



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

6. Oktober 2015

Erstprüfer: Rolf Findeisen

Zweitprüfer: Erik B.

Betreuer: Pablo

Aufgabenstellung

Thema

Zeitraum: 06 - 11

Das ist meine Aufgabenstellung

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
Abkürzungsverzeichnis	VI
Symbolverzeichnis	VII
1 Einleitung	1
1.1 TODO	1
1.1.1 Format	1
1.1.2 Inhalt	1
2 Grundlagen	3
2.1 Soft Constraints	3
2.1.1 Zusätzliche Slackvariable	3
2.1.2 Using the KS function	3
3 Algorithmus	4
3.1 Erweiterung für nichtlineare Constraints	4
3.1.1 Erweiterung für Quadratic Problems mit quadratic Constraints . .	4
3.1.2 Erweiterung für Second Order Cone Problems	6
3.1.3 Komplette Formulierung	7
3.2 Anpassung für test cases	8
3.2.1 Allgemeine Beschreibung der test cases	8
3.2.2 Algorithmus mit Prädiktionshorizont gleich eins	9

4 Ergebnisse	11
4.1 Testprobleme	11
4.1.1 Einfaches QP	12
4.1.2 QP in PCE Formulierung	12
4.1.3 Referenzproblem	12
4.1.4 QP in PCE Formulierung mit QC	12
4.1.5 QP in PCE Formulierung mit SOCC	12
4.1.6 QP in PCE Formulierung mit Softconstraints (KSF)	12
4.1.7 QP in PCE Formulierung mit Softconstraints (Slack)	13
4.2 Vergleich der Testergebnisse	13
Literaturverzeichnis	14
Selbstständigkeitserklärung	15

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

MPC Modellprädiktive Regelung

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 κ 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]

Tääast

2.1 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 werde ich 2 Möglichkeiten erläutern solche Soft Constraints einzuführen, die ich im Zuge meiner Masterarbeit bzgl der auswirkung des nichteinhaltens der exact penalty function untersuche und vergleiche.

2.1.1 Zusätzliche Slackvariable

2.1.2 Using the KS function

3 Algorithmus

3.1 Erweiterung für nichtlineare Constraints

Um mit dem effizienten Algorithmus aus [WB10] auch kompliziertere/komplexere [TODO] Probleme, in Kapitel 3.1.2 Second Order Cone Problems (SOCP), bzw in Kapitel 3.1.1 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.

3.1.1 Erweiterung für Quadratic Problems mit quadratic Constraints

Einführendes zu QCQPs

QCQP-Formulierung

Zusätzliche Ungleichungsnebenbedingung sieht wie folgt aus:

$$x^T \Gamma x + \beta^T x \leq \alpha \quad (3.1)$$

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

$$-f_j(z) = \alpha_j - z^T \Gamma_j z - \beta_j^T z, \quad j = 1 \dots p \quad (3.2)$$

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.

$$\nabla f_j(z) = 2z^T \Gamma_j + \beta_j^T \quad (3.3)$$

$$\nabla^2 f_j(z) = 2\Gamma_j \quad (3.4)$$

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

$$\nabla \phi(z) = \sum_{k=1}^{lT+f_f+p} \frac{1}{\begin{bmatrix} h_i \\ \alpha_j \end{bmatrix}_k - \begin{bmatrix} p_i^T \\ \beta_j^T + z^T \Gamma_j \end{bmatrix}_k} z \begin{bmatrix} p_i^T \\ \beta_j^T + 2z^T \Gamma_j \end{bmatrix}_k \quad (3.5)$$

Mit

$$\hat{P}(z) = \begin{bmatrix} P \\ \beta_1^T + z^T \Gamma_1 \\ \vdots \\ \beta_p^T + z^T \Gamma_p \end{bmatrix}, \quad \hat{h} = \begin{bmatrix} h \\ \alpha_1^T \\ \vdots \\ \alpha_p^T \end{bmatrix} \quad (3.6)$$

lässt sich das auch einfacher schreiben:

$$\nabla \phi(z) = \hat{P}^T(2z) \hat{d} \quad (3.7)$$

Wobei

$$\hat{d}_k = \frac{1}{\hat{h}_k - \hat{p}_k(z)z} \quad (3.8)$$

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

$$\nabla^2 \phi(z) = \hat{P}(2z) \text{diag}(\hat{d})^2 \hat{P}(2z) + \sum_{j=1}^p \left(\hat{d}_{(lT+l_f)+j} 2\Gamma_j \right) \quad (3.9)$$

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.

3.1.2 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]]:

$$\|Ax + b\|_2 \leq c^T x + d \quad (3.10)$$

Als generalized inequality nimmt Gleichung 3.10 leicht andere Form an:

$$\|Ax + b\|_2^2 \leq (c^T x + d)^2 \quad (3.11)$$

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

$$-f_j(x) = (c_j^T x + d_j)^2 - \|A_j x + b_j\|_2^2 \quad (3.12)$$

Alle k barrier function Funktionen lassen sich zu

$$-f_k(z) = - \begin{bmatrix} f_i \\ f_j \end{bmatrix}_k = \begin{bmatrix} h_i \\ 0 \end{bmatrix}_k - \begin{bmatrix} p_i^T z \\ \left(\|A_j z + b_j\|_2^2 - (c_j^T z + d_j)^2 \right) \end{bmatrix}_k \quad (3.13)$$

zusammenfassen. Dabei lässt sich f_j auch noch auflösen zu

$$f_j(z) = d_j^2 - b_j^T b_j - \left[- (c_j^T z + 2d_j) c_j^T + (z^T A_j^T + 2b_j^T) A_j \right] z \quad (3.14)$$

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.

$$\nabla f_k(z) = \begin{bmatrix} p_i^T \\ \left(-2 \left((c_j^T z + d_j) c_j - A_j^T (A_j z + b_j) \right) \right)^T \end{bmatrix}_k \quad (3.15)$$

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.

$$\nabla \phi(z) = \hat{P}^T(2z)\hat{d} \quad (3.16)$$

Mit

$$\nabla^2 f_k(z) = \begin{bmatrix} 0 \\ -2(c_j c_j^T - A_j^T A_j) \end{bmatrix}_k \quad (3.17)$$

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

$$\nabla^2 \phi(z) = \hat{P}(2z) \text{diag}(\hat{d})^2 \hat{P}(2z) + \sum_{j=1}^p \left(\hat{d}_{(l_T+l_f)+j} - 2(c_j c_j^T - A_j^T A_j) \right) \quad (3.18)$$

[TODO richtiger Index für d]

3.1.3 Komplette Formulierung

Ungleichungsnebenbedingungen

$$\hat{P}(z)z \leq \hat{h} \quad (3.19)$$

mit

$$\hat{P}(z) = \begin{bmatrix} P \\ \beta_1^T + z^T \Gamma_1 \\ \vdots \\ \beta_p^T + z^T \Gamma_p \\ -(c_1^T z + 2d_1) c_1^T + (z^T A_1^T + 2b_1^T) A_1 \\ \vdots \\ -(c_q^T z + 2d_q) c_q^T + (z^T A_q^T + 2b_q^T) A_q \end{bmatrix}, \quad \hat{h} = \begin{bmatrix} h \\ \alpha_1^T \\ \vdots \\ \alpha_p^T \\ d_1^2 - b_1^T b_1 \\ \vdots \\ d_q^2 - b_q^T b_q \end{bmatrix} \quad (3.20)$$

Daraus ergibt sich die Logarithmic barrier function zu

$$\phi(z) = \sum_{i=1}^{l_T+l_f+p+q} -\log(\hat{h}_i - \hat{p}_i^T(z)z) \quad (3.21)$$

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

$$\nabla\phi(z) = \hat{P}^T(2z)\hat{d} \quad (3.22)$$

mit

$$\hat{d}_i = \frac{1}{\hat{h}_i - \hat{p}_i^T(z)z} \quad (3.23)$$

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

Hessian der der Logarithmic barrier function für Berechnung des Φ

$$\begin{aligned} \nabla^2\phi(z) = & \hat{P}(2z)\text{diag}(\hat{d})^2\hat{P}(2z) \\ & + \sum_{i=lT+lf+1}^{lT+lf+p} \left(\hat{d}_i 2\Gamma_i \right) + \sum_{j=lT+lf+p+1}^{lT+lf+p+q} \left(-2\hat{d}_j (c_j c_j^T - A_j^T A_j) \right) \end{aligned} \quad (3.24)$$

3.2 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.

3.2.1 Allgemeine Beschreibung der test cases

Nach [MM97] haben die test cases folgende Form:

$$\begin{aligned} \min \quad & \hat{c}^T \hat{x} + \frac{1}{2} \hat{x}^T \hat{Q} \hat{x} \\ \text{s.t.} \quad & \hat{A} \hat{x} = \hat{b} \\ & \hat{l} \leq \hat{x} \leq \hat{u} \end{aligned} \quad (3.25)$$

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

$$\hat{b}_{lower} \leq \hat{A} \hat{x} \leq \hat{b}_{upper} \quad (3.26)$$

Vereinheitlicht für 3.25 und 3.26 schreiben

$$\begin{aligned} \min \quad & \hat{c}^T \hat{x} + \frac{1}{2} \hat{x}^T \hat{Q} \hat{x} \\ \text{s.t.} \quad & \hat{b}_{lower} \leq \hat{A} \hat{x} \leq \hat{b}_{upper} \\ & \hat{l} \leq \hat{x} \leq \hat{u} \end{aligned} \tag{3.27}$$

Wobei sich für

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

die Gleichungsnebenbedingungen

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

ergeben

3.2.2 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:

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

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

$$H = \frac{1}{2}\hat{Q}, \quad g = \hat{c}, \text{ nicht korrekt, einzelne Untermatrizen setzen} \quad (3.28)$$

gesetzt werden. Die Ungleichungsnebenbedingung

$$F_u u(t) + F_x x(t) + F_f x(t+1) \leq f = f_u + f_x \quad (3.29)$$

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

$$x(t+1) = Ax(t) + Bu(t) \quad (3.30)$$

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

$$\hat{b} = -Ax(t) \quad \text{mit} \quad A = -I \quad (3.31)$$

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 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:

$$x_{k+1} = . \quad (4.1)$$

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 σ_1 und σ_2 variiert, wobei die Werte für die Unsicherheit im angepassten System

$$x_{k+1} = \quad (4.2)$$

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.

4.1.1 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 $H \in \mathbb{R}^{6T \times 6T}$.

4.1.2 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.

4.1.3 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 (4.1.1) und Problem 2 (4.1.2) sowohl mit dem zugeschnittenen Algorithmus als auch ohne die Matrixstruktur auszunutzen gelöst.

4.1.4 QP in PCE Formulierung mit QC

4.1.5 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.

4.1.6 QP in PCE Formulierung mit Softconstraints (KSF)

Erhöht man die Unsicherheit des Systems weiter, so kommt es zu Fällen in denen der Algorithmus versagt, da kein feasilber 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.

4.1.7 QP in PCE Formulierung mit Softconstraints (Slack)

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

4.2 Vergleich der Testergebnisse

Verglichen werden hier

Besonderes Augenmerk auf Laufzeit, um den Vorteil des Grundalgorithmus zu zeigen.

Mit verschiedenen Horizontlängen. Vielleicht unconstraint QP ref \leftrightarrow QP

QP PCE ref \leftrightarrow 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.

Literaturverzeichnis

- [BV04] BOYD, Stephen ; VANDENBERGHE, Lieven: *Convex Optimization*. University Press, 2004
- [MM97] MAROS, István ; MÉSZÁROS, Csaba: A Repository of Convex Quadratic Programming Problems. (1997)
- [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.

6. Oktober 2015

Hannes Heinemann