Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 2351

Depth control of an Underwater Robot

Matthias van de Weyer

Course of Study:	Technische Kybernetik
Examiner:	Prof. Dr. rer. nat. habil. Paul Levi
Supervisor:	Dipl.-Ing. Tobias Dipper
Commenced:	25.07.2011
Completed:	24.01.2011
CR-Classification:	I.2.9, I.6.0, B.1.0

Zusammenfassung

Für autonome Unterwasserfahrzeuge ist es unerlässlich selbstständig eine vorgegebene Tiefe anfahren und halten zu können. Dieser Vorgang ist im allgemeinen nicht einfach, da eine Tiefenänderung ein nichtlinearer Vorgang ist. Außerdem müssen Störgrößen, wie beispielsweiße durch Wasserströmungen, kompensiert werden.
Diese Arbeit befasst sich mit der Entwicklung eines Tiefenreglers für eine sehr kleine Unterwasserplattform, welcher genau diese Aufgaben übernimmt. Dazu wird zuerst das Modell der Plattform aus den Bewegungsgleichungen hergeleitet und in Matlab Simulink modelliert. Mit diesem Modell können beliebige Regelstrukturen und Parameter in kurzer Zeit Simuliert werden. Da die Plattform sehr klein ist, werden einfache Reglerstrukturen wie PID oder Zustandsrückführung verwendet. Um die Nichtlinearität nicht zu vernachlässigen wird das Modell exakt linearisiert. Der entwickelte Regler wird dann auf dem Microcontroller Board, im Inneren der Plattform, implementiert und getestet.

Abstract

For autonomous underwater vehicles, it is absolutely essential to be able to reach and keep a given depth autonomously. This procedure is usually not trivial, because a depth change is a nonlinear event. In addition disturbances, like water currents have to be compensated.
This thesis deals with the development of a depth controller for a very small underwater platform and should perform this tasks. Therefore, the model of the platform is deduced from the equations of motions and simulated in Matlab Simulink. With this model, it is possible to simulate arbitrary control structures and control parameters in short time. Because the platform is very small, simple control structures like PID or state feedback will be used. To not neglect the nonlinearity, the system will be exact linearized. The designed controller will then be implemented on the micro controller board inside the platform.

Contents

List of Figures

List of Tables

1 Introduction

Underwater missions are always an expensive and sophisticated task. Divers can only operate in very limited depths and manned submarines are extremely expensive and very rare. Therefore, many missions are realized by the use of remote controlled or autonomous underwater robots. These systems can go up to very high depth without endangering any humans. Some tasks for underwater robots are the exploration of the sea, the search for underwater mines as well as working in different areas, like for example after the drowning of the oil platform Deepwater Horizon in 2010 [Joh12]. Another domain for especially small autonomous underwater robots (AUV) is the exploration of swarm algorithms [CoC13].
A main objective for all kinds of autonomous underwater robots is to reach and keep a given depth autonomously. A controller for this given task has to fulfill several requirements. First of all, it has to keep the depth within the tolerance under the disturbance of sea current or hydrodynamic forces. Second, the controller has to be simple enough to be implemented on the micro-controller of the robot. The following thesis is going to describe a minimalistic way of implementing a controller inside a very small AUV platform.
There are already several very complex depth control systems using high performance control algorithms like neuronal net control, fuzzy control or adaptive control [J.Y00]. All of this control structures are designed for underwater vehicles operating in the sea in a depth of up to 6000 meters [J.Y00]. These locations require a much better control algorithm because of the disturbances trough sea currents. All of this structures are very powerful but way to complicated and complex to be implemented inside the very small platform that is built to operate in an aquarium without massive disturbances [CoC13]. The platform is equipped with a variable compressor to change the depth. This method of depth changing is called the indirect method. The variable compressor is changing the volume and therefore the water displacement of the robot. This creates up- or down forces that move the robot up or down. In theory, this method only consumes energy when the depth is changed. When holding a constant depth, the forces pulling it up and down should be exactly equal, so there is no need to move the compressor. In the praxis, this point will never be reached exactly. This means that the compressor will be moving slightly all the time. An other method of changing the depth is the direct method. This method is easier to control, because the force pushing the robot under water can be controlled directly. The direct method uses a marine screw propeller to create a constant down force that holds the robot under water. If the platform is

balanced very well, it also consumes energy only for changing the depth. As soon as the mass of the robot is varying, the energy consumption will increase. The best diving method would be a combination of both. The indirect method to balance the robot and the direct to reach a given depth.

indirect	direct
energy only for depth changing	constant use of energy
needs a pressure tank	only needs a turbine
controlling is very complex	easy to control
can't go up without motor movement	self rising if the motor is not working

Table 1.1: Comparison of direct and indirect diving

1.1 Goal

The goal of this thesis is to design a preferably simple controller to reach and keep a given depth. This controller has to be designed, simulated and implemented on the experimentation platform.

1.2 Approach

The first objective to reach the given goal, is to create a model of the system. Therefore the equations of motion have to be set up. The equations have to be transformed into a Simulink modell to simulate them. Up next is the design of the controller. One kind possible controller for the given task is a PID controller. This kind of controller could work if the nonlinearity of the system can be neglected. Linearizing the system is a common way to simplify a system. In the given case, the linearized system should still match the reality because the robot is using very low speed. If the nonlinearity can't be ignored, it is necessary to compensate the nonlinearity. For this, the system can be transformed into a linear system with the method of exact linearization. The transformed system can then be easily controlled with a state feedback controller. If the simulated controller reaches the given goal, it can be implemented on the micro processor inside the robot.

2 Modeling

An underwater robot is a non linear and time invariant system [J.Y00]. Therefore it is absolutely necessary to have a model to simulate the effect of various control parameters. The base for the model are mathematical equations that describe the real system as good as possible. For the control unit design, it is often necessary to make some simplifications. The system model has to match the real system. So it is better to model it first without simplifications and then do the controller design with a simplified model.

2.1 Dynamics

The vertical dynamics of the underwater robot are described trough three parts, the weight (Sec. 2.1.1), buoyancy (Sec. 2.1.2) and the hydrodynamic resistance (Sec. 2.1.3) of the water. To transform the power balance into a differential equation, Newton's second law can be used:

$$F = m * a \tag{2.1}$$

With:

$$\begin{aligned} F &= \text{Force [N]} \\ m &= \text{mass [kg]} \\ a &= \text{acceleration } \left[\frac{m}{s^2}\right] \end{aligned}$$

2.1.1 Weight force

The weight force is the only force pulling the robot under water. It is made up of the mass of the robot and the gravity of the earth. In general, the gravity depends on the position where the experiments are made. But the difference is very low, so all calculations are made with a gravity of $9.81 \frac{m}{s^2}$. The weight force of the platform is constant.

$$F_g = m * g \tag{2.2}$$

With: [BKL05]

F_g = weight [N]

g = gravitation acceleration $\left[\frac{m}{s^2}\right]$ = 9.81

2.1.2 Buoyancy

The buoyancy is pointing in the opposite direction of the weight force. It is responsible for bringing the robot up again. If the buoyancy is bigger than the weight force, the robot is going upwards, if the buoyancy is lower than the weight force, the robot is sinking down. To hold a constant depth, both forces have to be exactly equal. Because of the compressor, the buoyancy is also responsible for every depth change. Trough pulling the compressor in, the volume and the buoyancy of the robot is lowered and the robot is sinking down. To go up again, the expanding compressor increases the buoyancy.

$$F_a = \rho * V * g \tag{2.3}$$

With: [BKL05]

F_a = acceleration power [N]

V = volume $[m^3]$

ρ = density of the water $\left[\frac{kg}{m^3}\right]$

2.1.3 Hydrodynamic resistance

The Hydrodynamic resistance occurs in both directions. It acts like a break and is lowering the diving speed of the robot. It represents the flow resistance through the density of the water and the shape of the robot. When holding a constant depth, there is no resistance.

$$F_s = \frac{1}{2} * \rho * v^2 * A * c_w \tag{2.4}$$

With: [BKL05]

F_s = Hydrodynamic resistance [N]

v = Diving speed $\left[\frac{m}{s}\right]$

A = surface area of the robot $[m^2]$

c_w = drag coefficient

Because of the bidirectional functionality of the robot, the equation has to be modified. The direction of the robot has no influence in this equation, because the speed is squared. With a negative velocity, the resistance would also point in the positive direction, so it would accelerate the robot.
To get the flow resistance working in both directions, the v^2 is replaced by $v * |v|$

$$F_s = \frac{1}{2} * \rho * v * |v| * A * c_w \tag{2.5}$$

2.1.4 Complete equations of motion

The equations (2.2) to (2.4) can be composed to only one, describing the power balance of the whole system.

$$F = m * g - \rho * V * g + \frac{1}{2} * \rho * v * |v| * A * c_w \tag{2.6}$$

This power balance can be transformed by the use of Eq. 2.1 in a differential equation, describing the dynamics of the system.

$$\ddot{x} = g - \frac{\rho * g * V}{m} + \frac{1}{2 * m} * \rho * \dot{x} * |\dot{x}| * A * c_w \tag{2.7}$$

The depth of the robot is changed by creating an up- or down force trough varying the volume. Unfortunately there is no feedback sensor leading back the actual position of the compressor to control the volume directly. Therefore the volume can't be used as input value. The volume is replaced with the integral over the motor time, which represents the force pushing the compressor up or down. The motor time is then used as input signal for the controller.

2.1.5 State space representation

In state space representation, a n - th order differential equation is described through n first order differential equations [Lun07]. If the model is given in state space representation, with a initial condition x_0, it is then possible to compute the unique

output parameter for each possible input parameter.

$$
\begin{aligned}
x_1 &= d \\
x_2 &= v = \dot{x_1} \\
x_3 &= \frac{m}{\rho} - V \\
\\
\dot{x_1} &= x_2 \\
\dot{x_2} &= k_1 * x_3 - k_2 * x_2 * |x_2| \\
\dot{x_3} &= k * u = -\dot{V} \\
\\
x(0) &= x_0, \text{with} x = [x_1, x_2, x_3]^T
\end{aligned}
\tag{2.8}
$$

With the limits:

$$
\begin{aligned}
4.25 * 10^{-4}\text{m}^3 \leqslant & V \leqslant 4.45 * 10^{-4}\text{m}^3 \\
2.5 * 10^{-6}\frac{\text{m}^3}{\text{s}} \leqslant & \dot{V} \leqslant 2.5^{-6}\frac{\text{m}^3}{\text{s}} \\
-1 \leqslant & u \leqslant 1
\end{aligned}
$$

$$
\begin{aligned}
k_1 &= \frac{\rho * g}{m} \\
k_2 &= \frac{1}{2 * m} * \rho * A * c_w \\
d &= \text{Depth of the robot [m]} \\
k &= \text{Transmission factor of the compressor} \\
u &= \text{input value}
\end{aligned}
$$

The diving speed of the robot has a quadratic influence on the hydrodynamic resistance. Thin nonlinearity would be much more difficult to control than a linear system. Because of the very low speed of the robot the error through the assumption that the speed is influencing the hydrodynamic resistance in a linear way would be very low, so the system is handled as linear system.
With this simplification done, the system can be transfered into state space repre-

sentation.

$$
\begin{aligned}
\dot{x} &= Ax + Bu \\
y &= C^T x + Du \\
\dot{x} &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & -k_2 * c_{lin} & k_1 \\ 0 & 0 & 0 \end{bmatrix} * x + \begin{bmatrix} 0 \\ 0 \\ k \end{bmatrix} * u \\
y &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} * x
\end{aligned}
\tag{2.9}
$$

With:

c_{lin} = Linearization constant

2.2 Model Parameters

There are only five Parameters that describe the whole system. The weight (m) , the surface (A) the drag coefficient of the robot(c_w), the transmission factor(k) between the motor and the volume - slider and the density of the water (ρ).

2.2.1 Weight (m)

The weight of the bot is affecting the force that pulls the robot under water. The weight can be measured easily by putting it on a scale. The robot weighs 0.434 kg.

2.2.2 Surface area (A)

When going up or down, the complete projection surface of the robot has to pass the water, so the surface is of great significance for the flow resistance. The part of the flow resistance referring to the shape of the bot is described through the product of surface area and drag coefficient. The robot has an elliptic shape. It is 12.5 cm long and 9.5 cm wide. So the projetion surface area is approximately $8 * 10^{-3} m^2$

2.2.3 Drag coefficient (c_w)

The drag coefficient reflects the resistance of the robot against the water. It is significantly influencing the hydrodynamic resistance. A lower drag coefficient means that there is less drag.The shape of the top and the bottom part of the robot are different. While the top has an elliptic shape and can be approximated by a circle (drag coefficient of 1.11 [Boe09]), the bottom is more like a sphere (drag coefficient of 0.4 [Boe09]). The actual coefficient is something in the middle. For all calculations, the drag coefficient will be 0.75.

2.2.4 Transmission Factor (k)

The transmission factor represents the factor between motor runtime and volume change. Therefore, it is necessary to know the volume of the robot with fully extended and fully retracted slider. This can be measured by putting the robot in the water and measuring the supplanted water. The complete volume is between 425 and 445 cm^3 so the volume can be altered by 20 cm^3. To fully extend the compressor, it takes about 8 seconds. So the changing rate is $2.5\frac{cm^3}{s}$ or $2.5 * 10^{-6}\frac{m^3}{s}$. The time to expand the compressor is depending on the speed of the motor and therefore on the battery voltage. To have an accurate value, the robot measures the times and calculates the transmission factor when the compressor is initialized. But the simulation is made with a constant factor of $2.5 * 10^{-6}\frac{m^3}{s}$.

2.2.5 Water - density (ρ)

In general, the density of water is depending on the pressure and temperature [ALL92]. It would be possible to compensated these influences with the data from the pressure sensor. Because of the very low difference, this isn't necessary and all calculations are made with a density of $1000\frac{kg}{m^3}$.

2.2.6 Linearization factor (c_{lin})

The purpose of the linearization factor is to minimize the error, that is made through the linearization of the influence of the velocity. Therefore the maximal diving speed of the robot has to be estimate. This speed is very low. So a value of $0.2\frac{m}{s}$ should match the reality very good. Then the polynomial is approximated with a straight line. A line with a slope of 0.15 matches very good. This value is determined with the Matlab function polyfit. Figure 2.1 shows the polynomial velocity (green) together with the linearized velocity (blue).

Parameter	Value
ρ	$1000kg * m^{-3}$
A	$8 * 10^{-3}m^2$
c_w	0.75
m	0.434kg
c_{lin}	0.15
k	$2.5 * 10^{-6}\frac{m^3}{s}$

Table 2.1: System Parameters

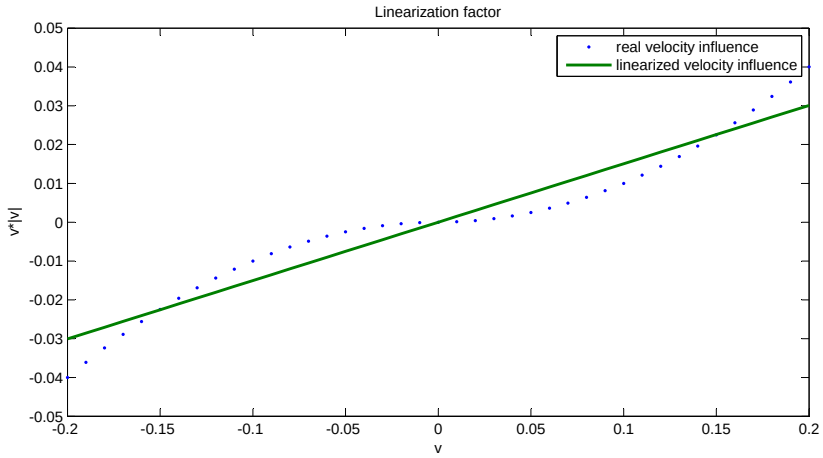

Figure 2.1: Comparison of the linearized velocity influence and the real velocity influence

2.3 Simulink model

To make sure the reactions of the modeled system and the real system match, the model has to match the real system as perfect as possible. The modeled system represents the complete equation of motion (Eq. 2.7). The Simulink model of the robot

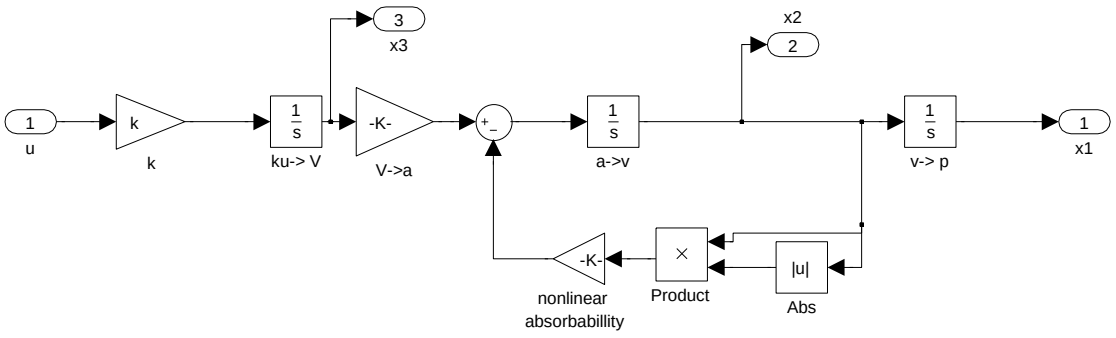

Figure 2.2: Simulink model of the dynamics of the robot

dynamics is shown in Fig. 2.2. The input variable u represents the output of the controller. The states x_1 (depth), x_2 (velocity) and x_3 (weight compensated volume) are representing the states of the system. The signals x_2 and x_3 are only used for monitoring, because they are not measurable in the real robot. The only value that is used for the control process is the depth x_1, which is also available on the real system trough the pressure sensor. The triangular blocks represent a multiplication

with a constant, while the square product block stands for the multiplication of two signals.

3 Control unit design

3.1 Stability

To make statements about the stability of the System, the eigenvalues of the control Matrix (2.9) have to be computed. Figure 3.1 shows the Pole-zero plot of the system.

$$\det(sI - A) = 0 \Rightarrow s_1 = 0, s_2 = 0, s_3 = -k2 * c_{lin} \tag{3.1}$$

Because of the two eigenvalues $s_{1,2} = 0$, the system is instable. To make it a stable system, a controller is necessary.

3.2 Controllability

To make sure that the System can be controlled, the controllability matrix Q_s (3.2) is computed.

$$Q_s = [b, Ab, A^2b] = \begin{bmatrix} 0 & 0 & k_1 * k \\ 0 & k_1 * k & -k * k_1 * k_2 * c_{lin} \\ k & 0 & 0 \end{bmatrix} \tag{3.2}$$

With: [Ebe11]

The matrix shows that$Rank(Q_s) = 3$ for $k \neq 0$ and the system is controllable.

3.3 Observability

If the PID-controller can't stabilize the system, it might be necessary to exact linearize the system and implement a state feedback controller. This kind of controller needs all states of the system, not only the measurable x_1. These other states can be calculated by a state observer, if the input parameter and the output parameter are measurable [Ebe11]. To make sure that the system is observable, the observability matrix is calculated.

$$Q_b = [c, A^T c, (A^2)^T c]^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -k_2 * c_{lin} & k_1 \end{bmatrix} \tag{3.3}$$

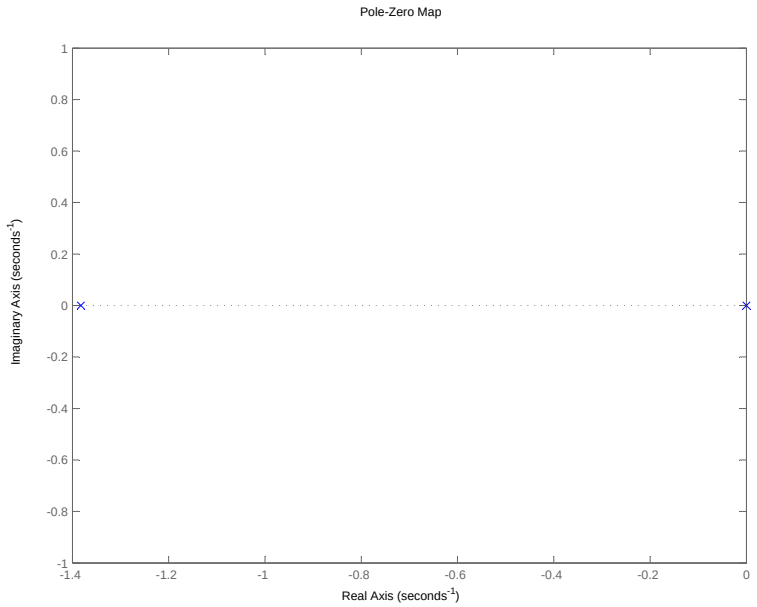

Figure 3.1: Pole-zero plot

As in the controllability matrix, the matrix has the full rank of 3 and is therefore observable for $k_1 \neq 0$.

3.4 Transfer Function

For further control technique analysis, the transfer function is necessary. It can be derivated from the state space representation.

$$G(s) = c^T(sI - A)^{-1}b = \frac{k_1 * k}{s^3 + k_2 * s^2 * c_{lin}} \tag{3.4}$$

3.5 PID-Controller

The PID-Controller (proportiona integral-derivative controller) is a linear controller consisting of three parts, the P-term, the I-term and the D-term. Because of the linearity of the PID-controller, the nonlinear system has to be linearized. Therefore the nonlinear resistance is approximated with a linear term. This might lead to some inaccuracie. Figure 3.2 shows the control loop with a PID-controller.

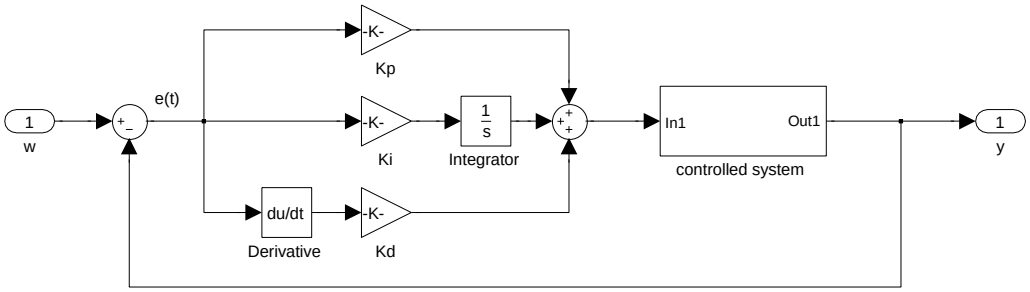

Figure 3.2: Model of the control loop with a PID-controller

3.5.1 P-term

The P-term is a simple proportional boost of the input signal (the deviation between desired and real depth), with the factor K_P. The main task of the P-term is to scale the input signal to get an output value that stabilizes the system.

$$P_{out} = e(t) * K_P \tag{3.5}$$

With:

$$K_P = \text{proportional factor}$$
$$e(t) = \text{desired depth - actual depth}$$

3.5.2 I-term

The I-term integrates the error between desired and actual position and multiplies it with a static factor K_I. Thus the value is increasing when the desired depth is differing from the real depth. The I-Term eliminates the permanent control deviation.

$$I_{out} = K_I * \int_0^t e(\tau)\, d\tau \tag{3.6}$$

With:

$$K_I = \text{integrating factor}$$

3.5.3 D - Term

The D - term is a differentiator. It does not directly react to the error $e(t)$ but to the change of the error. So the D - term alters the output value as soon as the error is varying. This makes the PID - controller a suitable choice for the given system.

$$D_{out} = K_D * \frac{d}{dt} e(t) \tag{3.7}$$

With:

K_D = differential factor

3.5.4 Anti-windup

The control variables like the volume and the changing rate are limited. An integrator together with limited control variables is always a bit more difficult. If a large control variable is limited by the saturation, the integrator is still integrating but the control variable isn't rising. If the error $e(t)$ is getting smaller again, there is a delay trough the high integrator value [Ada09]. This could lead to the escalation of the system or the system could begin to swing. It is necessary to implement an anti - windup strategy. This can be done by subtracting the value before and behind the limitation, multiplying it with a constant and subtracting it from the integrator. So the integrator value is decreased, when the control variable is limited by the saturation. Figure 3.3 shows the previous PID-controller with an anti - windup method.

3.5.5 Loop tuning

To stabilize a system with a PID-controller, it is necessary to appoint the three parameters K_P, K_D and K_I. This can be done by several methods.
The first method is to tune the loop automatically. A tool like Matlab needs the transfer function of the system and calculates the three parameters. The problem with this method is, that it's hard to tell how the system is stabilized and that parameters like the reaction time can't be influenced .
The second method is to use empirical determined rules, like the Ziegler Nichols adjustment tuning [Lun07]. With this method, the control parameters are determined with rules based on the gain, time constant and reaction time of the system. But this methods requires the system to be already stable. Thus the method can't

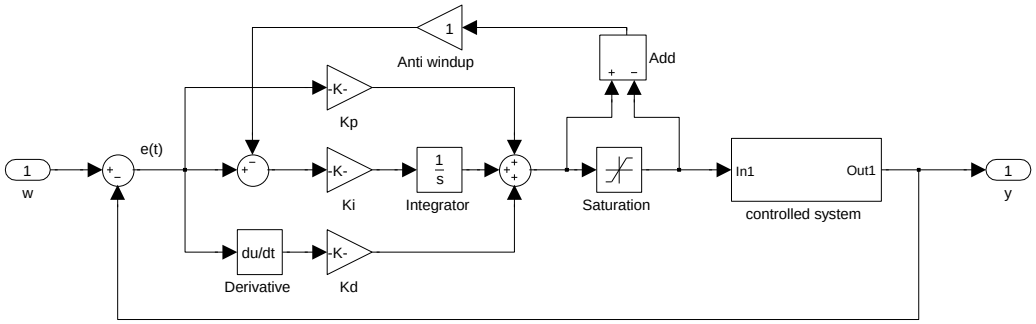

Figure 3.3: Structure of a PID-controller with anti-windup

be used, because the given system is instable.
The method that is used in this case, does a pole placement. It tries to compensate the problematic poles to stabilize the system. The control parameters can be calculated through the transfer function of the closed loop.

$$
\begin{aligned}
K(s) &= \frac{K_I + s * K_P + s^2 * K_D}{s};\ G(s) = \frac{k_1 * k}{s^3 + s^2 * k_2} \\
p(s) &= (s + \frac{1}{4}k_2)^4 = s^4 + s^3 * k_2 + \frac{3}{8} * s^2 * k_2^2 + \frac{1}{16} * s * k_2^3 + \frac{1}{256} * k_2^4 \\
G * K + 1 &= s^4 + s^3 * k_2 + s^2 * k_1 * k * K_D + s * k_1 * k * K_P + k_1 * k * K_I \stackrel{!}{=} p(s)
\end{aligned}
\tag{3.8}
$$

Unfortunately the closed loop is a fourth order system. The PID-controller ($K(s)$) has only three degrees of freedom and allows therefore only three poles to be placed arbitrary. To stabilize the system, the poles have to be placed, so that the s^3 term of the desired polynomial ($p(s)$) equals the s^3 term of $G * K + 1$. So the poles are placed to $(s + \frac{1}{4} * k_2)^4$.
The numerator of the closed loop can't be altered anyway. Therefore it is enough to calculated $G * K + 1$ instead of $\frac{G*K}{G*K+1}$. This keeps the equation easier.

Equating the coefficients provides the following control parameters:

$$
\begin{aligned}
K_D &= \frac{3 * k_2^2}{8 * k_1 * k} \\
K_P &= \frac{1 * k_2^3}{16 * k_1 * k} \\
K_I &= \frac{1 * k_2^4}{256 * k_1 * k}
\end{aligned}
\tag{3.9}
$$

These parameters can be manually fine tuned. The simulation shows that the best behavior of the system is reached with $K_{P,tuned} = \frac{K_P}{10}$, $K_{I,tuned} = \frac{K_I}{100}$ and $K_{D,tuned} = K_D$.

3.6 Nonlinear approach

The simulation of the linear PID-controller (Sec. 4.2) shows that the system can be stabilized, but the permanent control deviation can't be eliminated without a nonlinear controller. The robot is swinging around the desired position. To get better results, the nonlinearity can't be ignored.

3.6.1 Exact linearization

With the exact linearization, the nonlinear system is transformed into a linear system, that can easily be controlled by a linear controller. Therefore the nonlinearity on the entrance of the control path is being compensated by an inverse nonlinearity inside the controller. Figure 3.4 shows the principle of exact linearization on an example system, where a cubic function inside the control path is compensated by the cube root in the controller.
To exact linearize the system, it is necessary to calculate the new control variable u [FH08].

$$u = \frac{-L_f^3 h(x) + v}{L_g L_f^2 h(x)} \tag{3.10}$$

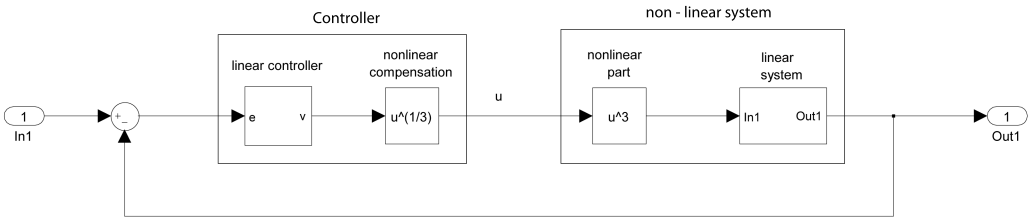

Figure 3.4: Principle of exact linearization [FH08]

Given the system in the form:

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{3.11}$$

Leads to:

$$\begin{aligned} f(x) &= \begin{pmatrix} x_2 \\ k_1 * x_3 - k_2 * x_2 * |x_2| \\ 0 \end{pmatrix}; g(x) = \begin{pmatrix} 0 \\ 0 \\ k \end{pmatrix} \\ y(x) &= x_1 \end{aligned} \tag{3.12}$$

The first step is, to compute the Lie - derivatives of the vector fields g(x) and f(x) and the scalar function h(x).

$$L_f h(x) = \left(\frac{\partial h}{\partial x}\right)^T * f(x) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}^T * f(x) = x_2 \tag{3.13}$$

The 2nd. step:

$$L_f^2 h(x) = \frac{\partial [L_f h(x)]}{\partial x} * f(x) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}^T * f(x) = k_1 * x_3 - k_2 * x_2 * |x_2| \tag{3.14}$$

The absolute value is not continuously differentiable because of the kink at $f(x) = 0$. Because of that, it can't be easily derived. In the given case, the derivation can be calculated with a trick. The problematic point $f(x) = 0$ can be neglected because the complete term becomes zero through the multiplication with x_2. The derivation represents the slope at the actual position. The slope of the absolute value function is -1 for $x < 0$ and $+1$ for $x > 0$. This is exactly the same behavior as the *sign* function. So the derivative of the absolute value function can be described through the *sign* function. Figure 3.5 shows the absolute value and the sign function as the derivative.

$$\begin{aligned} f(x) = |x|, \dot{f(x)} &= \begin{cases} -1, & x < 0 \\ 1, & x > 0 \end{cases} = sign(x) \\ \frac{d}{dx} x_2 * |x_2| &= |x_2| + x_2 * sign(x_2) = 2 * |x_2| \end{aligned} \tag{3.15}$$

So the complete Lie - derivation can be computed as follows.

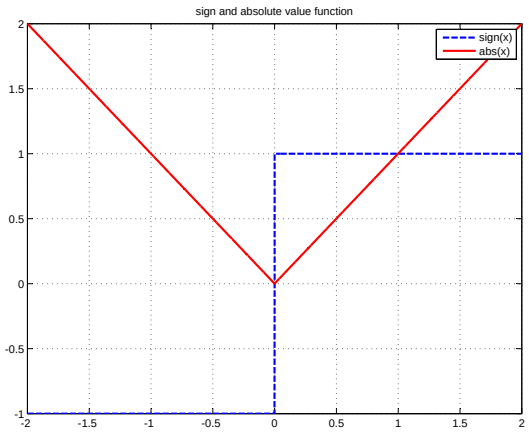

Figure 3.5: Plot of the sign function (green) and absolute value(red)

$$
\begin{aligned}
L_f^3 h(x) &= \begin{pmatrix} 1 \\ -|x_2| * k_2 - x_2 * sign(x_2) * k_2 \\ k_1 \end{pmatrix} * f(x) \\
&= -k_1 * k_2 * |x_2| * x_3 - k_1 * k_2 * x_3 * x_2 * sign(x_2) + k_2^2 * x_2^3 \\
&+ x_2^2 * |x_2| * k_2^2 * sign(x_2) \\
&\text{with } x_2 * sign(x_2) = |x_2| \text{ and } |x_2| * sign(x_2) = x_2 \\
&= -2 * k_1 * k_2 * |x_2| * x_3 + 2 * k_2^2 * x_2^3 = \alpha
\end{aligned}
\tag{3.16}
$$

$$
L_g h(x) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}^T * g(x) = 0
$$

$$
L_g L_f h(x) = \left[\frac{\partial L_f h(x)}{\partial x} \right]^T * g(x) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}^T * g(x) = 0
$$

$$
L_g L_f^2 h(x) = \begin{pmatrix} 0 \\ -|x_2| * k_2 - x_2 * sign(x_2) * k_2 \\ k_1 \end{pmatrix} * g(x) = k_1 * k = \beta
$$

Only the third term is not equal to zero, so the relative degree is 3 and the necessary condition is fulfilled.
With all Lie - derivations computed, the new control variable u can be calculated.

The control variable u is the signal that controls the motor power. To compute u, a new input signal v is introduced. This variable v represents the linear input for the controller.

$$\begin{aligned} u &= \frac{1}{\beta}(v - a) \\ &= \frac{1}{k_1 * k} * (v + 2 * k_1 * k_2 * |x_2| * x_3 + 2 * k_2^2 * x_2^3) \end{aligned} \tag{3.17}$$

With:

$$\begin{aligned} k_1 &= \frac{\rho * g}{m} \\ k_2 &= \frac{1}{2 * m} * \rho * A * c_w \\ u &= \text{control variable} \\ v &= \text{new input variable} \end{aligned}$$

This equations are modeled in Simulink to be able to simulate the controller. Figure 3.6 shows the previously calculated equation for u in Simulink.

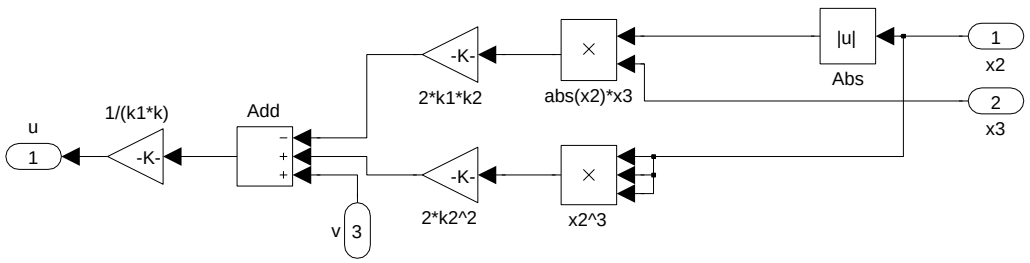

Figure 3.6: Model of the exact linearization equation

As mentioned above, the nonlinear system is transformed into a linear system,

to be controlled by a linear controller. This transformation is calculated as follows.

$$
\begin{aligned}
z &= \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} h(x) \\ L_f h(x) \\ L_f^2 h(x) \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ k_1 * x_3 - x_2 * |x_2| * k_2 \end{pmatrix} \\
\dot{z} &= \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 * k_1 - \dot{x}_2 * |x_2| * k_2 - sign(x_2) * x_2 * \dot{x}_2 * k_2 \end{pmatrix} \\
&= \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 * k_1 - \dot{x}_2 * |x_2| * k_2 - |x_2| * \dot{x}_2 * k_2 \end{pmatrix}
\end{aligned}
$$

$$
\begin{aligned}
\dot{z}_1 &= f_1(x) + g_1(x)u = x_2 \\
\dot{z}_2 &= f_2(x) + g_2(x)u = k_1 * x_3 - k_2 * x_2 * |x_2| \\
\dot{z}_3 &= f_3(x) + g_3(x)u = k * u
\end{aligned}
\tag{3.18}
$$

$$
\begin{aligned}
\dot{z} &= \begin{pmatrix} x_2 \\ k_1 * x_3 - k_2 * x_2 * |x_2| \\ k_1 * k * u - (k_1 * x_3 - k_2 * x_2 * |x_2|) * 2 * |x_2| * k_2 \end{pmatrix} \\
&= \begin{pmatrix} z_2 \\ z_3 \\ v \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} z + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} v \\
y &= z_1
\end{aligned}
$$

The transformed system is equal to a chain of three integrators and can be controlled very easily.

3.6.2 State Feedback

A internal feedback controller is a linear controller that stabilizes through feeding back all states multiplied by a factor. Therefore it is necessary to know all states. In the given case, only the position of the robot (x_1) can be measured. All other states have to be estimated. The feedback factors can be computed very easily by the direct solution process [Ebe11].

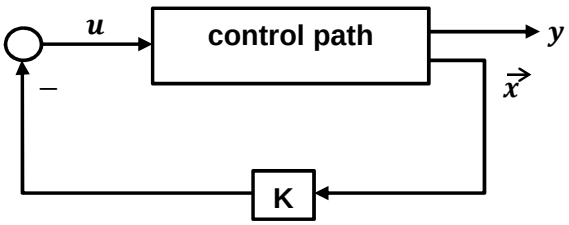

Figure 3.7: Structure of a state feedback controller

$$
\begin{aligned}
&A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, k = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix} \\
&\det(s * I - (A - b * k^T) \stackrel{!}{=} p(s) \\
&s^3 + s^2 * k_3 + s * k_2 + k_1 \stackrel{!}{=} p(s) \\
&\Rightarrow k_1 = 3, k_2 = 3, k_3 = 1
\end{aligned}
\tag{3.19}
$$

With:

$s =$ Laplace variable

$I =$ identity matrix

$p(s) =$ desired poles, here $(s+1)^3$

With these factors, it is possible to stabilize the system. But a regular state feedback controller doesn't eliminate the permanent control deviation. This problem can be solved by adding an integrator. The integrator integrates the error and therefore eliminates the permanent control deviation, like the I - term of the PID-controller. This leads to an integrator chain with four integrators, instead of three, what means that the feedback factors have to be changed slightly.

$$
\begin{aligned}
&A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, k = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{pmatrix} \\
&\det(s * I - (A - b * k^T) \stackrel{!}{=} p(s) \\
&s^4 + s^3 * k_4 + s^2 * k_3 + s * k_2 + k_1 \stackrel{!}{=} p(s) \\
&\Rightarrow k_1 = 1, k_2 = 4, k_3 = 6, k_4 = 4
\end{aligned}
\tag{3.20}
$$

With:

$p(s) =$ desired poles, $(s+1)^4$

Figure 3.8 shows the complete state feedback controller with the previously calculated parameters modeled in Simulink.

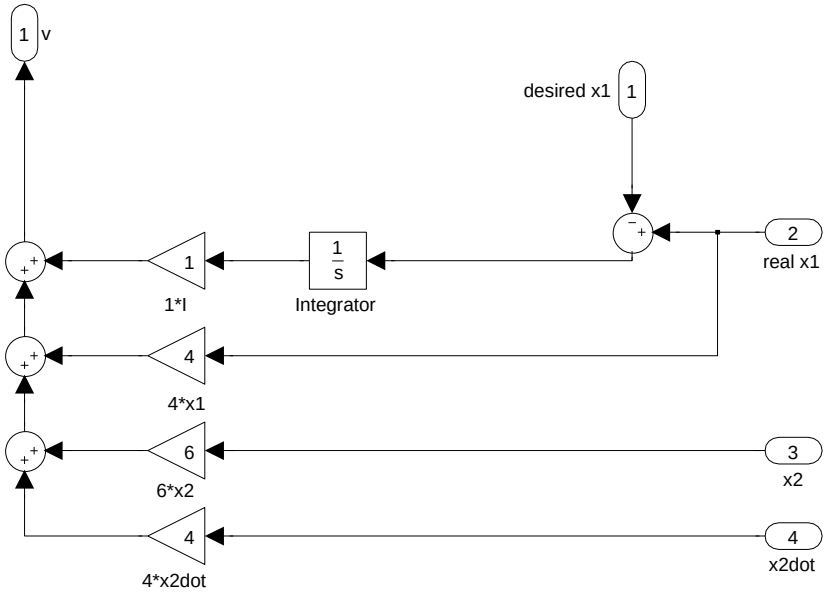

Figure 3.8: State feedback controller modeled in Simulink

3.6.3 State observer

The state feedback controller, as well as the exact linearization need the diving velocity and acceleration, depth and volume of the robot, to act properly. The only known state is the depth x_1 and the output signal u. All other states have to be computed, using only these two values.

The velocity, x_2 can be computed by deriving x_1.

$$x_2 = \frac{\partial}{\partial t} x_1 \tag{3.21}$$

For the state feedback, it is also essential to know the acceleration $\dot{x_2}$. The acceleration could be computed by the second derivation of x_1. This would lead to a very bad signal, because the noise in x_1 would be boosted through the derivations. The better solution is to compute it algebraic.

$$\dot{x_2} = k_1 * x_3 - k_2 * x_2 * |x_2| \tag{3.22}$$

The value of x_3 can be calculated by integrating the output signal, that equals the runtime of the motor and multiplying it with the transmission factor k.

$$x_3 = k * \int_0^t u \, dt \tag{3.23}$$

Figure 3.9 shows the previous equations modeled in Simulink.

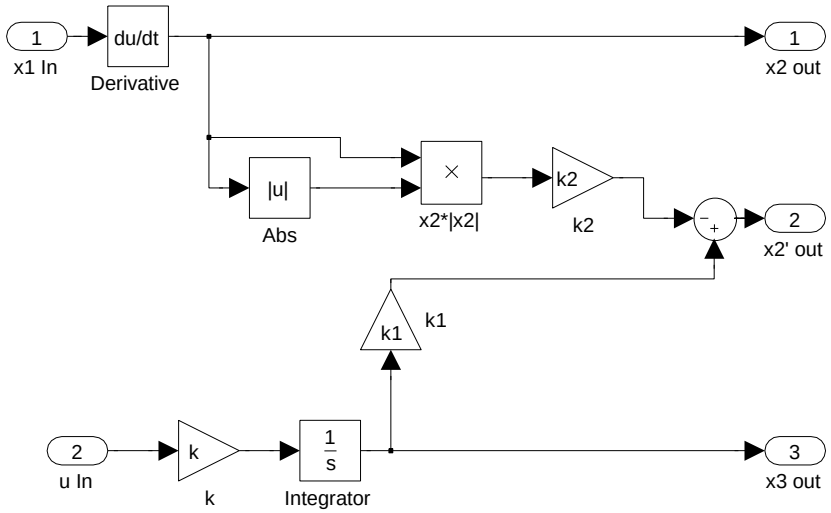

Figure 3.9: State observer modeled with Matlab Simulink

4 Simulation

The Simulation is a very important tool for the control unit design. By simulating every new controller parameter or new controller design, the response of the model is shown within seconds. Implementing these changes on the real platform would take much more time then just changing the simulation.

4.1 Input signal

To simulate the system, it is necessary to have an input signal. This signal should represent the later input signal of the real system. The input is generated trough three step functions. Two of them are positive and one negative. With this setup, it is possible to test the behavior of the system with different depth changes. Figure 4.1 shows the depth profile through the step functions (blue/solid) and the smoothened profile (red/flushed) that is then used for the simulation. Smoothing the profile makes the later results much better, because the overshoot is significantly decreased. The smoothing is realized with the filter function shown in Eq. 4.1.

$$G(s) = \frac{1}{s+1} \tag{4.1}$$

4.2 PID controlled system

First the PID-controller is tested. The PID-controller would be a very simple solution to control the system.

4.2.1 Simulink plan

To keep the plan as clear as possible, the different models are grouped in subsystems. Figure 4.2 shows the complete Simulink plan.
The controller block is the PID-controller with anti - windup (Fig. 3.3). The system model represents the behavior of the robot (Fig. 2.2). The evaluation block calculates the absolute deviation of the control loop. Therefore it subtracts the desired and real position of the robot, squares and integrates it (Fig. 4.3). The scope displays the results of the simulation.

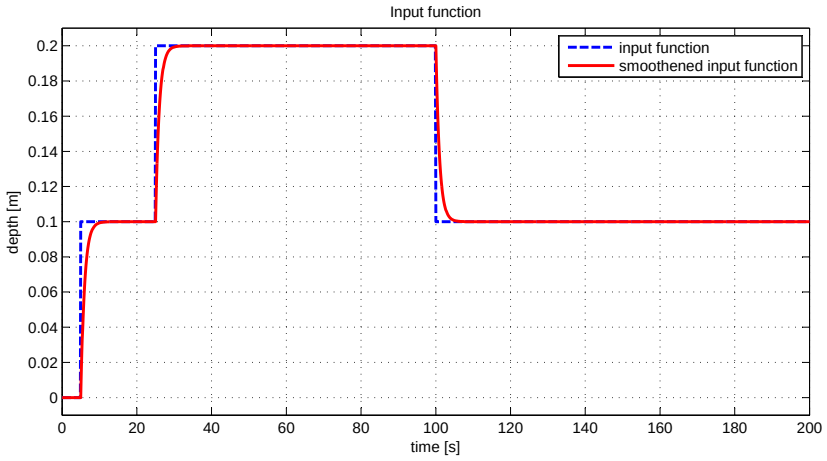

Figure 4.1: Simulated depth profile

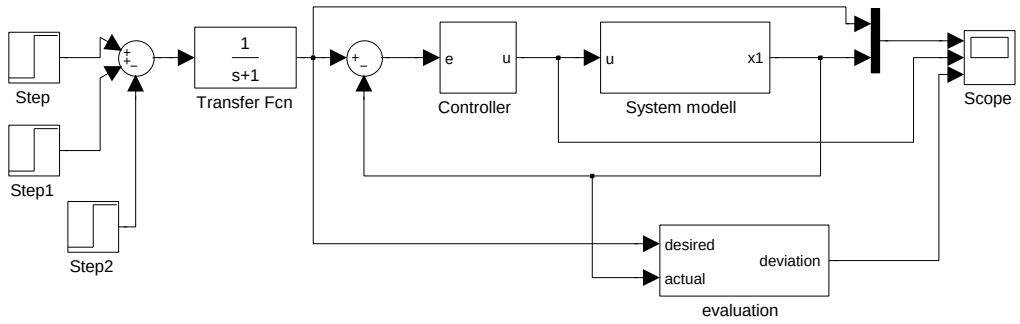

Figure 4.2: Model of the control loop with a PID-controller

4.2.2 Results

The advantage of the simulation is that the results are visible within seconds. There are three diagrams created through the scope block, the absolute deviation (Fig. 4.6), the control variable u (Fig. 4.5) and most important the process of the depth of the platform (Fig. 4.4).
Figure 4.4 shows the behavior of the controlled system, with the tuned parameters from Sec. 3.5.5. The red line represents the desired path of the platform, the blue line corresponds the real depth process. It shows, that the system gets stable and is not collapsing. But there is some overshot after the depth changes and the platform is oscillating around the desired position. This is caused by the nonlinearity of the system that has been linearized. For better results, it is necessary to design a nonlinear controller.

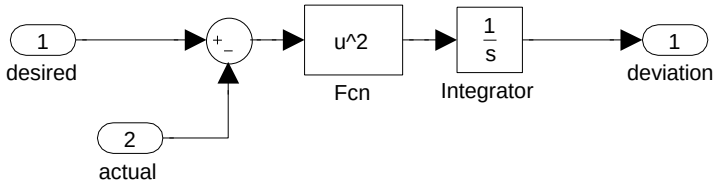

Figure 4.3: Calculation of the absolute deviation

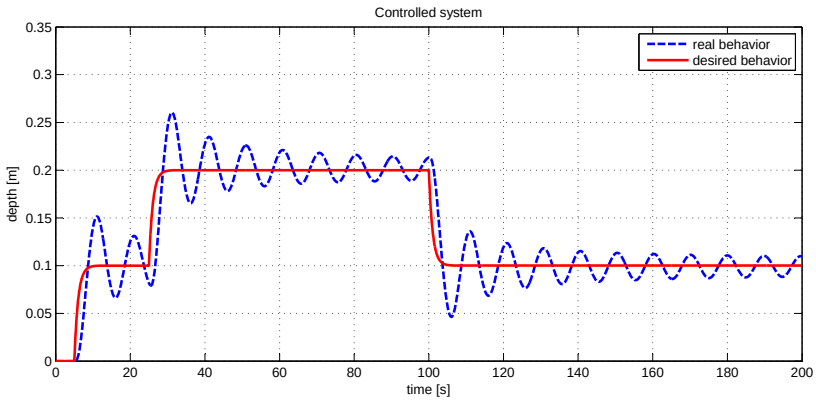

Figure 4.4: Behavior of the controlled PID system

The second digram (Figure 4.5) shows the control variable u created by the controller. This is the variable that controls the motor power in the real system between 100 and zero percent in both directions. The Diagram shows, that the controller hits the limitation several times, but due to the anti - windup strategy, the system is not collapsing. It also shows, that the motor would be running all the time. This leads to a high energy consumption and can be traced back to the swinging of the system.

The last diagram (Figure 4.6) shows the absolute error of the controlled system. It shows, that the error increases more after a change of the desired depth. This can be traced back to the overshoot. After that the slope is decreasing but never getting zero. The reason for that is the permanent oscillating around the desired position. A total error of 0.09 after 200 seconds doesn't look bad on the first view. But the error is squared and small deviations therefore result in a small increase of the total deviation. In addition to that, this error will grow as long as the system is running, because of the permanent control deviation. This error value can be used to compare different controllers.

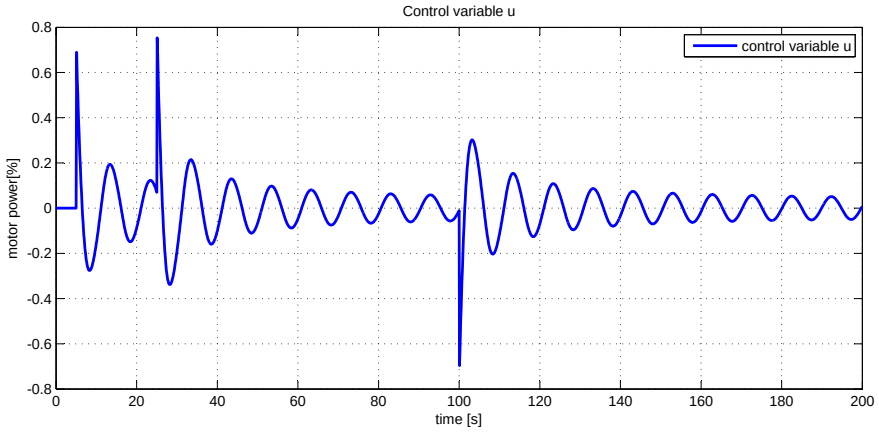

Figure 4.5: Control variable u of the PID controlled system

4.3 Nonlinear state feedback

The results of the nonlinear state feedback controller should be better then the linear PID - controller. To be able to compare the results, the input signal stays the same.

4.3.1 Simulink plan

Figure 4.7 shows the complete Simulink plan with the state feedback controller and the exact linearization to compensate the nonlinearity inside the system. The input function as well as the modell and the evaluation are the same subsystems as in the linear PID case. The observer (Figure 3.9) calculates the needed values for x_2, $\dot{x}_2$ and x_3 with the given position (x_1) and u.
The state feedback controller (Figure 3.8) uses the calculated values, and creates the output value v which is than processed in the exact linearization subsystem. The exact linearization subsystem uses the calculated values to calculate the inverse nonlinearity of the system to compensate it. This subsystem is shown in Figure 3.6.

4.3.2 Results

The variables plotted by the scope are the same like in the PID case.
Figure 4.8 shows the behavior of the system, controlled with the state feedback controller that is shown in Fig. 3.8. The solid red line represents the desired path while the dashed blue line is the real path. It shows, that the platform follows the path way better than with the PID-controller. There is only very little delay until

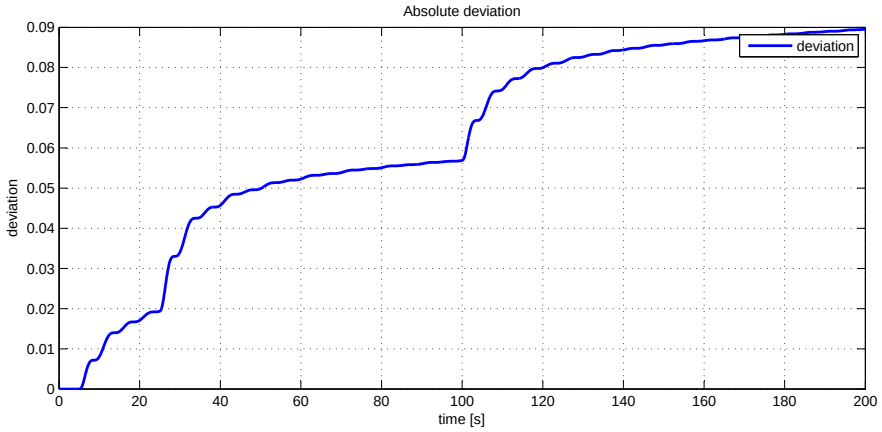

Figure 4.6: Absolute deviation of the closed loop with a PID-controller

the robot starts to follow the depth change. This can be traced back to the high inertia of the system. Compared to the PID controlled system, there is no swinging around the desired position. The platform is following the desired path perfectly.

The control variable u is plotted in figure 4.9. The maximum values of the control variable stays way smaller than in the PID controlled system and it doesn't hit the limitation. So there shouldn't be a windup problem. An anti-windup strategy isn't necessary in this case.

The absolute deviation value (Figure 4.10) is smaller than in the PID controlled System and it doesn't rise without a depth change. This is because of the eliminated permanent control deviation. The system reaches the desired depth and keeps it until there is a change in the desired position. So the only error is made when the depth is changed. This error can be decreased by making the system faster through placing the poles different. But this could also destabilize the system and therefore an error of 0.045 is tolerated.

4.3.3 Margin of error

The previously shown system is calculated with perfect values. These values can't be reached in the real system. Therefore some disturbances have to be simulated to see if the controller can handle them. The real sensor has noise for about 0.1 mm. This doesn't sound much but has a direct effect on the behavior of the system. This disturbance can be modeled by adding white noise to the calculated depth value (x_1). Another major source of error is the calculated volume (x_3). The value is calculated, with the assumption that the motor speed is linear. Because of the poor quality of the motor and the transmission, the actual motor speed is diverging very much. This has been modeled by multiplying the calculated volume with an factor

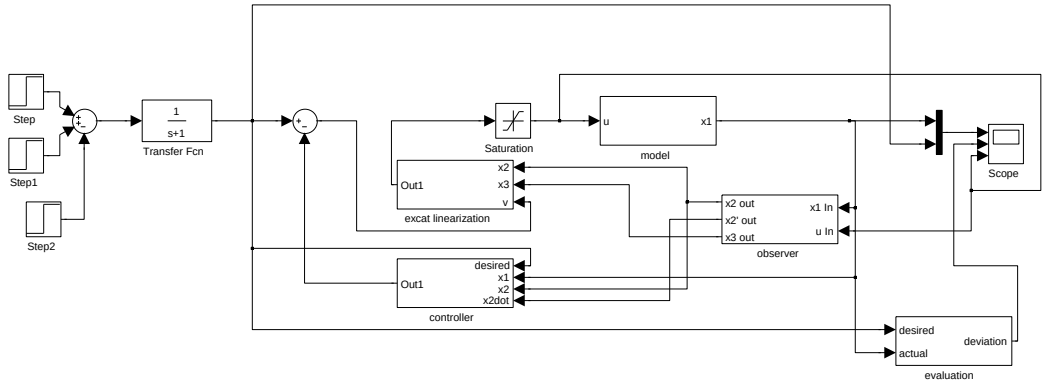

Figure 4.7: Complete Simulink plan with state feedback controller and exact linearization

of 1.2. This means a deviation of 20% and should simulate a behavior more closely to the reality. Figure 4.11 shows the effect of these changes in the simulation. There is no visible change compared to the ideal system. The control variable u (Figure 4.12) looks completely different. The motor is switching between full speed forward and full speed backward all the time. This would result in a massive energy consumption and a very bad behavior of the system, because of the inertia of the robot. The control variable has to be smoothened to make it applicable. This can be done by adding a PT1 term like shown in Eq. 4.2 after the controller.

$$G(s) = \frac{5}{s+5} \tag{4.2}$$

The effect is visible in Figure 4.13. The variable is still noisy but a larger smoothing would lead to too much phase shifting and the collapse of the whole system. This means that the motor is running all the time as well. But a behavior like in Figure 4.9 can't be achieved in reality. Because there is always a disturbance, through inaccuracies of the sensors or from outside the system, like water currents.

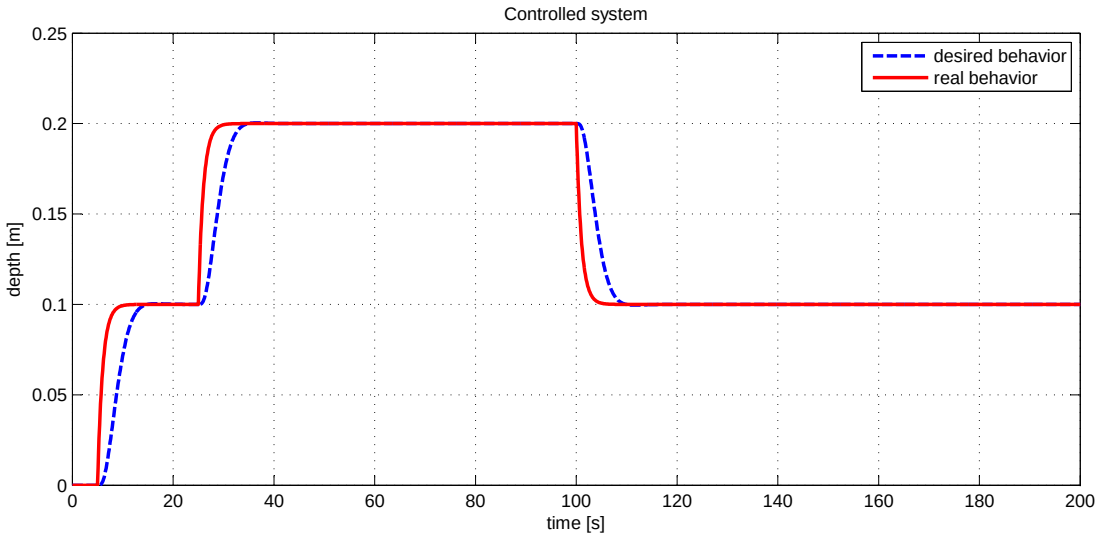

Figure 4.8: Behavior of the system with a state feedback controller

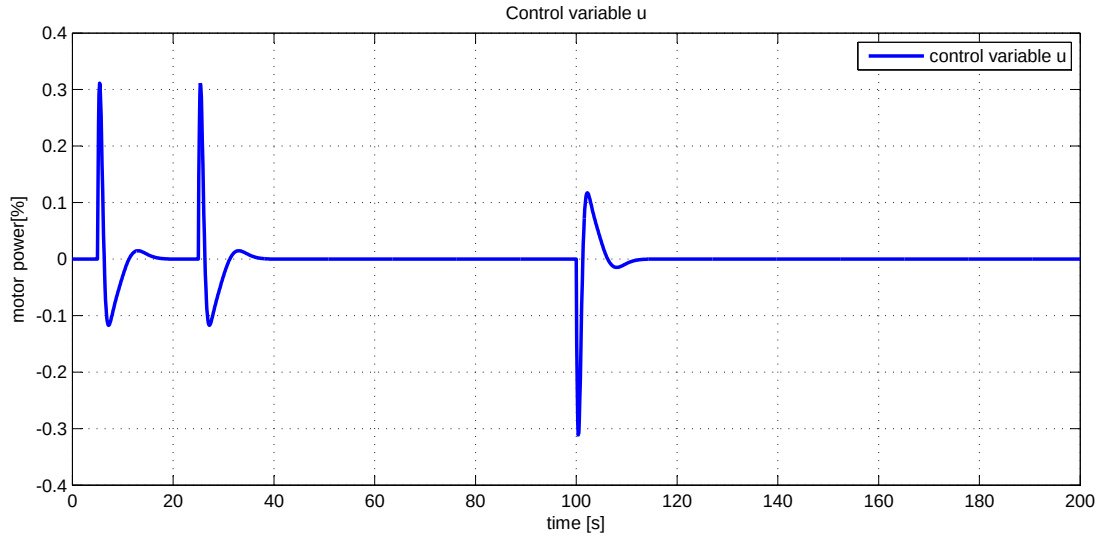

Figure 4.9: Control variable u of the state feedback controlled system

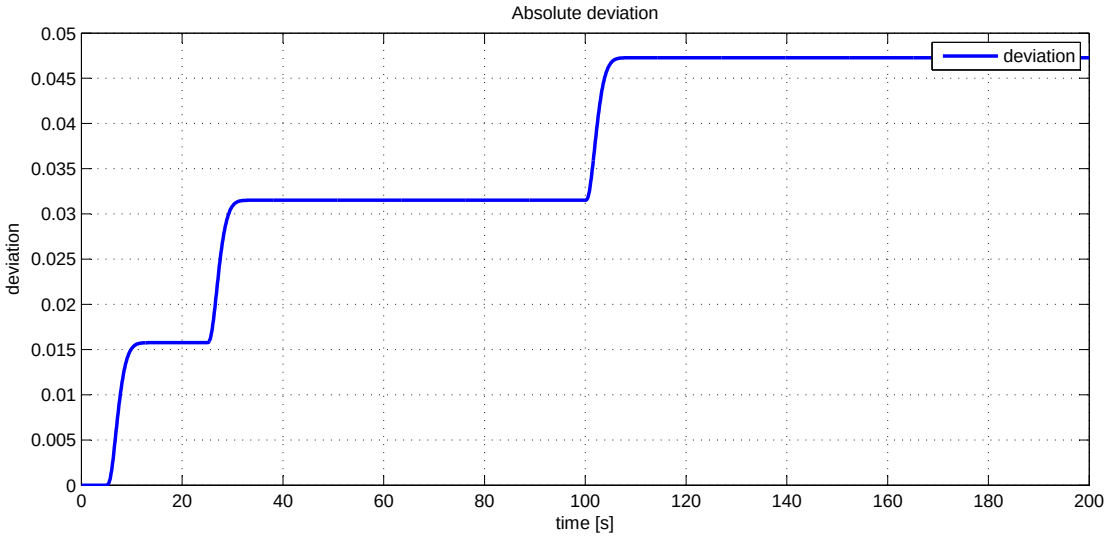

Figure 4.10: Absolute deviation of the state feedback controlled system

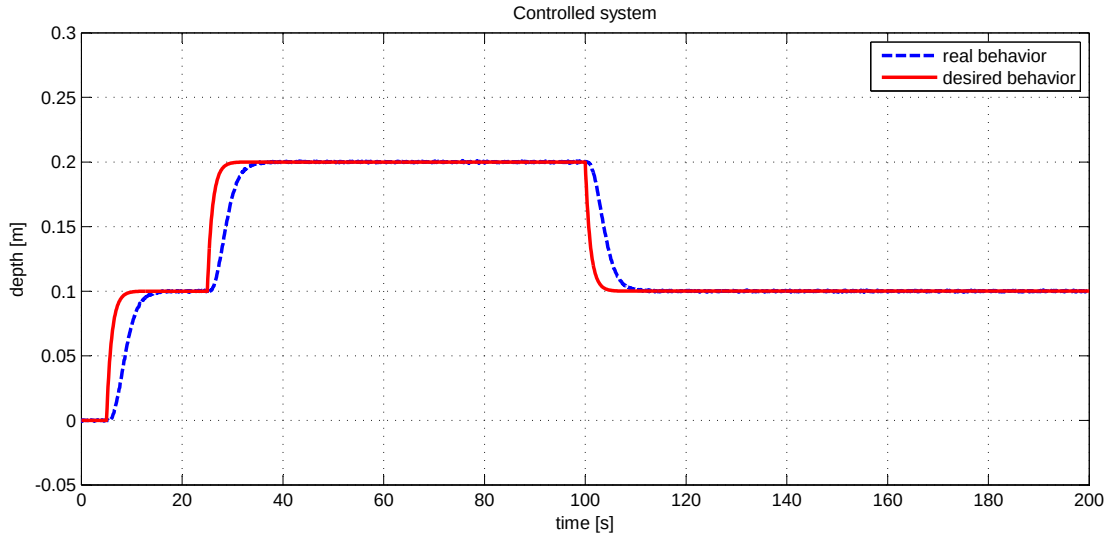

Figure 4.11: Sate feedback controlled system with simulated disturbances of white noise added to the simulated sensor value and a gain on the estimated volume

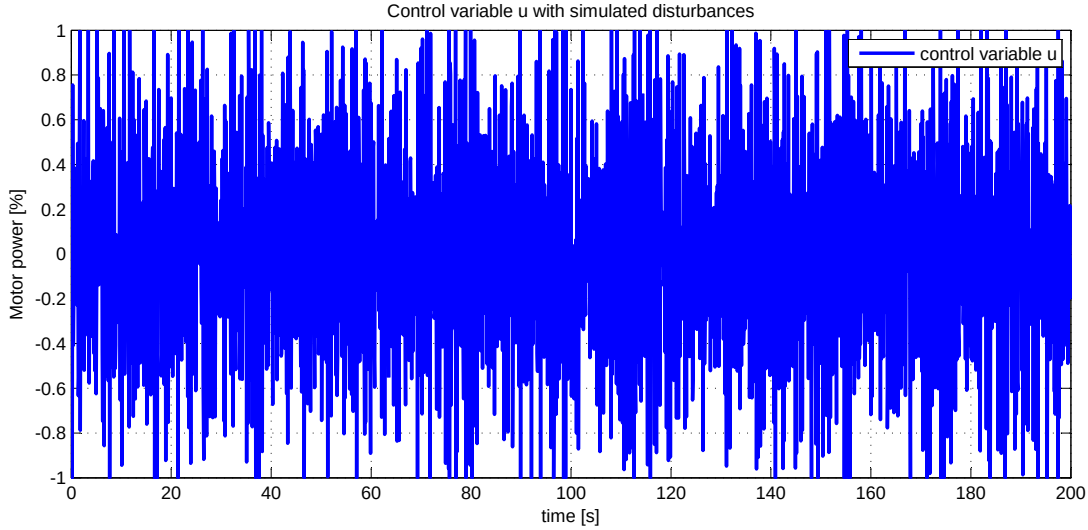

Figure 4.12: Control variable u with simulated disturbances through adding white noise to the depth value and a gain to the estimated volume.

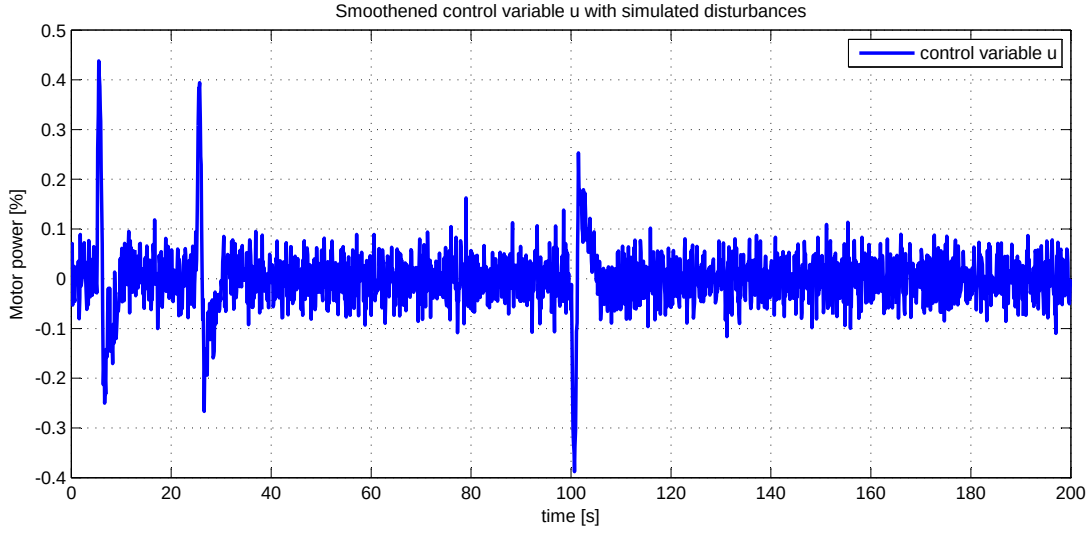

Figure 4.13: Control variable u with simulated disturbances smoothened by a PT1 term

5 Hardware

5.1 Platform

The objective for the platform was to make it as easy and cheap as possible. Therefore, the base for the complete robot is a remote controlled toy submarine. All the electronic has been removed and replaced with the microprocessor board. The original submarine was very small, so the top part has been replaced with a rapid prototyped cover with much more space for all the electronic to fit in. Using a prefabricated platform has several advantages and disadvantages. On the one hand developing the complete submarine with casing, drives, steering systems and getting all of it water tight is very costly. On the other hand, there are always compromises to be made. In this case, the controllability of the depth compressor is such a compromise. It would be much easier and more accurate to control the depth if the compressor could be controlled in a more accurate way. Like having a stepper motor, servo motor or a feedback at which position the compressor is at the moment. Adding these feedback sensor is prevented by an other compromise. Although the platform has already been enlarged, it is still very tight.

5.1.1 Experimentation Board

The Board is equipped with two ARM Cortex microprocessors, running at 50 MHz. For communication there are 4 Bluelight - modules and a RF - module. The first controller is responsible for two of the Bluelight - modules and the electrical sense system. The second one is dedicated for controlling the motors and the rest of the periphery. For the depth feedback, there is a MS5803 pressure sensor connected via I^2C. For further projects, the board is also equipped with a compass and an accelerometer.

5.1.2 Feedback Sensor

For the feedback, a MS5803 Sensor is used. This is a very small pressure sensor, which is mainly used for watches, or height control for model helicopters. It is able to measure the height with a resolution of up to 10 centimeters. Because the density of water is thousand times higher than the density of air, the resolution under water is theoretical $0,1$mm. With this sensor, it is possible to get a very accurate depth feedback, up to 3 meters total depth. The Sensor is connected via the I^2C Bus.

5.2 Implementation

All the programming is made in C with the programming environment μVision3.

5.2.1 I^{2}C communication

The pressure sensor MS5803 offers two types of communication, the SPI ans I^{2}C Bus. I^{2}C has been chosen, because it is easy to use and there are only two strands necessary. To retrieve data over the I^{2}C bus, there are usually two steps required. First the controller has to send a command that contains the address of the receiver and a command that initiates a reaction of the receiver. Then the sender switches to receive mode and waits for the answer. In the given case, the controller sends the command to initiate the conversion of the pressure and waits for the sensor to reply. The commands for the sensor are always 8 bit long, while the response is between 16 bit (constants) and 32bit (pressure and temperature values). Therefore 2 methods are programmed, one to send an 8 bit word to the sensor and one to receive n- bytes from the sensor.

5.2.2 Temperature and pressure conversion

To get the pressure data from the sensor, there are several steps to be made. The pressure value from the sensor has to be calculated using six 16 bit constants. These constants are stored in the internal PROM of the sensor. The constants are elected when the sensor is initiated. To retrieve the constants, the controller sends a command that contains the address of the desired constant and waits until the sensor returns the value. This step is repeated until all constants are retrieved. These constants are static and are only retrieved when the sensor is initiated.
To retrieve an actual pressure or temperature value, the controller sends a command including the desired variable and the sampling rate. The sensor is able to measure temperature and pressure in 5 different accuracies, between 256 and 4096 samples. A higher sample number means longer time until the value is available. After the command is sent, the controller has to wait a predefined time, depending on the sampling count, between 0.6 and 9 milliseconds. Then a command is sent that instructs the sensor to return the converted value. The received values are only raw data. To get the real pressure values, there are some calculations to be made.
For the temperature calculation, it is necessary to compute the difference between the actual and the reference temperature.

$$\begin{aligned} dT &= D2 - C_5 * 2^8 \\ TEMP &= 2000 + dT * \frac{C_6}{2^8} \end{aligned} \tag{5.1}$$

With: [dat10]

$$
\begin{aligned}
dT &= \text{Difference between actual and reference temperature} \\
TEMP &= \text{Actual Temperature} \\
D2 &= \text{Digital temperature value} \\
C_5 &= \text{Reference temperature (sensor constant)} \\
C_6 &= \text{Temperature coefficient (sensor constant)}
\end{aligned}
$$

The pressure itself depends on the temperature, because the temperature is influencing the measure electronic. This disturbance is eliminated through calculating the temperature compensated pressure.
To calculate the pressure, it is necessary to calculate the offset at the actual temperature , the sensitivity at the actual temperature and the compensated pressure.

$$
\begin{aligned}
OFF &= C_2 * 2^{16} + \frac{C_4 * dT}{2^7} \\
SENS &= C_1 * 2^{15} + \frac{C_3 * dT}{2^8} \\
P &= \frac{\frac{D1 * SENS}{2^{21}} - OFF}{2^{15}}
\end{aligned}
\tag{5.2}
$$

With: [dat10]

$$
\begin{aligned}
OFF &= \text{Offset at actual temperature} \\
SENS &= \text{Sensitivity at actual temperature} \\
P &= \text{Temperature compensated pressure} \\
D1 &= \text{Digital pressure value} \\
C_1 &= \text{Pressure sensitivity (sensor constant)} \\
C_2 &= \text{Pressure offset (sensor constant)} \\
C_3 &= \text{Temperature coefficient of pressure sensitivity (sensor constant)} \\
C_4 &= \text{Temperature coefficient of pressure offset (sensor constant)}
\end{aligned}
$$

The now calculated pressure is an integer value. These pressure value is used as feedback for the depth control.

5.2.3 Timing

There are several tasks inside the program that require an accurate timing. Therefore, one of the three internal 32 - bit counters is used as an timer, to generate an

interrupt every millisecond. This interrupt routine is used to increment the timing variables. The interrupt handler could also trigger the methods, like the depth controller directly. But this causes several problems, because the interrupt can only be triggered if the method is completed and one run of the depth control method takes more then one millisecond. Therefore the interrupt handler only increments variables and sets bits that trigger the methods later. Figure 5.1 shows the flow chart of the interrupt handler
The first variable is the system counter. This counter is incremented, all the time the controller is running. It is mainly used to trigger the command evaluation (Sec. 5.2.4).
The variable msDelay is used to delay the controller for a given time. To do so the method waitMilli(int) initiates the variable with the delay time and waits until it gets zero again. This method is used for example, to delay the controller while the sensor is converting the data.
The query over the motor direction increases or decreases the compressor position while the motor is running to be able to make assumptions about the actual position of the compressor.
The last part of the interrupt handler creates a cycle of 10Hz for the depth controller. The depth controller works with discretized states and needs to be called in a constant cycle. The variable pidTakt is increases until it reaches 100, then the flag pidGo is set to one. With this flag set, the main method triggers the depth control.

5.2.4 Communication

A communication system is essential for debugging and sending commands to the robot. There would be three possible ways of establishing a connection. The first possibility would be to use a wired UART interface. This would be very fast but the wires could distort the results because of the additional weight. The second possibility is to use the RF - receiver. But the given test platform isn't equipped with RF. So the communication is realized with Bluelight.
Bluelight is a simple module that can establish a UART connection with up to 9600 baud. It uses two LEDs for sending data and one receiver. The range of the Bluelight module is very limited. Figure 5.2 shows the Bluelight module that is used on the computer side. The LEDs on the left and right are for sending and the receiver is in the middle. The Platform is equipped with 4 Bluelight systems, to communicate with other robots around it. Only one of them is used here.

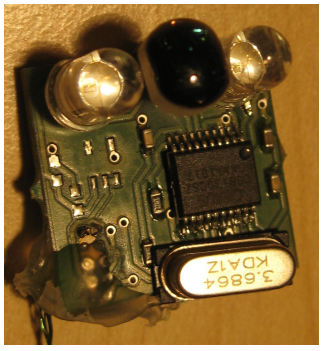

Figure 5.2: Bluelight module

The commands for controlling the robot are shown in table 5.1. All commands

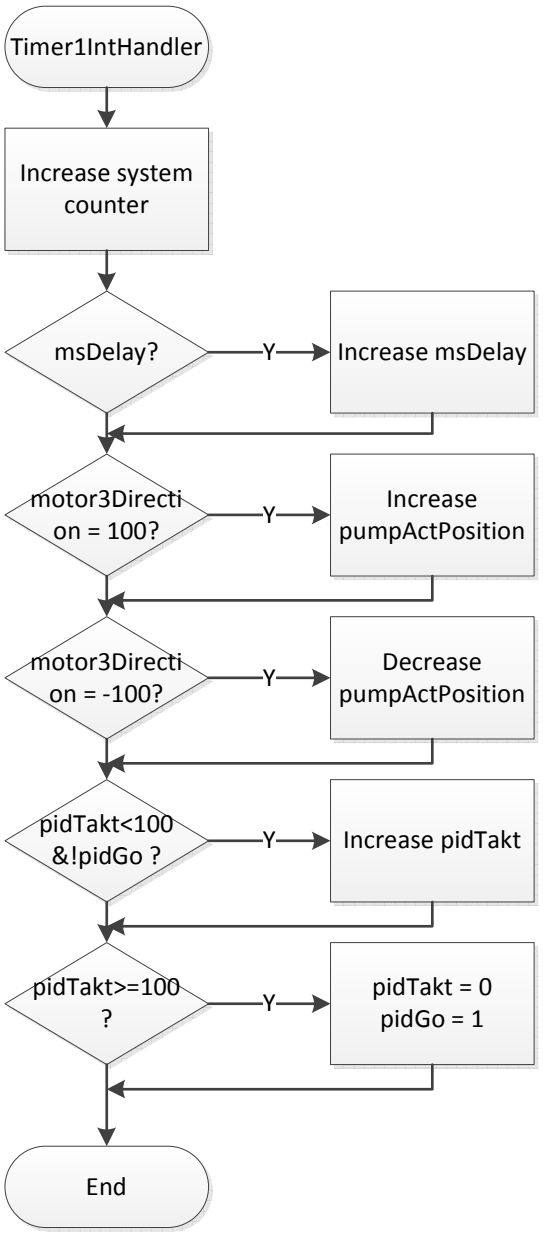

Figure 5.1: Flowchart of the Timer1IntHandler

begin with a # to indicate that a command is following. The main method checks if there are any new commands. Experiments showed, that there is a problem, if the check is done at every pass, so the robot checks for commands once a second. This doesn't mean a loss of commands, because the received commands are written in the UART buffer.
The first three commands are for controlling the robot manually.
Configuring the compressor means, that the compressor is pulled in, until the end switch is reached. Then it is pushed out again, up to the other end switch, while the controller is counting the time. This time is used as indicator for the compressor position.
Commands four and five return the actual depth or position of the compressor without initiating any moving reaction.
Command six is executed before a dive is initiated, while the robot is on the water surface. The robot stores the actual pressure value as offset, so the depth can be calculated to the water surface.
The most important command is the dive command. A dive is initiated by sending the desired depth in centimeters. To indicate, that the following command contains depth informations, the # is followed by a $.
Commands can be sent by every terminal program via serial - port, or by using the Matlab program that is programmed especially for that purpose.

Command	Meaning
# 0	Stop the compressor
# 1	Compressor push
# 2	Compressor pull
# 3	Configure the compressor
# 4	Return the actual position of the compressor
# 5	Return the actual depth
# 6	Take the actual pressure as zero position
# $ XXX	Go to specified depth in millimeters

Table 5.1: Commands for controlling the robot

5.2.5 Compressor feedback

Unfortunately, there is no sensor that measures the actual position of the compressor in an absolute or relative way. There are only two limit switches on both ends. These switches are triggering an interrupt that stops the motor, to prevent the motor from getting damaged. For the depth control, it is necessary to know the volume of the robot as accurate as possible. So the actual position of the compressor has to be estimated. This is done with the help of the timer handler, that is executed every

millisecond. When the compressor motor is running, the handler adds or subtracts a value, indicated by the speed of the motor, to the actual compressor position. This method is very inaccurate, because the motor takes some time to spin up and isn't linear because of the very imprecise transmission. To improve the accuracy, the compressor position is reseted, once the compressor hits a limiting switch, where the exact volume is known.

5.2.6 Depth control

The depth control is realized by a discrete controller. A time continuous controller can't be implemented into a digital system. Time discrete integrators and differentiators can be represented by simple equations.

Integrator:

$$\begin{aligned} e_{sum} &= e_{sum} + e \\ I &= T_a * eSum \end{aligned} \tag{5.3}$$

With:

$$\begin{aligned} e_{sum} &= \text{Sum of the error} \\ e &= \text{deviation between desired and actual position} \\ I &= \text{Integral value} \\ T_a &= \text{Time constant} \end{aligned}$$

Differentiator:

$$\begin{aligned} D &= \frac{1}{T_a} * (e - e_{old}) \\ e_{old} &= e \end{aligned} \tag{5.4}$$

With:

$$\begin{aligned} e_{old} &= \text{Last error } e \\ D &= \text{Differential value} \end{aligned}$$

With these equations, the rest of the observer can be constructed nearly directly out of the Simulink plan

$$\begin{aligned} x_2 &= \frac{1}{T_a} * (e - e_{old}) \\ \dot{x_2} &= k_1 * x_3 - k_2 * x_2 * |x_2| \end{aligned} \tag{5.5}$$

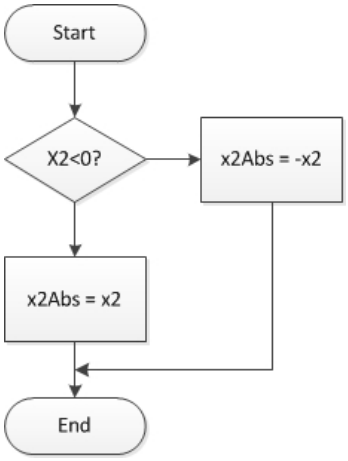

Figure 5.3: Calculation of $|x_2|$

Only the absolute value of x_2 is calculated by an if - prompt shown in figure 5.3. The estimated value for x_3 is calculated with the actual compressor position.

The control variable v is calculated through a simple equation, with the help of the previous estimated states. This equation equals the Simulink structure in Figure 3.8.

$$v = desiredDepth - (T_a * e_{sum} + 4 * actDepth + 6 * x_2 + 4 * \dot{x}_2) \tag{5.6}$$

Finally there is only the compensation of the nonlinearity inside the control path missing. This is done with the equation, visible in Figure 3.6

$$u_1 = \frac{1}{k_1 * k} * (v + 2 * {k_2}^2 * {x_2}^3 - 2 * k_1 * k_2 * |x_2| * x_3) \tag{5.7}$$

The motor signal has to be smoothened. This is done by using a PT1 term. The time continuous term is shown in Eq. 4.2. To implement this term, it has to be discretized. This can be done by using the Matlab method c2d. This method needs the system and the sample time (in this case: 0.1). There are several methods to discretize the system, zero order hold is used here.

$$u = \frac{0.2212}{u_1 - 0.7788} \tag{5.8}$$

The now calculated control variable is used to control the speed of the compressor motor by PWM. A u value of 1 means 100% motor power upwards, a value of -1 means 100% in the other direction. The motor is controlled by sending integer values between -100 and 100 to the motor control method. It also needs a PWM signal of at least 75% before it starts running. Therefore it is necessary to add an

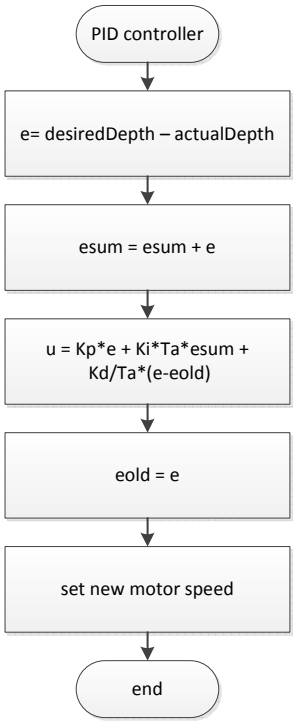

Figure 5.4: Flowchart of the PID controller

offset to the control variable. This is done with the following equation:

$$motorSpeed = \begin{cases} u * 25 - 75 & u < 0 \\ u * 25 + 75 & u > 0 \\ 0 & u = 0 \end{cases} \tag{5.9}$$

5.3 Implementation of the PID controller

To be able to compare the results, the PID controller is also implemented. The PID controller is not depending on estimated states. It calculates the control variable only with the measured pressure value (x_1). Therefore the controller at least stabilizes the system.
The implementation is easier than the state feedback controller. The control variable is calculated through three simple equations.

$$
\begin{aligned}
P_{term} &= K_P * e \\
I_{term} &= K_I * T_a * e_{sum} \\
D_{term} &= \frac{K_D}{T_a} * (e - e_{old}) \\
u &= P_{term} + I_{term} + D_{term}
\end{aligned}
\tag{5.10}
$$

The u value is then converted like described in Eq. 5.9 and is sent to the motor control method. The flowchart of the PID controller is shown in Fig 5.4.

6 Experiments

6.1 Setup

The tests are made in a large aquarium with a dimension of 2x1.4x0.6m. This allows to test in several depths and without large disturbances. All test are made at one side of the aquarium, to make sure the data from the robot can be received. The blue light module is placed outside the aquarium to ensure a good connection. For transmitting, the robot is using the Bluelight module on the front. The robot is sending the actual pressure value and the behavior is plotted with a Matlab program.

6.2 Holding depth with the state feedback controller

Because the state feedback controller showed the best behavior in the simulation, it is tested first. The tests with the real platform shows the difference between the simulation and the reality. Although the simulation is looking very good, the real system is behaving completely differently. The controller doesn't stabilize the system. Analyzing the estimated states showed that the states are very inaccurate. The compressor motor is switching very often. This leads to a very inaccurate estimated volume, because changing the direction of the compressor always takes some time of idle that would be very complicated to model. Also the velocity values are inaccurate. Although the noise on the pressure value is very small, through the derivation the velocity value gets useless. A state feedback controller is very difficult to use without reliable estimated states.

6.3 Holding depth with the PID-controller

The first test for the PID-controller is to keep a constant depth of 20 centimeters. The behavior of the platform is logged for 200 seconds and plotted. This plot is shown in Fig. 6.1. The figure shows, that the system is stable and not collapsing. But the control accuracy is not very good. There are two overlaid oscillations. One of them has a period of about 200 seconds and a peak of approximately 10 centimeters. The origin of this oscillation is not clear, it could be caused by the integrator. The second oscillation is faster, it has a period of only 20 seconds but a peak - peak distance of up to 20 centimeters. For this vibration, the origin is probably the linearization.

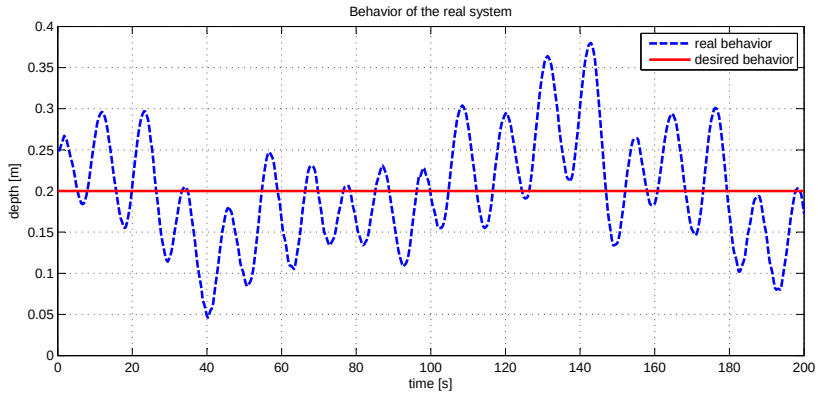

Figure 6.1: Test result with the PID controlled system trying to hold a constant depth of 0.2m

This swinging was already visible in the simulation, but with a smaller peak that was explained by the linearization of the system. The swinging is now boosted because of the bad controllability of the motor. With both vibrations combined, the system is swinging for up to 20 centimeters around the desired position. All tries to improve the accuracy by tuning the control parameters show no enhancements.

6.4 Step response with the PID-controller

In the second test, the robot is keeping a constant depth of 20 centimeters and gets a change of the desired depth to 30 centimeters just after the recording is started. This test result is visible in Fig. 6.2. The Figure shows, that the system is reacting on the change of the desired depth. It is reaching the new desired depth and swinging around the new position. Because of the high inertia of the system, this task is very slow. Also in this case, there are the two overlaid osillations from Sec. 6.3.

6.5 Results

The experiments show, that a very simple toy can be controlled by a PID-controller, but the accuracy isn't very good. The system is able to keep a specified depth and to react on a depth change. To reach the new desired depth it takes some time, because of the high inertia of the system. The accuracy could be improved by finding a way to estimate the states better or through modifying the platform.

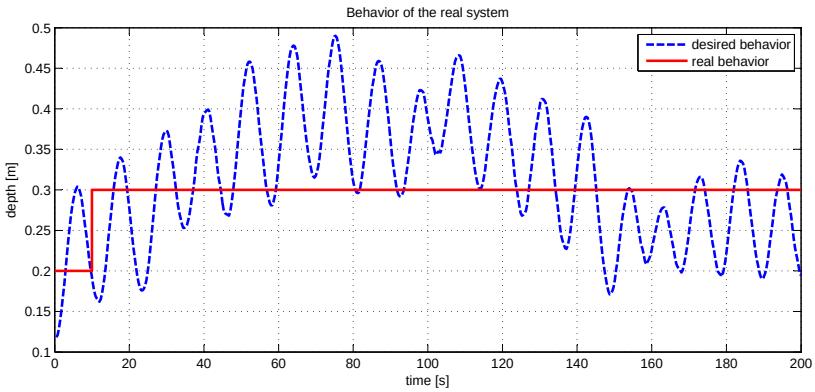

Figure 6.2: Test result with the PID controlled system performing a depth change from 0.2m to 0.3m

7 Conclusion

7.1 Summary

Summarized, the thesis shows the steps from a given platform with predefined sensors and actors to the implemented depth controller.
To reach this goal, the system has been modeled by deriving the equations of motion that describe the system. These equations have been modeled in Matlab Simulink to be able to simulate different control structures and parameters without using the real system.
Two different types of controllers were tested. The first one was a linear PID - controller. The simulation showed, that the system can be stabilized but the permanent control deviation can't be eliminated. The reason for this behavior was the nonlinearity of the control path. To improve the accuracy, the system was exact linearized and controlled by a state feedback controller. For this, it was necessary to estimated all states of the system. The results in the simulation were promising, the system reached the desired position in a very short time and kept it perfectly. Both controllers were implemented in C on the microprocessor board. To send commands to the board and to receive validation informations, a transmission system was implemented. This has been done by using the Bluelight modules, a system that implements an UART interface by the use of LEDs.
The complete system was then tested inside the aquarium.
At first, the state feedback controller was tested, because of the better behavior in the simulation. The results were the complete opposite of the simulation, the state feedback controller was not able to stabilize the system. This can be justified through the combination of inaccurate estimated states and inaccurate motor control. The PID controller showed a better behavior, it was stabilizing the system but the accuracy was not very good. The system was swinging around the desired position with two overlaid oscillations. One of these oscillations was caused by the nonlinearity of the system that has been linearized to be able to use a linear controller. The origin of the second oscillation could be caused by the integrator.

7.2 Future work

A major improvement would be a feedback sensor for the volume of the robot, or the ability to control the compressor in a more accurate way.

For a volume sensor, there are two strategies. The motor of the robot could be equipped with a incremental position encoder that triggers a signal on every rotation of the motor. The controller could count these signals and calculate the compressor position relative. Another method would be to attach a variable resistor to the compressor. The controller could then calculate the position absolute by using an AD converter.
Another method to increase the accuracy of the compressor would be to use a different kind of motor. By using a stepper or servo motor, the compressor could be controlled more accurately. Therefore it would be possible for the controller to adjust the desired volume. This would also reduce the complexity of the controller, because the volume could be controlled directly and not only the motor power.

Bibliography

[Ada09] J. Adamy. *Nichtlineare Regelungen*. Springer, 2009.

[ALL92] J. Ans, E. Lax, and M.D. Lechner. *Taschenbuch fuer Chemiker und Physiker: Physikalisch-chemische Daten*. Taschenbuch fuer Chemiker und Physiker. Springer, 1992.

[BKL05] J. Berber, H. Kacher, and R. Langer. *Physik in Formeln und Tabellen*. Teubner, 2005.

[Boe09] L. Boeswirth. *Technische Stroemungslehre: Lehr- und Uebungsbuch*. Viewegs Fachbuecher der Technik. Vieweg+Teubner Verlag, 2009.

[CoC13] CoCoRo. *Collective Cognitive robots FP7*. 2011 - 2013.

[dat10] *Data sheet MS5803-01BA miniature variometer module*. 2010.

[Ebe11] Prof. Dr.-Ing. Christian Ebenbauer. *Einfuehrung in die Regelungstechnik Zusammenfassung: Begriffe und Formeln*. 2011.

[FH08] Torsten Bertram Frank Hoffmann. *Nichtlineare Regelung*. 2008.

[Joh12] Anathol Johansen. *Tödliche Tiefen*. http://www.welt.de/print/die_welt/wissen/article13826422/Toedliche-Tiefen.html, 21.01.2012.

[J.Y00] J.Yuh. *Design and Control of Autonomous Underwater Robots: A Survey*. 2000.

[Lun07] Jan Lunze. *Regelungstechnik I*, volume 6. Auflage. 2007.

[PS08] W. Plaßmann and D. Schulz. *Handbuch Elektrotechnik: Grundlagen und Anwendungen für Elektrotechniker*. Vieweg+Teubner Verlag, 2008.

[RC90] Fortis A. Papoulias Anthony J. Healey Roberto Christ. *Adaptive Sliding Mode Control of Autonomous Underwater Vehicle in the Dive Plane*. 1990.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Matthias van de Weyer)