Inhaltsverzeichnis

Eine Python-Toolbox zur datengetriebenen Modellierung von Spritzgießprozessen und Lösung von Optimalsteuerungsproblemen zur Steuerung der Bauteilqualität

Alexander Rehmer[1], Andreas Kroll[1]

[1]Fachgebiet Mess- und Regelungstechnik, Universität Kassel
Mönchebergstr. 7, 34125 Kassel
E-Mail: {alexander.rehmer,andreas.kroll}@mrt.uni-kassel.de

Kurzfassung

Eine Voraussetzung für die Realisierung weitgehend autonomer Fertigungssysteme ist das Vorhandensein von mathematischen Modellen, mit denen das Verhalten einer Produktionsanlage mit ausreichend hoher Genauigkeit prädiziert werden kann. Solche Modelle können in modellbasierten Optimierungsverfahren eingesetzt werden, deren Ergebnis zur Unterstützung des Anlagenfahrers oder zur Selbstoptimierung der Produktionsanlage verwendet werden kann. Produktionsprozesse weisen hoft hochgradig dynamisches und nichtlineares Verhalten auf, sodass eine Modellbildung für Anwender nicht zu bewältigen ist. In diesem Beitrag wird eine Toolbox zur Unterstützung bei der Modellierung und modellbasierten Prozessoptimierung eines Produktionsverfahrens, dem Kunststoff-Spritzgießprozess, vorgestellt.

1 Einführung

Die Digital Twin of Injection Molding (DIM) Toolbox wurde mit Python 3.8 entwickelt und getestet. Sie stellt grundsätzlich einen Wrapper für das CasADi-Framework dar, einer Open-Source Bibliothek für nichtlineare Optimierung

und algorithmische Differentation [1]. Die prototypische DIM-Toolbox stellt Klassen und Funktionen bereit, welche die spezifischen Anforderungen bei der Modellbildung des Spritzgießprozesses berücksichtigt. Darüber hinaus werden numerische Optimalsteuerungsmethoden zur modellbasierten Prozessoptimierung bereitgestellt. Die Toolbox wird unter der BSD-Lizenz vertrieben und kann auf GitHub[1] heruntergeladen werden.

1.1 Digitale Zwillinge in Produktionsprozessen

Durch die im Kontext von Industrie 4.0 und dem Internet of Things (IoT) voranschreitenden Entwicklungen auf dem Gebiet der Automatisierung und Digitalisierung stehen erstmals umfassende und zeitlich hochaufgelöste Daten von physischen Produktionssystemen (PPS) wie dem Kunststoff-Spritzgießen zentral zur Verfügung. Dies ermöglicht die datengetriebene Bildung eines statischen oder dynamischen Modells, d.h. eines Digitalen Zwillings (DT), welcher in Verbindung mit dem physischen Produktionssystem ein cyber-physisches Produktionssystem bildet (CPPS). In Verbindung mit modellbasierten Regelungs- und Optimierungsverfahren kann der Digitale Zwilling zur Optimierung des Produktionsprozesses, meist hinsichtlich der Bauteilqualität, eingesetzt werden, und ist damit grundlegend für die Realisierung eines intelligenten Fertigungssystems [2].

Herausforderungen hierbei sind zum einen die Bildung eines Digitalen Zwillings, insbesondere falls eine dynamische Modellierung eines komplexen Produktionsprozesses wie dem Spritzgießen angestrebt wird. Zum anderen muss der Digitale Zwilling in das CPPS integriert werden. Während für die software- und datenseitige Integration des DT bereits Referenzarchitekturen entwickelt wurden [2, 3, 4], existieren keine Softwarebibliotheken die den Anwender bei der eigentlichen Bildung eines DT unterstützen sowie Methoden zur modellbasierten Prozessoptimierung bereitstellen. In diesem Beitrag wird daher eine im Rahmen des Projekts Digital Twin of Injection Molding[2] (DIM) entwickelte Toolbox zur datengetriebenen Modellbildung und modellbasierten Optimierung des Spritzgießprozesses vorgestellt.

[1] `https://github.com/MRT-RT/DigitalTwinInjectionMolding.git`
[2] `https://www.uni-kassel.de/go/DIM`

Bild 1: Spritzgießmaschine Arburg 470S. *Quelle: Fachgebiet Kunststofftechnik, IfW, Universität Kassel.*

1.2 Kunststoff-Spritzgießen

Das Spritzgießen ist ein urformendes Fertigungsverfahren bei dem plastifizierter Kunststoff unter Druck in eine Form, das Spritzgießwerkzeug, eingespritzt wird. Bild 1 zeigt die im Rahmen des Projektes verwendete Spritzgießmaschine. Der Prozess lässt sich üblicherweise in drei Phasen unterscheiden:

Einspritzphase: Eine definierte Schmelzmenge wird durch eine translatorische Bewegung der Schnecke in die Form injiziert. Diese Phase ist meist geschwindigkeitsgeregelt.

Nachdruckphase: Um den Materialschwund durch das Abkühlen der Schmelze in der Form zu kompensieren, wird ein definiertes Druckprofil aufgebracht. In dieser Phase wird meist der von der Maschine generierte hydraulische Druck geregelt.

Abkühlphase: In der drucklosen Abkühlphase soll das Bauteil soweit abkühlen, dass es sich deformationslos Entformen lässt.

Industrielle Spritzgießmaschinen verfügen meist über maschineninterne Regler, deren Parameter für den Nutzer unzugänglich sind. Darüber hinaus kann der Nutzer oft keine Sollwerttrajektorien $\{r_k\}_{k=0}^{T}$ vorgeben, sondern lediglich einige Maschinenparameter s, welche die Sollwerttrajektorie parametrieren. Hierbei sei k die diskrete Zeit und T die Dauer eines Spritzgießzyklus. Die gemessenen Prozessgrößen $\{p_k\}_{k=0}^{T}$ spiegeln den Verlauf des Prozesses wider und bestimmen auf unbekannte Weise die Eigenschaften Q des hergestellten

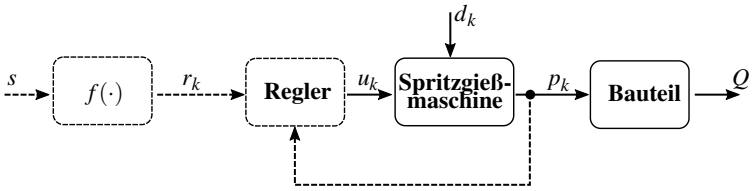

Bild 2: Schematische Darstellung des geregelten Spritzgießprozesses.

Bauteils. Häufig werden ausschließlich sogenannte maschinenseitige Prozessgrößen wie Geschwindigkeit v_k und Position der Schnecke x_k sowie Druck im Hydraulikaggregat p_k^{hyd} erfasst, da es sich hierbei um die Zielgrößen der maschineninternen Regler handelt. Entscheidend für die Bauteilqualität sind jedoch Prozessgrößen in der Kavität, bspw. der Werkzeuginnendruck p_k^{cav} und die Temperatur im Werkzeug T_k^{cav}. Bild 2 zeigt ein Blockschaltbild der gesamten Prozesskette.

2 Konzept der Modellbildung und Optimalsteuerung

Die Toolbox soll Anwender in Industrie und Praxis bei der Modellierung und Optimierung Ihrer Produktionsanlagen unterstützen. Produktionsanlagen unterscheiden sich oft bspw. hinsichtlich der implementierten Regelungskonzepte und verfügbaren Messgrößen. Eine Anforderung an die Toolbox ist somit ein modularer Aufbau, welcher dem Anwender erlaubt nur die für ihn sinnvollen Funktionalitäten zu nutzen.

Die in Bild 3 dargestellten von der Toolbox bereitgestellten wesentlichen Funktionalitäten sind

- Dynamische Qualitätsmodelle: Die Bild von Prozessgrößentrajektorien $\{p_k\}_{k=0}^{T}$ auf ein einzelnes Qualitätsdatum Q nach Ablauf der diskreten Zykluszeit T wird durch Modellstrukturen mit interner Dynamik realisiert. Diese Modelle können verwendet werden, um den optimalen Prozessgrößenverlauf für eine geforderte Bauteilqualität Q zu ermitteln.

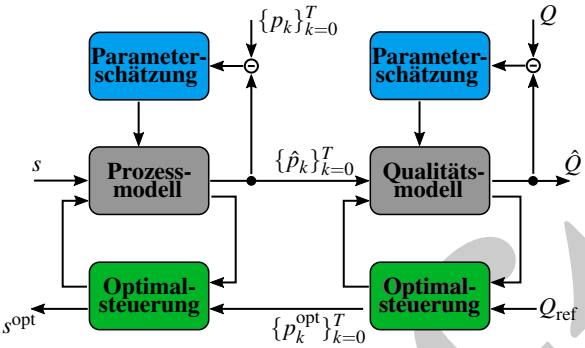

Bild 3: Modellierungs- und Optimierungskonzept.

- Dynamische Prozessmodelle: Ermöglichen die Bild von Stellgrößentrajektorien $\{u_k\}_{k=0}^T$ oder Maschinenparametern s_k auf die resultierenden Prozessgrößentrajektorien $\{p_k\}_{k=0}^T$.
- Parameteroptimierung: Methoden zur online und offline Parameteroptimierung der verwendeten Modellstrukturen.
- Optimalsteuerungsmethoden: Auf den Spritzgießprozess zugeschnittene Methoden zur Lösung von Optimalsteuerungsproblemen.

3 Toolbox

3.1 Datenstruktur

Um Kompatibilität zwischen allen Klassen zu gewährleisten, ist ein einheitliches Datenformat erforderlich. Für numerische und sequenzielle Daten existieren bereits etablierte Datentypen. Daher wurde beschlossen als einheitliches Datenformat für die Klassen der DIM-Toolbox ein Dictionary zu verwenden, in welchem die Daten in einer definierten Weise gespeichert werden. Die im einzelnen verwendeten Datentypen sind Listen, der `DataFrame`-Datentyp der `pandas`-Bibliothek und der `ndarray`-Datentyp der `numpy`-Bibliothek. verwendet. Wie in Bild 4 dargestellt, werden die Daten in einem Dictionary mit den Keys `'io_data'`, `'init_state'` und `'switch'` organisiert:

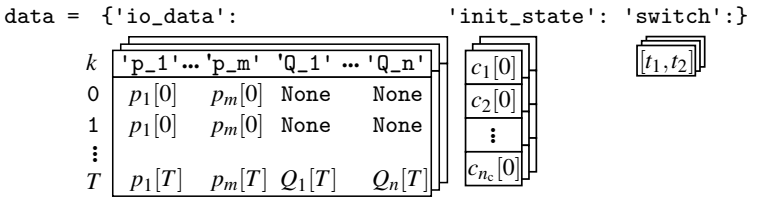

Bild 4: Standardisiertes Datenformat

`'io_data'`: Eine Liste von `pandas-DataFrames`. Jeder `DataFrame` enthält die mit einem Spritzgießzyklus korrespondierenden Prozessgrößen $\{p_k\}_{k=0}^{T}$ und Qualitätsgrößen Q_T. Da Q_T vektoriell ist, sind auch Werkzeuge mit mehreren Kavitäten abbildbar.

`'init_state'`: Eine Liste von `numpy-ndarrays`. Nur erforderlich für dynamische Modelle. Jedes Array repräsentiert den initialen Zustandsvektor des Modells zum Zyklusbeginn und hat eine der Modellordnung `dim_c`, siehe Bild 5, entsprechende Anzahl an Zeilen.

`'switch'`: Eine Liste von Listen, welche jeweils die diskreten Umschaltpunkte für den korrespondierenden Zyklus als `int` enthalten. Nur erforderlich, wenn mehrere zeitinvariante dynamische Modelle mittels einer der Wrapper-Klassen, siehe Bild 7, zu einem zeitvarianten schaltenden Modell verbunden werden. Auf diese Weise wird dem Modell mitgeteilt, zu welchen diskreten Zeitpunkten von einem zum nächsten Teilmodell umzuschalten ist.

3.2 Modellklassen

3.2.1 Basisklassen

Die Basisklasse für alle Modelle ist `Model`. `Model` stellt grundlegende Attribute, wie die Anzahl an Eingangsgrößen `dim_u` und Ausgangsgrößen `dim_y` sowie Funktionen, wie die Einschrittprädiktion `one_step_prediction()`, bereit, siehe Bild 5. Aus der `Model`-Klasse wurden wiederum zwei Klassen abgeleitet: `Dynamic` und `Static`. Die Klasse `Dynamic` stellt alle grundlegenden

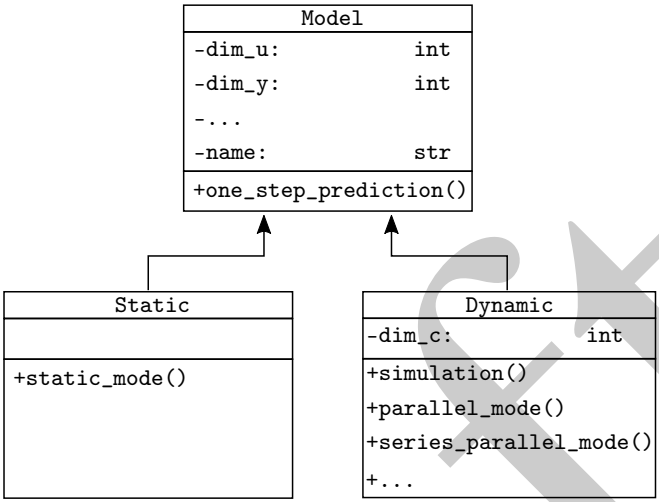

Bild 5: Basisklasse `Model` und daraus abgeleitete Klassen `Static` und `Dynamic`

Methoden eines dynamischen Modells bereit, d.h. die Auswertung in parallel Konfiguration `parallel_mode()` oder seriell-paralleler Konfiguration `series_parallel_mode()`. Die dynamische Ordnung des Modells wird durch das Attribut `dim_c` festgelegt. Sowohl interne Dynamikmodelle

$$\begin{aligned} x_{k+1} &= h(x_k, u_k) \\ y_k &= g(x_k) \end{aligned} \tag{1}$$

mit Eingangsgröße $u_k \in \mathbb{R}^{n_\mathrm{u}}$, Zustandsgröße $x_k \in \mathbb{R}^{n_\mathrm{x}}$ und Ausgangsgröße $y_k \in \mathbb{R}^{n_\mathrm{y}}$ als auch externe Dynamikmodelle

$$y_k = f(y_{k-1}, \ldots, y_{k-m}, u_{k-1}, \ldots, u_{k-n}) \tag{2}$$

können aus dieser Basisklasse abgeleitet werden, siehe Bild 6. Die Modellgleichungen müssen durch den Nutzer in der Methode `Initialize()` der abgeleiteten Klasse als CasADi MXFunction definiert werden. Eine Reihe gängiger rekurrenter Modellstrukturen, wie die Gated Recurrent Unit (GRU) und das Long Short-Term Memory (LSTM) Netz, sind bereits vorimplementiert. Indem nur noch die Modellgleichungen nach einem vorgegebenen Schema

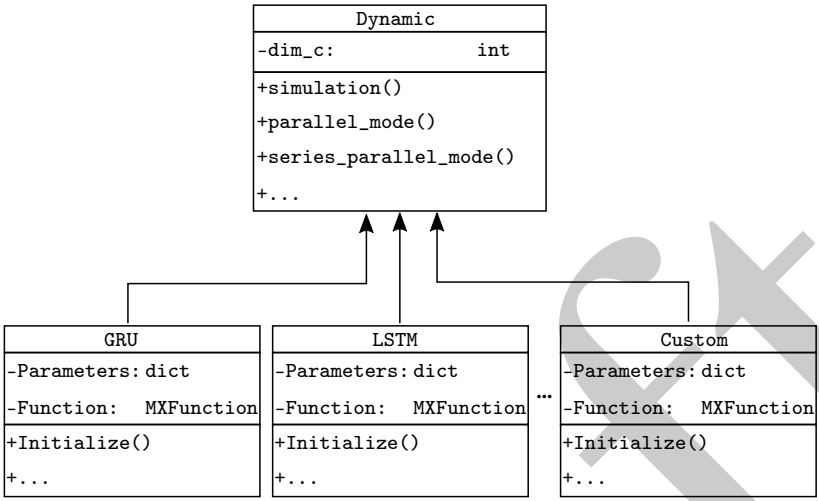

Bild 6: Basisklasse `Dynamic` und daraus abgeleitete Klassen

definiert werden müssen, soll dem Nutzer die die Implementierung eigener Modellansätze einfachst möglich gemacht werden.

Zudem wurden die Modellklassen so konstruiert, dass dasselbe Objekt vom Nutzer sowohl an Funktionen zur Parameterschätzung als auch zur Lösung numerischer Optimalsteuerungsprobleme übergeben werden kann. Hierfür werden bei der Definition der Modellgleichungen sowohl die Eingängsgrößen als auch die Modellparameter als symbolische Variablen definiert. Bei der Modellparameteroptimierung bzw. Optimierung der Eingangsgrößen werden dann die Eingangsgrößen bzw. Modellparameter automatisch von symbolischen in numerische Variablen umgewandelt. Aus der Basisklasse `Static` können statische Modellklassen, deren Gleichung vom Typ

$$y = f(u)$$

ist, abgeleitet werden. Verbreitete statische Modellstrukturen, wie Polynome und Multilayer-Perceptrons (MLP), sind bereits implementiert.

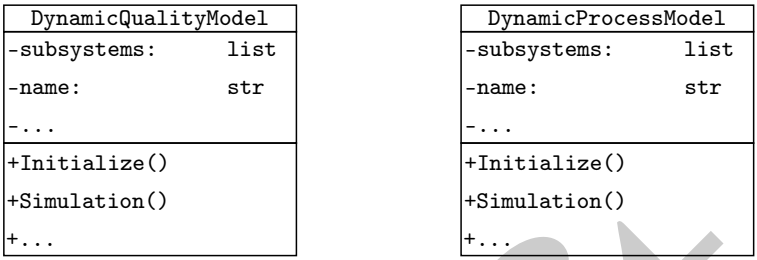

Bild 7: Wrapper-Klassen `DynamicQualityModel` und `DynamicProcessModel` zur Realisierung schaltender dynamischer Modelle.

3.2.2 Wrapper-Klassen

Wie in Abschnitt 1.2 erläutert, lässt sich der Spritzgießprozess in drei Phasen mit unterschiedlichem dynamischen Verhalten unterteilen. Bei einer dynamischen Modellierung wird dem Rechnung getragen, indem ein dediziertes Modell für jede Phase geschätzt wird. Das entstehende Gesamtmodell ist somit ein zeitvariantes schaltendes System bestehend aus drei zeitinvarianten Subsystemen $i = 1,2,3$ mit Umschaltpunkten t_1 und t_2:

$$\begin{aligned} x_{k+1} &= h^i\left(x_k, u_k; \Theta^i\right), \quad x_0 = 0, \quad k \in \mathbb{Z}^+, \quad i = \begin{cases} 1 \,\forall\, k \le t_1, \\ 2 \,\forall\, t_1 < k \le t_2, \\ 3 \,\forall\, k > t_2 \end{cases} \\ y_k &= g^i\left(x_k; \Phi^i\right) \end{aligned} \tag{3}$$

Damit ein aus mehreren Teilmodellen bestehendes Modell genauso ausgewertet, optimiert und für Optimalsteuerungsprobleme verwendet werden kann wie Modelle der Klasse `Dynamic()`, wird jeweils eine Wrapper-Klasse für Qualitätsmodelle `DynamicQualityModel` und für Prozessmodelle `DynamicProcessModel` bereitgestellt, siehe Bild 7. Die Teilmodelle vom Typ `Dynamic` werden der Wrapper-Klasse als Attribut `subsystems` übergeben. Die Teilmodelle werden dann mittels der `Simulation`-Methode in der richtigen Reihenfolge ausgewertet und die internen Modellzustände an den definierten Umschaltpunkten übergeben.

ParamOptimizer	
-model:	int
-data_train:	data
-data_val:	data
-...	
+optimize()	
-...	

Bild 8: Klasse `ParamOptimizer` zur Optimierung der Parameter von Objekten der Klasse `Model`

3.3 Parameteroptimierung

Der Optimierer minimiert die Modellparameter θ so, dass die quadratische Kostenfunktion

$$\mathscr{L}(\theta) = \frac{1}{2} \left(\hat{y}_T(\theta) - y_T\right)^T \left(\hat{y}_T(\theta) - y_T\right) \tag{4}$$

minimiert wird. Bei der Erzeugung einer Instanz der Klasse `ParamOptimizer` wird eine Instanz der CasADi-Klasse `Opti` erstellt mithilfe derer das Optimierungsproblem formuliert wird. Die Modellparameter θ des zu optimierenden Modells `model` werden als Zielgrößen des Optimierungsproblems definiert. Durch Aufrufen der `optimize`-Methode wird das Modell (in Abhängigkeit der symbolischen Modellparameter) sowohl auf den Trainingsdaten `data_train` als auch auf den Validierungsdaten `data_val` ausgewertet. Die Parameterschätzung erfolgt auf den Trainingsdaten. Der Parametersatz, welcher während der Optimierung die geringsten Validierungskosten ergab, wird an den Nutzer zurückgegeben. Zur Parameterschätzung wird IPOPT [5] verwendet. Neben den zuvor beschriebenen grundlegenden Funktionalitäten, sind folgende zusätzliche Funktionalitäten in der Toolbox implementiert:

- Parallelisierung: Um die Initialisierungsabhängigkeit nichtlinearer Optimierungsprobleme zu berücksichtigen, werden oft Multistart-Strategien verfolgt. `ParamOptimizer` ermöglicht die parallele Optimierung mehrerer Initialisierungen.
- Online-Optimierung: Durch Übergabe von Optionen an den Solver - IPOPT können bspw. eine approximative Berechnung der Hesse-Matrix,

eine maximale Anzahl an Iterationen oder eine maximale Optimierungsdauer eingestellt werden.

- Randbedindungen: Es können Randbedingungen an den Optimierer übergeben werden.
- Parameter einfrieren: Parameter, die nicht optimiert werden sollen, können vom Nutzer eingefroren werden.

3.4 Numerische Optimalsteuerung

Wie aus Bild 3 hervorgeht, soll die Toolbox dem Nutzer die Möglichkeit bieten, zwei Optimalsteuerungsprobleme zu lösen:

- Die Ermittlung der optimalen einzustellenden Maschinenparameter s^{opt} gegeben die Referenztrajektorien für eine oder mehrere Prozessgrößen $\{p_k^{\text{ref}}\}_{k=0}^{T}$ unter Verwendung eines dynamischen Prozessmodells.
- Die Ermittlung der optimalen Prozessgrößentrajektorien $\{p_k^{\text{opt}}\}_{k=0}^{T}$ gegeben eine Referenz der zu realisierenden Bauteilqualität Q_{ref} unter Verwendung eines dynamischen Qualitätsmodells.

Für die Lösung des erstgenannten Optimierungsproblems stellt die Klasse `ProcessMultiStageOptimizer` eine auf den Spritzgießprozess zugeschnittene Variante des sogenannten Mehrfachschießverfahrens (*engl. direct multiple shooting*) bereit. Hinsichtlich des Spritzgießprozesses zu berücksichtigende Anforderungen umfassen insbesondere, dass zu definierten Zeitinstanzen zwischen Teilmodellen umgeschaltet wird. Sind die Eingangsgrößen des Prozessmodells zudem Sollwerttrajektorien für maschineninterne Regler, so können diese meist nicht beliebig vorgegeben werden, sondern sind Funktionen einiger weniger Maschinenparameter. `ProcessMultiStageOptimizer` erlaubt, die funktionale Abhängigkeit der Sollwerttrajektorien `reference_input` von den einstellbaren Maschinenparametern `ref_param` zu definieren. Somit sind dann nicht die gesamten Sollwerttrajektorien Gegenstand der Optimierung, sondern nur einige wenige Maschinenparameter.

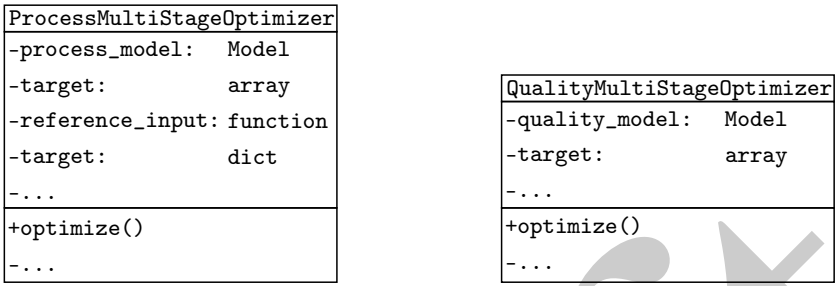

Bild 9: Klassen `QualityMultiStageOptimizer` zur Lösung von Optimalsteuerungsproblemen mit einem Einschießverfahren und `ProcessMultiStageOptimizer` zur Lösung von Optimalsteuerungsproblemen mit einem Mehrfachschießverfahren.

Die Ermittlung der optimalen Prozessgrößentrajektorien $\{p_k^{\text{opt}}\}_{k=0}^{T}$ kann nicht mit einem Mehrfachschießverfahren gelöst werden, da für die Ausgangsgröße, d.h. die Bauteilqualität, keine Trajektorie sondern nur ein Endwert zur Verfügung steht. Daher wurde mit der Klasse `QualityMultiStageOptimizer`, siehe Bild 9, ein auf Einfachschießverfahren (*engl. direct single shooting*) basierender Löser implementiert. Das Umschalten zwischen Teilmodellen zu definierten Zeitpunkten wurde hier ebenfalls berücksichtigt sowie die Kostenfunktion an die Erreichung eines Endwertes angepasst.

4 Anwendungsbeispiel

In diesem Anwendungsbeispiel soll die Identifikation eines dynamischen Qualitätsmodells mittels der Toolbox demonstriert werden. Sowohl die Daten als auch der vollständige Code in Form eines Jupyter-Notebooks für diese Fallstudie sind in dem einleitend erwähnten GitHub-Repository verfügbar.

Aus umfassenden Voruntersuchungen [6] ist bekannt, dass eine Modellstruktur mit interner Dynamik, die zu guten Ergebnissen führt, die sogenannte Gated Recurrent Unit (GRU) ist. Die Zustandsgleichung dieser rekurrenten Netzarchitektur lautet:

$$\hat{c}_{k+1} = f_{\text{z}} \odot \hat{c}_k + (1 - f_{\text{z}}) \odot f_{\text{c}}. \tag{5}$$

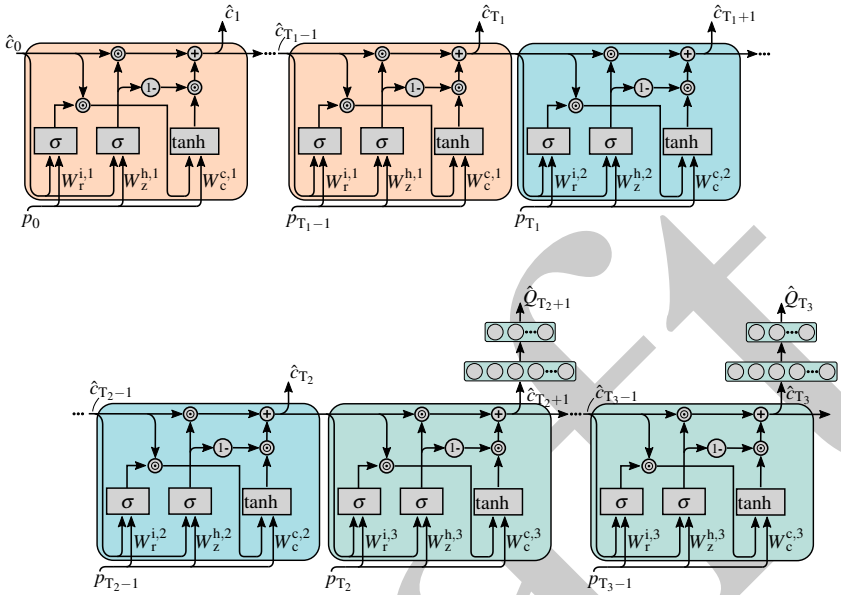

Bild 10: Zeitvariantes schaltendes Qualitätsmodell bestehend aus drei GRU Netzen und einem gewöhnlichen MLP als Ausgabegleichung.

Der Operator $\odot$ steht für das Hadamard-Produkt. Die Aktivierungen der sogenannten *Gates* f_r, f_z und f_c ergeben sich gemäß

$$\begin{aligned} f_\mathrm{r} &= \sigma\left(W_\mathrm{r} \cdot [\hat{c}_k, u_k]^T + b_\mathrm{r}\right), \\ f_\mathrm{z} &= \sigma\left(W_\mathrm{z} \cdot [\hat{c}_k, u_k]^T + b_\mathrm{z}\right), \\ f_\mathrm{c} &= \tanh\left(W_\mathrm{c} \cdot \left[\tilde{\hat{c}}_k, u_k\right]^T + b_\mathrm{c}\right), \end{aligned} \tag{6}$$

mit $\tilde{\hat{c}}_k = f_\mathrm{r} \odot \hat{c}_k$.

Daher soll ein dynamisches Qualitätsmodell bestehend aus drei GRUs, jeweils für Einspritz-, Nachdruck- und Abkühlphase geschätzt werden, siehe Bild 10. Qualitätsrelevante Prozessgrößen sind die unmittelbar in der Form gemessene Temperatur `T_wkz_ist` und der Druck `p_wkz_ist`. Modelliert werden soll der Innendurchmesser `D_i` des produzierten Bauteils, [1] in Bild 11.

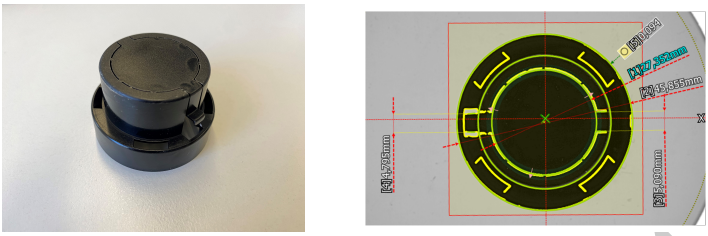

Bild 11: Foto des produzierten Bauteils (links) sowie Draufsicht mit optisch erfasster Bauteilgeometrie. *Quelle: Fachgebiet Kunststofftechnik, IfW, Universität Kassel.*

Die Bezeichnung der Ein- und Ausgangsgrößen des Modells muss mit den Spaltenbezeichnungen in den `DataFrames`, siehe Bild 4, übereinstimmen. Algorithmus 1 zeigt, wie das in Bild 10 dargestellte Modell mit der DIM-Toolbox generiert werden kann. Im Folgenden werden beispielhaft GRUs mit einem internen Modellzustand `dim_c=1` verwendet, `dim_c=1` entspricht n_x in (1). Die Ausgabegleichung, vergleiche (1), wird durch ein vorwärtsgerichtetes Neuronales Netz mit `dim_hidden=10` verdeckten Neuronen approximiert.

Algorithmus 1: Initialisierung des Modells in Bild 10

```
from DIM.models.model_structures import GRU
from DIM.models.injection_molding import DynamicQualityModel

u_label = ['p_wkz_ist','T_wkz_ist']
y_label = ['D_i']

options = {'dim_u':2,'dim_c':1,'dim_hidden':10,
    'dim_out':1,'u_label':u_label,'y_label':y_label}

GRU_inj = GRU(**options,name='GRU_inj')
GRU_press = GRU(**options,name='GRU_press')
GRU_cool = GRU(**options,name='GRU_cool')

QModel = DynamicQualityModel(
    subsystems=[GRU_inj,GRU_press,GRU_cool], name='QModel')
```

Die Parameteroptimierung erfolgt, indem das zeitvariante schaltende Qualitätsmodell zusammen mit Trainingsdaten `data_train` und Validierungsdaten `data_val`, welche wie in Abschnitt 3.1 beschrieben in einem Dictionary or-

ganisiert sind, an ein Objekt der Klasse `ParamOptimizer` übergeben werden und dessen `optimize`-Methode aufgerufen wird, siehe Algorithmus 2.

Algorithmus 2: Parameteroptimierung des Modells in Bild 10

```
import pickle
from DIM.optim.param_optim import ParamOptimizer

data_train, data_val = pkl.load(open('data.pkl'))

param_optimizer = ParamOptimizer(model=QModel,
    data_train=data_train,data_val=data_val,
    initializations=10)

optim_results = param_optimizer.optimize()
```

`optimize` gibt den Wert der Kostenfunktion auf den Trainings- und Validierungsdaten sowie die zugehörigen Parametersätze als `DataFrame` zurück. Der Nutzer kann dann entscheiden, welche Parameter er dem Modell letztendlich zuweisen möchte. Für gewöhnlich wird es sich dabei um den Parametersatz handeln, welcher mit den geringsten Kosten auf den Validierungsdaten korrespondiert. Für die Zuweisung der Parameter zu allen Teilmodelle verfügt `DynamicQualityModel` über die `SetParameters()`-Methode, siehe Algorithmus 3.

Algorithmus 3: Zuweisung der Modellparameter

```
val_min = optim_results['loss_val'].idxmin()

QModel.SetParameters(optim_results.loc[val_min,'params_val']
```

Das dynamische Qualitätsmodell kann dann zur Prädiktion der Bauteilgüte verwendet werden, indem es simulativ mittels der `parallel_mode`-Methode auf Prozessgrößenverläufen ausgewertet wird.

Algorithmus 4: Auswertung des Modells in Bild 10 auf Prozessgrößenverläufen

```
sim_val = QModel.parallel_mode(data_val)
```

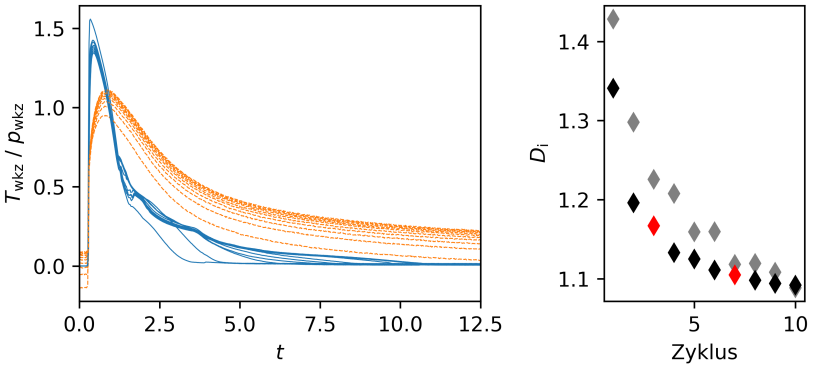

Bild 12: Links: Normierte Prozessgrößenverläufe der zehn betrachteten Produktionszyklen. Rechts: Normierter Bauteildurchmesser, Trainingsdaten in schwarz, Validierungsdaten in rot, Modellprädiktion in grau.

In Bild 12 sind die Ergebnisse der simulativen Auswertung auf den Trainings- und Validierungsdaten für einige Spritzgießzyklen mit gleichen Prozessparametern s visualisiert. Es ist deutlich zu erkennen, dass das dynamische Qualitätsmodell in der Lage ist, die variierende Bauteilqualität (bei identischen Prozessparametern s) mittels der Prozessgrößenverläufe zu prädizieren.

5 Zusammenfassung

Die (dynamische) Modellierung komplexer Produktionsprozesse ist eine herausfordernde Aufgabe. Mit der auf dem CasADi-Framework vorgestellten DIM-Toolbox wird diesbezüglich ein Beitrag zur Unterstützung von Anwendern in Industrie und Wissenschaft geleistet. Die Toolbox stellt in Klassen implementierte Modellstrukturen bereit, welche von einfachen statischen Modellen bis hin zu dynamischen schaltenden Modellen reichen. Anhand einer ausführlichen Dokumentation und bereits vorimplementierter Modelle (GRU, LSTM, MLP, Polynome) ist der Anwender in der Lage, ohne großen Aufwand eigene Modelle zu implementieren. Die Modellstrukturen sind uneingeschränkt kompatibel zu den implementierten Klassen zur Parameteroptimierung zur Lösung von Optimalsteuerungsproblemen. Alle Optimierungsverfah-

ren basieren auf dem erprobten und gut dokumentierten Löser IPOPT. Es handelt sich hierbei um ein Innere-Punkte-Verfahren, sodass auch eine Optimierung unter Nebenbedingungen möglich ist.

Neben dem Spritzgießprozess sind die Methoden der Toolbox auf andere Batch-Fertigungsprozesse übertragbar. Insbesondere solche, bei denen das Problem der Abbildung von Zeitreihen auf einen Endwert auftritt.

Danksagung

Dieses Projekt wird vom Land Hessen und dem Europäischen Fond für regionale Entwicklung (EFRE 2014-2020) gefördert, Projekt: Digital Twin of Injection Molding (DIM) FKZ: 0107/20007409.

Ein Dank an die Kollegen, insbesondere Marco Klute, des Instituts für Werkstofftechnik (IfW) - Fachgebiet Kunststofftechnik der Universität Kassel für die Planung und Durchführung der Experimente zum Zweck der Erhebung der in dieser Publikation verwendeten Daten.

Literatur

[1] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings und M. Diehl. „CasADi – A software framework for nonlinear optimization and optimal control“. In: *Mathematical Programming Computation*. 2018.

[2] Y. Lu, C. Liu, I. Kevin, K. Wang, H. Huang und X. Xu. „Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues“. In: *Robotics and Computer-Integrated Manufacturing* 61. S. 101837. 2020.

[3] P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz und A. Wortmann. „Model-driven development of a digital twin for injection molding“. In: *International Conference on Advanced Information Systems Engineering*. S. 85-100. 2020.

[4] F. Tao und M. Zhang. „Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing“. In: *Ieee Access* 5. S. 20418–20427. 2017.

[5] A. Wächter und L. T. Biegler. „On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming“. In: *Mathematical programming* 106. S. 25-57. 2006.

[6] A. Rehmer, M. Klute, A. Kroll und H.-P. Heim. „An internal dynamics approach to predicting batch-end product quality in plastic injection molding using Recurrent Neural Networks“. In: *Proc. of the 6th IEEE Conference on Control Technology and Applications (CCTA)*. 2022.

[7] A. Rehmer, M. Klute, A. Kroll und H.-P. Heim. „A Digital Twin for Part Quality Prediction and Control in Plastic Injection Molding“. In: *Modeling, Identification, and Control for Cyber-Physical Systems Towards Industry 4.0*, eingereicht. 2022.