Otto-von-Guericke-Universität Magdeburg

Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungstechnik

Masterarbeit

Meine Master Thesis

Heinemann, Hannes
22. Oktober 2015

Erstprüfer:	Rolf Findeisen
Zweitprüfer:	Erik B.
Betreuer:	Pablo

Aufgabenstellung

Thema

Zeitraum: 06 - 11

Das ist meine Aufgabenstellung

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Fast MPC	Primal Barrier Interior-Point Methode nach [WB10]
KS	Kreisselmeier-Steinhauser
log barrier	logaritmische Straffunktion
MPC	Modellprädiktive Regelung
QCQP	Quadratic Constrained Quadratic Program
SOCP	Second Order Cone Problem

Symbolverzeichnis

1 Einleitung

1.1 TODO

1.1.1 Format

- Zitierstil
- Papierformat
- Schriftgröße, Schriftart ...
- Zeilenabstand

1.1.2 Inhalt

- Beispiele helfen um den Algorithmus und das Bilden der Matrizen zu verstehen, falls noch Seiten gefüllt werden müssen ;)
- "Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization" hier findet man in kapitel 5 ein paar hinweise zu der Regularization
- "Primal Barrier Methods For Linear Programming" Kapitel 3: schwierigkeiten bei der Wahl des StartKappa und Startwerte, wie man es günstig wählt Bei einem guten Startwert, kann der Barrierparameter entsprechend klein gewählt werden, ohne das Probleme bei der Konvergenz gegen ein Optimum auftreten + S.29 Tested kappa [0.0001, 1] multiplied with (c'x)/n Value of Costfunction durch dimension to have the value of barrier function in same order as the costvalue + lienear modifications to the barrier function + where to stop + The Nullspace Methods + Hinweise zu Cholesky factorization

- Ausprobieren Kappa kleiner zu wählen, wenn in ermittelter Suchrichtung keine Verbesserung des Funktionswertes möglich ist
- Tabelle mit den Ergebnissen der gelösten Tests

[MM99] Modellprädiktive Regelung (MPC)

2 Problemstellungen

Dieser Abschnitt befasst sich mit verschiedenen hardware- und software technischen Problemen, die beachtet oder gelöst werden müssen.

2.1 Hardware - Microcontroller

Auf einem Microcontroller treten verschiedene Probleme auf, die bei der Programmierung mit dem PC keine Schwierigkeiten machen.
32bit Gleitkommazahlen
kaum Verwendung von Bibliotheken
Schwierigkeiten beim Debuging
...

2.2 Software

Condegeneration
Speicherallozierung
Laufzeit optimierung
...

2.3 Mathematisch

Kein Feasibl Punkt
positiv semidefinit
...

3 Grundlagen

- generalized inequalities
- Originalalgorithmus von Wang und Boyd [WB10]

3.1 Modellprädiktive Regelung

Viele Regelungskonzepte werden in der Industrie angewand. Ein Ansatz ist die optimale Reglung, bei der online oder auch offline das Reglungsgestz so bestimmt wird das eine gewisse Kostenfunktion minimal ist. Besteht die Möglichkeit dieses Reglungsgestz offline zu berechnen ist alles schön und gut. Aber in vielen lässt muss dieses Gesetz für jeden Zeitschritt erneut ausgerechnet werden und somit online zur Laufzeit des Systems ständig neu berechnet werden. Nun gibt es Syteme bei denen die eine schnelle Reaktionszeit des Reglers erforden, was unter Umständen dazu führen kann, dass möglichst effiziente Methoden benutzt werden.
Eine spezielle Form der optimalen Reglung ist die Modellprädiktive Regelung (MPC).
AUS OTTOCAR-BERICHT GEKLAUT
Bei der modellprädiktiven Regelung (MPC) handelt es sich um eine Form der optimalen Regelung, bei der wiederholt eine Berechnung der optimalen Steuerung für ein System ausgehend von dessen aktuellem Zustand stattfindet. In vielen Bereichen finden MPCs immer häufiger Anwendung, da sie eine direkte Berücksichtigung von Beschränkungen erlauben und eine Form des strukturierten Reglerentwurfs ausgehend von der modellierten Systemdynamik darstellen. Dabei kann durch die geeignete Wahl der Kostenfunktion und deren Wichtungsparametern die Güte des Reglers gezielt beeinflusst werden. Allerdings ergeben sich auch Schwierigkeiten bei der Verwendung von MPCs. Zum einen ist die Konvergenz der Optimierung gegen einen optimalen Wert für die Optimierungsvariablen und die Stabilität des geschlossenen Kreises insbesondere bei nichtlinearen Systemmodellen

oft nur schwierig nachweisbar und zum anderen stellt das wiederholte Lösen des meist hochdimensionalen Optimierungsproblems während der Laufzeit in genügend schneller Geschwindigkeit eine große Herausforderung dar.
Es ergibt sich ein Optimierungsproblem:
m realen Anwendungsfall des oTToCAR-Projekts eignet sich eine Systemdarstellung in zeitdiskreter Form ([Ada09]), bei der die Lösung des Optimierungsproblems weniger komplex ist und die ebenfalls zeitdiskreten Messwerte vom realen System weniger kompliziert integriert werden können. Demnach sind die diskretisierten Systemgleichungen wie folgt gegeben:

$$\begin{aligned} \boldsymbol{x}(k+1) &= \boldsymbol{f}\left(\boldsymbol{x}(k), \boldsymbol{u}(k)\right) \\ \boldsymbol{y}(k) &= \boldsymbol{g}\left(\boldsymbol{x}(k)\right) \end{aligned}$$

mit den nichtlinearen mehr Text Funktionen $\boldsymbol{f}(\cdot)$ und $\boldsymbol{g}(\cdot)$, wobei

$$\begin{aligned} \boldsymbol{x}(k) &\in \mathcal{X} \subset \mathbb{R}^n \\ \boldsymbol{u}(k) &\in \mathcal{U} \subset \mathbb{R}^m \\ \boldsymbol{y}(k) &\in \mathcal{Y} \subset \mathbb{R}^r \end{aligned}$$

Ausgehend vom aktuell gemessenen Zustand $\boldsymbol{x}(k)$ des zu regelnden Systems, der wenn nicht messbar geschätzt werden muss, wird anhand des bekannten Systemmodells das zukünftige Systemverhalten

$$\boldsymbol{x_p} = \{\boldsymbol{x}(k+1), \ldots, \boldsymbol{x}(k+n_p)\}$$

bis zum Prädiktionshorizont n_p unter der Optimierung einer Sequenz von Eingängen

$$\boldsymbol{u} = \{\boldsymbol{u}(k), \ldots, \boldsymbol{u}(k+n_c-1)\}$$

bis zum Stellhorizont n_c vorhergesagt. Aus der gefundenen optimalen Eingangssequenz $\boldsymbol{u}^*$ wird der erste Eintrag $\boldsymbol{u}^*(k)$ auf das zu regelnde System angewandt. Im nächsten Zeitschritt kann der neue Zustand gemessen bzw. geschätzt werden und die Optimierung beginnt von neuem. Ziel dabei ist es einer Referenztrajektorie $\boldsymbol{x_r}$ zu folgen.
Für das an jedem Zeitschritt k zu lösende Minimierungsproblem wurde die benötigte Kostenfunktion J in quadratische Form mit $\boldsymbol{x_p}$ und $\boldsymbol{u}$ als Optimierungsvariablen aufge-

stellt:

$$\min_{\boldsymbol{x_p},\boldsymbol{u}} J := \sum_{i=k+1}^{k+n_p} \left[\boldsymbol{x}_p(i) - \boldsymbol{x}_r(i)\right]^T \boldsymbol{Q}_i \left[\boldsymbol{x}_p(i) - \boldsymbol{x}_r(i)\right] + \sum_{j=k}^{k+n_c-1} \boldsymbol{u}^T(j)\boldsymbol{R}_j\boldsymbol{u}(j)$$

$$s.t.\ \boldsymbol{x_p}(i+1) = \boldsymbol{f}\left(\boldsymbol{x_p}(i), \boldsymbol{u}(i)\right), \quad i = k, ..., k+n_c-1$$

$$\boldsymbol{x_p}(i+1) = \boldsymbol{f}\left(\boldsymbol{x_p}(i), \boldsymbol{u}(k+n_c-1)\right), \quad i = k+n_c, ..., k+n_p-1$$

Mit den Vektoren

$$\boldsymbol{x}_p(k) = \left[\boldsymbol{x}_p(k+1 \mid k), \ldots, \boldsymbol{x}_p(k+n_p \mid k)\right]^T$$
$$\boldsymbol{x}_r(k) = \left[\boldsymbol{x}_r(k+1), \ldots, \boldsymbol{x}_r(k+n_p)\right]^T$$
$$\boldsymbol{u}(k) = \left[\boldsymbol{u}(k), \ldots, \boldsymbol{u}(k+n_c-1)\right]^T$$

und den dazugehörigen positiv definiten Wichtungsmatrizen $\boldsymbol{Q}_i \in \mathbb{R}^{n\times n}$ $(i = 1, ..., n_p)$ und $\boldsymbol{R}_j \in \mathbb{R}^{m\times m}$ $(j = 0, ..., n_c - 1)$. Weiterhin lässt sich das Optimierungsproblem um einfache Beschränkungen der Eingänge

$$\boldsymbol{u}_{min} \leq \boldsymbol{u}(i) \leq \boldsymbol{u}_{max}, \quad i = k, ..., k+n_c-1$$

und Zustandsbeschränkungen der Form

$$\boldsymbol{A}\boldsymbol{x}_p(i) \leq \boldsymbol{b} \quad i = k+1, ..., k+n_p$$

erweitern.

3.2 Boyds Grundlagen

3.2.1 Konvexität

3.2.2 Kontinuierliche Differenzierbarkeit

3.2.3 Konvexex Optimierungsproblem

$$\min f_0(x) \qquad \text{s.t.} f_i(x) \leq 0, \quad i = 1, \ldots, m \qquad Ax = b \tag{3.1}$$

ist ein konvexes Optimierungsproblem mit dem sich diese Arbeit beschäfigt, wenn die Funktionen $f_0, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$ konvexe Funktionen und zweimal kontinuierlich differenzierbar sind, vgl. [BV04].

3.2.4 Strikte Feasiblität

Es sei ein beschräntes Optimierungsproblem gegeben. Dieses Optimierungsproblem ist strikt feasibl, wenn ein $x \in \mathcal{D}$ existiert, dass alle Gleichungsnebenbedingnung und Ungleichungsnebenbedingungen erfüllt, vgl. [BV04].

3.2.5 Cholesky Zerlegung

Beides siehe [SK11]
Bedingungen
positiv definit
Ist eine Matrix A positiv definit und symmetrisch, so kann sie in der Form $A = LL^T$ zerlegt werden. Wobei L eine untere Dreicksmatrix und deren transponierte L^T dementsprechend eine obere Dreiecksmatrix ist. Die Elemente $l_{11}, \ldots, l_{nn}$ der Matrix

$$L = \begin{bmatrix} l_{11} & 0 & \ldots & 0 \\ l_{21} & l_{22} & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ l_{n1} & l_{n2} & \ldots & l_{nn} \end{bmatrix}$$

lassen sich dazu wir folgt berechnen

3.2.6 Newton Methode

3.2.7 Interior-Point Methode

Diese Arbeit befasst sich mit mit der Implementierung eines Lösungsalgorithmus für das konvexe Optimierungsproblem 3.1, bei denen Ungleichungsnebenbedingungen berücksichtigt werden müssen. Dazu wurde eine Interior-Point Methode implementiert, deren Grundlagen in diesem Abschnitt beschrieben werden. Es wird vorausgesetzt, dass das Optimierungsproblem strikt feasibl ist.
Die Interior-Point Methode löst das Optimierungsproblem 3.1 indem die Newton Methode auf ein System von linear gleichungsbeschränkten Optmierungsproblemen angewandt wird.

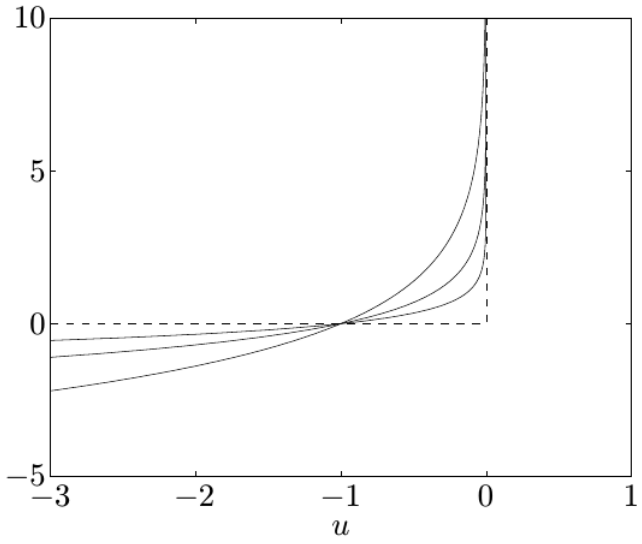

Abbildung 3.1: Verschiende Funktionen für verschiedene κ [BV04]

3.2.8 Primal-Dual vs. Primal

Es gibt Methoden die das Duale Optimierungsproblem und deren Variablen zur Lösung des ungleichungsnebenbedingten Systems heranziehen. Diese Methoden nennt man primal-dual Methoden. In dieser Arbeit wurde allerdings mit einer primal Methode gearbeitet, der sogenannten barrier Methode.
Es wäre schön, hier ein paar Vor- und Nachteile lesen zu können.

3.2.9 Grundlagen logarithmic barrier Methode

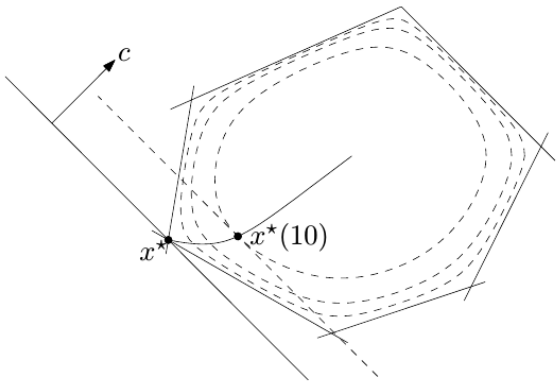

Abbildung 3.2: Central 0 path für verschiedene κ [BV04]

3.3 Ungleichungsnebenbedingungen

Wie schon angeführt, ist ein Vorteil des MPC Ansatzes die direkte Berücksichtigung von Nebenbedingungen. Das gilt sowohl für Gleichungsnebenbedingungen als auch Ungleichungsnebenbedingungen. Bei dem später beschriebenen Algorithmus werden ausschließlich die zur Prädiktion nötigen linearen Dynamiken eines zu regelnden Systems als Gleichungsnebenbedingungen der Form

$$x_{k+1} = Ax_k + Bu_k$$

berücksichtigt.
Zusätzlich zu der Einhaltung der Systemdynamik wird allgemein gefordert, dass sowohl Zustände als auch Stellgrößen aus erlaubten Mengen von Werten

$$\boldsymbol{x}(k) \in \mathcal{X} \subset \mathbb{R}^n$$
$$\boldsymbol{u}(k) \in \mathcal{U} \subset \mathbb{R}^m$$

stammt ([MRRS00]). Dieses wird, je nach Eigenschaften von $\mathcal{X}$ und $\mathcal{U}$, mit Hilfe von Ungleichungsnebenbedingungen in verschiedenen Formen, die im folgenden Abschnitt kurz erläutert werden, sichergestellt. Begonnen wird mit einfachen Ungleichungsnebenbedingungen in linearer Form, es folgen spezielle Formen nichtlinearer Ungleichungsnebenbedingungen, die häufig Anwendung finden.

3.3.1 Lineare Ungleichungsnebenbedingungen

In Regelungsproblemen treten häufig Fälle auf, in denen für einzelne oder alle Zustände beispielsweise aus Sicherheitsgründen bestimmte Werte nicht unter- bzw. überschritten werden dürfen. Änliches git auch Systemeingänge bei denen aus beispielsweise technischen Gründen nur nur Werte aus einem bekannten Intervall auf das System gegeben werden können. Solche einfachen Zustandsbeschränkungen der Form

$$x_{min} \leq x(k) \leq x_{max}, \quad k = 0, 1, \ldots$$
$$u_{min} \leq u(k) \leq u_{max}, \quad k = 0, 1, \ldots$$

werden auch als box constraints bezeichnet ([WB10]). Dabei gilt $x_{min}, x_{max} \in \mathbb{R}^n$ und $u_{min}, u_{max} \in \mathbb{R}^m$.

Mixed constraints guck mal muao mpc wegen beschreibung.
Alle so beschriebenen linearen Ungleichungsnebenbedingungen lassen sich als ein System von l linearen Ungleichungen

$$F_x x(k) + F_u u(k) \leq f, \quad k = 0, 1, \ldots \tag{3.2}$$

mit $F_x \in \mathbb{R}^{l \times n}, F_u \in \mathbb{R}^{l \times m}$ und $f \in \mathbb{R}^l$ zusammenfassen.

3.4 Quadratic Constraints

Was genau, wofür QC?

3.5 Second Order Cone Constraints

Was genau, wofür SOCC?

3.6 Soft Constraints

Manchmal liegen die Lösungen der Optimierungsprobleme an Rande der feasible Regionen und so kann es dazu kommen, dass durch Störungen auf das System, Lösungsvektoren nicht mehr feasibl sind. Da es besser ist, den Algorithmus nicht abzubrechen sondern mit einer nicht erlaubten Punkt weiter zu arbeiten, gibt es die Möglichkeit Soft Constraints einzuführen. Dieses weichen wie der Name schon sagt, die harten Ungleichungsnebenbedingungen auf, sodass diese Verletzt werden können. diese Verletzung geht als zusätzliche Strafe mit in die Kostenfunktionen ein. Bei richtiger Wichtung der Verletzung, sorgt er Optimierungsalgorithmus dafür, dass diese Grenzen nur verletzt werden, wenn keiner Punkt mehr feasibl ist.
Im folgenden Abschnitt werden 2 Möglichkeiten erlärt, solche Soft Constraints einzuführen, die ich im Zuge meiner Masterarbeit bzgl der auswirkung des nichteinhaltens der exact penalty function untersuche und vergleiche.

3.6.1 Soft Constraints mittels Einführung zusätzlicher Slackvariablen

Die übliche Methode, weil auch die kofortabelste, statt harten Ungleichungsnebenbedingungen weichen Ungleichungsnebenbedingungen (oder Soft Constraints) zu verwenden,

ist die Einführung von Schlupfvariablen (slack variables) ([Ric13]). Das lässt sich unkompliziert durch die Erweiterung des Stellgrößen Vektors um eine zusätzliche Variable $s(k)$ ohne phsikalische Bedeutung zu $\begin{bmatrix} u^T(k) & s(k) \end{bmatrix}^T$ für jeden Zeitschritt k durchführen. Analog zu Ungleichung 3.2 lassen sich dann die soft constriants als

$$\begin{bmatrix} \tilde{F}_x \\ 0 \end{bmatrix} x(k) + \begin{bmatrix} \tilde{F}_u & -1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} u(k) \\ s(k) \end{pmatrix} \leq \begin{bmatrix} \tilde{f} \\ 0 \end{bmatrix}, \quad k = 0, 1, \ldots \tag{3.3}$$

formulieren. $s(k)$ ist dabei die maximale Verletzung der der Ungleichungsnebenbedingungen im $k$ten Zeitschritt, $s(k) = 0$ sofern alle Ungleichungsnebenbedingungen erfüllt sind und wird in der Kostenfunktion entsprechend bestraft, sodass $s(k) \neq 0$ nur dann, wenn andernfalls kein valider Schritt möglich ist. Es ist zu erwähnen dass zur Berrücksichtigung der Soft Constraints in dieser Formulierung keine besondere Berücksichtigung im Algorithmus vorgesehen sein muss. Lediglich die Formulierung der Matrizen und Dimensionen für das zu lösende Optimierungsproblem müssen angepasst werden.

3.6.2 Soft Constraints mittels Einführung eines Strafterms in Form der Kreisselmeier-Steinhauser Funktion

In [Ric13] wird eine weitere Möglichkeit vorgeschlagen, um lineare Ungleichungsnebenbedingungen als soft constraints zu nutzen. Der Artikel ist darauf fokussiert die soft constraints zugeschnitten zu dem Algorithmus aus [WB10] einzuführen. Dazu wird die sogenannte Kreisselmeier-Steinhauser (KS) Funktion

$$KS'(g(x)) = \frac{1}{\rho} \log \left[\sum_j \exp(\rho g_j(x)) \right]$$

nach [Kre79] [darf ich das?] genutzt. Um Probleme mit zu starkem exponentiellen Wachstum zu vermeiden nimmt man genauer gesagt eine equivalente Form der KS Funktion, bei der diese Probleme nicht auftreten:

$$KS(g(x)) = g_{max}(x) + \frac{1}{\rho} \log \left[\sum_j \exp(\rho(g_j(x) - g_{max(x)})) \right]. \tag{3.4}$$

Ähnlich wie bei die log barrier den Übergang von feasibl zu infeasibl annähert, dient die KS Funktion dazu, die Nichtkontinuität des Gradienten zu durch eine kontinuierliche Funktion zu approximieren. Analog zu der Reduzierung von κ führt eine Erhöhung von

ρ dazu, dass man sich der exakten Funktion nähert.
Um mit der KS Funktion soft constraints in den Algorithmus von Boyd zu integrieren sind folgende Schritte notwendig. Soft constraints analog zu 4.8 formulieren

$$\tilde{P}z \leq \tilde{h}.$$

Sodass der Strafterm

$$\sum_i \max\{0, \tilde{p}_i^T z - \tilde{h}_i\}$$

mit Hilfe der KS Fnuktion als

$$\theta(z) = \sum_i \frac{1}{\rho} \log\left[1 + \exp(\rho(\tilde{p}_i^T z - \tilde{h}_i))\right]$$

formuliert werden kann.

$$e_i^+$$
$$e_i^-$$

für Gradient der Straffunktion

$$\nabla \tag{3.5}$$

mit

$$\tilde{d} \tag{3.6}$$

Und die Hessian

$$\nabla^2 \tag{3.7}$$

mit

$$\hat{d} \tag{3.8}$$

Φ und r_d anpassen.
Auf Struktur geachtet, die die vorher ausgenutzte Struktur nicht verändert.

3.6.3 Exact Penalty Functions

Siehe [KM00]
In der ersten Variante der Verwendung von soft constraints ist es möglich mit der Richtigen Wichtung der Strafe für die Verletzung der Ungleichungsnebenbedingungen zu garantieren, dass bei ausreichend genauer Lösung des Optimierungsproblems auch die exakte Lösung gefunden wird, sofern diese feasibl ist und wirklich nur bei keinen feasiblen Lösungen Verletzungen zugelassen werden. Hier wird grob angerissen, was dazu nötig ist, um eine exact penalty function zu haben. Im Rahmen dieser Masterarbeit soll noch untersucht werden, ob der Verlust der exact penalty function Garantie bei der KS Funktion Variante in kritischen und unkritischen Fällen mit Nachteilen zu rechnen ist.

3.7 Polynimial Chaos Expansion

Wie bekomme ich aus einem normalen QP ein PCE? Fangen wir erstmal an mit [LZKF15]

3.8 Konvergenz für nummerisch schwierige und infeasibl Probleme

Um die Konvergenz bei Optmierungsproblemen mit gewissen Schwierigkeiten sicher zu stellen, wird der gewählte Optimierungsalgorithmus durch verschiedene Ansätze erweitert. Dazu gehören Soft Constraints, mit denen ein Scheitern des Algorithmus bei infeasibl Problemen vermieden werden kann. Ein anderer kritischer Punkt ist die Nummerik, gerade auf rechenarmer Hardware, auf der eventuell nur 32bit große Gleitkommazahlen verwendet werden können.

3.8.1 Regularization

The most important step to solve the system of linear equations is to compute the Cholesky factorization of every block in Φ. For Cholesky factorization the blocks have to be symmetric and positive definite ([BV04]). In order to have all blocks positive definite and so ensure the Cholesky factorization of every block in Φ exists, we introduce a

regularization term in the system of linear equations. We now solve

$$\begin{bmatrix} \Phi + \epsilon I & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta v \end{bmatrix} = - \begin{bmatrix} r_d \\ r_p \end{bmatrix} \tag{3.9}$$

and tolerate a small error we make instead of solving the original system of linear equations.

3.8.2 Iterative Refinement

All you need is find in [MB12] Seite 16
For solving the optimization problem as application of a MPC, it is often sufficient to use the solution of the linear equations with regularization term. If it is necessary to get the solution of the original linear equations we can compensate the error of regularization by iterative refinement as described in [MB12]. Let

$$Kl = r$$

the original system of linear equations and

$$\tilde{K} l^{(0)} = r$$

the system with regularization we can solve easily. Setting $k = 0$ a correction $\delta l^{(k)}$ can be approximated by solving

$$\tilde{K} \delta l^{(k)} = \left(r - K l^{(k)} \right),$$

which takes less effort because we already calculated the operator $\tilde{K}^{-1}$ before. Then one has to update

$$l^{(k+1)} = l^{(k)} + \delta l^{(k)}$$

and iterate k. After repeating both steps until

$$\left\| r - K l^{(k)} \right\|$$

is sufficiently small, $l^{(k)}$ can be used as good approximation for l.

3.9 Weitere Mathematische Grundlagen

Was muss man wissen, um spätere Umstellungen und Ableitungen und Berechnungen zu verstehen?

4 Algorithmus

4.1 Primal Barrier Interior-Point Methode

Das Optimierungsproblem, welches den Kern der Reglung mittles MPC bildet, muss zu jedem Zeitschritt online gelöst werden. Dabei ist es wichtig, dass eine hinreichend genaue Lösung möglichst schnell und zuverlässig gefunden werden kann. Hinreichend genau soll hier bedeuten, dass die Performance des geschlossenen Regelkreises gegeben sein muss. Sowohl in [WB10] erwähnt als auch durch die Ergebnisse dieser Arbeit bestätigt, wird eine sehr gute closed loop Regelung auch für eine nicht exakte Lösung des Optmierungsproblems erreicht. In dieser Arbeit wurde dazu eine Interior-Point Methode genutzt, bei der die Ungleichungsnebenbedingung näherungsweise durch Zuhilfenahme von logaritmische Straftermen (im folgenden als log barrier bezeichnet) berücksichtigt werden. Diese Methode wird als Primal Barrier Interior-Point Methode bezeichnet. Im Speziellen wurde hier der Ansatz von [WB10] verfolgt, bei dem besondere Rücksicht auf die Ausnutzung der spezifischen Struktur, wie sie bei MPC-Problemen auftritt, gelegt wird.

4.1.1 Optimierungsproblem für Fast-MPC Algorithmus

Der Primal Barrier Interior-Point Methode nach [WB10] oder kurz Fast MPC auf die sich später erklärte Erweitungen beziehen liegt folgende Problembeschreibung zu Grunde.
Die Dynamik des zu regelnden linearen zeitinvarianten Systems liegt in zeitdisreter Formulierung vor und lautet

$$x(k+1) = Ax(k) + Bu(k) + w(k), \quad k = 0, 1, \ldots, T. \tag{4.1}$$

Dabei ist k der jeweilige Zeitschritt bis hin zum Prädikitonshorizont T, $x(k) \in \mathbb{R}^n$ der Vektor der Zustände und $u(k) \in \mathbb{R}^m$ der Vektor der Stellgrößen. Die Systemmtrizen $A \in \mathbb{R}^{n \times n}$ und $B \in \mathbb{R}^{n \times m}$ werden zu diesem Zeitpukt erstmal als bekannt vorausgesetzt, bzw. sind die Nominalwerte von $A(\theta)$ und $B(\theta)$. Als Ungleichungsnebenbedingungen

werden in [WB10] lineare Ungleichungsnebenbedingungen der Form 3.2 betrachtet. Damit ergibt sich das Optimierungsproblem für die MPC zu

$$\min \tag{4.2}$$

$$\min \tag{4.3}$$

mit den Optimierungsvariablen $x(t+1), \ldots, x(t+T)$ und $u(t), \ldots, u(t+T-1)$. Wobei

$$l() \tag{4.4}$$

und

$$l_f() \tag{4.5}$$

die kosten convexen Kostenfunktionen für den jeweiligen Zeitschritt $k, \ldots, k+T-1$ bzw die finale Kostenfunktion zum Zeitschritt $k+T$ sind. Mit einer abweichenden finalen Kostenfunktion lässt sich für Stabilität sorgen [MRRS00]. Das gleiche gilt für stage constraints und finalen constraints. Mit dem Vektor $z = (u(t), x(t+1)), \ldots, u(t+T-1), x(t+T)$ für alle Optimierungsvariablen zusammengefasst lässt sich dieses Optimierungsproblem auch kompakt als

$$\min \tag{4.6}$$

formulieren, worauf sich weitere Erklärungen zur Lösung dieses Optimierungsproblems beziehen.

4.1.2 Fast-MPC Algorithmus

Wie löst Boyd sein QP?
Wie schon oben erwähnt werden die Ungleichungsnebenbedingungen durch Zuhilfenahme von log barriers berücksichtigt. Dadurch ergibt sich das einfacher zu lösendes Optimierungsproblem

$$\min \tag{4.7}$$

mit κ und Φ, bei dem keine direkten Ungleichungsnebenbedingungen behandeln werden müssen.
z muss immer strikt

$$Pz \leq h \tag{4.8}$$

erfüllen.

$$H = \begin{bmatrix} 0 \end{bmatrix}$$
$$P = \begin{bmatrix} 0 \end{bmatrix}$$
$$C = \begin{bmatrix} 0 \end{bmatrix}$$
$$g = \begin{bmatrix} 0 \end{bmatrix}$$
$$h = \begin{bmatrix} 0 \end{bmatrix}$$
$$b = \begin{bmatrix} 0 \end{bmatrix}$$

4.2 Erweiterung für nichtlineare Constraints

Um mit dem effizienten Algorithmus aus [WB10] auch kompliziertere/komplexere [TODO] Probleme, in Kapitel 4.2.2 Second Order Cone Problems (SOCP), bzw in Kapitel 4.2.1 Quadratic Programs/Problems mit Quadratic Constraints (QCQP) zu lösen, wurde der Algorithmus wie gleich folgt erweitert. Dabei wurde speziell darauf geachtet nicht die Struktur der entstehenden Matrizen zu verändern, sodass diese auch weiterhin ausgenutzt werden kann. Allerdings sind für ein SOCP bzw QCQP nun die Matrizen für die Ungleichungsnebenbedingungen nicht mehr konstant sondern hängen von $z(k)$ ab, sodass sie in jedem MPC Schritt angepasst werden müssen, was einen erhöhten Rechenaufwand bedeutet.

4.2.1 Erweiterung für Quadratic Problems mit quadratic Constraints

Einführendes zu QCQPs

QCQP-Formulierung

Zusäliche Ungleichungsnebenbedingung sieht wie folgt aus:

$$x^T \Gamma x + \beta^T x \leq \alpha \tag{4.9}$$

Mit $x = z$ kommen somit zu den $lT + l_f$ ursprünglichen mit den linearen Ungleichungsnebenbedingungen assoziierten Funktionen zusätzliche p Funktionen für die logarithmic barrier function hinzu

$$-f_j(z) = \alpha_j - z^T \Gamma_j z - \beta_j^T z, \quad j = 1 \dots p \tag{4.10}$$

Für den Algorithmus wird nun weiterhin die Ableitung (Gradient und Hessian) der logarithmic barrier function $\phi(z)$ benötigt, die aus unter anderem $\nabla f_k(z)$ und $\nabla^2 f_k(z)$ gebildet werden.

$$\nabla f_j\,(z) = 2z^T \Gamma_j + \beta_j^T \tag{4.11}$$

$$\nabla^2 f_j\,(z) = 2\Gamma_j \tag{4.12}$$

Daraus ergibt sich der Gradient der barrier function $\phi(z)$ zu

$$\nabla\phi\left(z\right) = \sum_{k=1}^{lT+f_f+p} \frac{1}{\begin{bmatrix} h_i \\ \alpha_j \end{bmatrix}_k - \begin{bmatrix} p_i^T \\ \beta_j^T + z^T\Gamma_j \end{bmatrix}_k z} \begin{bmatrix} p_i^T \\ \beta_j^T + 2z^T\Gamma_j \end{bmatrix}_k \tag{4.13}$$

Mit

$$\hat{P}(z) = \begin{bmatrix} P \\ \beta_1^T + z^T\Gamma_1 \\ \vdots \\ \beta_p^T + z^T\Gamma_p \end{bmatrix}, \quad \hat{h} = \begin{bmatrix} h \\ \alpha_1^T \\ \vdots \\ \alpha_p^T \end{bmatrix} \tag{4.14}$$

lässt sich das auch einfacher schreiben:

$$\nabla\phi\left(z\right) = \hat{P}^T(2z)\hat{d} \tag{4.15}$$

Wobei

$$\hat{d}_k = \frac{1}{\hat{h}_k - \hat{p}_k(z)z} \tag{4.16}$$

und $\hat{p}_k(2z)$ die Zeilen in $\hat{P}(2z)$ sein sollen. Um Φ zu bilden wird noch die zweite Ableitung von der barrier function benötigt

$$\nabla^2\phi(z) = \hat{P}(2z)\mathrm{diag}(\hat{d})^2\hat{P}(2z) + \sum_{j=1}^{p}\left(\hat{d}_{(lT+l_f)+j}2\Gamma_j\right) \tag{4.17}$$

Die Berechnungen der Ableitungen haben also analog Struktur wie [WB10] bis auf den zusätzlichen Term, der bei der Hessian hinzukommt. Aber da dieser Term in der Implementierung nicht durch die Multiplikation einer großen Matrix *Gamma* mit z sonder identischer Teilmatrizen $Gamma_k$ mit den T x_k ist sicher gestellt das keine größeren Blöcke als $n \times n$ an den richtigen Stellen erzeugt werden. Das bedeutet, dass auch hier die Struktur der Matrix Φ nicht verändert wird.

4.2.2 Erweiterung für Second Order Cone Problems

Bei der Ungleichungsnebenbedingungen für die Second Order Cone Constraints lässt sich die Berechnung leider nicht so Überschaubar darstellen, da hier Funktion und Ableitung nicht mehr so schön ähnlich sind. Die Anpassungen müssen daher wie folgt aussehen.

SOCP Formulierung

Zusäliche Ungleichungsnebenbedingung sieht wie folgt aus [TODO: woher [BV04]]:

$$\|Ax + b\|_2 \leq c^T x + d \tag{4.18}$$

Als generalized inequality nimmt Gleichung 4.18 leicht andere Form an:

$$\|Ax + b\|_2^2 \leq \left(c^T x + d\right)^2 \tag{4.19}$$

Im folgenden lässt sich die Ungleichungsnebenbedingung so leichter umformen und ableiten, speziell hebt sich so später ein Wurzelterm auf. Mit $x = z$ ergeben sich die zusätzlichen j Funktionen für die logarithmic barrier function somit zu

$$-f_j\left(x\right) = \left(c_j^T x + d_j\right)^2 - \|A_j x + b_j\|_2^2 \tag{4.20}$$

Alle k barrier function Funktionen lassen sich zu

$$-f_k\left(z\right) = -\begin{bmatrix} f_i \\ f_j \end{bmatrix}_k = \begin{bmatrix} h_i \\ 0 \end{bmatrix}_k - \begin{bmatrix} p_i^T z \\ \left(\|A_j z + b_j\|_2^2 - \left(c_j^T z + d_j\right)^2\right) \end{bmatrix}_k \tag{4.21}$$

zusammenfassen. Dabei lässt sich f_j auch noch auflösen zu

$$f_j(z) = d_j^2 - b_j^T b_j - \left[-\left(c_j^T z + 2d_j\right) c_j^T + \left(z^T A_j^T + 2b_j^T\right) A_j\right] z \tag{4.22}$$

Für den Algorithmus wird nun weiterhin die Ableitung (Gradient und Hessian) der logarithmic barrier function $\phi(z)$ benötigt, die aus unter anderem $\nabla f_k(z)$ und $\nabla^2 f_k(z)$ gebildet werden.

$$\nabla f_k\left(z\right) = \begin{bmatrix} p_i^T \\ \left(-2\left(\left(c_j^T z + d_j\right) c_j - A_j^T\left(A_j z + b_j\right)\right)\right)^T \end{bmatrix}_k \tag{4.23}$$

Das Transponieren der unteren Zeile ergibt sich dadurch, dass man zu P eine Spalte anhängt also zu P^T die transformierte dieser Spalte. Damit lässt sich auch hier $\nabla f_k(z)$ wie schon bei die quadratic constraint schreiben.

$$\nabla\phi\left(z\right) = \hat{P}^T(2z)\hat{d} \tag{4.24}$$

Mit

$$\nabla^2 f_k\left(z\right) = \begin{bmatrix} 0 \\ -2\left(c_j c_j^T - A_j^T A_j\right) \end{bmatrix}_k \tag{4.25}$$

ergibt sich $\nabla^2\phi(z)$ zu

$$\nabla^2\phi(z) = \hat{P}(2z)\text{diag}(\hat{d})^2\hat{P}(2z) + \sum_{j=1}^{p}\left(\hat{d}_{(lT+l_f)+j} - 2\left(c_j c_j^T - A_j^T A_j\right)\right) \tag{4.26}$$

[TODO richtiger Index für d]

4.2.3 Komplette Formulierung

Ungleichungsnebenbedingungen

$$\hat{P}(z)z \leq \hat{h} \tag{4.27}$$

mit

$$\hat{P}(z) = \begin{bmatrix} P \\ \beta_1^T + z^T\Gamma_1 \\ \vdots \\ \beta_p^T + z^T\Gamma_p \\ -\left(c_1^T z + 2d_1\right)c_1^T + \left(z^T A_1^T + 2b_1^T\right)A_1 \\ \vdots \\ -\left(c_q^T z + 2d_q\right)c_q^T + \left(z^T A_q^T + 2b_q^T\right)A_q \end{bmatrix}, \quad \hat{h} = \begin{bmatrix} h \\ \alpha_1^T \\ \vdots \\ \alpha_p^T \\ d_1^2 - b_1^T b_1 \\ \vdots \\ d_q^2 - b_q^T b_q \end{bmatrix} \tag{4.28}$$

Daraus ergibt sich die Logarithmic barrier function zu

$$\phi(z) = \sum_{i=1}^{lT+l_f+p+q} -\log\left(\hat{h}_i - \hat{p}_i^T(z)z\right) \tag{4.29}$$

Wobei $\hat{p}_i^T(z)$ die Zeilen aus $\hat{P}(z)$ sind.
Bei der Berechnung des Residuums wird der Gradient der logarithmic barrier function benötigt

$$\nabla\phi(z) = \hat{P}^T(2z)\hat{d} \tag{4.30}$$

mit

$$\hat{d}_i = \frac{1}{\hat{h}_i - \hat{p}_i^T(z)z} \tag{4.31}$$

Dabei muss $\hat{P}^T(z)$ mit $2z$ aufgerufen werden wie oben beschrieben.
Hessian der der Logarithmic barrier function für Berechnung des Φ

$$\begin{aligned}\nabla^2\phi(z) =& \hat{P}(2z)\mathrm{diag}(\hat{d})^2\hat{P}(2z) \\ &+ \sum_{i=lT+lf+1}^{lT+lf+p} \left(\hat{d}_i 2\Gamma_i\right) + \sum_{j=lT+lf+p+1}^{lT+lf+p+q} \left(-2\hat{d}_j \left(c_j c_j^T - A_j^T A_j\right)\right)\end{aligned} \tag{4.32}$$

4.3 Erweiterungen um Soft Constraints

Was Soft Constraints sind wurde bereits beschrieben. Im benutzten Algorithmus lassen sie sich wie folgt ohne Probleme verwenden.

4.4 Selecting Kappa

If κ convergeres to zero, the gap between these minimizers does, too. Therefore in [BV04] it is proposed to decrease κ by factor 10 to 20 in every inner optimization step.
Additionally we bound κ to a minimum value $\kappa_{min} > 0$ to avoid nummerical problems for values of the cost term $z^T Mz + g^T z \leq 0$. Choosing κ with this approach leads to only small changes in its value for every execution of the MPC, so the warm start still works fine. Since κ only converges against κ_{min} after a few MPC executions, the gap between the minimizers of (4.6) and (4.7) never disappears. Nevertheless with this trade-off we find sufficiently exact solutions in much less inner iterations.

4.5 Anpassung für test cases

Der implementierte Algorithmus wie in [Paper:, Section:] beschrieben kann auch verwendet werden, um Optimierungsprobleme zu lösen, die ihren Ursprung nicht in der Anwendung von MPC haben. Dazu sind keine wirklichen Anpassungen des Algorithmus notwendig. Da der Algorithmus allerdings die Struktur der bei MPC auftretenden Matrizen ausnutzt, muss der jeweilige test case so "transformiert" werden, dass dieser eine ähnliche Struktur aufweist.

4.5.1 Allgemeine Beschreibung der test cases

Nach [MM99] haben die test cases folgende Form:

$$\begin{aligned} \min \quad & \hat{c}^T\hat{x} + \frac{1}{2}\hat{x}^T\hat{Q}\hat{x} \\ \text{s.t.} \quad & \hat{A}\hat{x} = \hat{b} \\ & \hat{l} \leq \hat{x} \leq \hat{u} \end{aligned} \tag{4.33}$$

Aber es existieren auch test cases mit weiteren Ungleichungsnebenbedingung der Form:

$$\hat{b}_{lower} \leq \hat{A}\hat{x} \leq \hat{b}_{upper} \tag{4.34}$$

Vereinheitlicht für 4.33 und 4.34 schreiben

$$\begin{aligned} \min \quad & \hat{c}^T\hat{x} + \frac{1}{2}\hat{x}^T\hat{Q}\hat{x} \\ \text{s.t.} \quad & \hat{b}_{lower} \leq \hat{A}\hat{x} \leq \hat{b}_{upper} \\ & \hat{l} \leq \hat{x} \leq \hat{u} \end{aligned} \tag{4.35}$$

Wobei sich für

$$\hat{b} = \hat{b}_{lower} = \hat{b}_{upper}$$

die Gleichungsnebenbedingungen

$$\hat{A}\hat{x} = \hat{b}$$

ergeben

4.5.2 Algorithmus mit Prädiktionshorizont gleich eins

Um die test cases lösen zu können, muss der Prädiktionshorizont $T = 1$ gewählt werden. Die Optimierungsvariable beschränkt sich damit auf

$$z = (u(t), x(t+T)) \in \mathbb{R}^{(m+n)}, \quad T = 1$$

Die strukturierten Matrizen im Algorithmus zum lösen des Optimierungsproblems

$$\begin{aligned} \min \quad & z^T H z + g^T z \\ \text{s.t.} \quad & Pz \leq h, \quad Cz = b \end{aligned}$$

reduzieren sich damit auf folgende Form:

$$H = \begin{bmatrix} R & 0 \\ 0 & Q_f \end{bmatrix}$$
$$P = \begin{bmatrix} F_u & 0 \\ 0 & F_f \end{bmatrix}$$
$$C = \begin{bmatrix} -B & I \end{bmatrix}$$
$$g = \begin{bmatrix} r + 2S^T x(t) \\ q \end{bmatrix}$$
$$h = \begin{bmatrix} f - F_x x(t) \\ f_f \end{bmatrix}$$
$$b = \begin{bmatrix} Ax(t) \end{bmatrix}$$

Um den Algorithmus nun mit den test cases nach [MM99] zu testen muss

$$H = \frac{1}{2}\hat{Q}, \quad g = \hat{c}, \text{nicht korrekt, einzelne Untermatrizen setzen} \tag{4.36}$$

gesetzt werden. Die Ungleichungsnebenbedingung

$$F_u u(t) + F_x x(t) + F_f x(t+1) \leq f = f_u + f_x \tag{4.37}$$

ergeben sich zu

$$F_u = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$
$$F_x = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$
$$F_f = \begin{bmatrix} 0 & 0 \end{bmatrix}$$
$$f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$f_f = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Als Gleichungsnebenbedingungen bleibt im implementierten Algorithmus

$$x(t+1) = Ax(t) + Bu(t) \tag{4.38}$$

Da allerdings x(t+1) auch zu dem Vektor der Optimierungsvariablen gehört muss

$$\hat{b} = -Ax(t) \quad \text{mit} \quad A = -I \tag{4.39}$$

gesetzt werden. Zusätzlich wird mit weiteren Ungleichungsnebenbedingung dafür gesorgt, dass $x(t+1)$ im Optimum nahe der Null liegt. Das bedeutet aber auch, dass in der Auswertung der Güte der eingehaltenen Gleichungsnebenbedingungen auch die Genauigkeit der zusätzlichen Ungleichungsnebenbedingung betrachtet werden muss.

5 Ergebnisse

Um zu beurteilen, wie robust der implementierte Algorithmus funktioniert, wurden verschiedene Testscenarien erstellt und die Ergebnisse hinsichtlich der Laufzeit und Häufigkeit der Beschränkungsverletzungen miteinander vergleichen.

5.1 Marosh Mezarosh

Die beiden Menschen haben sich Gedanken darüber gemacht, mit welchen Testproblemen ein Algorithmus getest werden kann. Beispielsweise hier mal die Lösung einzelner QPs. allerdings ohne die MPC-Struktur ausnutzen zu können, die hier nicht vorhanden ist.

5.2 Testprobleme

Um zu beurteilen, wie robust der implementierte Algorithmus funktioniert, wurden verschiedene Testscenarien erstellt und die Ergebnisse hinsichtlich der Laufzeit und Häufigkeit der Beschränkungsverletzungen miteinander vergleichen.
Als Optimierungsbeispiel wurde dazu in verschiedenen angepassten Varianten das Beispiel aus herangezogen. Dabei handelt es sich um ein lineares System dass die Bewegung eines Flugzeuges beschreibt. Dieses System besitzt 5 Zustände und einen Eingang und sieht folgendermaßen aus:

$$x_{k+1} = . \tag{5.1}$$

Mit x_1, des angel of attack x_2, x_3, der Höhe x_4, und x_5 als Zustände und u_1 als Eingang. Eine Schwierigkeit bei der Regelung ist die Unsicherheit bei den ermittelten Systemmatrizen. Deshalb wurde für die Simulation des Systems die Zustandsmatrizen mit den Unsicherheiten σ_1 und σ_2 variert, wobei die Werte für die Unsicherheit im angepassten

System

$$x_{k+1} = \tag{5.2}$$

während jeder Simulation in allen Zeitschritten konstant ist und nur zwischen den Simulationen wechselt. Für die Prädiktion in der MPC wurden jeweils weiterhin die nominalen Werte für die Systemmatrizen A und B verwendet.

5.2.1 Einfaches QP

Stellt man das Optimierungsproblem als einfaches QP auf, sieht es wie folgt aus. Die Optimierungsvariable z hat die Dimension $(n+m)T = 6T$ und demzufolge gilt Matrix $H \in R^{6T \times 6T}$.

5.2.2 QP in PCE Formulierung

Wie schon erwähnt ist es hilfreich die Modellunsicherheiten mittels stochastischer Abschätzungen zu erfassen. Dazu wurde das einfache QP als PCE augestellt. Nun hat die Optimierungsvariable z Dimension $31 * T$ und H dann natürlich $31T \times 31T$. Es wird insbesondere hier erwartet, dass das ausnutzen der blockdiagonalen Struktur der Matrix H große Vorteile mit sich bringt.

5.2.3 Referenzproblem

Der implementierte Algorithmus nutzt die Struktur des Optimierungsproblems aus, die spezifisch für die Verwendung als MPC entsteht. Um zu zeigen, wie der Vorteil des Algorithmus die nötige Zeit der Optimierung verringert wurden Problem 1 (5.2.1) und Problem 2 (5.2.2) sowohl mit dem zugeschnittenen Algorithmus als auch ohne die Matrixstruktur auszunutzen gelöst.

5.2.4 QP in PCE Formulierung mit QC

5.2.5 QP in PCE Formulierung mit SOCC

Besser ist es das PCE als SOCP zu lösen. Um so besser mit den Unsicherheiten umgehen zu können. Die implementierte Erweiterung lässt dies nun zu.

5.2.6 QP in PCE Formulierung mit Softconstraints (KSF)

Erhöht man die Unsicherheit des Systems weiter, so kommt es zu Fällen in den der Algorithmus versagt, da kein feasibler Punkt für die Optimierungsvariable da ist, bzw nicht in einem Schritt erreicht werden kann. wenn es sich hierbei allerdings um Grenzen handelt, deren Verletzung kurzzeitig vertretbar wäre, kann diese durch Soft Constraints zugelassen werden. Dafür als Vergleich das Problem mit Softconstraints als KSF.

5.2.7 QP in PCE Formulierung mit Softconstraints (Slack)

Eine andere Möglichkeit Softconstraints zu integrieren wurde beschrieben und implementiert. Dazu wurde eine zusätzliche Slackvariable eingeführt, die Dimensinen sehen daher bei dieser Variante wie folgt aus.

5.3 Vergleich der Testergebnisse

Verglichen werden hier
Besonderes Augenmerk auf Laufzeit, um den Vorteil des Grundalgorithmus zu zeigen.
Mit verschiedenen Horizontlängen. Vielleicht unconstraint
QP ref <–> QP
QP PCE ref <–> QP PCE
QP PCE mit verschiedenen Constraints (also lineare Constraints, QC, SOCC, Soft KSF, Soft Slack (Soft statt lin)) um dort Laufzeiten und Violations zu vergleichen
Ergebnisse könnten hinsichtlich innerer Steps vergleichen werden. Dafür feste Horizontlängen
Softconstraints mehr untereinander vergleichen, als gegen die anderen, weil hier noch der zusätliche Parameter dazu kommt, der Vergleich mit anderen verfälscht.

6 Auswertung

6.1 Zusammenfassung

6.2 Schlussfolgerung

6.3 Ausblick

Literaturverzeichnis

[Ada09] Adamy, Jürgen: *Nichtlineare Regelungen.* Springer-Verlag, 2009

[BV04] Boyd, Stephen ; Vandenberghe, Lieven: *Convex optimization.* Cambridge university press, 2004

[KM00] Kerrigan, Eric C. ; Maciejowski, Jan M.: Soft constraints and exact penalty functions in model predictive control. In: *Control 2000 Conference, Cambridge*, 2000

[Kre79] Kreisselmeier, G: Systematic control design by optimizing a vector performance index. In: *IFAC Symp. Computer Aided Design of Control Systems, Zurich, Switzerland, 1979*, 1979

[LZKF15] Lucia, Sergio ; Zometa, Pablo ; Kögel, Markus ; Findeisen, Rolf: Efficient stochastic model predictive control based on polynomial chaos expansions for embedded applications. In: *Proceedings of the 54th IEEE Conference on Decision and Control (In press)*, 2015

[MB12] Mattingley, Jacob ; Boyd, Stephen: CVXGEN: a code generator for embedded convex optimization. In: *Optimization and Engineering* 13 (2012), Nr. 1, S. 1–27

[MM99] Maros, Istvan ; Mészáros, Csaba: A repository of convex quadratic programming problems. In: *Optimization Methods and Software* 11 (1999), Nr. 1-4, S. 671–681

[MRRS00] Mayne, David Q. ; Rawlings, James B. ; Rao, Christopher V. ; Scokaert, Pierre O.: Constrained model predictive control: Stability and optimality. In: *Automatica* 36 (2000), Nr. 6, S. 789–814

[Ric13] Richards, Arthur: Fast model predictive control with soft constraints. In: *Control Conference (ECC), 2013 European* IEEE, 2013, S. 1–6

[SK11] Schwarz, Hans R. ; Köckler, Norbert: *Numerische mathematik*. Springer-Verlag, 2011

[WB10] Wang, Yang ; Boyd, Stephen: Fast model predictive control using online optimization. In: *Control Systems Technology, IEEE Transactions on* 18 (2010), Nr. 2, S. 267–278

Selbstständigkeitserklärung

Hiermit versichere ich, Hannes Heinemann, dass ich die vorliegende Bachelorarbeit mit dem Thema „Umsetzung eines Kanalmodells für Petri-Netz modellierte Funksysteme“ selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

22. Oktober 2015 Hannes Heinemann