Bachelorarbeit

Linear State Space Control of Tethered Flying Objects

Thilo Bronnenmeyer

23.09.2014

Albert-Ludwigs-Universität Freiburg im Breisgau
Technische Fakultät
Institut für Mikrosystemtechnik

Bearbeitungszeitraum
23. 06. 2014 – 23. 09. 2014

Gutachter
Prof. Dr. Moritz Diehl
Prof. Dr. Leonhard Reindl

Betreuer
Greg Horn, M. Sc.

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Unterschrift	Ort, Datum

Contents

Kurzzusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der linearen Regelung im Zustandsraum. Angewandt wird die Regelung auf ein Karussell, das durch seine Rotation einen mit einer Leine am Karussell befestigten Ball anhebt — etwa wie bei einem Kettenkraussell. Zur Regelung wird das System zunächst linearisiert. Dann wird ein Discrete Quadratic Regulator verwendet, um am Punkt der Linearisierung eine optimale Regelungsvorschrift abzuleiten. Zur Komplementierung des Zustandsreglers wird ein Kalman Filter zur Zustandsschätzung eingeführt. Dazu kommen noch einige Erweiterungen, um den Zustandsregler robuster gegenüber Modellierungsfehlern zu machen. Nachdem der Regler in Simulationen hinreichend getestet wurde, wird er im tatsächlichen Versuchsaufbau implementiert und getestet. Da dieser letzte Schritt nicht in den zeitlichen Rahmen der Arbeit passt, endet sie stattdessen mit einer Skizze der geplanten Experimente.

Abstract

This thesis concers itself with linear state space control. The control in question is applied to a carousel that lifts a ball tethered to the carousel arm through rotation — just like a chairoplane. The system is first linearized. Then a Discrete Quadratic Regulator is used to derive a optimal feedback law at the point of linearization. To complement the state space controller, one introduces a Kalman filter for state estimation. Additionally there are some extensions in order to make the controller more robust concerning modeling errors. After the controller is tested sufficiently in simulation, it will be implemented and tested on the actual experimental setup. Since this last step could not be completed in the time frame of this thesis, there will be an outline of the planned experiments instead.

Thanks and Acknowledgements

Thanks to Prof. Dr. Moritz Diehl, Prof. Dr. Leonhard Reindl and all members of the chair of System Theory and Optimization for their helpful attitude and warm work atmosphere. Thanks to Heike Dietel for the productive cooperation when deriving the carousel model. I especially express my gratitude to Dr. Andrew Wagner and my supervisor Greg Horn for their support and helpful feedback.

List of Figures

List of Tables

1 Introduction

The public's increasing awareness concerning the ecological consequences of the last 250 years' industrial expansion has peaked in the time of catastrophes such as the accidents in the Fukushima nuclear power plant in Japan 2011 or Chernobyl 1987 in Ukraine. These events have become a symbol for the widespread opinion that nuclear power does not have the capacity be a clean alternative to coal and oil as the main power source for a growing industry. Due to the inadequacies of nuclear power, renewable energy sources such as wind, solar energy or water have become more attractive to both consumers and producers.

Conventional wind power has one big limitation concerning its cost–effectiveness both in terms of economical and ecological factors: to harness the wind power of stronger and more consistent high altitude winds or to just increase the area of a wind turbine, the size of the supporting towers also has to be increased — up to the point where the size of those towers is not justified by the amount of power produced by the turbine anymore. This limitation seems absurd considering that the part of the wind turbine that actually produces the power is up high in the air while the limiting part is just there to attach and connect the turbine to the ground and absorb its torque.

Instead of trying to solve this problem by improving an inherently limited concept, one has to think about clever alternatives. One such alternative makes it possible to both harvest the stronger high altitude winds *and* decrease the amount of space and material needed for buildings on the ground. The idea is to harvest the wind energy with airborne kites that are tethered to a generator at the ground-station. The kite uses the force of the wind to unroll the tether that is wound up on the generator's coil, thus generating power (see Fig. 1). In order to effectively produce energy with this setup, the kite must fly so called *pumping cycles* that are visualized in Fig. 2. First the kite spirals outwards, rolling out the tether and thus generating energy. Since the tether has a finite length, the setup must reel in the tether after a certain time, thus consuming energy. The cycle then begins anew. It is obvious that in order to effectively generate power, the setup has to generate more power in the first part of the cycle than it loses in the second. The ratio of energy created to energy consumed is very much dependent on the trajectory of the kite with respect to the wind direction.

To fully realize the potential of airborne wind energy, the kites trajectory has to be automated. This makes it a control problem. The task of controlling the kites has already been tackled with various optimal control approaches like Nonlinear Model Predictive Control (NMPC) [18, 17, 4, 9],

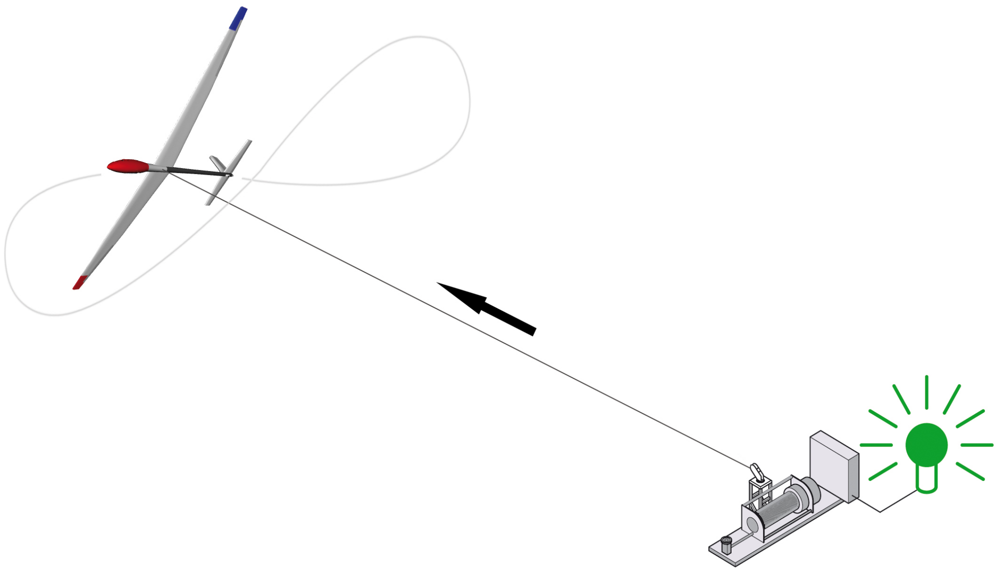
Figure 1: Airborne kite tethered to a generator coil at the ground station [1].

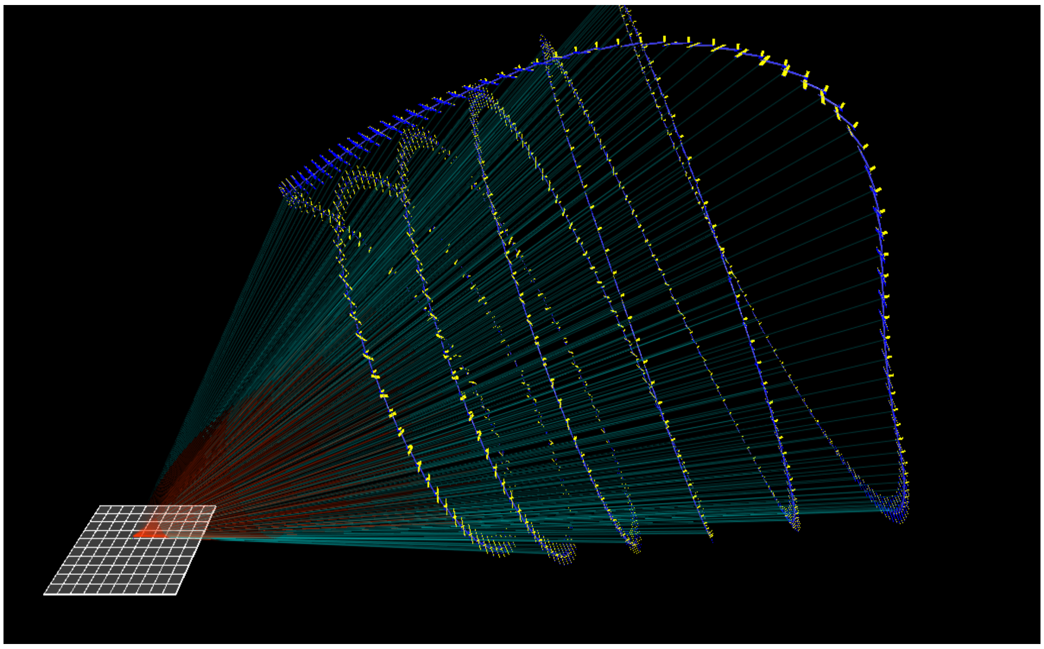
Figure 2: Kite flying a pumping cycle consisting of power generating and consuming part. Illustration by Greg Horn.

Moving Horizon Estimation (MHE) [18, 6] and Robust Optimal Control [15]. While most papers concern themselves with the question of how to optimally control the kites once they are in the air, it is also important to think about how to get them airborne. Since the kites are already tethered to a ground-station, it seems natural to use this connection for the start-up process. To do so, the kite can be rotated by a carousel spinning around its central axis to lift it into the air [7, 8]. This setup has the additional advantage that the kite does not need any on-board means of propulsion, thus making it cheaper and easier to produce.

As a first step toward experimental kite start-up, the team of the Highwind project has constructed a simplified experimental setup. Instead of a glider, a ball is attached to the carousel by means of a carbon fiber stick (that replaces the tether). The goal of this experimental setup is to control the height (or more precisely, the elevation angle) of the ball via the rotational speed of the carousel. The topic of this thesis is how to achieve this goal with the means of Linear State Space Control. The thesis content is outlined as follows:

In Section 2 the problem is first approached by modeling the ball and carousel and understanding the behavior of the open loop system. This includes a linearization of the system in both continuous and discrete time and a closer look at the eigenvalues and eigenvectors of the linearized system. In Section 3 the loop is then closed in simulation using a Linear Quadratic Regulator (LQR) complemented with a Kalman filter for state estimation. After showing that the LQR's performance surpasses that of a PID controller, the control setup is refined some more to make it more robust in Section 4. A pseudo-force is added to the model used for state estimation in order to account for potential mismodeling and thus decrease the steady state error. This is followed by a discussion of a special kind of mismodeling: using the linearized system equations for state estimation instead of the non-linear ones.

All computations, simulations and plots are done with `MATLAB 2013b` unless otherwise specified. The equations of motion for the carousel are derived with the `Symbolic Math Toolbox`. For control work the `Control System Toolbox` and `Optimization Toolbox` are used. The `MATLAB` function `ode15s` is employed to integrate the system equations. Every other function and script written for the purpose of this thesis can be looked up in the Appendix A. The experimental data and `MATLAB` files can be downloaded from `https://github.com/thilobro/state_space_control_thesis`.

2 Modeling the Carousel

As a very first step the experimental setup of the carousel is discussed. This is followed by a description of how the carousel is modeled and subsequently linearized. After that an investigation of the carousel's open loop behavior follows.

2.1 Experimental Setup of the Carousel

The carousel that has been constructed by Highwind is situated at the IMTEK of the University of Freiburg. It is approximately 4 m high and made out of aluminum (see Fig. 3). It is mounted on a steel trailer for transportation and has an arm span r_A of 2 m (4 m in diameter). On one end of the arm the line angle sensor is positioned. The sensor is in turn connected to a plastic ball by a carbon fiber rod with length $l_T = 1.82$ m (see Fig. 4). Also attached to the end of the carousel arm is a cradle to ensure soft and controlled landings of the ball (and thus lessen the strain on the carbon fiber

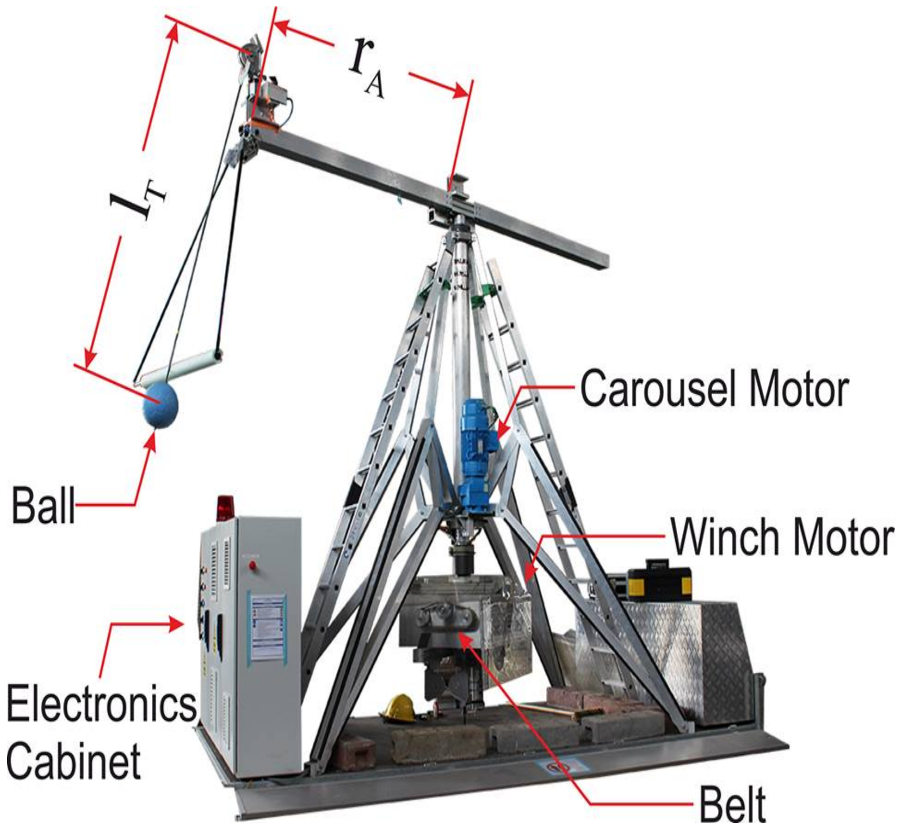

Figure 3: Experimental setup of the carousel.

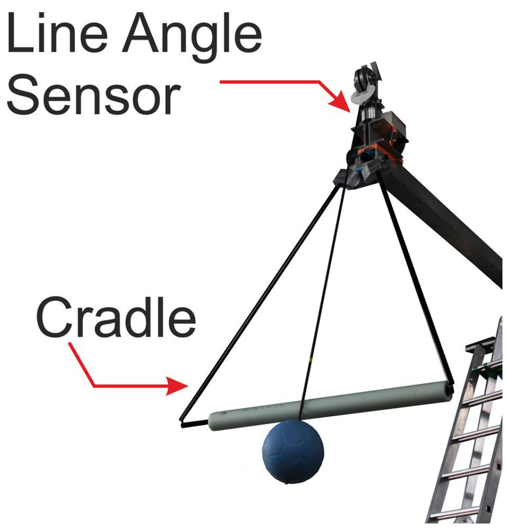

Figure 4: Close up of the carousel's line angle sensor, cradle and ball.

stick). The carousel is rotated by a induction motor (with a gear ratio R_G of 22.29) which transfers its torque to the carousel by means of a rubber belt. The motor torque (before the gear ratio is applied) should not exceed 20 N m. It is also possible to attach the ball to a tether instead of the carbon fiber rod. This tether can then be extended or retracted with a winch motor. Even though it is planned to eventually replace the rod with a tether, the winch motor remains unused during the course of this thesis.

The carousel motor is controlled by a desktop PC stored in the electronics cabinet. Also situated in the cabinet are the motor's drives and brakes system. To conduct experiments, one generally does not use the PC directly but via SSH. The line angle sensor measures the angular displacement of the carbon fiber rod in two dimensions and transmits the data online to the PC. In addition the motor measures its own rotational speed. Safety measures include a net around the whole carousel, two flashing red lights when the carousel is operating and several emergency stop buttons.

2.2 Basic Properties of the Model

The model of the carousel is derived using the Lagrange formalism.[1] The generalized coordinates q_i are

1. the angle of the motor δ_M

[1]The model has been derived in cooperation with Heike Dietl while writing her bachelor thesis 'Nonlinear Dynamic System Models of Tethered Flight' [3].

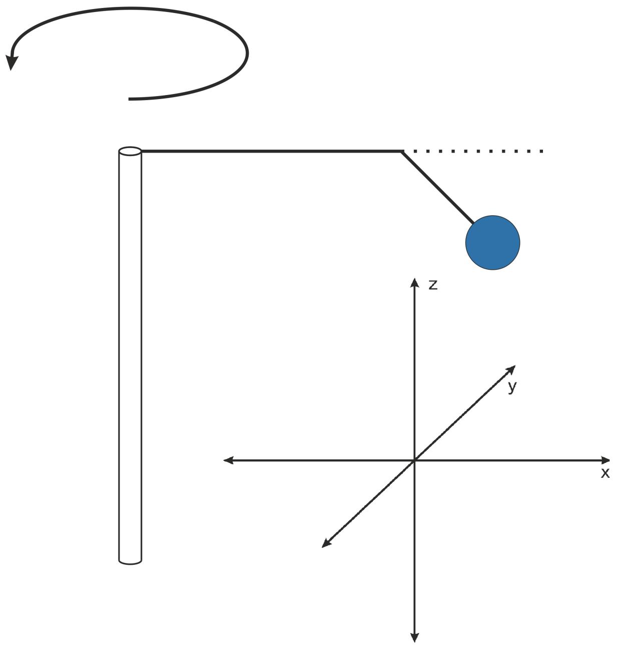

Figure 5: Coordinate system of the model. The directions of the axis are North-East-Down.

2. the angle of the carousel arm δ_{A}
3. the elevation angle α
4. the azimuth angle β

Thus

$$q = \begin{bmatrix} \delta_{\mathrm{M}} \\ \delta_{\mathrm{A}} \\ \alpha \\ \beta \end{bmatrix}$$

Fig. 5 shows how the coordinate system is chosen for the model. The North-East-Down convention is used to assign the directions of the axes. The definitions of the angles within this coordinate system are shown in Fig. 6.

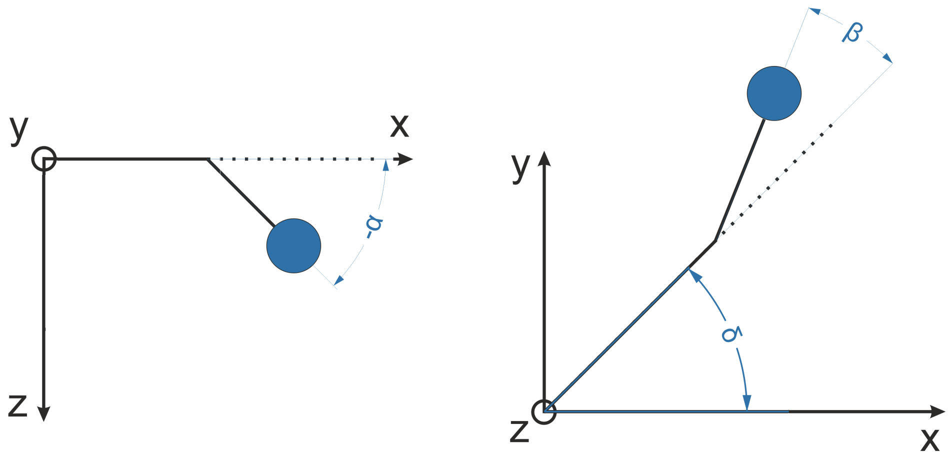

Figure 6: Angles of the model.

One assumes further that the position of the line angle sensor p_{LA} in the absolute coordinates is given by

$$p_{\mathrm{LA}} = r_{\mathrm{A}} \cdot \begin{bmatrix} \sin(\delta_{\mathrm{A}}) \\ \cos(\delta_{\mathrm{A}}) \\ 0 \end{bmatrix}$$

The position of the ball p_{BA} is then given by

$$p_{\mathrm{BA}} = p_{\mathrm{LA}} + l_{\mathrm{T}} \cdot \begin{bmatrix} \cos(\alpha)\sin(\delta_{\mathrm{A}} + \beta) \\ \cos(\alpha)\cos(\delta_{\mathrm{A}} + \beta) \\ -\sin(\alpha) \end{bmatrix}$$

For the kinetic energy E_{K} of the system, the following terms are considered

1. the translatory motion of the point mass of the ball m_{BA}
2. the rotational motion of the carousel arm around its z axis in δ_{A} direction with moment of inertia I_{A}
3. the rotational motion of the carousel motor around its z axis in δ_{M} direction with moment of inertia I_{M}
4. the rotational motion of the carbon fiber stick around the two axis at its end point perpendicular to the stick in α and β direction with moment of inertia I_{T}

The expression for the kinetic energy is

$$\begin{aligned} E_{\mathrm{K}} = &\frac{1}{2} m_{\mathrm{BA}} \dot{p}_{\mathrm{BA}}^2 \\ &+ \frac{1}{2} I_{\mathrm{A}} \dot{\delta}_{\mathrm{A}}^2 \\ &+ \frac{1}{2} I_{\mathrm{M}} \dot{\delta}_{\mathrm{M}} \\ &+ \frac{1}{2} I_{\mathrm{T}} (\dot{\alpha}^2 + \dot{\beta}^2) \end{aligned}$$

The term for the rotational motion of the carbon fiber stick is strictly speaking not correct, since is disregards the rotation in δ_{A} direction. The term would then change to $\frac{1}{2} I_{\mathrm{T}} (\dot{\alpha}^2 + (\dot{\beta} + \dot{\delta}_{\mathrm{A}})^2)$. However since $I_{\mathrm{A}} \gg I_{\mathrm{T}}$ this addition is disregarded.
For the potential energy E_{P} of the system, one considers the terms

1. the potential energy of m_{BA}
2. the potential energy of the belt between motor and carousel arm, that is modeled like a rotational spring with spring constant k_{BE}

The expresion for the potential energy is

$$\begin{aligned} E_{\mathrm{P}} = &m_{\mathrm{BA}} \cdot g \cdot l_{\mathrm{T}} \sin(\alpha) \\ &+ \frac{1}{2} k_{\mathrm{BE}} (\delta_{\mathrm{M}} - \delta_{\mathrm{A}})^2 \end{aligned}$$

The non-conservative generalized forces $F_i^\star$ of the system are

1. the air friction of the ball with direction opposite to $\dot{p}_{\mathrm{BA}}$
2. the dampening of the belt in both δ_{A} and δ_{M} direction (although with different signs)
3. the friction of the line angle sensor in both α and β direction
4. the friction of the carousel shaft's rotation μ_{SH} in δ_{A} direction
5. the motor torque τ_{M} in δ_{M} direction

The expression for the generalized forces is

$$F^\star = \begin{bmatrix} F^\star_{\delta_\mathrm{M}} \\ F^\star_{\delta_\mathrm{A}} \\ F^\star_\alpha \\ F^\star_\beta \end{bmatrix} = \begin{bmatrix} R_\mathrm{G} \cdot \tau_\mathrm{M} + \tau_\mathrm{BE} \\ -\tau_\mathrm{BE} + \tau_\mathrm{SH} \\ \tau_{\mathrm{LA},\alpha} \\ \tau_{\mathrm{LA},\beta} \end{bmatrix} + F^\star_\mathrm{AIR}$$

with the gear ratio of the motor R_G and

$$\begin{aligned} \tau_\mathrm{BE} &= -c_\mathrm{BE} \cdot (\dot{\delta}_\mathrm{M} - \dot{\delta}_\mathrm{A}) \\ \tau_\mathrm{SH} &= -\mu_\mathrm{SH} \cdot 1 \\ \tau_{\mathrm{LA},\alpha} &= \mu_{\mathrm{LA},\alpha} \cdot \dot{\alpha} \\ \tau_{\mathrm{LA},\beta} &= \mu_{\mathrm{LA},\beta} \cdot \dot{\beta} \\ F^\star_\mathrm{AIR} &= \underbrace{\left[\frac{dp_\mathrm{BA}}{dq}\right]^T}_{J} \cdot F_\mathrm{AIR} = J^T(-\frac{1}{2}\rho_\mathrm{AIR} A_\mathrm{BA} c_\mathrm{W} \dot{p}_\mathrm{BA} ||\dot{p}_\mathrm{BA}||_2) \end{aligned} \tag{1}$$

$F^\star_\mathrm{AIR}$ is derived by multiplying F_AIR with the transposed Jacobian J and thus transforming F_AIR into the generalized coordinates (see Eq. (1)). This can be derived with

$$\begin{aligned} x &= f(q) \\ \delta x &= \underbrace{\frac{df(q)}{dq}}_{J} \delta q \\ \delta x^T F &= \delta q^T F^\star \\ \delta q^T J^T F &= \delta q^T F^\star \\ J^T F &= F^\star \end{aligned}$$

To control the system one introduces $\dot{\delta}_\mathrm{M,SP}$ which is the setpoint for the motor speed that is given to the internal P controller of the drives with a constant k_P. The generalized motor torque is then

$$\tau_\mathrm{M} = -k_\mathrm{P}(\dot{\delta}_\mathrm{M} - \dot{\delta}_\mathrm{M,SP}) \tag{2}$$

At first, the control u for the system is the setpoint $\dot{\delta}_\mathrm{M,SP}$. This is easy to implement in the experimental setup and can be used to compare simulation results and experimental data with the script `step_response_experiment`. These comparisons are used to verify the model and choose the constants by fitting the simulation to experimental data. The result of this fitting process can be seen in Fig. 7. The plots are done by replicating a step response experiment where $\dot{\delta}_\mathrm{M,SP}$ changes from $1.54\,\mathrm{rad\,s^{-1}}$ to $1.67\,\mathrm{rad\,s^{-1}}$. One can

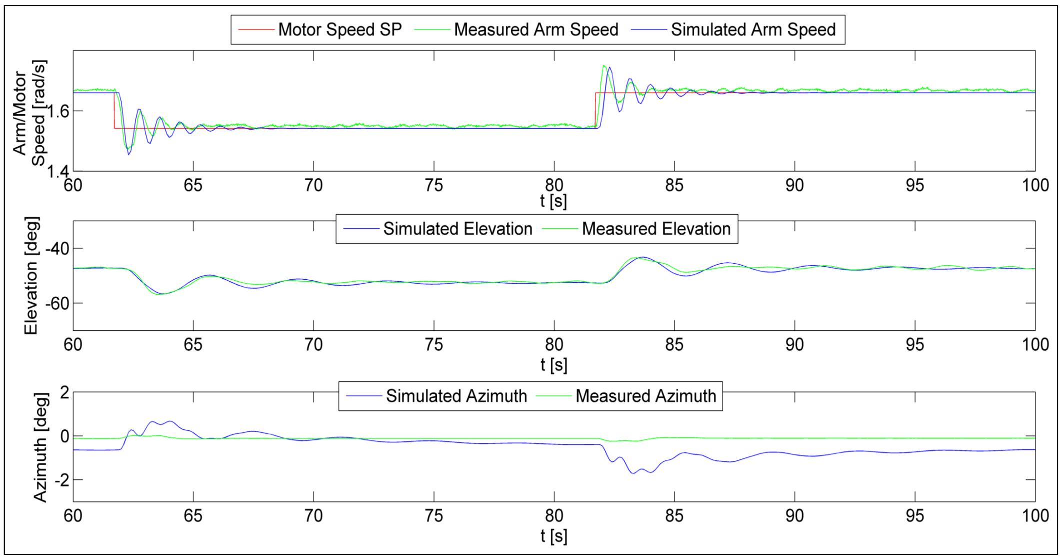

Figure 7: Step response of elevation, azimuth and arm speed for $\dot{\delta}_{\mathrm{M,SP}}$ changing from $1.54\,\mathrm{rad\,s^{-1}}$ to $1.67\,\mathrm{rad\,s^{-1}}$ and back to $1.54\,\mathrm{rad\,s^{-1}}$. Experimental data is compared to simulation data to verify the model.

see that the model is a good approximation of the real carousel setup. Thus it will be used for further simulation and experiments.

The constants that are used in the model can be looked up in Table 1. Most of the constants that could not be physically measured have been roughly estimated and then fine-tuned by fitting the model to the experimental data by hand. The choice of setting μ_{SH} to zero might be counter intuitive but can be explained as follows: in the real setup, the drives' internal PI controller takes care of the constant friction of the carousel shaft by supplying enough additional torque. Since the PI controller is modeled as a simple P controller, this is not possible in the simulation. However assuming that the friction will always be compensated by the integral term of the internal PI controller, one can just leave out both to simplify the model.

With the help of the `MATLAB` function `lagrange_formalism` one can derive the explicit equations of motion in the form of

$$\dot{x} = f(x, u)$$

Constant	Physical Meaning	Value
m_{BA}	mass of the ball	$0.57\,\mathrm{kg}$
l_{T}	length of the tether/carbon fiber stick	$1.82\,\mathrm{m}$
r_{A}	radius of the carousel arm	$2\,\mathrm{m}$
A_{BA}	effective area of the ball regarding air friction	$36 \times 10^{-4}\,\mathrm{m}^2$
ρ_{AIR}	density of the air	$1.184\,\mathrm{kg\,m}^{-3}$
c_{W}	air friction constant for a sphere	0.5
g	gravitational constant	$9.81\,\mathrm{m\,s}^{-2}$
$\mu_{\mathrm{LA},\beta}$	friction constant of the line angle sensor in β direction	$30\,\mathrm{N\,m\,rad}^{-1}$
$\mu_{\mathrm{LA},\alpha}$	friction constant of the line angle sensor in α direction	$1\,\mathrm{N\,m\,rad}^{-1}$
μ_{SH}	friction constant of the carousel shaft in δ_{A} direction	$0\,\mathrm{N\,m\,rad}^{-1}$
k_{BE}	spring constant of the motor belt	$11\,419\,\mathrm{N\,m\,rad}^{-1}$
c_{BE}	dampening constant of the motor belt	$0.1\,\mathrm{N\,m\,rad}^{-2}$
I_{A}	moment of inertia of the carousel arm	$200\,\mathrm{kg\,m}^2$
I_{M}	moment of inertia of the motor	$6.7838 \times 10^{-4}\,\mathrm{kg\,m}^2$
I_{T}	moment of inertia of the tether/carbon fiber stick	$1.5561\,\mathrm{kg\,m}^2$
k_{P}	proportional gain of the internal P controller of the drives	400
R_{G}	gear ratio of the motor	22.29

Table 1: All constants used in the model.

for

$$x(t) = \begin{bmatrix} \delta_{\mathrm{M}} \\ \delta_{\mathrm{A}} \\ \alpha \\ \beta \\ \dot{\delta}_{\mathrm{M}} \\ \dot{\delta}_{\mathrm{A}} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix}$$

To do so, `lagrange_formalism` formulates the Lagrange equation

$$L = E_{\mathrm{K}} - E_{\mathrm{P}}$$

and then solves the Euler-Lagrange equation for each generalized coordinate q_i

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = F_i^\star$$

The resulting system of implicit differential equations is then solved for the explicit $\ddot{q}$. The equations of motion are then

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix}$$

The full expression of $f(x, u)$ is too long to be usefully presented on paper, but can easily be generated with the `lagrange_formalism` script.

2.3 Linearization and Discretization of the Model

For further analysis of the model, we need to linearize it. In order to be able to change from continuous to discrete time later on, the linearization is done twice: once in discrete time and once in continuous time. While the continuous form is used for mathematical analysis of system, the discrete form is used for the `MATLAB` implementation and simulation. The x_{SS}, u_{SS}, where the difference equation is linearized, are computed numerically with `solve_steady_state_lsq` that solves the least squares problem[2]

$$\begin{aligned} &\min_{x,u} ||g(x,u)||_2^2 \\ &\text{with } g(x,u) = f(x,u) - \begin{bmatrix} \dot{\delta}_{\text{M,SS}} \\ \dot{\delta}_{\text{A,SS}} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{aligned} \tag{3}$$

for a given α_{SS}. It is assumed that for all steady states $\dot{\delta}_{\text{A,SS}} = \dot{\delta}_{\text{M,SS}}$. By looking at equation (3) one can see that not all components of $f(x, u)$ are set to zero. This is due to the fact that the interesting steady states are the ones where the carousel is moving but the ball is standing still in reference to the rotating coordinate system of the carousel. Due to the way that the motor toque τ_{M} is defined (see Eq. (2)), in every steady state there has to be a difference between $\dot{\delta}_{\text{M}}$ and $\dot{\delta}_{\text{M,SP}}$ to supply a constant torque that is equal to the negative torque of friction and air restistance. Setting $u = \dot{\delta}_{\text{M,SP}}$ means that to control x to a specified x_{SS} a constant feedforward term $u_{\text{FF}} = u_{\text{SS}} \neq \dot{\delta}_{\text{M}}$ is needed. For more advanced closed loop control with a Linear Quadratic Regulator it is more convenient to work with a setup that ensures $u_{\text{SS}} = 0$. To achieve this, the old control $\dot{\delta}_{\text{M,SP}}$ is added as a

[2]The problem is rewritten as a least squares problem because they are easy to implement in `MATLAB`.

new state while the new control is now $\ddot{\delta}_{\mathrm{M,SP}}$. This artificially complicates the system by adding another state but it does not change the controllability of the system. The $u_{\mathrm{FF}} \neq 0$ of the old setup now becomes the new state's steady state $\dot{\delta}_{\mathrm{M,SP,SS}}$.
In discrete time the linearization is

$$x_k = x_{\mathrm{SS}} + \delta x_k,\ u_k = u_{\mathrm{SS}} + \delta u_k \ \text{ and }\ y_k = y_{\mathrm{SS}} + \delta y_k \tag{4}$$

with

$$\begin{aligned} \delta x_{k+1} = &\quad A\delta x_k + B\delta u_k \\ \delta y_k = &\quad C\delta x_k + D\delta u_k \end{aligned}$$

where

$$\begin{aligned} A &= \frac{\partial x_{k+1}}{\partial x_k}(x_{\mathrm{SS}}, u_{\mathrm{SS}}), \quad B = \frac{\partial x_{k+1}}{\partial u_k}(x_{\mathrm{SS}}, u_{\mathrm{SS}}), \\ C &= \frac{\partial f_{\mathrm{SENS}}}{\partial x_k}(x_{\mathrm{SS}}, u_{\mathrm{SS}}), \quad D = \frac{\partial f_{\mathrm{SENS}}}{\partial u_k}(x_{\mathrm{SS}}, u_{\mathrm{SS}}) = 0 \end{aligned}$$

The Jacobians A, B and C are computed numerically by the `MATLAB` function `disc_syscreator` that in turn uses finite differences with `fingrad`. Since the system that is being linearized is continuous, `disc_syscreator` numerically integrates $f(x, u)$ with initial condition $[x_0, u_0] = [x_{\mathrm{SS}}, u_{\mathrm{SS}}]$ over a small timestep T_{S} to get x_{k+1}.

In continous time, the linearization is more straightforward [16, p. 26]. For the same system it is

$$x(t) = x_{\mathrm{SS}} + \delta x(t),\ u(t) = u_{\mathrm{SS}} + \delta u(t) \ \text{ and }\ y(t) = y_{\mathrm{SS}} + \delta y(t)$$

with

$$\begin{aligned} \delta f(x(t), u(t)) = \delta\dot{x}(t) = &\quad A\delta x(t) + B\delta u(t) \\ \delta y(t) = &\quad C\delta x(t) + D\delta u(t) \end{aligned}$$

where

$$\begin{aligned} A &= \frac{\partial f(x,u)}{\partial x}(x_{\mathrm{SS}}, u_{\mathrm{SS}}), \quad B = \frac{\partial f(x,u)}{\partial u}(x_{\mathrm{SS}}, u_{\mathrm{SS}}), \\ C &= \frac{\partial f_{\mathrm{SENS}}}{\partial x}(x_{\mathrm{SS}}, u_{\mathrm{SS}}), \quad D = \frac{\partial f_{\mathrm{SENS}}}{\partial u}(x_{\mathrm{SS}}, u_{\mathrm{SS}}) = 0 \end{aligned}$$

This time, the Jacobians A, B and C are computed with the `MATLAB` function `cont_syscreator` that is based on the same function for finite differences `fingrad`.

The system of ODEs that has been derived with in Section 2.2 is in continous time. All further simulations in this thesis however are in discrete time. The discretization of the system is done by calculating $x_{k+1} = f(x_k, u_k)$

by numerical integration with the MATLAB function ode15s.

2.4 Open Loop Dynamics of the Carousel

For analyzing the general open loop behavior of the system, the continuous time linearization is used as described in Section 2.3. The eigenvalues of the system can be computed with the MATLAB function eig(A). In total there are eight eigenvalues. Four of them are complex eigenvalues $\lambda_{3,4,5,6}$ with real parts smaller than zero, three real eigenvalues $\lambda_{1,2,8}$ smaller than zero and one real eigenvalue λ_7 equal to zero. The eigenvalues are shown in Table 2 and Fig. 8.

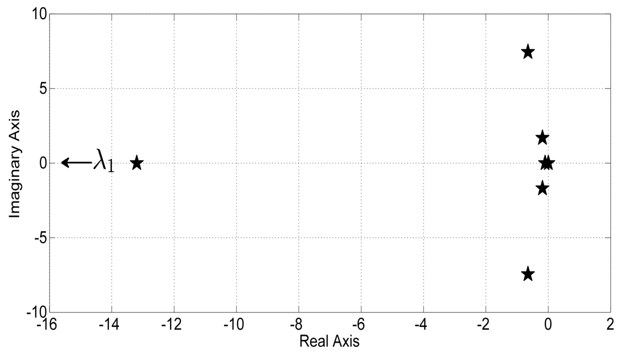

Figure 8: Eigenvalues of the open loop system plotted on the complex plane.

λ_i	$\mathrm{Re}\,\lambda_i$	$\mathrm{Im}\,\lambda_i$
λ_1	-1.31×10^7	0
λ_2	-1.32×10^1	0
λ_3	-6.53×10^{-1}	7.44
λ_4	-6.53×10^{-1}	- 7.44
λ_5	-1.88×10^{-1}	1.69
λ_6	-1.88×10^{-1}	-1.69
λ_7	0	0
λ_8	-1.04×10^{-1}	0

Table 2: Eigenvalues of the open loop system.

Of particular interest are eigenvalues λ_1 and λ_7 since one is very large and the other one zero. To facilitate further analysis, one looks at the solution of the homogeneous system

$$\delta\dot{x}(t) = A\delta x(t) = A(x(t) - x_{\mathrm{SS}}(t))$$

that can be written as a linear combination of terms of $e^{\lambda_i t}$ (see Eq. (5)) [10, p. 147f].

$$\begin{aligned} \delta x(t) &= e^{At}\delta x_0 = e^{M\Lambda M^{-1}t} = Me^{\Lambda t}M^{-1}\delta x_0 \\ &= c_1 e^{\lambda_1 t}\delta\tilde{x}_{0,1} + c_2 e^{\lambda_2 t}\delta\tilde{x}_{0,2} + \ldots + c_7 e^{\lambda_7 t}\delta\tilde{x}_{0,7} + c_8 e^{\lambda_8 t}\delta\tilde{x}_{0,8} \\ &= \sum_{i=1}^{8} c_i e^{\lambda_i t}\delta\tilde{x}_{0,i} \end{aligned} \tag{5}$$

where $\tilde{x}_{0,i} = h(x_0)$ and the constants c_i are dependent on the system's eigenvectors.

It is important to see the steady state behavior of the system, which is equivalent to the homogeneous system in the limit that $t \to \infty$. With the aforementioned properties of the eigenvalues (especially $\lambda_7 = 0$) one can compute

$$\lim_{t\to\infty} \sum_{i=1}^{8} c_i e^{\lambda_i t} \delta\tilde{x}_{0,i} = c_7 \delta\tilde{x}_{0,7} \tag{6}$$

Eq. (6) shows that the system is converging to a constant $c_7\delta\tilde{x}_{0,7}$ for $t \to \infty$. This can be interpreted as follows: if one exerts a little force onto the linearized system, it starts oscillating and turning in δ_{M} and δ_{A}. As time progresses, the oscillation subsides (since the complex eigenvalues have negative real parts) and the turning motion in δ_{M} and δ_{A} stops because the system is losing energy due to friction. This way, δ_{M} and δ_{A} converge to a constant value while the rest of the states δx settle to zero again. This interpretation is encouraged by the fact that the eigenvector corresponding to λ_7 is $v_7 \approx [0.7, 0.7, 0, 0, 0, 0, 0, 0]^T$. The system behavior it describes is thusly a motion in δ_{M} and δ_{A}.

The second eigenvalue of interest is λ_1. Compared to the other eigenvalues it is several magnitudes larger. This means that one state of the system decays very fast, namely $\dot{\delta}_{\mathrm{M}}$ since it is controlled by the internal P controller of the drives to follow $\dot{\delta}_{\mathrm{M,SP}} = u_{\mathrm{SS}}$. As the k_{P} of this P controller is fairly large, the corresponding eigenvalue is also large. Again the eigenvector corresponding to λ_1 supports this claim. Since $v_1 \approx [0, 0, 0, 0, -1, 0, 0, 0]^T$, $\dot{\delta}_{\mathrm{M}}$ is indeed the fast decaying state.

In order to later judge the efficiency of the closed loop controls one has to examine the open loop step response of the non-linear system. For this the setup with $u = \dot{\delta}_{\mathrm{M,SP}}$ is used. The following criteria are used to quantify the performance of the control setup:

1. overshoot
$$H_{\%}^{\mathrm{R/F}} = 100 \cdot \frac{\alpha_{\mathrm{MAX/MIN}} - \alpha_{\mathrm{REF}}}{\Delta\alpha_{\mathrm{REF}}} \%$$

2. rise/fall time
$$\begin{aligned} &T_{\mathrm{R}} = t(\alpha_{90\%}) - t(\alpha_{10\%}) \\ &T_{\mathrm{F}} = t(\alpha_{10\%}) - t(\alpha_{90\%}) \\ &\text{with} \quad \alpha_{90\%} = 0.9 \cdot \alpha_{\mathrm{REF}} \quad \text{and} \quad \alpha_{10\%} = 0.1 \cdot \alpha_{\mathrm{REF}} \end{aligned}$$

3. settling time in reference to the time of the step t_{STEP}
$$\alpha(t - t_{\mathrm{STEP}} > T_{\mathrm{SET}}^{\epsilon}) \in [(1-\epsilon)\alpha_{\mathrm{REF}}, (1+\epsilon)\alpha_{\mathrm{REF}}] \text{ with } \epsilon = 0.01$$

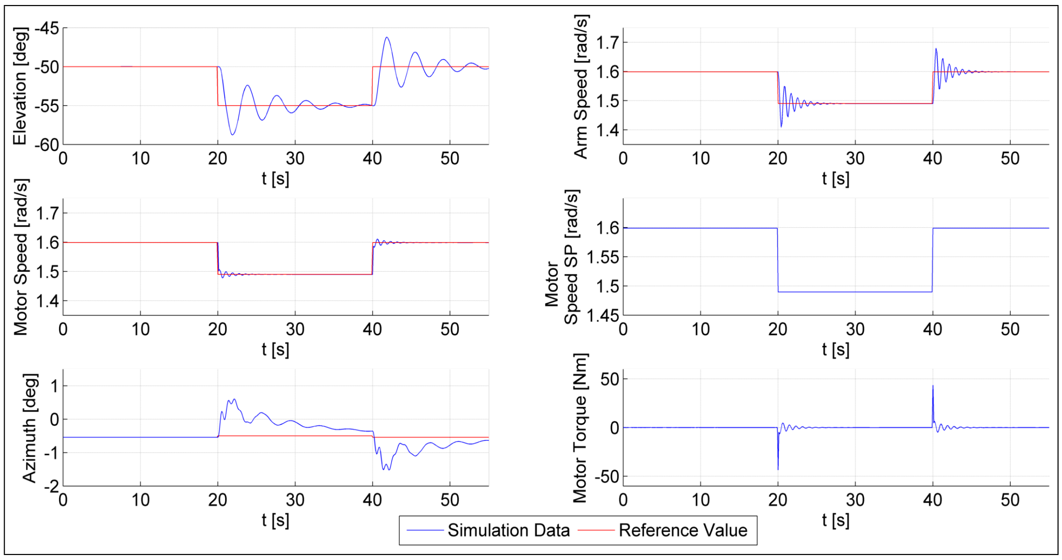

Figure 9: Overview of the state behavior for an open loop step response with α changing from $-50°$ to $-55°$ and back to $-50°$.

Evaluation Criteria	Value
$H^{\mathrm{S}}_{\%}$	75.6 %
$H^{\mathrm{F}}_{\%}$	75.6 %
T_{R}	0.5 s
T_{F}	0.6 s
$T^{\epsilon}_{\mathrm{SET}}$	11.7 s

Table 3: Evaluation criteria for open loop feedforward control setup.

Fig. 9 shows an overview of the simulated open loop dynamics of the system for a step in α_{REF} from $-50°$ to $-55°$ and then back again to $-50°$. Fig. 10 is a more detailed version of the elevation step response since this is the most important response for judging the performance of the control setup. In this and all further simulations in this thesis the propagation of the system behavior is done with the non-linear system equations with the script `carousel_dynamics`.

The criteria mentioned above are summarized in Table 3. The shown values act as a baseline for evaluating all further control setups that should perform *at least* as good as the open loop response.

It is now established how the model of the carousel is derived using the Lagrange formalism and how it is subsequently linearized. It was further investigated how the open loop system behaves with $\dot{\delta}_{\mathrm{M}}$ as control u. A closer look at the eigenvalues and eigenvectors of the linearized system has

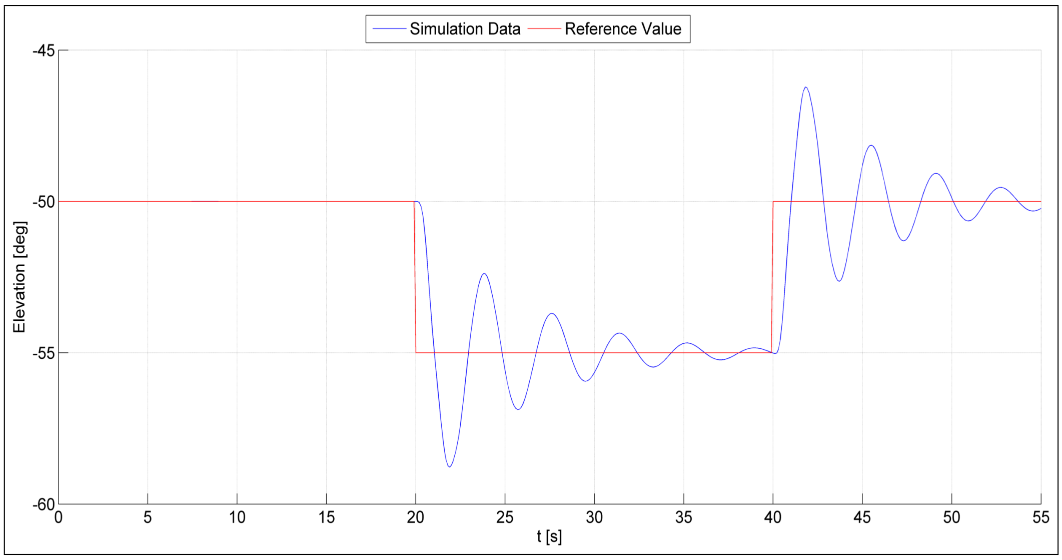

Figure 10: Elevation step response of the open loop system with α changing from $-50°$ to $-55°$ and back to $-50°$.

explained the most important aspects of the carousel's dynamics.

3 Closing the Loop

Up until now the open loop behavior of the system has been investigated. As a next step, one needs to improve the behavior of the system by decreasing the overshoot and settling time of the open loop system (see Table 3). To do so, it is necessary to implement a feedback controller and thus close the loop. A Discrete Linear Quadratic Regulator (DLQR) is chosen for that task. After comparing the DLQR's performance to a PID controller, it is complemented with a Kalman filter for state estimation.

3.1 Discrete Linear Quadratic Regulator

To implement a DLQR, we must look for a feedback matrix K with

$$\bar{u}_k = -K\bar{x}_k,\ k = 0, 1, 2, \dots, N \tag{7}$$

In order to control x_k to a desired reference value $x_{\text{REF},k}$, one sets $\bar{x}_k = x_k - x_{\text{REF},k}$. A feedforward term is implemented with $\bar{u}_k = u_k - u_{\text{FF},k}$. $\ddot{\delta}_{\text{M,SP}}$ is used as control. With this setup, u_{FF} is equal to zero and thus $\bar{u}_k = u_k$. The controller $\bar{u}_k$ and with it the feedback are linearly proportional to the difference of the state vector x_k and the desired state $x_{\text{REF},k}$. The feedback law $\bar{u}_k(x_k)$ is an affine map. All reference values are steady states and are calculated using the `solve_steady_state_lsq` function described in Section 2.3. Since δ_M and δ_A do not have a steady state that could be used as a reference, they are updated in each iteration using the difference equations

$$\delta_{\text{M,REF},k+1} = \delta_{\text{M,REF},k} + \dot{\delta}_{\text{M},k} \cdot T_\text{S}$$
$$\delta_{\text{A,REF},k+1} = \delta_{\text{A,REF},k} + \dot{\delta}_{\text{A},k} \cdot T_\text{S}$$

It is necessary to chose the feedback matrix K for Eq. (7). To do so, one can choose the approach of a DLQR that states that the feedback matrix K should minimize the cost function

$$J = \sum_{k=0}^{\infty} (\bar{x}_k^T Q \bar{x}_k + \bar{u}_k^T R \bar{u}_k + 2\bar{x}_k^T N \bar{u}_k)$$

for a system

$$\bar{x}_{k+1} = A\bar{x}_k + B\bar{u}_k \tag{8}$$

With the aforementioned forms for $\bar{x}$ and $\bar{u}$ Eq. (8) becomes

$$x_{k+1} - x_{\text{REF},k+1} = A(x_k - x_{\text{REF},k}) + B(x_k - x_{\text{REF},k})$$

This procedure can be concisely rephrased as the optimization problem

$$\begin{aligned} \min_{\bar{u}} \quad & \sum_{k=0}^{\infty} (\bar{x}_k^T Q \bar{x}_k + \bar{u}_k^T R \bar{u}_k + 2\bar{x}_k^T N \bar{u}_k) \\ \text{s.t. } & \bar{x}_{k+1} - A\bar{x}_k + B\bar{u}_k = 0 \\ & \bar{x}_0 - x_0^\star = 0 \\ \text{with } & \bar{x}_k = x_k - x_{\text{REF},k} \\ & \bar{u}_k = u_k - u_{\text{REF},k} \end{aligned} \tag{9}$$

The minimization problem presented in Eq. (9) is a linear quadratic problem of the form

$$\begin{aligned} \min_{x,u} \quad & \sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} Q_k & N_k^T \\ N_k & R_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + x_N^T P_N x_N \\ \text{s.t. } & x_{k+1} - A_k x_k + B_k u_k = 0 \\ & x_0 - x_0^\star = 0 \end{aligned}$$

This kind of optimization problem is solved by first calculating the *Difference Riccati Equation* (see Eq. (10)) and then solving for the optimal $u_k(x_k)$ by forward simulation with Eq. (11).

$$P_k = Q_k + A_k^T P_{k+1} A_k - (N_k^T + A_k^T P_{k+1} B_k)(R_k + B_k^T P_{k+1} B_k)^{-1}(N_k + B_k^T P_{k+1} A_k) \tag{10}$$

$$u_k(x_k) = -(R_k + B_k^T P_{k+1} B_k)^{-1}(N_k + B_k^T P_{k+1} A_k) x_k \tag{11}$$

This solution can be simplified by setting $N \to \infty$ and assuming that P_k converges to a P_∞. One can then set $P_k = P_{k+1} = P_\infty$ and transform Eq. (10) to the *Discrete Algebraic Ricatti Equation* (see Eq. (12)). All indices k of the matrices are dropped and replaced by ∞.[3] The optimal $u_k(x_k)$ is then obtained with Eq. (13), which is already in the form of a feedback rule like Eq. (7). [2, p. 52–55]

$$P = Q + A^T PA - (N^T + A^T PB)(R + B^T PB)^{-1}(N + B^T PA) \tag{12}$$

[3]For enhanced readability, the ∞ is omitted in Eq. (12), thus $P_\infty = P$ and so on.

$$u_k(x_k) = -\underbrace{(R + B^T P B)^{-1}(N + B^T P A)}_{=K} x_k \tag{13}$$

It should be noted that the optimal $u_k(x_k)$ is only a function of x_k and is independent of the initial value x_0. Also the feedback matrix K is constant for all times k. The DLQR can be implemented in `MATLAB` using the function `DLQR(A,B,Q,R,N)`, which solves the *Discrete Algebraic Ricatti Equation* and returns the optimal feedback matrix K.

By comparing the optimization problem in Eq. (9) to Eq. (4) one can see that it corresponds to a linearization at $x_{\mathrm{REF},k}$, $u_{\mathrm{REF},k}$. Thus for the feedback matrix K to be optimal, all $x_{\mathrm{REF},k}$, $u_{\mathrm{REF},k}$ need to be close to the point of linearization. For the point of linearization the x_{SS} corresponding to $\frac{\alpha_{\mathrm{SS},0}+\alpha_{\mathrm{SS},1}}{2}$ is chosen, with a step in α from $\alpha_{\mathrm{SS},0}$ to $\alpha_{\mathrm{SS},1}$. When n_x is the dimension of the state vector and n_u is the dimension of the controls, $Q \in \mathbb{R}^{n_x \times n_x}$, $R \in \mathbb{R}^{n_u \times n_u}$ and $N \in \mathbb{R}^{n_x \times n_u}$.
For choosing Q, R, and N, we have to keep the following criteria in mind:

1. $\bar{u}_k$ should be reasonably small for all k
2. $\bar{x}_{k,3}$ is of particular interest since it corresponds to the systems oscillation of α around the desired reference value
3. time is of interest. The faster $\bar{x}_k$ converges to zero the better
4. Q and R have to be positive definite. N can be zero [12]
5. the system has to be stabilizable [12]

In addition to Q, R, and N, one has to supply `DLQR` with the matrices A and B of the linearized system. The matrices for the regulator are

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad R = [2^2], \quad N = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

In Q one can see a strong penalty for α deviating from its reference. Also deviations of $\dot{\alpha}$ are penalized, although less. N is zero because cross terms of

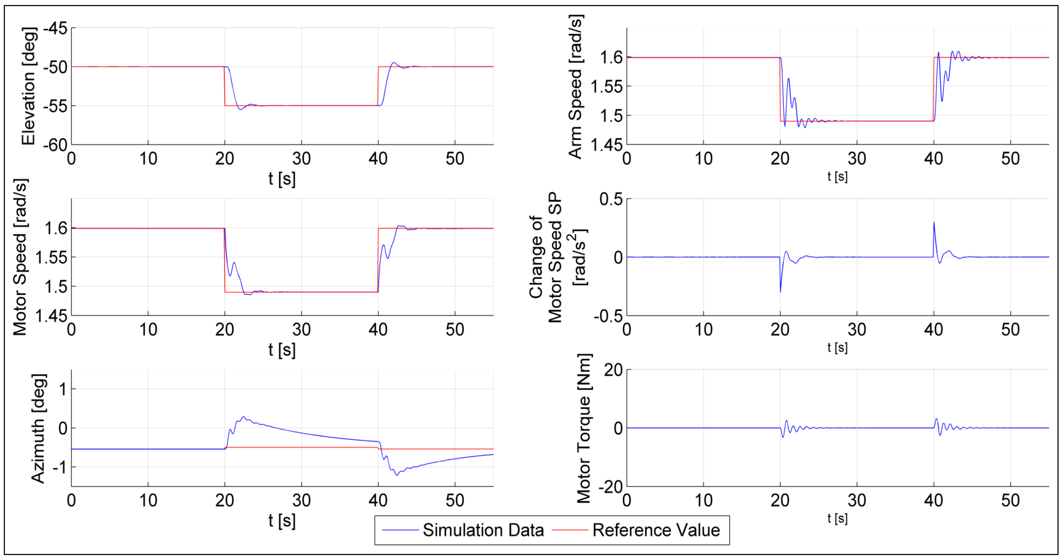

Figure 11: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. The fact that $\dot{\alpha}$ is not penalized results in an overshoot.

the form $2\bar{x}_k N \bar{u}_k$ are not needed in the cost function. By choosing $\ddot{\delta}_{\mathrm{M,SP}}$ as the control u, one can directly penalize the change of $\dot{\delta}_{\mathrm{M,SP}}$ and thus ensure that the applied motor torque stays below the torque limit. Choosing the matrices in this way results in a feedback matrix

$$\begin{aligned} K =&[5.9810, -5.9809, -1.2141, -1.0758 \times 10^{-1}, 4.5442 \times 10^{-7}, \\ &4.9958 \times 10^{-2}, 2.0995, 1.2904 \times 10^{-1}, 3.5599] \end{aligned}$$

In Fig. 11 one can see the simulation results for a DLQR *without* the aforementioned penalty for deviation of $\dot{\alpha}$. Introducing the penalty for $\dot{\alpha}$ reduces the overshoot (see Fig. 12) because the change of α is slower and less abrupt. This combination of penalties for α and $\dot{\alpha}$ produces the best results for the controller setup. Fig. 13 shows a more detailed plot comparing the elevation step response of the DLQR and open loop.

To compare the step response with DLQR to the open loop response, the same evaluation criteria as in Section 2.4 are applied (see Table 4). Compared to the open loop control (see Table 3) both $H^{\mathrm{S}}_{\%}$ and $H^{\mathrm{F}}_{\%}$ have decreased. The overshoot $H^{\mathrm{S}}_{\%}$ has decreased by approximately 99.9 % while $H^{\mathrm{F}}_{\%}$ has become so small that it appears to be 0 % with the current simulation timestep $T_{\mathrm{S}} = 0.1\,\mathrm{s}$. The settling time $T^{\epsilon}_{\mathrm{SET}}$ has decreased by 85 %. The rise time T_{R} and the fall time T_{F} on the other hand have increased by approximately 80 %. Looking at Fig. 13 one can see that despite the increased T_{R} and T_{F}, the step response for the DLQR controller looks much better than the open

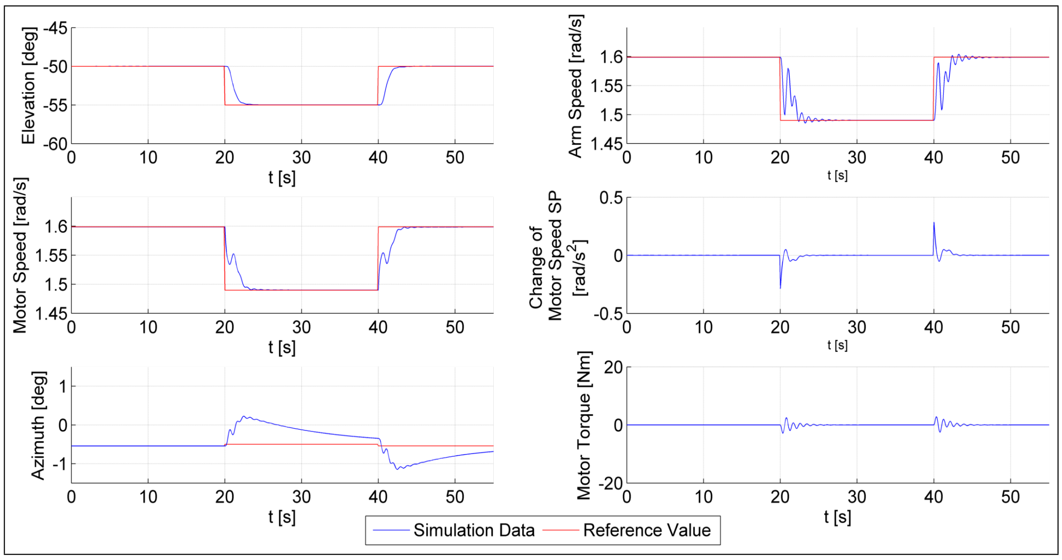

Figure 12: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. Penalization of $\dot{\alpha}$ makes the overshoot disappear.

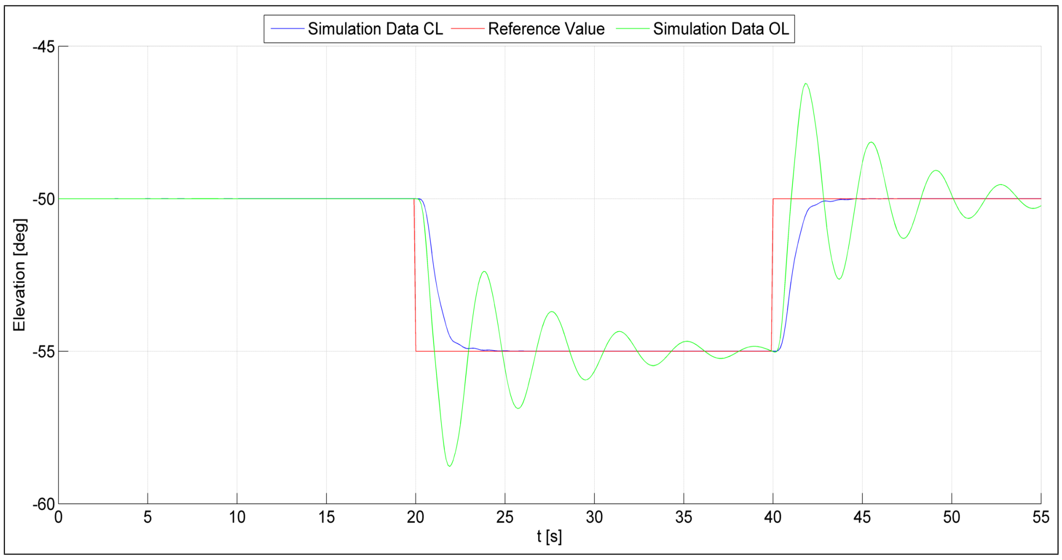

Figure 13: Elevation step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. Penalization of $\dot{\alpha}$ makes the overshoot disappear. The open loop step response is shown for comparison.

Evaluation Criteria	CL Value	OL Value
$H^{\mathrm{S}}_{\%}$	$0.02\,\%$	$75.6\,\%$
$H^{\mathrm{F}}_{\%}$	$\approx 0\,\%$	$75.6\,\%$
T_{R}	$0.9\,\mathrm{s}$	$0.5\,\mathrm{s}$
T_{F}	$0.9\,\mathrm{s}$	$0.6\,\mathrm{s}$
$T^{\epsilon}_{\mathrm{SET}}$	$1.7\,\mathrm{s}$	$11.7\,\mathrm{s}$

Table 4: Evaluation criteria for closed loop DLQR setup. The open loop values are shown in comparison.

loop response. Comparing the motor torque for the two setups (see Fig. 9 and Fig. 12) it becomes apparent that while the open loop controller exceeds the torque limit, the DLQR stays well below the limit. In summary the DLQR exerts less torque on the carousel but still gets better results.

3.2 DLQR versus PID controller

The DLQR is a fairly complex form a of controller, especially since it requires an estimator (in this case a Kalman filter). Is the performance of the DLQR worth the effort[4] or can similar or even better results be achieved by a much simpler controller? This can be checked by comparing the DLQR to a discrete PID controller — a much less complex setup that requires no estimator. The PID controller is implemented with

$$u_k = -k_{\mathrm{P}}\bar{\alpha}_k - k_{\mathrm{I}} \cdot T_{\mathrm{S}} \cdot \sum_{k=1}^{n} \bar{\alpha}_k - k_{\mathrm{D}} \frac{\bar{\alpha}_k - \bar{\alpha}_{k-1}}{T_{\mathrm{S}}}$$
$$\text{with} \quad \bar{\alpha}_k = \alpha_k - \alpha_{\mathrm{REF},k}$$

The values for k_{P}, k_{I} and k_{D} are derived using the Ziegler-Nichols method. For this method, a simple P controller is implemented and its gain increased until the system becomes unstable. The gains are then derived using Eq. (14) with the help of the values of the critical P gain k_{CR} and the period T_{CR} [10, p. 443]. Fig. 14 shows the system with $k_{\mathrm{CR}} = 0.42$, oscillating with $T_{\mathrm{CR}} = 3.9\,\mathrm{s}$. Using the corresponding gains results in the simulation shown in Fig. 15. This is used as a baseline for fine-tuning the gains by hand to improve the PID (see Fig. 16). The gains — both before and after fine-tuning — are shown in Table 5.

[4]The DLQR is harder to implement and has more degrees of freedom for tuning than e.g. a PID controller. For this reason it is also harder to debug. Most importantly however, a DLQR needs a good model of the system to work properly.

Constant	Ziegler-Nichols Value	Fine-Tuned Value
k_{P}	0.252	0.3
k_{I}	0.129	0.001
k_{D}	0.118	0.08

Table 5: Constants used to tune the PID controller.

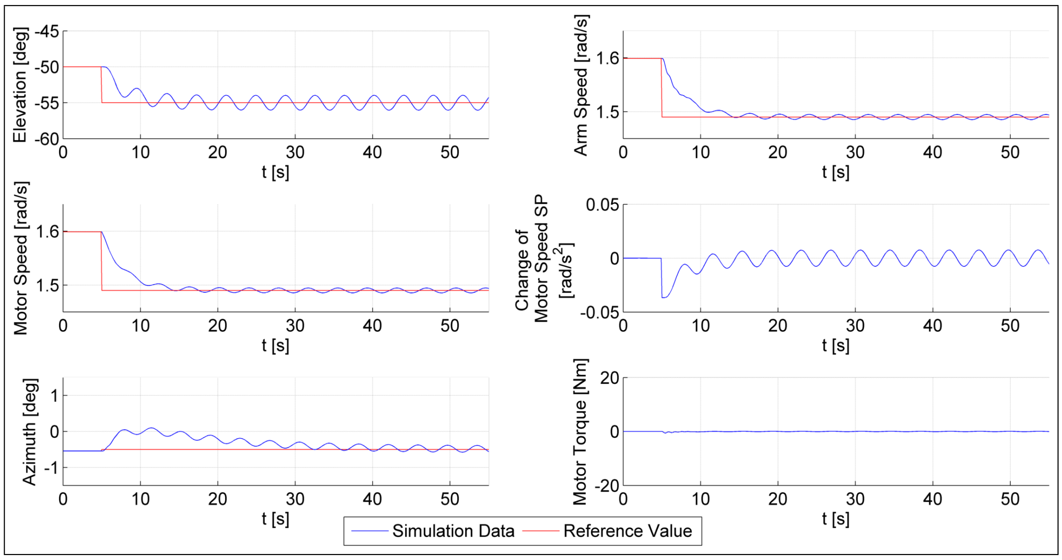

Figure 14: Overview of the system behavior for a step response with a P controller and critical gain. The system oscillates with period T_{CR}.

$$\begin{aligned} k_{\mathrm{P}} &= 0.6 \cdot k_{\mathrm{CR}} \\ k_{\mathrm{I}} &= 1.2 \cdot \frac{k_{\mathrm{CR}}}{T_{\mathrm{CR}}} \\ k_{\mathrm{D}} &= 0.072 \cdot k_{\mathrm{CR}} T_{\mathrm{CR}} \end{aligned} \tag{14}$$

Fig. 17 shows a comparison of the elevation step response for the PID and DLQR setup. The corresponding values for the evaluation criteria are presented in Table 6. When comparing the system behavior and evaluation criteria it becomes apparent that the DLQR's performance surpasses that of the PID controller. The rise time T_{R} increased by 800 % compared to the DLQR, T_{F} by 822 %. The settling time $T^{\epsilon}_{\mathrm{SETTLE}}$ increased by 612 %. The question posed at the beginning of this section can therefore be answered: a simple PID cannot achieve similar or better performance than the DLQR in this case.

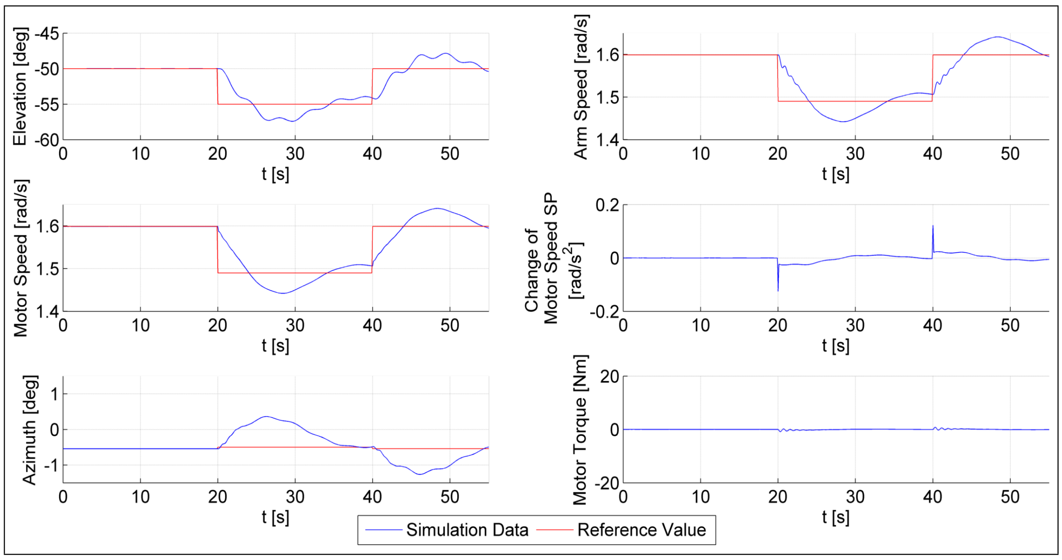

Figure 15: Overview of the system behavior for a step response with a PID controller and α changing from $-50°$ to $-55°$ and back to $-50°$. The gains of the PID have been derived with the Ziegler-Nichols method.

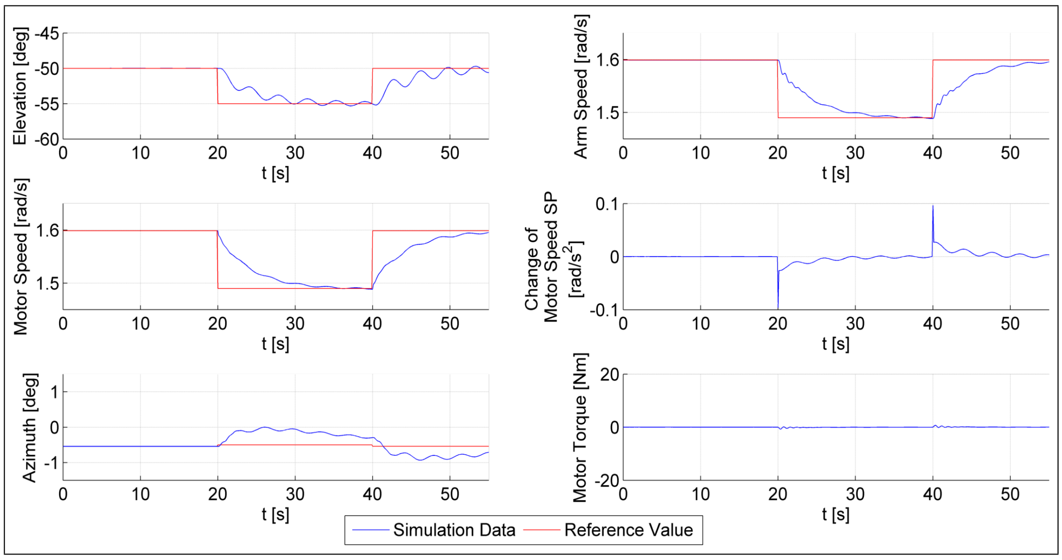

Figure 16: Elevation step response with PID controller with α changing from $-50°$ to $-55°$ and back to $-50°$. The gains of the PID have been derived with the Ziegler-Nichols method and then fine-tuned by hand.

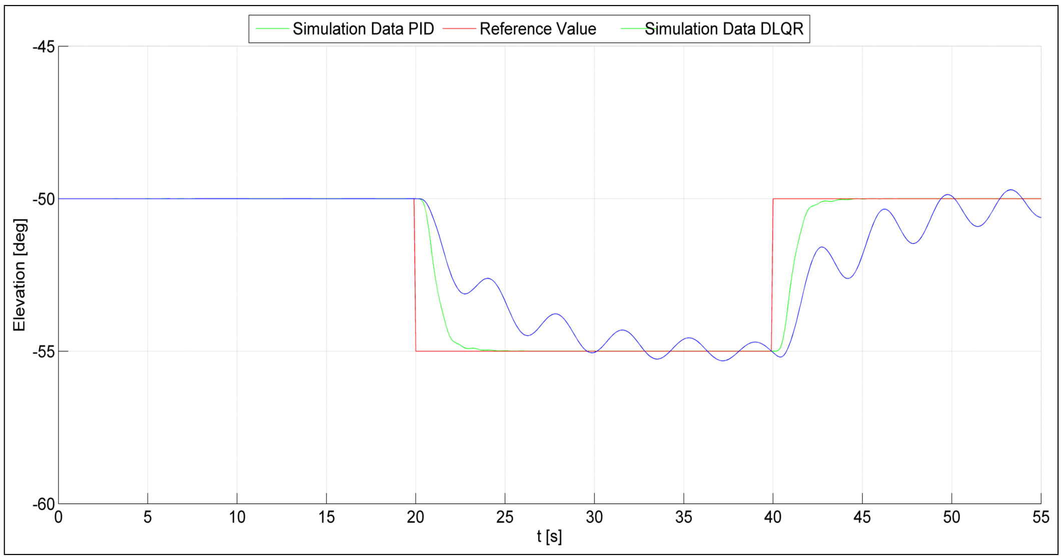

Figure 17: Comparison of elevation step response of PID and DLQR with α changing from $-50°$ to $-55°$ and back to $-50°$.

Evaluation Criteria	Value PID	Value DLQR
$H^{\mathrm{S}}_{\%}$	5.8 %	0.02 %
$H^{\mathrm{F}}_{\%}$	6.2 %	≈ 0 %
T_{R}	8.1 s	0.9 s
T_{F}	8.3 s	0.9 s
$T^{\epsilon}_{\mathrm{SET}}$	12.1 s	1.7 s

Table 6: Evaluation criteria for closed loop PID controller setup. The values for DLQR are shown in comparison.

3.3 State Estimation with Kalman Filter

The feedback law of Eq. (7) requires that the full state vector x_k is known to the controller. This would in turn necessitate the exact measurement of all states at all times k. The way the carousel is set up however, only the states α and β are measured. For further ease of notation, the system output vector is defined in Eq. (15), with y_α and y_β as the measured values returned from the sensors. The sensors are modeled with the sensor function $f_{\mathrm{SENS}}(x_k, u_k)$ (see Eq. (16)), which is implemented in `MATLAB` with the function `sensor`.

$$y_k = [y_{\alpha,k}, y_{\beta,k}]^T \tag{15}$$

$$f_{\mathrm{SENS}}(x_k, u_k) = Cx_k = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix} \cdot x_k \tag{16}$$

Instead of trying to measure the system as accurately as possible and then trusting these measurements to reflect x_k, one can use an estimator, or more specifically, a Kalman filter. Assume a system

$$\begin{aligned} x_{k+1} &= f_{\mathrm{D}}(x_k, u_k) + Gw_k, \text{ with system noise } w_k \\ y_k &= f_{\mathrm{SENS}}(x_k, u_k) + v_k, \text{ with sensor noise } v_k \end{aligned}$$

The system noise w_k and sensor noise v_k are both white noise with $w_k \sim N(0, \Sigma(w_k))$ and $v_k \sim N(0, \Sigma(v_k))$ [11, 14, p. 375]. The general principle of a Kalman filter is shown in Algorithm 1.[5]

Algorithm 1 Working principle of a Kalman filter.

```
$x_0 = \hat{x}_{0|0}$
**while** 1 **do**
    Read in $u_k$
    $\hat{x}_{k+1|k} = f_{\mathrm{D}}(\hat{x}_{k|k}, u_k)$
    $\hat{y}_{k+1|k} = f_{\mathrm{SENS}}(\hat{x}_{k+1|k})$
    Read in $y_{k+1}$
    $\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + M(y_{k+1} - \hat{y}_{k+1|k})$
    Output $\hat{x}_{k+1|k+1}$
    $k++$
**end while**
```

This procedure is shown in Fig. 18. It can be summarized into a more compact form with the equations

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + M(y_k - \hat{y}_{k|k-1}) \\ \hat{x}_{k+1|k} &= f_{\mathrm{D}}(\hat{x}_{k|k}, u_k) \end{aligned} \tag{17}$$

Contrary to a standard Kalman filter the $f_{\mathrm{D}}(\hat{x}_{k|k}, u_k)$ used to predict x_{k+1} is not the linearized system equations but the non-linear one. The system of difference equations $\hat{x}_{k+1|k} = f(\hat{x}_{k|k}, u_k)$ is integrated with `ode15s` just as mentioned in Section 2.3. With the Kalman filter Eq. (7) changes to $\bar{u}_k = -K(\hat{x}_{k|k} - x_{\mathrm{REF},k})$. The Kalman filter uses an optimal matrix M, which means that it minimizes $\Sigma(\bar{x}_k - \hat{x}_{k|k})$ [14, p. 377]. To derive M the `MATLAB` function `DLQE(A,G,C,QE,RE)` from the `Control System Toolbox` is used. A and C are matrices supplied by the linearization. When n_x, n_y are again defined as in Section 3.1 and n_y is the output dimension, $G \in \mathbb{R}^{n_x \times n_x}$, $Q_E \in \mathbb{R}^{n_x \times n_x}$ and $R_E \in \mathbb{R}^{n_y \times n_y}$. G determines how w_k is weighed.[6] Q_E and

[5]In this notation $x_{k+1|k}$ is the estimate of x_{k+1} at timestep k.

[6]This is redundant but part of the `DLQE` function. In order to use it, one has to assign

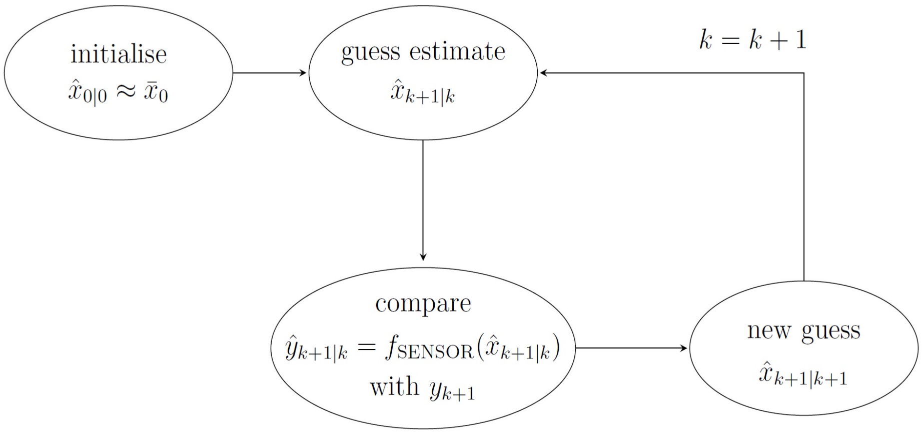

Figure 18: Working principle of a Kalman filter.

R_E are chosen so that

$$\begin{aligned} Q_E &= E\{w_k w_k^T\} = \Sigma(w_k) \\ R_E &= E\{v_k v_k^T\} = \Sigma(v_k) \end{aligned}$$

Noise can only enter the system via forces respectively $\dot{\delta}_{\text{M}}$, $\dot{\delta}_{\text{A}}$, $\dot{\alpha}$ and $\dot{\beta}$, thus Q_E is chosen to only apply to $x_{k,5}$, $x_{k,6}$, $x_{k,7}$ and $x_{k,8}$. In order to estimate the system, the matrices for the estimator are chosen as follows:

1. Q_{E} is a sparse matrix with only $Q_{\text{E},5,5}$, $Q_{\text{E},6,6}$, $Q_{\text{E},7,7}$ and $Q_{\text{E},88}$ set to w_{N}^2 where $Q_{\text{E},ij}$ is the component of Q_{E} in the i'th row and j'th column
2. $R_E = \begin{bmatrix} v_{\text{N}}^2 & 0 \\ 0 & v_{\text{N}}^2 \end{bmatrix}$
3. G is a $n_x \times n_x$ identity matrix.

G is chosen as an identity matrix under the assumption that the system noise distributes equally over all of x_{k+1}. The choice for R_E implies that all sensors

a G. The matrix Q_{E} is chosen to describe the system noise.

are equally noisy. The resulting estimator matrix is

$$M = \begin{bmatrix} -2.2631 \times 10^{-3} & 2.861 \times 10^{-5} \\ -2.2635 \times 10^{-3} & 7.6034 \times 10^{-4} \\ 3.0853 \times 10^{-2} & -4.3369 \times 10^{-7} \\ -2.4835 \times 10^{-3} & -6.3411 \times 10^{-5} \\ -4.3369 \times 10^{-7} & 1.0829 \times 10^{-2} \\ -1.3643 \times 10^{-3} & 8.9856 \times 10^{-4} \\ 5.0099 \times 10^{-3} & 1.7257 \times 10^{-3} \\ -4.8434 \times 10^{-4} & 2.0402 \times 10^{-3} \\ 0 & 0 \end{bmatrix}$$

The entries $M_{9,1}$ and $M_{9,2}$ are zero because they correspond the the carousel speed setpoint $\dot{\delta}_{\mathrm{M,SP}}$. Since this state is just the integral of $u = \ddot{\delta}_{\mathrm{M,SP}}$, it is not affected by noise.

For a first simulation w_{N} is set to 0.001 and v_{N} to 0.01. To simulate the noise, the `MATLAB` function `mvnrnd` from the `Statistics Toolbox` is used. It returns a matrix with random numbers chosen from the zero-mean multivariate normal distribution [13]. In principle there is a difference between then noise w_{N} and v_{N} assumed when tuning the Kalman filter and the simulated noise w_{SIM} and w_{SIM}. For the next two simulations $w_{\mathrm{N}} = w_{\mathrm{SIM}}$ and $v_{\mathrm{N}} = v_{\mathrm{SIM}}$. The results can be seen in Fig. 19. There is a visible difference between x_k and $\hat{x}_{k|k}$, but the estimator follows the system trajectory closely. With increased noise, this becomes more difficult. Fig. 20 shows the simulation results for both w_{N} and v_{N} increased by a factor of 10. The controller still has a reasonably good performance, but $\hat{x}_{k|k}$ is not able to follow x_k as closely as before. With increasing w_{N} and v_{N} increases also the control u_k and thus the motor torque. Even with the noise increased to a point where the estimator starts having difficulties following the system trajectory, the torque does not exceed its limit. For further simulations $w_{\mathrm{N}} = 0.001$ and $v_{\mathrm{N}} = 0.01$ regardless of how much noise is simulated with w_{SIM} and v_{SIM}.

The loop has now been closed with a Discrete Linear Quadratic Regulator that supplies an optimal feedback matrix K for the feedback law of Eq. (7) by solving the optimization problem of Eq. (9). It has been shown that the DLQR outperforms a PID controller and is therefore a useful control setup. The optimal feedback of the DLQR is complemented with a Kalman filter as an optimal state estimator that minimizes $E\{\bar{x}_k - \hat{x}_{k|k}\}$. This setup of a DLQR and Kalman filter guarantees both a good control behavior and noise compensation.

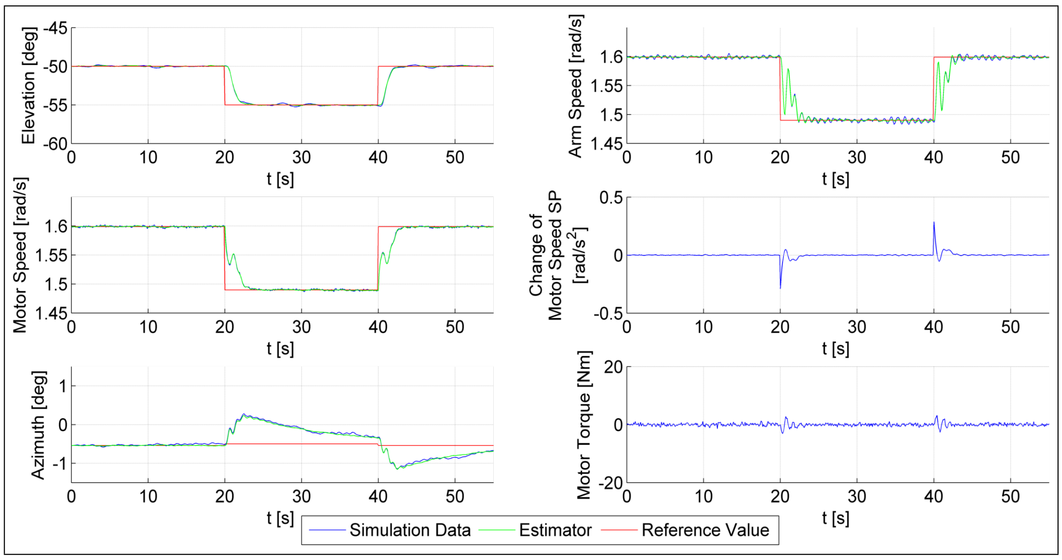

Figure 19: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. A Kalman filter is implemented with low system and sensor noise.

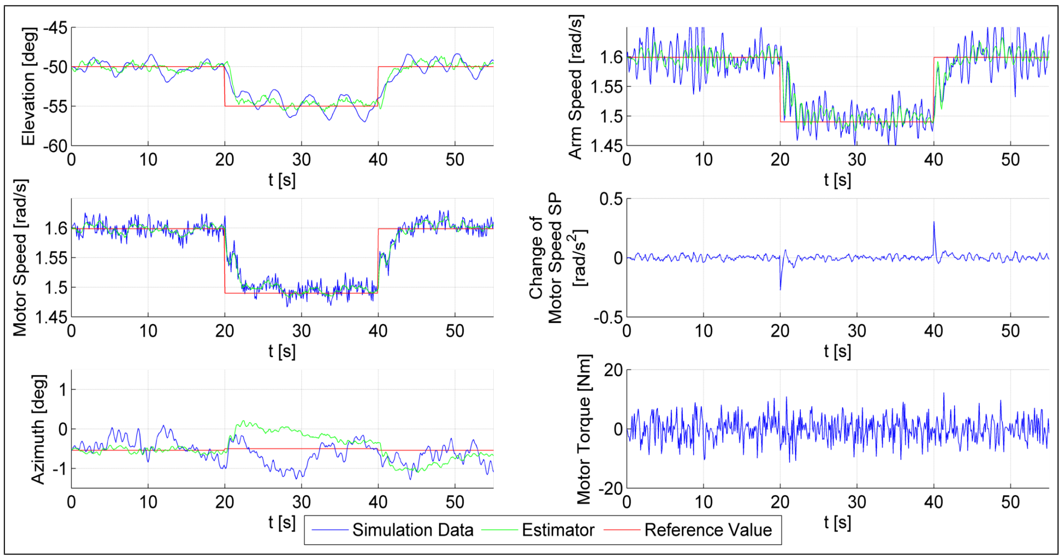

Figure 20: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. A Kalman filter is implemented with high system and sensor noise.

4 Mismodeling

Due to the limitations of modeling one can assume that there will always be a certain mismatch between the behavior of the real carousel and the model. In order to improve the controller that has been designed so far, it is helpful to simulate this mismodeling. Until now, there is only one set of ODEs that is $f(x, u)$ to both estimate the system and simulate it. Now we duplicate $f(x, u)$ and introduce $f_{\mathrm{MODEL}}(x, u)$ and $f_{\mathrm{REAL}}(x, u)$. While $f_{\mathrm{MODEL}}(x, u)$ is exactly equal to $f(x, u)$ (but for a different name), $f_{\mathrm{REAL}}(x, u)$ has slightly changed constants. Now $f_{\mathrm{REAL}}(x, u)$ is used to simulate the system but linearization, state estimation and steady state computation is done with $f_{\mathrm{MODEL}}(x, u)$.

4.1 Effects of Simulated Mismodeling

For a first simulation, the constants that are most difficult to measure (or estimate) in the real setup are changed. More specifically this means that the arm's moment of inertia I_{A} and the belt's spring constant k_{BE} are increased by $50\,\%$ each for $f_{\mathrm{REAL}}(x, u)$. To make the significant effects more visible, the system and sensor noise are turned off for the simulations in Sections 4.1 to 4.2. The result of the simulation can be seen in Fig. 21. The difference between x_k and $\hat{x}_{k|k}$ can hardly be seen. Also the overall trajectory of the elevation step response does not seem to have changed compared to the simulation without mismodeling (see Fig. 13).

But what happens if constants are changed that affect the elevation angle more directly? One variable that obviously does so is the tether length l_{T}, that is now increased by $10\,\%$ in $f_{\mathrm{REAL}}(x, u)$.[7] The simulation results are shown in Fig. 22.

Due to the differences in the two models a steady state error occurs. There is as well a mismatch between x_k and $\hat{x}_{k|k}$ as between $\hat{x}_{k|k}$ and $x_{\mathrm{REF},k}$. To successfully control the setup one must first fix the mismatch between estimator and system trajectory. The obvious thought would be to treat the mismatch just like any other system noise and try to decrease it by adding a component in Q_{E} responsible for noise in α. This is however ill-advised because the Kalman filter described in Section 3.3 only works for zero-mean Gaussian noise [14, p. 375].

[7]The constants I_{A} and k_{BE} are changed back to their initial values.

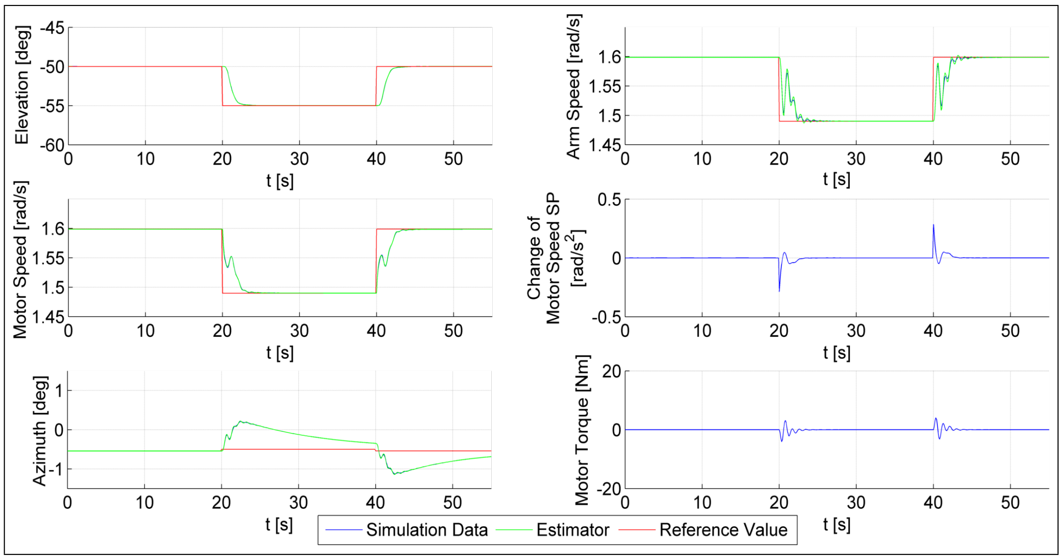

Figure 21: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. The constants I_{A} and k_{BE} are increased by 50 % for $f_{\mathrm{REAL}}(x, u)$ to show the controller's reaction to mismodeling. System and sensor noise are turned off.

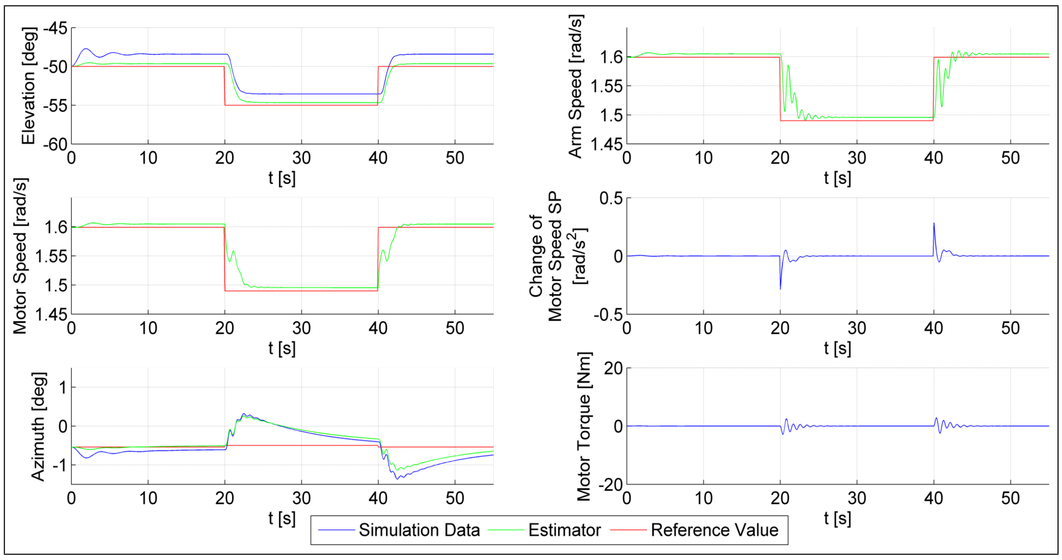

Figure 22: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. The constant l_{T} is increased by 10 % for $f_{\mathrm{REAL}}(x, u)$ to show the controller's reaction to mismodeling. System and sensor noise are turned off.

4.2 Addition of a Pseudo-Force

A different approach is chosen by introducing a generalized pseudo-force S_P in α direction as a new state seemingly responsible for the effects of the mismodeling. By doing so, $f(x,u)$ changes with

$$\begin{aligned}\ddot{\alpha} &= \cdots + S_\mathrm{P} \\ \dot{S}_\mathrm{P} &= \frac{-S_\mathrm{P}}{\tau} + w_\mathrm{N,S} \quad \text{with} \quad w_\mathrm{N,S} \sim N(0, \Sigma(S_\mathrm{P}))\end{aligned} \tag{18}$$

By setting the component $w_\mathrm{N,S}$ of Q_E responsible for S_P to 0.1 one ensures that the estimator will learn the value of S_P and correctly adjust $\hat{x}_{k|k}$ to fit the degree of mismodeling that the value of the pseudo-force represents. $\dot{S}_\mathrm{P}$ is set to a first-order Gauss-Markov term so that the system remains at least stabilizable [5, p. 44f]. The time constant τ is set to 100 s since S_P should decline slower than the other time constants of the system. The feedback and estimator matrices thus change to

$$\begin{aligned}K =& [5.9810, -5.9809, -1.2141, -1.0758 \times 10^{-1}, 4.5442 \times 10^{-7}, \\ & 4.9958 \times 10^{-2}, 2.0995, 1.2904 \times 10^{-1}, 3.5599, 1.5223]\end{aligned}$$

$$M = \begin{bmatrix} -4.2742 \times 10^{-4} & 1.0309 \times 10^{-5} \\ -1.9221 \times 10^{-3} & 7.5257 \times 10^{-4} \\ 5.7198 \times 10^{-1} & -1.8954 \times 10^{-3} \\ -3.8768 \times 10^{-2} & 3.7148 \times 10^{-5} \\ -1.8954 \times 10^{-3} & 1.0842 \times 10^{-2} \\ -6.3224 \times 10^{-3} & 9.3547 \times 10^{-4} \\ 2.3721 & -1.1167 \times 10^{-2} \\ -1.8835 \times 10^{-1} & 3.0575 \times 10^{-3} \\ 0 & 0 \\ 6.5144 & -3.4605 \times 10^{-2} \end{bmatrix} \tag{19}$$

Repeating the simulation with increased l_T produces Fig. 23. The approach has the desired result: α_k now converges to $\hat{\alpha}_{k|k}$ and the steady state error almost disappears. The kink in α at approximately 1 s stems from the fact that the estimator needs a finite time to learn S_P before it can adjust. Also the controller is not able anymore to follow all of the references — fixing the steady state error in α has resulted in a steady state error in the other state. This however is not a problem and easily understood: since the reference values for each state are derived using f_MODEL, they do not reflect the system's actual steady states. Following any other state than α would therefore be pointless.

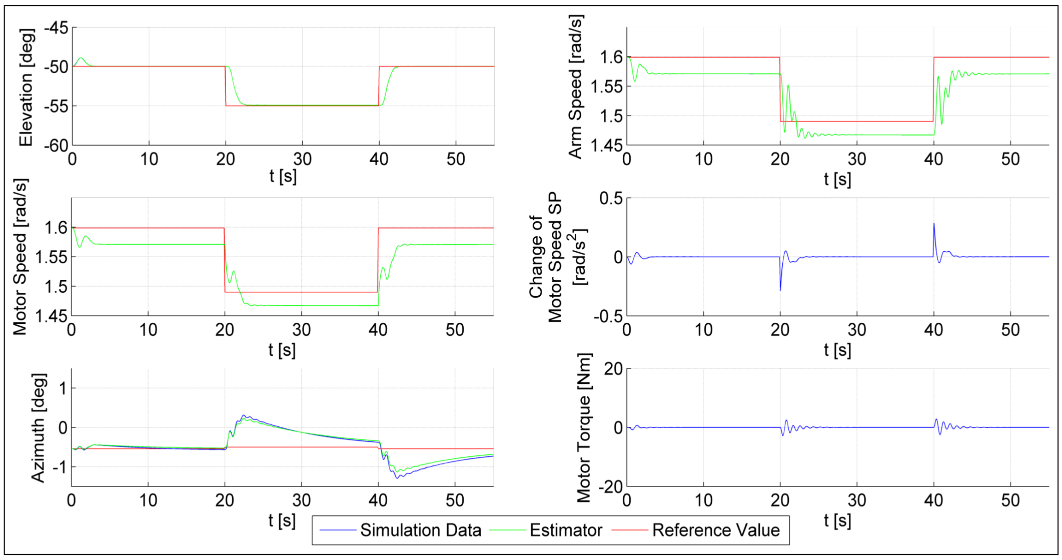

Figure 23: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. The constant l_{T} is increased by 10 % for $f_{\mathrm{REAL}}(x,u)$ to show the controller's reaction to mismodeling. System and sensor noise are turned off. A new state S_{P} is added to decrease the steady state error.

4.3 Noise Compensation of the Enhanced DLQR

To ensure that the new control setup with an integral term can still compensate system noise as well as before, w_{SIM} is set to 0.001 and v_{SIM} to 0.01 again (just like in the simulation seen in Fig. 19). The component $w_{\mathrm{SIM,S}} = 0$ because no noise enters the system via the pseudo-force. The results of the new simulation can be seen in Fig. 24. Compared to the simulation without mismodeling all states oscillate a lot more, especially the arm speed $\dot{\delta}_{\mathrm{A}}$. Besides the motor torque has increased (even tough it still stays below its limit). This behavior is not desirable. The unwanted oscillation of the system is the result of the Kalman filter making S_{P} responsible for most of the system noise w_{N} (since the component $w_{\mathrm{N,S}}$ of Q_{E} responsible for S_{P} is 100 times larger than w_{N}). This results in a badly tuned matrix M with too big components for S_{P} and $\dot{\alpha}$ (see Eq. (19)). To solve this problem one sets

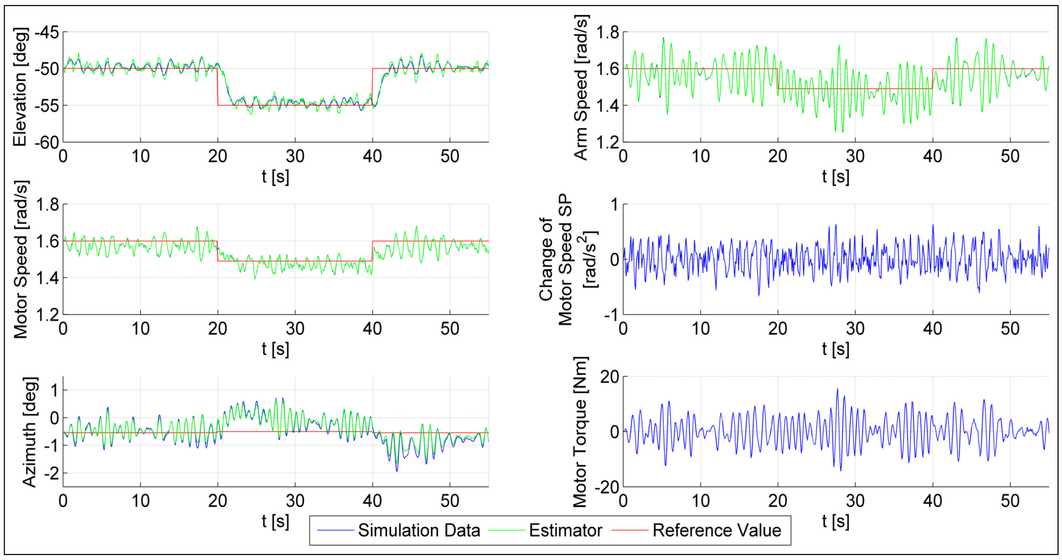

Figure 24: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. The constant l_{T} is increased by 10 % for $f_{\mathrm{REAL}}(x, u)$ to show the controller's reaction to mismodeling. System noise w_{N} is set to 0.001 and sensor noise v_{N} to 0.01. A new state S_{P} is added to decrease the mismatch between x_k and $\hat{x}_{k|k}$.

$w_{\mathrm{N,S}} = w_{\mathrm{N}}$, resulting in

$$M = \begin{bmatrix} -2.0970 \times 10^{-3} & 2.6636 \times 10^{-5} \\ -2.1023 \times 10^{-3} & 7.5838 \times 10^{-4} \\ 6.5531 \times 10^{-2} & -6.8076 \times 10^{-6} \\ -5.2434 \times 10^{-3} & -6.2946 \times 10^{-5} \\ -6.8076 \times 10^{-6} & 1.0829 \times 10^{-2} \\ -1.3302 \times 10^{-3} & 8.9858 \times 10^{-4} \\ 2.2385 \times 10^{-2} & 1.7033 \times 10^{-3} \\ -1.9302 \times 10^{-3} & 2.0422 \times 10^{-3} \\ 0 & 0 \\ 9.3440 \times 10^{-2} & -3.0876 \times 10^{-5} \end{bmatrix}$$

The huge entries of M for $\dot{\alpha}$ and S_{P} are the same size as the other entries of the matrix. Repeating the previous simulation with $w_{\mathrm{N,S}} = w_{\mathrm{N}} = 0.001$ produces the plot shown in Fig. 25.

Comparing Fig. 24 and Fig. 25 one can see that the change in $w_{\mathrm{N,S}}$ had the desired effect. The system's oscillation has decreased and the system trajectory can follow the reference much better. Also the motor torque has decreased by a factor of ten.

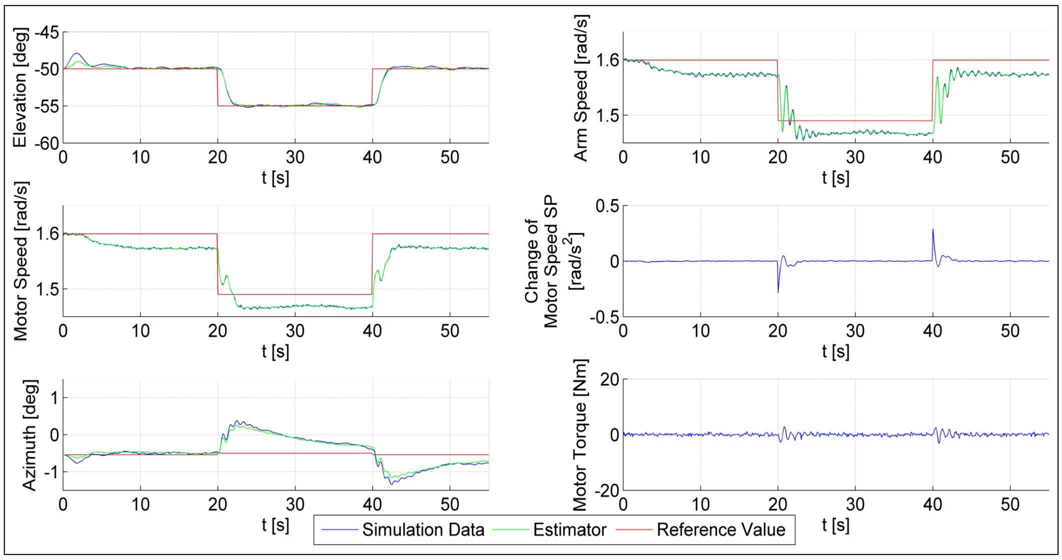

Figure 25: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. The constant l_{T} is increased by 10 % for $f_{\mathrm{REAL}}(x, u)$ to show the controller's reaction to mismodeling. System noise w_{N} is set to 0.001 and sensor noise v_{N} to 0.01. A new state S_{P} is added to decrease the steady state error. To decrease oscillation $w_{\mathrm{N,S}}$ is set to 0.001.

4.4 Simplification of the Kalman Filter

A special kind of mismodeling is the simplification of the Kalman filter. In the previous simulations the Kalman filter worked as shown in Eq. (17) where the $f_{\mathrm{D}}(x, u)$ to estimate x_{k+1} is a non-linear system of difference equations. Since the feedback matrix K and estimator matrix M is only optimal for small steps around the point of linearization x_{SS}, one should be able to use the linearized system to estimate x_{k+1} without losing too much accuracy. In addition to simplifying the port to the carousel, the linearization of the Kalman filter also improves the computation time since only a linear system of differential equations has to be integrated intead of a non-linear one. Using the linearized system equations instead of the non-linear ones is a kind of mismodeling that can be counteracted in the way that has been discussed in this section. The equations for the Kalman filter change to

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + M(y_k - \hat{y}_{k|k-1})$$
$$\hat{x}_{k+1|k} = x_{\mathrm{SS}} + A(\hat{x}_{k|k} - x_{\mathrm{SS}}) + Bu_k$$

Repeating the previous step response simulation with an increased tether length (but this time without any additions to get rid of mismodeling errors and no noise) produces the plot in Fig. 26. As expected there is again a

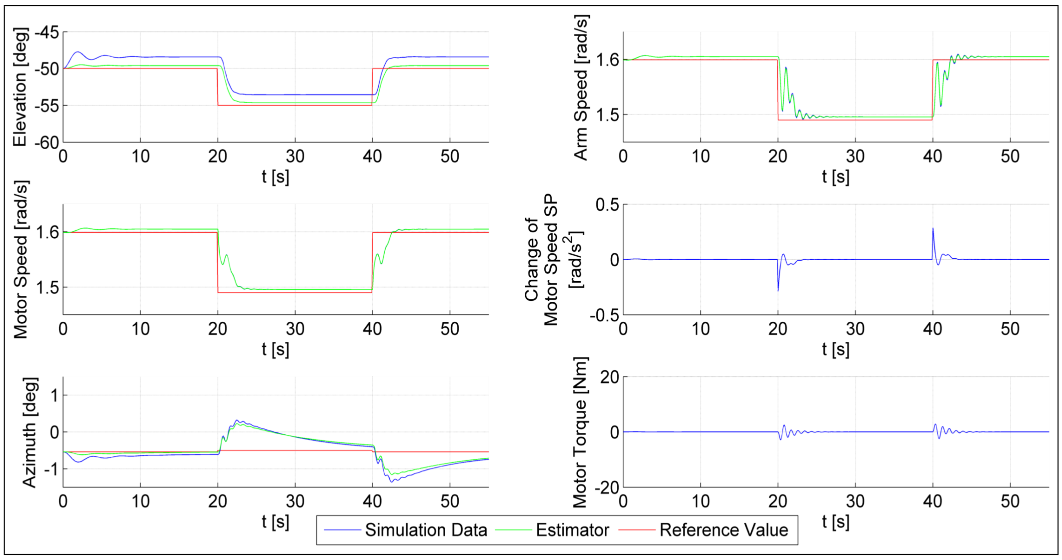

Figure 26: Overview of the system behavior for a step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. A Kalman filter with linear state estimation has been implimented. The thether length l_T has been increased by 10 % to show the effects of mismodeling.

steady state error and a mismatch between x_k and $\hat{x}_{k|k}$.

In order to get rid of the steady state error and the mismatch between x_k and $\hat{x}_{k|k}$ the additional state S_P is introduced again into the system equations $f(x, u)$ before linearization and discretization. The simulation results for this setup are shown in Fig. 27. The result of this simulation can hardly be distiguished from the corresponding simulation with non-linear state estimation (see Fig. 23). The assumption that the non-linear state estimation of the Kalman filter can be exchanged for a linear one has been supported by the simulation results. The performance of the linear and the non-linear state estimation is equally good as long as one stays near the point of linearization.

In order to decrease the steady state error even more, one would have to add an integral term. The addition of an integral term should be done cautiously due to issues like wind-up. With the current size of the steady state error an additional integral term is not necessary, but should be kept in mind for further improvements of the controller.

The implementation of a pseudo-force S_P in the system equations is a successful way of decreasing steady state errors. When staying near to the point of linearization, one can exchange the non-linear state estimation of the Kalman filter for a linear one without losing much accuracy. This is a way of simplifying the Kalman filter and thus cutting down on computation

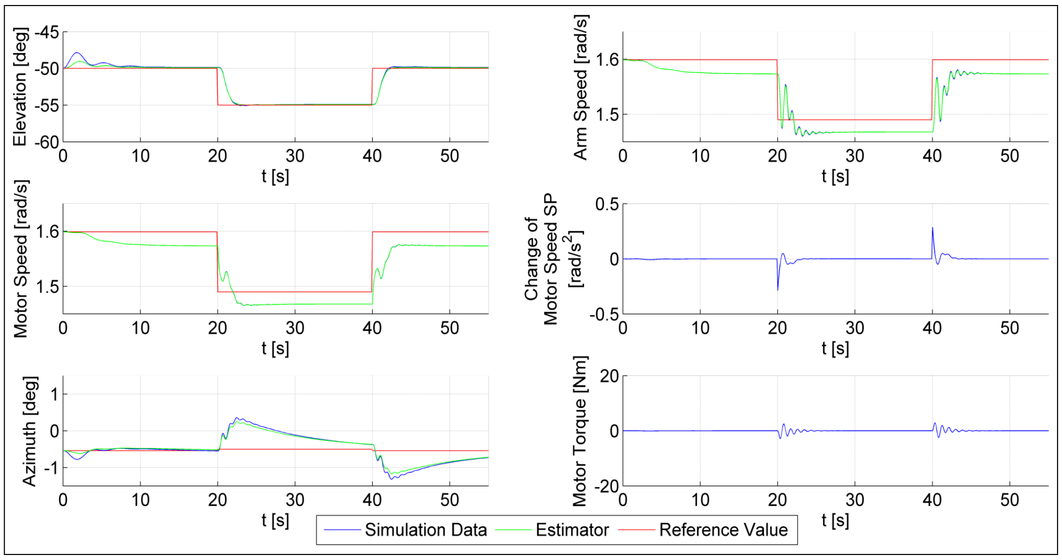

Figure 27: Elevation step response with DLQR and α changing from $-50°$ to $-55°$ and back to $-50°$. A Kalman filter with linear state estimation has been implimented. The thether length l_T has been increased by $10\,\%$ to show the effects of mismodeling. A new state S_P is added to decrease the steady state error.

time while increasing the portability of the setup by reducing its complexity.

5 Summary and Outlook

This thesis' premise was to discuss the possibility of using linear state space control to control for the experimental carousel setup of the Highwind project. The outcome was a Discrete Linear Quadratic Regulator with a Kalman filter for state estimation that has been thoroughly tested in simulation.

The thesis starts with an investigation of the modeling process of the carousel in Section 2. Section 2.1 concerns itself with Highwind's experimental carousel setup. Section 2.2 discusses how the carousel was modeled using the Lagrange formalism. The derivation of the system equations is automated with the `MATLAB` script `lagrange_formalism`. Some of the constants used in the model have been measured. The rest were roughly estimated and then fitted to experimental data by hand. The result of this fit can be seen in Fig. 7 which shows a comparison of the experimental data and the model.

Section 2.3 describes the process of finding the steady states of the model and then linearizing it at these steady states. In order to find steady states, the function `solve_steady_state_lsq` is employed. The linearization is done twice: once in continuous time and once in discrete time. While the discrete linearization is used for the `MATLAB` implementation and simulations, the continuous linearization is used to analyze the system behavior. The analysis of the carousel's open loop dynamics happens in Section 2.4. By looking at the system eigenvalues and eigenvectors one can conclude that all states except δ_M and δ_A return to their steady state values after the system is perturbed, while δ_M and δ_A converge to a constant. To establish a baseline for the performance of the controllers in Section 3, the open loop system is simulated for a step in α from $-50°$ to $-55°$ and back (see Fig. 9).

After this investigation of the open loop system, the loop is closed in Section 3. Section 3.1 describes how this is done using a Discrete Linear Quadratic Regulator. The DLQR finds an optimal feedback matrix K for the feedback law of Eq. (7) by solving the optimization problem of Eq. (9). In `MATLAB` this is implemented using the `DLQR` function. A comparison of the open loop and DLQR step responses in Fig. 13 shows how good the DLQR's performance is. To further drive this point home, the DLQR is compared to a PID controller in Section 3.2. The PID controller is tuned using the Ziegler-Nichols method with some fine-tuning by hand. As one can see in Fig. 17, the DLQR outperforms the PID. In Section 3.3 a Kalman filter is added for state estimation to complement the DLQR. The Kalman filter uses the optimal matrix M, which is chosen in a way that minimizes $\Sigma(\bar{x}_k - \hat{x}_{k|k})$. To derive M, the function `DLQE` is used (analogous to `DLQR` for matrix K). Fig. 19 and Fig. 20

show that the estimator can cope with varying degrees of system and sensor noise.

Both the DLQR and the Kalman filter rely on an accurate model. The better the model aligns with the experimental data, the better the controller will work. Even though it has been shown in Section 2.2 that the model aligns quite well with the experimental data, one has to consider the possibility of modeling errors. Section 4 discusses the possible effects of mismodeling and what measures one can take to avoid those effects. Section 4.1 shows what kind of mismodeling can lead to a steady state error in α and presents an example where the length of the tether is 10 % longer in the real system than in the model.

Section 4.2 then introduces the idea of adding a pseudo-force S_{P} to the model that is seemingly responsible for the steady state error. When this pseudo-force is implemented in the model as a new state in the right way, the Kalman filter can estimate the amount of mismodeling. This results in the correct controls and a decrease of the steady state error.

All simulations of Section 4 up to this point were done without system or sensor noise. Whether the noise compensation of the Kalman filter still works as well as before adding the additional state S_{P} is investigated in Section 4.3. When $w_{\mathrm{N,S}}$ (the component of the tuning matrix Q_{E} responsible for S_{P}) is large, the Kalman filter is able to learn the state's value very quickly. The downside however is that if $w_{\mathrm{N,S}} \gg w_{\mathrm{N}}$, the kalman filter makes the states S_{P} and $\dot{\alpha}$ responsible for most of the mismodeling, resulting in a badly tuned matrix M. By setting $w_{\mathrm{N,S}} = w_{\mathrm{N}}$ one ensures that the estimator is able to learn the value of S_{P} and compensates the system and sensor noise.

A special kind of mismodeling is discussed in Section 4.4: the Kalman filter that has been used in this thesis works with a system of non-linear difference equations to estimate x_k. Since the controller is only optimal near the point of linearization, one can use the linearized system of difference equations for the state estimation of the Kalman filter without losing much accuracy. In addition, the aforementioned pseudo-force S_{P} improves the performance of the linearized state estimation.

Now that the DLQR with Kalman filter proved to be both successful and robust in simulation, the next step is to implement it on the actual carousel. Sadly enough, this was not possible in the time frame of this thesis. However, it still seems prudent to outline how such an implementation should proceed and what measures have already been taken.

The `MATLAB` code of this thesis is ported to `C++`. The port to the carousel will proceed iteratively to make the troubleshooting process easier. For a first implementation, the DLQR and Kalman filter are set up without the additional state S_P. To decrease the complexity of the port, the Kalman filter with linear state estimation is chosen. The first experiments should still be open loop to ensure that the estimator works before closing the loop. As an additional safety measure one can output the controls of the DLQR without actually applying them to the motor in order to check if they are reasonable. After the complementary pair of DLQR and Kalman filter is set up in the carousel and produces reasonable controls and estimation, it has to be decided how to progress further. As established in Section 4 it is to be expected that there is a steady state error due to different kinds of mismodeling. To decrease the steady state error it is possible to implement the S_P state just as outlined in Section 4.2 or try an integral term if this approach does not show the desired results. Analogous to the previous iteration it should first be verified that the estimator works properly before closing the loop with a changed control setup. Judging by the simulation results it should be possible to implement a well-functioning controller this way.

References

[1] Ampyx Power: `http://www.ampyxpower.com/files/get/213.png`. visited on 12.09.2014, 12:02.

[2] Diehl, Moritz: *Lecture Notes on Optimal Estimation and Control.* Freiburg, 2014.

[3] Dietl, Heike: *Nonlinear Dynamic System Models of Tethered Flight.* Bachelor Thesis, Albert-Ludwigs-Universität Freiburg, 2014.

[4] H.J. Ferreau, B. Houska, K. Geebelen and M. Diehl: *Real-Time Control of a Kite-Carousel Using an Auto-Generated Nonlinear MPC Algorithm.* Proceedings of the IFAC Wolrd Congress, 2011.

[5] Gebre-Egziabher, Demoz: *Design and Performance Analysis of a Low-Cost Aided Dead Reckoning Navigator.* Dissertation, Standford University, 2004.

[6] K. Geebelen, A. Wagner, S. Gros, J. Swevers and M. Diehl: *Moving Horizon Estimation with a Huber Penalty Function for Robust Pose Estimation of Tethered Airplanes.* Proceedings of the 2012 American Control Conference, 2013.

[7] K. Geebelen and J. Gills: *Modelling and Control of Rotational Start-Up Phase of Tethered Aeroplanes for Wind Energy Harvesting.* Master Thesis, K. U. Leuven, 2010.

[8] G. Horn, S. Gros and M. Diehl: *Numerical Trajectory Optimization for Airborne Wind Energy Systems Described by High Fidelity.* In: M. Diehl: Airborn Wind Energy, 2013.

[9] A. Ilzhoefer, B. Houska and M. Diehl: *Nonlinear MPC of Kites under Varying Wind Conditions for a New Class of Large Scale Wind Power Generators.* International Journal of Robust and Nonlinear Control, 17, p. 1590–1599, 2007.

[10] Lunze, Jan: *Regelungstechnik 1. Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen.* Berlin, 2008.

[11] `MATLAB R2013b` Documentation: function `DLQE`, `Control System Toolbox`.

[12] `MATLAB R2013b` Documentation: function `DLQR`, `Control System Toolbox`.

[13] `MATLAB R2013b` Documentation: function `mvnrnd`, `Statistics Toolbox`.

[14] S. Skogestad, I. Postlethwaite: *Multivariable Feedback Control. Analysis and Design.* Chichester/ New York, 2006.

[15] J. Sternberg, B. Houska and M. Diehl: *A Structure Exploiting Algorithm for Approximate Robust Optimal Control with Application to Power Generating Kites.* Proceedings of the American Control Conference, 2012.

[16] Unbehauen, Heinz: *Regelungstechnik I.* Wiesbaden, 2008.

[17] M. Vukov, A. Domahidi, H.J. Ferreau, M. Morari and M. Diehl: *Auto-Generated Algorithms for Nonlinear Model Predictive Control on Long and on Short Horizons.* Proceedings of the 52nd IEEE Conference on Decision and Control, Firenze, Italy, 2013.

[18] M. Zanon, G. Horn, S. Gros and M. Diehl: *Control of Dual-Airfoil Airborne Wind Energy Systems Based on Nonlinear MPC and MHE.* Proceedings of the European Control Conference, 2014.

Appendices

A MATLAB Code

A.1 carousel_dynamics.m

```
%This script simulates and plots the system behavior for a ...
    repeated step
%from alpha0 to alpha1. It is possible to choose between a ...
    Discrete Linear
%Quadratic Regulator and a PID controller. In order to ...
    change the control
%for open loop simulations or simulate without the ...
    pseudo-force s, one has
%to change the matrix dimensions of the steady states and ...
    LQG matrices
%accordingly.

%constants
k_p = 400; %internal PID of model

%specify alpha reference values
alpha0 = -50*pi/180;
alpha1 = -55*pi/180;
alphass = (alpha0 + alpha1)/2;

%compute steady states
xopt0 = solve_steady_state_lsq(alpha0);
xopt1 = solve_steady_state_lsq(alpha1);
xoptss = solve_steady_state_lsq(alphass);
xss = [xoptss(1), xoptss(2), alphass, xoptss(3), ...
    xoptss(5), xoptss(5), 0, 0,xoptss(6), 0].';
xss0 = [xopt0(1), xopt0(2), alpha0, xopt0(3), xopt0(5), ...
    xopt0(5), 0, 0,xopt0(6), 0].';
xss1 = [xopt1(1), xopt1(2), alpha1, xopt1(3), xopt1(5), ...
    xopt1(5), 0, 0,xopt1(6), 0].';

%check if steady state were computed correctly
xdot_zero0 = carousel_lagrange(xss0, 0)
xdot_zero1 = carousel_lagrange(xss1, 0)

%set size of time step ts and number of time steps
n=700;
ts=0.1;

```

```
%get system matrices A, B and C by linerization
[A,B,C] = disc_syscreator(xss, 0, ts);

%check controllability and observability
rank_ctrb = rank(ctrb(A, B))
rank_obsv = rank(obsv(A, C))

%set PID gains
%Kp_pid = 0.3;
%Ki_pid = 0.001;
%Kd_pid = 0.08;

%dimension of state vector nx and output vector ny
nx = length(xss0);
ny = size(C, 1);

%set matrices Q and R for DLQR
penalty_delta_motor = 0;
penalty_delta_arm = 0;
penalty_alpha = 8;
penalty_beta = 0;
penalty_ddelta_motor = 0;
penalty_ddelta_arm = 0;
penalty_dalpha = 5;
penalty_dbeta = 0;
penalty_ddelta_motor_sp = 0;
penalty_s = 0;
Q = diag([penalty_delta_motor, penalty_delta_arm, ...
    penalty_alpha, penalty_beta, penalty_ddelta_motor, ...
    penalty_ddelta_arm, penalty_dalpha, penalty_dbeta, ...
    penalty_ddelta_motor_sp, penalty_s].^2);
R = (2).^2;

%set matrices G, QE and RE for DLQE (Kalman filter)
w_n = 0.001;
G = eye(nx);
variance_delta_motor = 0;
variance_delta_arm = 0;
variance_alpha = 0;
variance_beta = 0;
variance_ddelta_motor = w_n;
variance_ddelta_arm = w_n;
variance_dalpha = w_n;
variance_dbeta = w_n;
variance_ddelta_motor_sp = 0;
variance_s = w_n;
QE = diag([variance_delta_motor, variance_delta_arm, ...
    variance_alpha, variance_beta, variance_ddelta_motor, ...
    variance_ddelta_arm, variance_dalpha, variance_dbeta, ...
    variance_ddelta_motor_sp, variance_s].^2);
RE = diag([0.01,0.01].^2);
QE_sim = diag([variance_delta_motor, variance_delta_arm, ...
    0, variance_beta, variance_ddelta_motor, ...
    variance_ddelta_arm, variance_dalpha, variance_dbeta, ...
    variance_ddelta_motor_sp, 0].^2);
RE_sim = diag([0.01,0.01].^2);

%get matrix K for controls and M for Kalman filter
[K, ~, ~] = dlqr(A, B, Q, R);
[M, ~, ~, ~] = dlqe(A, G, C, QE, RE);

%Create matrices X, Xref, Xest, U and T to log simulation ...
    results
X = zeros(nx, n + 1);
Xest = X;
U = zeros(1, n);
Ysens = zeros(ny, n);
T = 0:ts:(n*ts);

%Create matrices for integral control
ERROR = zeros(ny,n);
%Ki = [0.08,0];

%Initialize logging matrices
Xest(:,1) = xss0;
X(:,1) = xss0;
Xref = zeros(nx,n);

for k = 1:n

    %get simulated sensor
    Ysens(:, k) = sensor(X(:, k)) + mvnrnd(zeros([2, 1]), ...
        RE_sim)';

    %run estimator
    yest = sensor(Xest(:, k));
    Xest(:, k) = Xest(:, k) + M*(Ysens(:, k) - yest);

    %reference value changing periodically from xss0 to xss1
    if mod(T(k),40) < 20
        xref = xss0;
    else
        xref = xss1;
    end

    %update reference value for delta_motor and delta_arm
    if k > 2
        xref(1) = Xest(1, k - 1) + Xest(5, k)*ts;
```

```
          xref(2) = Xest(2 ,k - 1) + Xest(6, k)*ts;

      end

      %log reference value
      Xref(:, k) = xref;

      %apply full state feedback controller
      U(k) = -K*(Xest(:, k)-Xref(:, k));

      %apply PID controller
%       ERROR(:, k + 1) = ERROR(:, k) + [X(3, k) - xref(3); ...
    X(4, k) - xref(4)];
%       bar_ak = (X(3, k) - Xref(3, k));
%       if k > 1
%           bar_akminus1 = X(3, k - 1) - Xref(3, k - 1);
%           U(k) = -(Kp_pid*(bar_ak) + Ki_pid*ts*ERROR(1, k) ...
    + Kd_pid*(bar_ak - bar_akminus1)/ts);
%       else
%           U(k) = -(Kp_pid*bar_ak + Ki_pid*ts*ERROR(1, k));
%       end

      %simulate system
      X(:, k + 1) = integrator(X(:,k),U(k),ts) + ...
          mvnrnd(zeros(nx, 1), QE_sim)';

      %run estimator
      Xest(:, k + 1) = integrator(Xest(:, k), U(k), ts); ...
          %non-linear
      %Xest(:, k + 1) = xss + A*(Xest(:, k) - xss) + B*U(k); ...
          %linear

end

%plots
figure(1);
clf;

%plot elevation
ax(1) = subplot(3, 2, 1);
hold on;
plot(T, 180/pi*X(3, :), 'b');
plot(T, 180/pi*Xest(3, :), 'g');
axis([0 55 -60 -45])
xlabel('t [s]')
ylabel('Elevation [deg]')
plot(T(1:end-1), 180/pi*Xref(3, :), 'r')
legend('Simulation Data', 'Estimator', 'Reference Value')
grid on;
```

```

%plot arm speed
ax(2) = subplot(3, 2, 2);
hold on;
plot(T, X(6, :));
plot(T(1:end - 1), Xref(6, :), 'r')
plot(T, Xest(6, :), 'g');
grid on;
axis([0 55 1.45 1.65])
xlabel('t [s]')
ylabel('Arm Speed [rad/s]')
legend('Simulation Data', 'Estimator', 'Reference Value')

%plot motor speed
ax(3) = subplot(3, 2, 3);
hold on;
grid on;
axis([0 55 1.45 1.65])
plot(T, X(5, :));
plot(T(1:end - 1), Xref(5, :), 'r')
plot(T, Xest(5, :), 'g');
xlabel('t [s]')
ylabel('Motor Speed [rad/s]')
legend('Simulation Data', 'Estimator', 'Reference Value')

%plot u
ax(4) = subplot(3, 2, 4);
hold on;
grid on;
plot(T(1:end - 1), U);
xlabel('t [s]')
ylabel('Change of Motor Speed SP [rad/s^2]')

%plot azimuth
ax(5) = subplot(3, 2, 5);
hold on;
grid on;
axis([0 55 -1.5 1.5])
plot(T(1:end - 1), Xref(4, :)*180/pi, 'r')
plot(T, X(4, :)*180/pi, 'b')
plot(T, 180/pi*Xest(4, :), 'g')
xlabel('t [s]')
ylabel('Azimuth [deg]')
legend('Simulation Data', 'Estimator', 'Reference Value')

%plot motor torque
ax(6) = subplot(3, 2, 6);
hold on;
grid on;
```

```
plot(T, -k_p*(X(5, :) - X(9, :)), 'b')
axis([0 55 -20 20])
xlabel('t [s]')
ylabel('Motor Torque [Nm]')

linkaxes(ax, 'x')
```

A.2 carousel_dynamics_SE.m

```
function [T,X,U] = carousel_dynamics_SE(carouselspeed_sp, ...
    timestamp)

%This function takes the experimental data for the ...
    ddelta_motor setpoint as
%controls and simulates the system.

%set time step and number of steps
ts = 0.1;
n = 1000;

%set initial conditions
ddelta0 = 1.44;
xss0 = [0, 0, -57*pi/180, 0, ddelta0, ddelta0, 0, 0]';

%initialize matrices
nx = length(xss0);
X = zeros(nx, n + 1);
X(:, 1) = xss0;
T = 0:ts:(n*ts);

%resample data
carouselspeedsetpointresamp = ...
    interp1(timestamp,carouselspeed_sp,T,'spline');

%set U
U = carouselspeedsetpointresamp(200:end - 1);

for k = 1:(n - 200)
    x = X(:, k);
    u = U(k);
    x = integrator(x, u, ts);
    X(:, k + 1) = x;
end

end
```

A.3 carousel_lagrange.m

```
function [xdot] = carousel_lagrange(x, u)

%This function contains the non-linear system equations. ...
    In order to change the control
%for open loop simulations or simulate without the ...
    pseudo-force s or implement mismodeling one has
%to change xdot or the constants accordingly.

%state vector
delta_motor = x(1);
delta_arm = x(2);
alpha = x(3);
beta = x(4);
ddelta_motor = x(5);
ddelta_arm = x(6);
dalpha = x(7);
dbeta = x(8);
ddelta_motor_sp = x(9);
s = x(10);

%control
dddelta_motor_sp = u;

%constants
m_ball = 0.57;
l_tether = 1.82;
I_arm = 200;
I_motor = 0.015121/(14.86*(3/2));
I_tether = 4.5*1/3*m_ball*l_tether;
r_arm = 2.05;
k_beltspring = 11419;
c_beltdampening = 0.1;
my_shaft = 0;
g = 9.81;
c_w = 0.5;
roh_air = 1.184;
A_ball = 36*10^(-4);
my_alpha_LA = 1;
my_beta_LA = 30;
k_p = 400;
tau = 100;

%Since the expression for ddq derived by ...
    lagrange_formalism.m is too big
```

```
%to be usefully presented on paper, ddq = [dddelta_motor; ...
    dddelta_arm;
%ddalpha; ddbeta] in the following expression for xdot.

%system equations
xdot = [
ddelta_motor;
ddelta_arm;
dalpha;
dbeta;
dddelta_motor;
dddelta_arm;
s + ddalpha;
ddbeta;
ddelta_motor_sp;
-s/tau];

end
```

A.4 cont_syscreator.m

```
function [A, B, C] = cont_syscreator(x0, u0)

%This function returns the matrices A, B and C of a ...
    continous linearization
%of carousel_lagrange at x0, u0.

A = fingrad(@(x)carousel_lagrange(x,u0), x0, 1e-6);
B = fingrad(@(u)carousel_lagrange(x0,u), u0, 1e-6);
C = fingrad(@sensor, x0, 1e-6);
Contr = ctrb(A ,B);

if rank(Contr) < rank(A)
    disp('Syscreator: System not controllable.')
end

end
```

A.5 disc_syscreator.m

```matlab
function [A, B, C] = disc_syscreator(x0, u0, ts)

%This function returns the matrices A, B and C of a ...
    discrete linearization
%of carousel_lagrange at x0, u0.

A = fingrad(@(x)integrator(x,u0,ts), x0, 1e-6);
B = fingrad(@(u)integrator(x0,u,ts), u0, 1e-6);
C = fingrad(@sensor, x0, 1e-6);
Contr = ctrb(A, B);

if rank(Contr) < rank(A)
    disp('Syscreator: System not controllable.')
end

end
```

A.6 fingrad.m

```matlab
function [JF] = fingrad(F, x0, h)

%Numerical calculation of the Jacobian of function F(x), ...
    evaluated at x0 with accuracy h

F0 = F(x0);
[nrow,ncol] = size(F0);
assert(ncol == 1, 'need a column vector')
JF = zeros(nrow, length(x0));
e = eye(length(x0));

for j = 1:length(x0)
    JF(:, j) = (F(x0 + h*e(:, j)) - F0)/h;
end

end
```

A.7 integrator.m

```
function xnext = integrator(x0, u0, dt)

%This function integrates carousel_lagrange numerically ...
    with initial value
%x0, u0 over time dt.

options = odeset('RelTol', 5e-14);
[~, Y] = ode15s(@(t, x)carousel_lagrange(x, u0), [0, dt], ...
    x0, options);
xnext = Y(end, :)';

end
```

A.8 lagrange_formalism.m

```
%This script derives xdot2 = [dddelta_motor, dddelta_arm, ...
    ddalpha, ddbeta]'
%for the carousel model using the Lagrange formalism.

%ground frame: NED centered at the carousel axis at arm level

syms M_motor %torque of motor [Nm]
syms p_ball real %position of the ball in ground frame, [m]
syms r_arm real %length of arm [m]
syms l_tether real %length of tether [m]
syms m_ball real %mass of the ball [kg]
syms I_arm real %moment of inertia of carousel [kg m^2]
syms I_motor real %moment of inertia of motor [kg m^2]
syms I_tether real %moment of intertia of the tether [kg m^2]
syms p_sensor real %position of line angle sensor in ...
    ground frame [m]
syms delta_arm ddelta_arm dddelta_arm real %angle of arm ...
    rotating around carousel axis with 0 aligned with ...
    east, positive for counter-clockwise rotation [rad]
syms delta_motor ddelta_motor dddelta_motor real %angle of ...
    motor rotating around carousel axis with 0 aligned ...
    with east, positive for counter-clockwise rotation [rad]
syms alpha dalpha ddalpha real %elevation angle of rope, ...
    positive above horizontal [rad]
syms beta dbeta ddbeta real %azimuth angle of rope, ...
    positive if ball is ahead of the arm [rad]
syms k_beltspring real %spring constant of the belt [Nm/rad]
syms c_beltdampening real %dampening coefficient of the ...
    belt [Nm/rad^2]
syms my_shaft real %friction constant of motor shaft [Nm/rad]
syms my_beta_LA real %friction constant of LA-sensor in ...
    beta direction [Nm/rad]
syms my_alpha_LA real %friction constant of LA-sensor in ...
    alpha direction [Nm/rad]
syms g real %gravitational constant
syms roh_air real %density of the air
syms A_ball real %area of the ball
syms c_w %air friction constant
syms k_p %proportional constant of the motor control [-]
syms ddelta_motor_sp %setpoint for the motor controller ...
    [rad/s]

%defining generalized coordinates
q = [delta_motor; delta_arm; alpha; beta];
dq = [ddelta_motor; ddelta_arm; dalpha; dbeta];
ddq = [dddelta_motor; dddelta_arm; ddalpha; ddbeta];

%defining geometric subexpressions
p_sensor = r_arm*[sin(delta_arm);cos(delta_arm);0];
p_ball = p_sensor + l_tether*[cos(alpha)*sin(delta_arm + ...
    beta); cos(alpha)*cos(delta_arm + beta); -sin(alpha)];

%velocity of the ball
v_ball = jacobian(p_ball, q)*dq;

%kinetic energy
KE = 0;
KE = KE + 1/2*m_ball*(v_ball.'*v_ball);
KE = KE + 1/2*I_arm*ddelta_arm^2;
KE = KE + 1/2*I_motor*ddelta_motor^2;
KE = KE + 1/2*I_tether*(dalpha^2+dbeta^2);

%potential energy
PE = m_ball*g*(l_tether*sin(alpha));
PE = PE + 1/2*k_beltspring*(delta_motor-delta_arm)^2;

%lagrangian
L = KE - PE;

%define motor control
M_motor = -k_p*(ddelta_motor - ddelta_motor_sp);

%generalized forces
J = jacobian(p_ball,q);
f_airfriction = -1/2*roh_air*A_ball*c_w*v_ball*norm(v_ball);
M_beltfriction = -c_beltdampening*(ddelta_motor - ddelta_arm);
```

```
M_shaftfriction = -my_shaft;
gen_airfriction = J.'*f_airfriction;
M_LA_friction_alpha = -my_alpha_LA*dalpha;
M_LA_friction_beta = -my_beta_LA*dbeta;
gen_forces = [14.86*(3/2)*M_motor+M_beltfriction; ...
    -M_beltfriction + M_shaftfriction; ...
    M_LA_friction_alpha; M_LA_friction_beta] + ...
    gen_airfriction;

%implicit ODE
Lq = jacobian(L, q).';
Ldq = jacobian(L, dq);
Ldqt = jacobian(Ldq, q)*dq + jacobian(Ldq, dq)*ddq;
impl_ode = Ldqt - Lq - gen_forces;

%explicit ODE
sol = solve(impl_ode(1), impl_ode(2), impl_ode(3), ...
    impl_ode(4), dddelta_motor, dddelta_arm, ddalpha, ddbeta);
xdot2 = simplify(expand([sol.dddelta_motor; ...
    sol.dddelta_arm; sol.ddalpha; sol.ddbeta]));
```

A.9 sensor.m

```
function [y] = sensor(x)

%This is the sensor function.

y = [x(3), x(4)]';

end
```

A.10 solve_steady_state_lsq.m

```
function xopt = solve_steady_state_lsq(alpha_ref)

%This function solves for steady states by solving a least ...
    squares problem.

options = optimoptions('lsqnonlin', 'TolFun', ...
    1e-25,'TolX', 1e-25);
xopt = lsqnonlin(@(x)g(x, alpha_ref), [0, 0, -0.5*pi/180, ...
    1.6, 1.6, 1.6], [], [], options);

```

```
end

function ret = g(dv, alpha_ref)

delta_motor = dv(1);
delta_arm = dv(2);
alpha = alpha_ref;
beta = dv(3);
ddelta_motor = dv(4);
ddelta_arm = dv(5);
ddelta_sp = dv(6);
u = 0;

x = [delta_motor; delta_arm; alpha; beta; ddelta_arm; ...
    ddelta_arm; 0; 0; ddelta_sp; 0];
xdot = carousel_lagrange(x,u) - [ddelta_arm; ddelta_arm; ...
    0; 0; 0; 0; 0; 0; 0; 0];
ret = xdot;

end
```

A.11 step_response_experiment.m

```
%This script compares experimental data of a step response ...
    to the
%corresponding simulation data in order to judge how good ...
    the model fits
%the real system. The .nc files can be downloaded from the ...
    git respository
%https://github.com/thilobro/state_space_control_thesis
%ATTENTION: to run this script successfully, ...
    carousel_lagrange has to be
%changed to ddelta_motor_sp control

%read in experimental data
ncid1 = netcdf.open('siemensSensorsData6.nc');
timestamp1 = netcdf.getVar(ncid1, 0);
carouselspeed_sp = netcdf.getVar(ncid1, 6);
ncid2 = netcdf.open('lineAngleSensor2Data6.nc');
timestamp2 = netcdf.getVar(ncid2, 0);
azimuth = netcdf.getVar(ncid2, 1);
elevation = netcdf.getVar(ncid2, 2);
ncid3 = netcdf.open('armboneLisaSensorsData6.nc');
timestamp3 = netcdf.getVar(ncid3, 0);
qractualspeed = netcdf.getVar(ncid3, 3);

```

```
%resample data
qractualspeed_resamp = interp1(timestamp3, qractualspeed, ...
    timestamp1, 'spline');
elevation_resamp = interp1(timestamp2, elevation, ...
    timestamp1, 'spline');
azimuth_resamp = interp1(timestamp2, azimuth, timestamp1, ...
    'spline');

%get simulation data
[T, X, U] = carousel_dynamics_SE(carouselspeed_sp, ...
    timestamp1);

%shift time axis in order to align simulation and ...
    experimental data
Tshift = 20;

%plot data
figure(4);
clf;
ax(1) = subplot(3, 1, 1);
plot(timestamp1, carouselspeed_sp, 'r')
hold on;
plot(timestamp1, qractualspeed_resamp, 'g')
plot(T + Tshift, X(6, :), 'b')
axis([60 100 1.4 1.8]);
xlabel('t [s]')
ylabel('Arm/Motor Speed [rad/s]')
legend('Motor Speed SP','Measured Arm Speed','Simulated ...
    Arm Speed')
ax(2) = subplot(3, 1, 2);
plot(T + Tshift, 180/pi*X(3, :), 'b');
hold on;
plot(timestamp1, 180/pi*elevation_resamp, 'g')
axis([60 100 -70 -30]);
legend('Simulated Elevation','Measured Elevation')
xlabel('t [s]')
ylabel('Elevation [deg]')
ax(3) = subplot(3, 1, 3);
plot(T + Tshift, 180/pi*X(4, :), 'b')
hold on;
plot(timestamp1, azimuth_resamp, 'g')
axis([60 100 -3 2]);
legend('Simulated Azimuth', 'Measured Azimuth')
xlabel('t [s]')
ylabel('Azimuth [deg]')

linkaxes(ax, 'x')
```