



OTTO VON GUERICKE  
UNIVERSITÄT  
MAGDEBURG

EIT

FAKULTÄT FÜR  
ELEKTROTECHNIK UND  
INFORMATIONSTECHNIK

Otto-von-Guericke-Universität Magdeburg

Fakultät für Elektrotechnik und Informationstechnik  
Institut für Automatisierungstechnik

## Masterarbeit

## Meine Master Thesis

Heinemann, Hannes

19. Oktober 2015

Erstprüfer: Rolf Findeisen

Zweitprüfer: Erik B.

Betreuer: Pablo

# Aufgabenstellung

## Thema

**Zeitraum: 06 - 11**

Das ist meine Aufgabenstellung

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>V</b>
<b>Tabellenverzeichnis</b>	<b>VI</b>
<b>Abkürzungsverzeichnis</b>	<b>VII</b>
<b>Symbolverzeichnis</b>	<b>VIII</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 TODO . . . . .	1
1.1.1 Format . . . . .	1
1.1.2 Inhalt . . . . .	1
<b>2 Grundlagen</b>	<b>3</b>
2.1 Modellprädiktive Regelung . . . . .	3
2.2 Boyds Grundlagen . . . . .	5
2.2.1 Interior-Point Methode . . . . .	5
2.2.2 Primal vs. Dual . . . . .	5
2.3 Ungleichungsnebenbedingungen . . . . .	5
2.3.1 Lineare Ungleichungsnebenbedingungen . . . . .	6
2.4 Quadratic Constraints . . . . .	7
2.5 Second Order Cone Constraints . . . . .	7
2.6 Soft Constraints . . . . .	7
2.6.1 Soft Constraints mittels Einführung zusätzlicher Slackvariablen . .	7
2.6.2 Soft Constraints mittels Einführung eines Strafterms in Form der Kreisselmeier-Steinhauser Funktion . . . . .	8
2.7 Polynimial Chaos Expansion . . . . .	8

2.8	Weitere Mathematische Grundlagen . . . . .	8
<b>3</b>	<b>Algorithmus</b>	<b>9</b>
3.1	Primal Barrier Interior-Point Methode . . . . .	9
3.1.1	Optimierungsproblem für Fast-MPC Algorithmus . . . . .	9
3.1.2	Fast-MPC Algorithmus . . . . .	10
3.2	Erweiterung für nichtlineare Constraints . . . . .	10
3.2.1	Erweiterung für Quadratic Problems mit quadratic Constraints . .	10
3.2.2	Erweiterung für Second Order Cone Problems . . . . .	12
3.2.3	Komplette Formulierung . . . . .	13
3.3	Erweiterungen um Soft Constraints . . . . .	14
3.4	Anpassung für test cases . . . . .	14
3.4.1	Allgemeine Beschreibung der test cases . . . . .	15
3.4.2	Algorithmus mit Prädiktionshorizont gleich eins . . . . .	15
<b>4</b>	<b>Ergebnisse</b>	<b>18</b>
4.1	Marosh Mezarosh . . . . .	18
4.2	Testprobleme . . . . .	18
4.2.1	Einfaches QP . . . . .	19
4.2.2	QP in PCE Formulierung . . . . .	19
4.2.3	Referenzproblem . . . . .	19
4.2.4	QP in PCE Formulierung mit QC . . . . .	19
4.2.5	QP in PCE Formulierung mit SOCC . . . . .	19
4.2.6	QP in PCE Formulierung mit Softconstraints (KSF) . . . . .	20
4.2.7	QP in PCE Formulierung mit Softconstraints (Slack) . . . . .	20
4.3	Vergleich der Testergebnisse . . . . .	20
<b>5</b>	<b>Schluss</b>	<b>21</b>
5.1	Zusammenfassung . . . . .	21
5.2	Schlussfolgerung . . . . .	21
5.3	Ausblick . . . . .	21
	<b>Literaturverzeichnis</b>	<b>22</b>

**Selbstständigkeitserklärung**

**23**

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

KS	.....	Kreisselmeier-Steinhauser
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 + linear 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

[MM97] Modellprädiktive Regelung (MPC)

## 2 Grundlagen

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

### 2.1 Modellprädiktive Regelung

Viele Regelungskonzepte werden in der Industrie angewandt. Ein Ansatz ist die optimale Regelung, bei der online oder auch offline das Regelungsgesetz so bestimmt wird, dass eine gewisse Kostenfunktion minimal ist. Besteht die Möglichkeit dieses Regelungsgesetz offline zu berechnen, ist alles schön und gut. Aber in vielen Fällen muss dieses Gesetz für jeden Zeitschritt erneut ausgerechnet werden und somit online zur Laufzeit des Systems ständig neu berechnet werden. Nun gibt es Systeme bei denen die eine schnelle Reaktionszeit des Reglers erfordern, was unter Umständen dazu führen kann, dass möglichst effiziente Methoden benutzt werden.

Eine spezielle Form der optimalen Regelung 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 Wichtungsparemtern 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:

Im realen Anwendungsfall des oTToCAR-Projekts eignet sich eine Systemdarstellung in zeitdiskreter Form ([?]), 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:

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

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

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

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

$$\mathbf{x}_p = \{\mathbf{x}(k+1), \dots, \mathbf{x}(k+n_p)\}$$

bis zum Prädiktionshorizont  $n_p$  unter der Optimierung einer Sequenz von Eingängen

$$\mathbf{u} = \{\mathbf{u}(k), \dots, \mathbf{u}(k+n_c-1)\}$$

bis zum Stellhorizont  $n_c$  vorhergesagt. Aus der gefundenen optimalen Eingangssequenz  $\mathbf{u}^*$  wird der erste Eintrag  $\mathbf{u}^*(k)$  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  $\mathbf{x}_r$  zu folgen.

Für das an jedem Zeitschritt  $k$  zu lösende Minimierungsproblem wurde die benötigte

Kostenfunktion  $J$  in quadratische Form mit  $\mathbf{x}_p$  und  $\mathbf{u}$  als Optimierungsvariablen aufgestellt:

$$\begin{aligned} \min_{\mathbf{x}_p, \mathbf{u}} J &:= \sum_{i=k+1}^{k+n_p} [\mathbf{x}_p(i) - \mathbf{x}_r(i)]^T \mathbf{Q}_i [\mathbf{x}_p(i) - \mathbf{x}_r(i)] + \sum_{j=k}^{k+n_c-1} \mathbf{u}^T(j) \mathbf{R}_j \mathbf{u}(j) \\ \text{s.t. } \mathbf{x}_p(i+1) &= \mathbf{f}(\mathbf{x}_p(i), \mathbf{u}(i)), \quad i = k, \dots, k+n_c-1 \\ \mathbf{x}_p(i+1) &= \mathbf{f}(\mathbf{x}_p(i), \mathbf{u}(k+n_c-1)), \quad i = k+n_c, \dots, k+n_p-1 \end{aligned}$$

Mit den Vektoren

$$\begin{aligned} \mathbf{x}_p(k) &= [\mathbf{x}_p(k+1 | k), \dots, \mathbf{x}_p(k+n_p | k)]^T \\ \mathbf{x}_r(k) &= [\mathbf{x}_r(k+1), \dots, \mathbf{x}_r(k+n_p)]^T \\ \mathbf{u}(k) &= [\mathbf{u}(k), \dots, \mathbf{u}(k+n_c-1)]^T \end{aligned}$$

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

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

und Zustandsbeschränkungen der Form

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

erweitern.

## 2.2 Boyds Grundlagen

### 2.2.1 Interior-Point Methode

### 2.2.2 Primal vs. Dual

## 2.3 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

$$\mathbf{x}(k) \in \mathcal{X} \subset \mathbb{R}^n$$

$$\mathbf{u}(k) \in \mathcal{U} \subset \mathbb{R}^m$$

stammt ([?]). 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.

### 2.3.1 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. Ähnliches gilt auch Systemeingänge bei denen aus beispielsweise technischen Gründen nur Werte aus einem bekannten Intervall auf das System gegeben werden können. Solche einfachen Zustandsbeschränkungen der Form

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

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

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

$$F_x x(k) + F_u u(k) \leq f, \quad k = 0, 1, \dots \quad (2.1)$$

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

## 2.4 Quadratic Constraints

Was genau, wofür QC?

## 2.5 Second Order Cone Constraints

Was genau, wofür SOCC?

## 2.6 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 feasil sind. Da es besser ist, den Algorithmus nicht abubrechen 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 feasil 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.

### 2.6.1 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 2.1 lassen sich dann die soft constriants als

$$\begin{bmatrix} \tilde{F}_x \\ 0 \end{bmatrix} x(k) + \begin{bmatrix} \tilde{F}_u & -1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} u(k) \\ s(k) \end{pmatrix} \leq \begin{bmatrix} \tilde{f} \\ 0 \end{bmatrix}, \quad k = 0, 1, \dots \quad (2.2)$$



formulieren.  $s(k)$  ist dabei die maximale Verletzung 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 Berücksichtigung 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.

### **2.6.2 Soft Constraints mittels Einführung eines Strafterms in Form der Kreisselmeier-Steinhauser Funktion**

## **2.7 Polynomial Chaos Expansion**

Wie bekomme ich aus einem normalen QP ein PCE?

## **2.8 Weitere Mathematische Grundlagen**

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

## 3 Algorithmus

### 3.1 Primal Barrier Interior-Point Methode

Das Optimierungsproblem, welches den Kern der Regelung mittels MPC bildet, muss zu jedem Zeitschritt online gelöst werden. Dabei ist es wichtig, dass eine ausreichend genaue Lösung möglichst schnell und zuverlässig gefunden werden kann. In dieser Arbeit wurde dazu eine Interior-Point Methode genutzt, bei der die Ungleichungsnebenbedingung näherungsweise durch Zuhilfenahme von logarithmische 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.

#### 3.1.1 Optimierungsproblem für Fast-MPC Algorithmus

Der Primal Barrier Interior-Point Methode nach [WB10] oder kurz Fast MPC auf die sich später erklärte Erweiterungen beziehen liegt folgende Problembeschreibung zu Grunde.

Beschreibung des allgemeinen Problems das von Boyd gelöst wird. Der Prädiktionshorizont  $T$

Damit ergibt sich Optimierungsproblem für die MPC zu

$$\min \tag{3.1}$$

mit den Optimierungsvariablen  $x(t+1), \dots, x(t+T)$  und  $u(t), \dots, u(t+T-1)$ . Dieses Problem lässt sich kompakt auch als

$$\min \tag{3.2}$$

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

### 3.1.2 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

$$\min \quad (3.3)$$

mit  $\kappa$  und  $\Phi$ , bei dem keine direkten Ungleichungsnebenbedingungen behandeln werden müssen.

## 3.2 Erweiterung für nichtlineare Constraints

Um mit dem effizienten Algorithmus aus [WB10] auch kompliziertere/komplexere [TODO] Probleme, in Kapitel 3.2.2 Second Order Cone Problems (SOCP), bzw in Kapitel 3.2.1 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.

### 3.2.1 Erweiterung für Quadratic Problems mit quadratic Constraints

Einführendes zu QCQPs

#### QCQP-Formulierung

Zusätzliche Ungleichungsnebenbedingung sieht wie folgt aus:

$$x^T \Gamma x + \beta^T x \leq \alpha \quad (3.4)$$

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

$$-f_j(z) = \alpha_j - z^T \Gamma_j z - \beta_j^T z, \quad j = 1 \dots p \quad (3.5)$$

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.

$$\nabla f_j(z) = 2z^T \Gamma_j + \beta_j^T \quad (3.6)$$

$$\nabla^2 f_j(z) = 2\Gamma_j \quad (3.7)$$

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

$$\nabla \phi(z) = \sum_{k=1}^{lT+l_f+p} \frac{1}{\begin{bmatrix} h_i \\ \alpha_j \end{bmatrix}_k - \begin{bmatrix} p_i^T \\ \beta_j^T + z^T \Gamma_j \end{bmatrix}_k} z \begin{bmatrix} p_i^T \\ \beta_j^T + 2z^T \Gamma_j \end{bmatrix}_k \quad (3.8)$$

Mit

$$\hat{P}(z) = \begin{bmatrix} P \\ \beta_1^T + z^T \Gamma_1 \\ \vdots \\ \beta_p^T + z^T \Gamma_p \end{bmatrix}, \quad \hat{h} = \begin{bmatrix} h \\ \alpha_1^T \\ \vdots \\ \alpha_p^T \end{bmatrix} \quad (3.9)$$

lässt sich das auch einfacher schreiben:

$$\nabla \phi(z) = \hat{P}^T(2z) \hat{d} \quad (3.10)$$

Wobei

$$\hat{d}_k = \frac{1}{\hat{h}_k - \hat{p}_k(z)z} \quad (3.11)$$

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

$$\nabla^2 \phi(z) = \hat{P}(2z) \text{diag}(\hat{d})^2 \hat{P}(2z) + \sum_{j=1}^p \left( \hat{d}_{(lT+l_f)+j} 2\Gamma_j \right) \quad (3.12)$$

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  $z$  sondern 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  $\Phi$  nicht verändert wird.

### 3.2.2 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

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

$$\|Ax + b\|_2 \leq c^T x + d \quad (3.13)$$

Als generalized inequality nimmt Gleichung 3.13 leicht andere Form an:

$$\|Ax + b\|_2^2 \leq (c^T x + d)^2 \quad (3.14)$$

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

$$-f_j(x) = (c_j^T x + d_j)^2 - \|A_j x + b_j\|_2^2 \quad (3.15)$$

Alle  $k$  barrier function Funktionen lassen sich zu

$$-f_k(z) = - \begin{bmatrix} f_i \\ f_j \end{bmatrix}_k = \begin{bmatrix} h_i \\ 0 \end{bmatrix}_k - \begin{bmatrix} p_i^T z \\ \left( \|A_j z + b_j\|_2^2 - (c_j^T z + d_j)^2 \right) \end{bmatrix}_k \quad (3.16)$$

zusammenfassen. Dabei lässt sich  $f_j$  auch noch auflösen zu

$$f_j(z) = d_j^2 - b_j^T b_j - \left[ - (c_j^T z + 2d_j) c_j^T + (z^T A_j^T + 2b_j^T) A_j \right] z \quad (3.17)$$

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.

$$\nabla f_k(z) = \begin{bmatrix} p_i^T \\ \left( -2 \left( (c_j^T z + d_j) c_j - A_j^T (A_j z + b_j) \right) \right)^T \end{bmatrix}_k \quad (3.18)$$

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.

$$\nabla \phi(z) = \hat{P}^T(2z) \hat{d} \quad (3.19)$$

Mit

$$\nabla^2 f_k(z) = \begin{bmatrix} 0 \\ -2 (c_j c_j^T - A_j^T A_j) \end{bmatrix}_k \quad (3.20)$$

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

$$\nabla^2 \phi(z) = \hat{P}(2z) \text{diag}(\hat{d})^2 \hat{P}(2z) + \sum_{j=1}^p \left( \hat{d}_{(l_T+l_f)+j} - 2 (c_j c_j^T - A_j^T A_j) \right) \quad (3.21)$$

[TODO richtiger Index für d]

### 3.2.3 Komplette Formulierung

Ungleichungsnebenbedingungen

$$\hat{P}(z)z \leq \hat{h} \quad (3.22)$$

mit

$$\hat{P}(z) = \begin{bmatrix} P \\ \beta_1^T + z^T \Gamma_1 \\ \vdots \\ \beta_p^T + z^T \Gamma_p \\ -(c_1^T z + 2d_1) c_1^T + (z^T A_1^T + 2b_1^T) A_1 \\ \vdots \\ -(c_q^T z + 2d_q) c_q^T + (z^T A_q^T + 2b_q^T) A_q \end{bmatrix}, \quad \hat{h} = \begin{bmatrix} h \\ \alpha_1^T \\ \vdots \\ \alpha_p^T \\ d_1^2 - b_1^T b_1 \\ \vdots \\ d_q^2 - b_q^T b_q \end{bmatrix} \quad (3.23)$$

Daraus ergibt sich die Logarithmic barrier function zu

$$\phi(z) = \sum_{i=1}^{lT+lf+p+q} -\log \left( \hat{h}_i - \hat{p}_i^T(z)z \right) \quad (3.24)$$

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

$$\nabla \phi(z) = \hat{P}^T(2z)\hat{d} \quad (3.25)$$

mit

$$\hat{d}_i = \frac{1}{\hat{h}_i - \hat{p}_i^T(z)z} \quad (3.26)$$

Dabei muss  $\hat{P}^T(z)$  mit  $2z$  aufgerufen werden wie oben beschrieben.

Hessian der der Logarithmic barrier function für Berechnung des  $\Phi$

$$\begin{aligned} \nabla^2 \phi(z) = & \hat{P}(2z)\text{diag}(\hat{d})^2\hat{P}(2z) \\ & + \sum_{i=lT+lf+1}^{lT+lf+p} \left( \hat{d}_i 2\Gamma_i \right) + \sum_{j=lT+lf+p+1}^{lT+lf+p+q} \left( -2\hat{d}_j (c_j c_j^T - A_j^T A_j) \right) \end{aligned} \quad (3.27)$$

### 3.3 Erweiterungen um Soft Constraints

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

### 3.4 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.

### 3.4.1 Allgemeine Beschreibung der test cases

Nach [MM97] haben die test cases folgende Form:

$$\begin{aligned} \min \quad & \hat{c}^T \hat{x} + \frac{1}{2} \hat{x}^T \hat{Q} \hat{x} \\ \text{s.t.} \quad & \hat{A} \hat{x} = \hat{b} \\ & \hat{l} \leq \hat{x} \leq \hat{u} \end{aligned} \tag{3.28}$$

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

$$\hat{b}_{lower} \leq \hat{A} \hat{x} \leq \hat{b}_{upper} \tag{3.29}$$

Vereinheitlicht für 3.28 und 3.29 schreiben

$$\begin{aligned} \min \quad & \hat{c}^T \hat{x} + \frac{1}{2} \hat{x}^T \hat{Q} \hat{x} \\ \text{s.t.} \quad & \hat{b}_{lower} \leq \hat{A} \hat{x} \leq \hat{b}_{upper} \\ & \hat{l} \leq \hat{x} \leq \hat{u} \end{aligned} \tag{3.30}$$

Wobei sich für

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

die Gleichungsnebenbedingungen

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

ergeben

### 3.4.2 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

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



reduzieren sich damit auf folgende Form:

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

Um den Algorithmus nun mit den test cases nach [MM97] zu testen muss

$$H = \frac{1}{2}\hat{Q}, \quad g = \hat{c}, \text{ nicht korrekt, einzelne Untermatrizen setzen} \quad (3.31)$$

gesetzt werden. Die Ungleichungsnebenbedingung

$$F_u u(t) + F_x x(t) + F_f x(t+1) \leq f = f_u + f_x \quad (3.32)$$

ergeben sich zu

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

Als Gleichungsnebenbedingungen bleibt im implementierten Algorithmus

$$x(t+1) = Ax(t) + Bu(t) \tag{3.33}$$

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

$$\hat{b} = -Ax(t) \quad \text{mit} \quad A = -I \tag{3.34}$$

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.

## 4 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 verglichen.

### 4.1 Marosh Mezarosh

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

### 4.2 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 verglichen.

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:

$$x_{k+1} = . \tag{4.1}$$

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  $\sigma_1$  und  $\sigma_2$  variiert, wobei die Werte für die Unsicherheit im angepassten

System

$$x_{k+1} = \tag{4.2}$$

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.

#### 4.2.1 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  $H \in R^{6T \times 6T}$ .

#### 4.2.2 QP in PCE Formulierung

Wie schon erwähnt ist es hilfreich die Modellunsicherheiten mittels stochastischer Abschätzungen zu erfassen. Dazu wurde das einfache QP als PCE aufgestellt. 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.

#### 4.2.3 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 (4.2.1) und Problem 2 (4.2.2) sowohl mit dem zugeschnittenen Algorithmus als auch ohne die Matrixstruktur auszunutzen gelöst.

#### 4.2.4 QP in PCE Formulierung mit QC

#### 4.2.5 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.

#### 4.2.6 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.

#### 4.2.7 QP in PCE Formulierung mit Softconstraints (Slack)

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

### 4.3 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 verglichen werden. Dafür feste Horizontlängen

Softconstraints mehr untereinander vergleichen, als gegen die anderen, weil hier noch der zusätzliche Parameter dazu kommt, der Vergleich mit anderen verfälscht.

## 5 Schluss

### 5.1 Zusammenfassung

### 5.2 Schlussfolgerung

### 5.3 Ausblick

# Literaturverzeichnis

- [BV04] BOYD, Stephen ; VANDENBERGHE, Lieven: *Convex Optimization*. University Press, 2004
- [MM97] MAROS, István ; MÉSZÁROS, Csaba: A Repository of Convex Quadratic Programming Problems. (1997)
- [Ric13] RICHARDS, Arthur: Fast model predictive control with soft constraints. In: *Control Conference (ECC), 2013 European IEEE*, 2013, S. 1–6
- [WB10] WANG, Yang ; BOYD, Stephen: Fast Model Predictive Control Using Online optimization. In: *IEEE Transactions on Control Systems Technology* 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 Funkssysteme“ selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

19. Oktober 2015

Hannes Heinemann