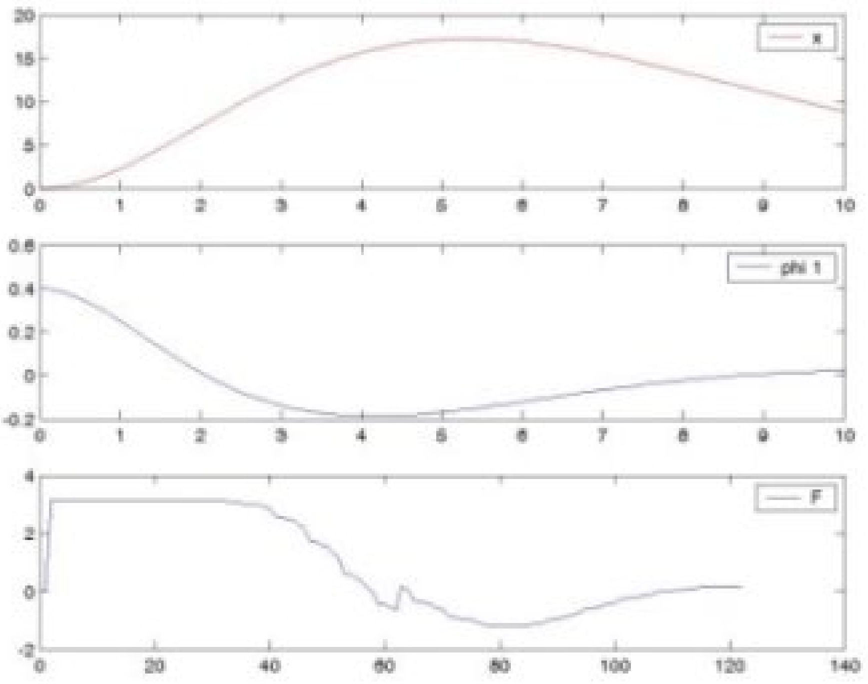


Figure 13: Plot with $\lambda = -0,5$ (PD-controller)

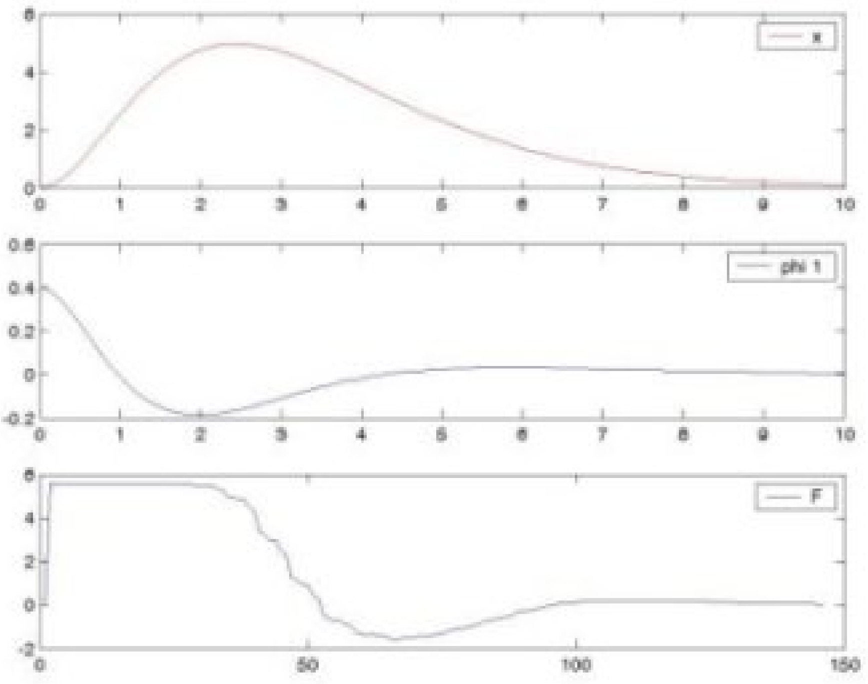


Figure 14: Plot with $\lambda = -1$ (PD-controller)

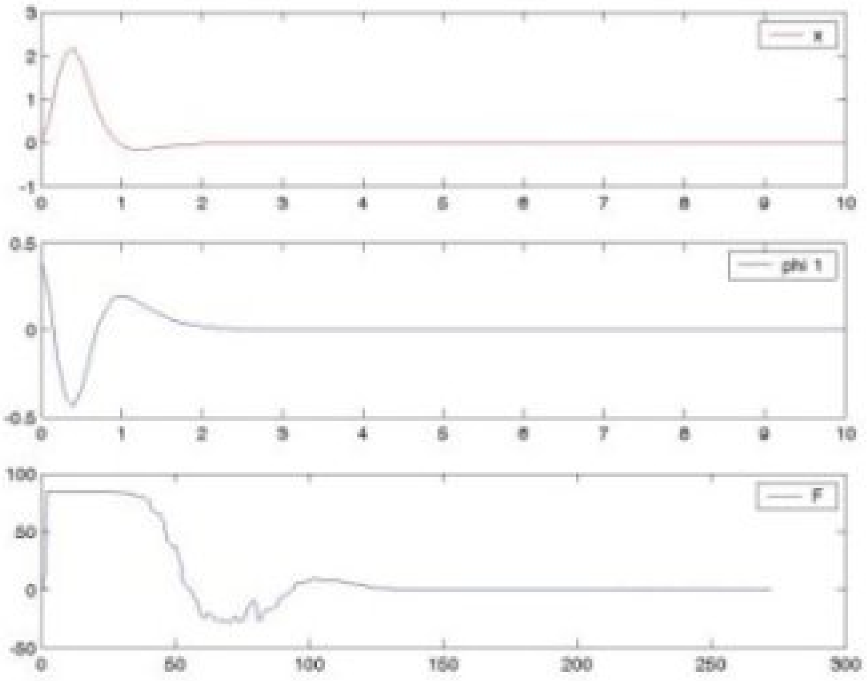


Figure 15: Plot with $\lambda = -4$ (PD-controller)

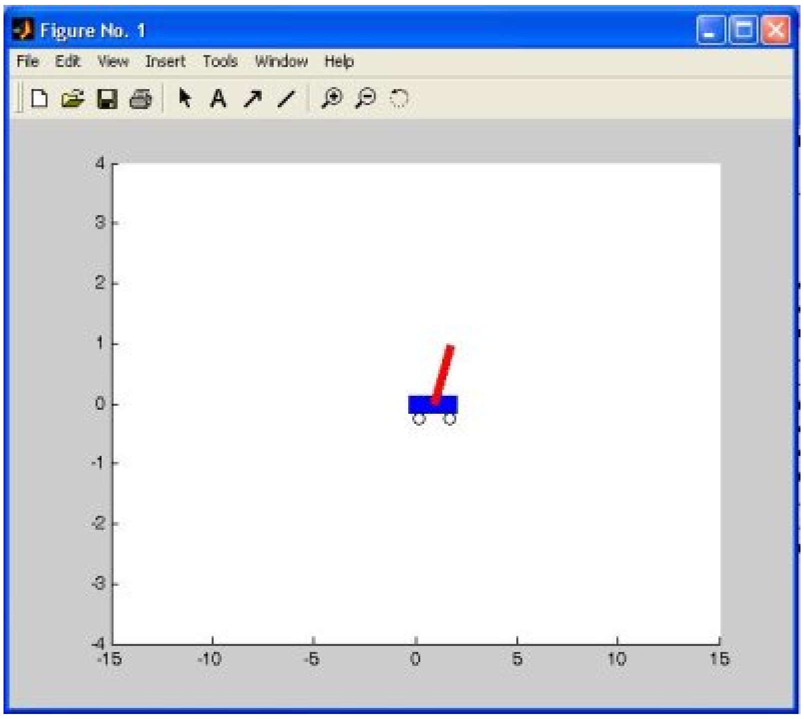


Figure 16: Animation of the pendulum)

6.1.2 Simulation with the neural network

For the simulation of the inverted pendulum we trained a neural network using reinforcement learning like described in section 6.3. The network had about 4,000 trials to learn. Afterwards it was able to control a pendulum with an initial angle of about 30 degrees, also depending on the initial position of the cart. It is not possible to exactly quantify the performance of the neural network as it is non-deterministic. The reason for the non-determinism is that the decision about the applied force depends also on random values. The applied force is either +10 N or -10 N. This restricts the possible interval of initial angles as for higher deflections, especially in the beginning of the control cycle, stronger forces are needed.
A plot of the cart position, cart velocity, bar angle and bar velocity for an initial angle of 30 degrees can be seen in figure 17.

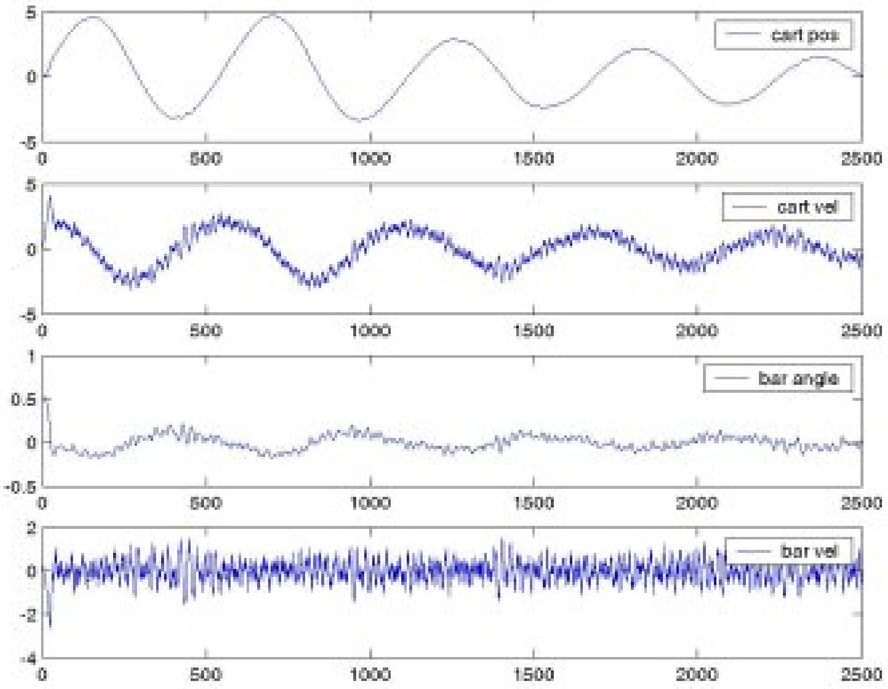


Figure 17: Plot of neural network results with initial angle of 30 °

Different trials have shown that starting with random values and training them, a quite similar control behaviour shows up. In figure 18 are examples for the simulation with an initial deflection of 5.8 degree with two different neural networks each after 1,000 learning steps.
After training several neural networks it has shown up that about 4,000 learning steps are enough so that the neural network can control the inverted pendulum.
The following weights have shown good results

$$a = \begin{pmatrix} -0.72193 & -0.38412 & -1.7838 & -1.0804 & -0.44322 \\ -0.92651 & -0.80744 & -1.2158 & -0.92695 & -0.81483 \\ -0.77189 & -0.84096 & -1.2998 & -1.0309 & -0.78035 \\ -0.83051 & -0.53704 & -1.4489 & -0.91007 & -0.55633 \\ -0.74199 & -0.63616 & -1.4718 & -0.88074 & -0.69155 \end{pmatrix}$$

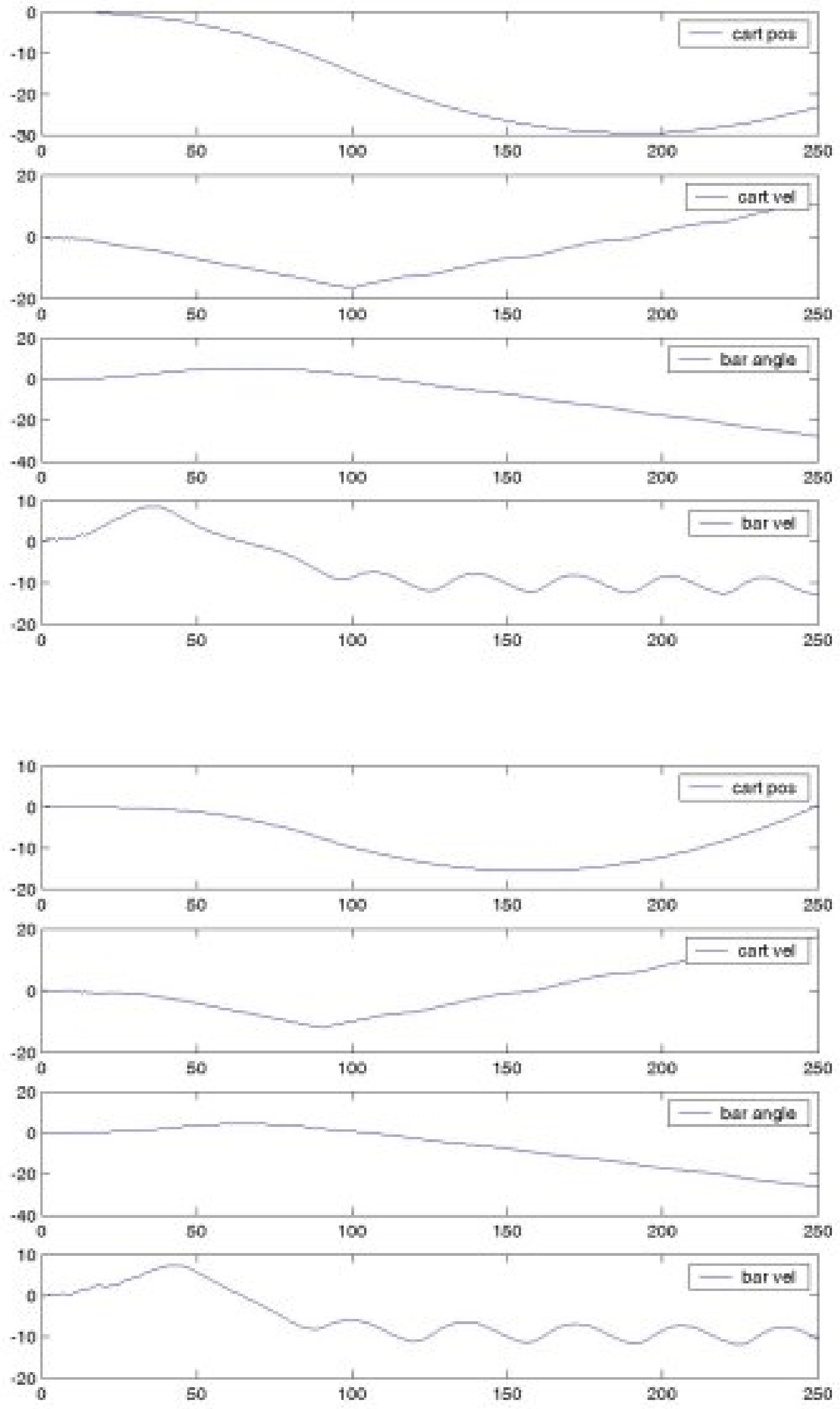


Figure 18: Plots of neural network results after 1000 learning steps

$$b = \begin{pmatrix} -0.46577 \\ -0.27866 \\ -1.0052 \\ -0.61087 \\ 3.0188 \end{pmatrix} c = \begin{pmatrix} -2.4621 \\ -0.67372 \\ -0.88071 \\ -0.84916 \\ -1.0529 \end{pmatrix}$$

$$d = \begin{pmatrix} -0.87518 & -1.0326 & -0.35068 & 0.12587 & -0.14769 \\ -0.76652 & -1.0859 & -0.21711 & 0.05435 & -0.25859 \\ -0.87144 & -1.1044 & -0.34958 & 0.021331 & -0.37435 \\ -0.9112 & -1.0643 & -0.35439 & 0.076197 & -0.29504 \\ -0.88561 & -1.1752 & -0.22411 & -0.011083 & -0.33166 \end{pmatrix}$$

$$e = \begin{pmatrix} 0.77669 \\ -4.9028 \\ 13.367 \\ 8.8845 \\ -4.4849 \end{pmatrix} f = \begin{pmatrix} -6.223 \\ -5.8976 \\ -5.623 \\ -5.8571 \\ -5.6119 \end{pmatrix}$$

We have also tried a feed-forward network with supervised learning, but it has not converged into a good local minimum, although we had done many tries. So the stabilisation of the network was not possible with this network.

6.2 Double pendulum

We have tried to develop a PD-Controller according to the results of section 5.3.2. We have applied many different eigenvalues, also -1 and -3, which were successfully used in [6]. Unfortunately the PD-Controller was not able to stabilise the double pendulum. Moreover in the case of the double pendulum, the control is such complex, that it is not possible to develop a control strategy intuitively.
Also the approach with the neural network was not successful. With the reinforcement network, that showed good results in the one-bar case, is very specialised and cannot be simply transformed to work for the two-bar case. We could not try the feed-forward network as our PD-controller did not work well and so we had no teacher for the network.

7 Conclusions

In the paper a model of the n-bar inverted pendulum problem was shown. According to this model, the corresponding ODE were listed. Based on this description in state space form, the controller equations for the one-bar and the two-bar pendulum were deduced. Moreover, an aproach using neural networks was applied. Two different types of neural networks were used, a feed-forward-network and a reinforcement network.
We implemented the model of the inverted n-bar pendulum and the deduced controllers in Matlab. To watch the simulation, we wrote a visualisation of the cart with the pendulum. We also added a graphical user interface, for an easy usage of the program.
The code was built modular to be extendable, so it is possible to integrate other controllers very easily. We also put emphasis on the detailed documentation of the code, so that the reuse of the program is enabled.

We have not examined the following aspects yet:

- An extension of the inverted pendulum is to allow movements not only in one dimension, but instead to move on a two-dimensional plane. Research concerning this topic is done by the chair of Professor Schmidhuber by Georg Fette.
- There are many other approaches in the field of machine learning, which could be used instead of a feed-forward or a reinforcement network, like genetic algorithms or recurrent networks.
- Further controllers can be developped, so that inverted pendulums with more than two bars can be stabilised. But as the equations become much more complex for higher dimensions, it will be more difficult to develop a controller for these cases.
- Another approach is the usage of a fuzzy controller, which was already successfully applied in this area. The design of a fuzzy controller is also supported of Matlab with the fuzzy toolbox.

References

[1] Anderson, C. W. Code for neuronal networks and reinforcement learning. *http://www.cs.colostate.edu/˜anderson/code/*.

[2] Anderson, C. W. Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine* (1989).

[3] B. Karasözen, P. Rentrop, Y. W. Inverted n-bar model in descriptor and in state space form. *Mathematical Modelling of Systems, Vol. 4, No. 4, pp. 272 - 285* (1995).

[4] K. Furuta, T. Okutani, H. S. Computer control of a double inverted pendulum. *Computers and Electrical Engineering, 5, pp. 67 - 84* (1978).

[5] P. J. Larcombe, R. Z. The controllability of a double inverted pendulum by symbolic algebra analysis. *Glasgow University, Proceedings of IEEE - Systems, Man and Cybernetics Conference* (1993).

[6] Wagner, Y. Modellierung differential-algebraischer gleichungen und neuronale netze zur lösung des stabbalance-problems. *TU München, Mathematisches Institut* (1994).