Anbindung eines aktiven Steuerkraftsystems an eine Echtzeitsimulation

Henrik Wüstenberg[1], Jürgen Gotschlich[2], Umut Durak[2]
[1]Hochschule Ostfalia
[2]DLR Institut für Flugsystemtechnik
he.wuestenberg@ostfalia.de
{ juergen.gotschlich, umut.durak }@dlr.de

In der Flugsimulation werden aktive Steuerkraftsysteme häufig für Forschungszwecke genutzt. Ein Grund dafür ist die Verbesserung der realistischen Umgebung durch simulierte Steuerkräfte. Die aktiven Steuerorgane erzeugen diese Steuerkräfte, um dem Piloten das Gefühl eines realen Luftfahrzeugs zu vermitteln. Ein Steuerkraftsystem funktioniert grundsätzlich autonom. Jedoch bringt eine Anbindung an die Simulationsinfrastruktur Vorteile, indem Steuerkraftkonfigurationen flexibel geändert werden können.

Das Deutsche Zentrum für Luft- und Raumfahrt e. V. (DLR), Institut für Flugsystemtechnik, betreibt am Standort Braunschweig den Air VEhicle Simulator (AVES). Es wird ein Airbus A320 Simulator und ein Eurocopter EC135 Simulator genutzt. Die Simulatoren werden über die im DLR entwickelte Simulations-Infrastruktur *2Simulate* verbunden, welche für eine deterministische Ausführung von Prozessen sorgt. Jeder Prozess erfüllt eine spezifische Aufgabe, wie z.B. Verbindungen zu externen Geräten oder auch die Berechnung dynamischer Modelle. Beide Simulatoren verwenden leistungsfähige aktive Steuerkraftsysteme der Fa. Wittenstein Aerospace & Simulation GmbH. Die jeweiligen Achsen werden nach einer Force-Feel Charakteristik geregelt. Im Umfang dieser Publikation wird die Schnittstelle zwischen Simulation und dem aktiven Steuerkraftsystem vorgestellt, die in *2Simulate* mittels eines WCLS-Tasks (Wittenstein Control Loading System) implementiert ist. Anhand einer Beispielanwendung werden Konfigurationsänderungen am Steuerkraftsystem demonstriert.

1 Einleitung

Aktive Steuerkraftsysteme werden in Flugsimulatoren häufig für Forschungszwecke genutzat. Eine steigende Anzahl Simulatoren wird mit diesen Systemen betrieben, da die taktile Wahrnehmung, als weiterer Kanal zum Piloten, ein großer Vorteil ist [1]. Sie verbessert das Bewusstsein des Piloten für eine Flugsituation und verstärkt die realistische Umgebung im Simulator.

Das DLR betreibt am Standort Braunschweig den AVES, einen Airbus A320 Simulator und einen Eurocopter EC135 Simulator. Diese Arbeit befasst sich mit dem aktiven Steuerkraftsystem im Hubschrauber Simulator [2]. Das Steuerkraftsystem trägt die Produktbezeichnung „Reconfigurable Helicopter Mission Trainer“ (RHMT). Dieser setzt sich hauptsächlich aus einer Regelungseinheit (SCM), mehreren Versorgungseinheiten (SPS) und den Aktuatoren für jede Achse zusammen [3]. Das Steuerkraftsystem wird im AVES grundsätzlich zur Forschung genutzt. Die Realitätsnähe der Flugsimulation wird hier durch Handling Quality Studien mit Piloten getestet. Dafür ist es entscheidend die hinterlegten Kraft-Positions-Kennlinien während der Simulation zu beeinflussen, um sie auf besondere Flugzustände anzupassen. Beispielsweise werden durch die Umströmung der Steuerflächen am Flugzeug Kräfte erzeugt, die bei einem mechanischen System entgegen den Steuereingaben des Piloten wirken. Dafür simuliert das aktive Steuerkraftsystem diese Kräfte. Die Modifikationen an den Parametern der Regelung können durch eine graphische Benutzeroberfläche der Fa. Wittenstein Aerospace & Simulation GmbH, namentlich AktivToolkit, durchgeführt werden [4].

Diese Publikation befasst sich mit der Schnittstelle zwischen der Simulation und dem aktiven Steuerkraftsystem. Dafür wird die Simulationsinfrastruktur *2Simulate* des DLR verwendet, eine Software für verteilte Echtzeitsimulationsanwendungen. Diese Software stellt dem Anwender Prozesse zur Verfügung die spezifische Aufgaben erfüllen und bei der Programmentwicklung unterstützen. Die Schnittstelle zum Steuerkraftsystem wird durch einen *WCLS-Task* in *2Simulate* verwendet. Des Weiteren wird eine Beispielanwendung vorgestellt, die Veränderungen am Wittenstein System mithilfe der WCLS-Task Methoden durchführt. Die Funktionen des Prozesses übertragen Parameterwerte über ein spezifisches Daten-Transfer-Protokoll an das SCM des Steuerkraftsystems. Zur Programmentwicklung wird für die Beispielanwendung der Toolkit Protokoll Simulator der Fa. Wittenstein verwendet. Der Emulator verfügt über grundsätzliche Funktionen des aktiven Steuerkraftsystems, wie das Wechseln von Achsenzuständen [5].

Das zweite Kapitel umfasst die Idee aktiver Steuerkraftsysteme im Allgemeinen und geht auf Details des Wittenstein Systems ein. Im Anschluss be-

schreibt das dritte Kapitel den Aufbau des 2Simulate-Frameworks und erläutert den Wittenstein-Task. Das vierte Kapitel beschreibt die Beispielanwendung und erläutert ihre Funktionen. Ein Ausblick folgt im fünften Kapitel dieser Publikation.

2 Das aktive Steuerkraftsystem

Während ein Flugzeug fliegt, fließt eine Luftströmung über die Oberfläche und erzeugt einen Widerstand. Besonders an den Steuerflächen, also an den Rudern des Luftfahrzeugs, entsteht durch Reibung ein hoher nichtlinearer Widerstand. Dieser muss bei mechanischen Flugsteuerungen vom Piloten überwunden werden. Um diese Steuerkräfte in der Simulation zu realisieren, wird ein geregelter Antrieb benötigt. Dafür werden hydraulische oder elektronische Aktuatoren verwendet [7]. Das aktive Steuerkraftsystem der Wittenstein AG ist ein elektronisches System mit leistungsstarken Gleichstrommotoren. Diese Motoren geben eine Kraftrückmeldung in Abhängigkeit von einer Kraft-Positions-Kennlinie, wie in Abbildung 1 dargestellt. Somit wird die vom Piloten aufzubringende Kraft aus der Kennlinie und in Abhängigkeit von der Auslenkung des Steuerelements berechnet [3].

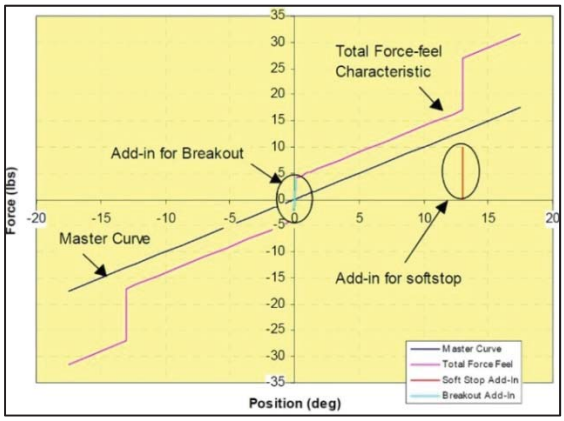

Abbildung 1 Kraft-Position Kennlinie [3]

Die flexiblen Einstellungen des Wittenstein Systems für die Kraft-Positions-Kennlinien werden durch zahlreiche Funktionen beim Konfigurieren umgesetzt. Beispielsweise kann die Grundkurve (Master-Curve) durch weitere zusätzliche Kurven (Addin-Curves) erweitert werden. Die Zusatzkurven ermöglichen Soft-Stop's, Breakout's und weitere besondere Rückmeldungen zum Piloten [4]. Abbildung 1 zeigt wie die Regelungssoftware des Steuerkraftsystems „Aktiv8" die Kennlinie aus einigen Kurven zusammensetzt.

2.1 Die Regelungsstruktur

Die Entwicklung des RHMT basiert auf der Verwendung von „Commercial-Off-The-Shelf" (COTS) Komponenten. Weswegen die Leistung der einzelnen Geräte weitestgehend ausgeschöpft werden soll [7]. Dafür wurde von Wittenstein eine dezentrale Regelung des Systems entworfen, siehe Abbildung 2. Das SCM (System Control Module) dient neben der Regelung auch für die Kommunikation im Steuerkraftsystem. Dafür besitzt es mehrere Schnittstellen, wie Ethernet und CAN-Bus, zur Kommunikation mit Aktuatoren, den SPS oder anderen externen Geräten [3].

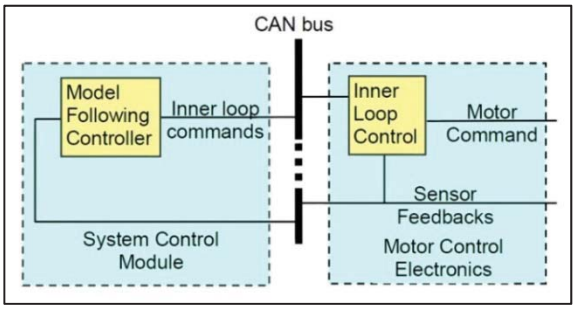

Abbildung 2 Regelungsstruktur [7]

Abbildung 2 zeigt im linken Bereich das SCM. Zur Regelung nimmt das SCM Positions- und Kraftmesswerte von der Motorregelungseinheit auf. Die Position entspricht der Auslenkung des Steuerorgans und die Kraft entspricht der aufgebrachten Steuerkraft des Piloten. Der sogenannte Model-Following-Regler des SCM besitzt ein hinterlegtes dynamisches Modell über das das Antwortverhalten der einzelnen Aktuatoren berechnet wird. Eingangsgrößen der Regelung sind die Messwerte der Sensoren am Aktuator. Das Antwortverhalten wird folglich als Steuerbefehl über ein CAN-Bus Netzwerk an den entsprechenden Aktuator übertragen. Die innere Regelung dient hier als Steuerelektronik für den Gleichstrommotor und erzeugt entsprechend dem Antwortverhalten des Modells und der Kraft-Position Kennlinie ein Gegenmoment auf das Steuerelement [7].

2.2 Die Softwarestruktur

Das SCM betreibt die Regelungssoftware *Aktiv8*. Sie unterteilt sich in die drei Komponenten *Aktiv8 Control*, *Aktiv8 DLM* und *Configuration File Set*. *Aktiv8 Control* ist generell für die Regelung zuständig, siehe Kapitel 2.1. Die *Aktiv8 DLM*- und *Configuration File Set*-Komponenten speichern Dateien zwischen und stellen sie bei Bedarf der Regelung zur Verfügung [3].

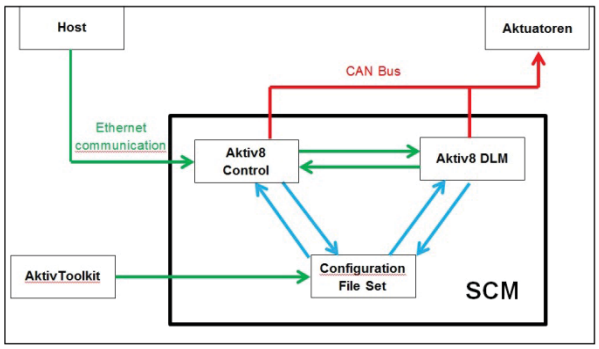

Abbildung 3 Die Wittenstein Softwarestruktur

Abbildung 3 zeigt das SCM mit seinen Software-Komponenten. Hier wird im oberen Bereich die Kommunikation des SCM mit dem Host dargestellt. Diese Ethernet Verbindung dient der Übertragung von Steuerbefehlen aus der laufenden Simulation an das Steuerkraftsystem [3]. Die Grundlagen zur Nutzung dieser Verbindung werden in den folgenden Kapiteln erläutert. Abgebildet ist ebenfalls die CAN-Bus-Verbindung zwischen dem SCM und den Aktuatoren.

Der untere Bereich von Abbildung 3 zeigt das AktivToolkit, eine Software die von einem Standard-Personal-Computer mit Microsoft Windows Betriebssystem verwendet wird. Sie dient als die bisherige Schnittstelle, um die Systemparameter des aktiven Steuerkraftsystems zu modifizieren. Diese Anwendung enthält alle Funktionen die im Umfang von Forschungsprojekten während der Simulation genutzt werden sollen, um die Kraft-Positions-Kennlinie und andere Parameter zu verändern. Zum Beispiel kann die Trimmungsposition, die Haft- und Gleitreibung, die Kopplung von Achsen, etc. eingestellt werden. Diese veränderten Konfigurationen am Steuerkraftsystem werden in dem *Configuration File Set* gespeichert. Weitere Funktionen des AktivToolkit umfassen die Messung der getätigten Steuereingaben während oder im Anschluss an die Simulation. Das *Aktiv8 DLM* speichert das dynamische Modell für die Regelung und stellt es *Aktiv8 Control* zur Verfügung [4].

3 2Simulate

Am DLR wird *2Simulate* als Framework für verteilte zyklisch geplante Echtzeitanwendungen entwickelt. Es wird in C++ programmiert und teilt sich in drei Komponenten auf, namentlich 2Simulate Real-Time-Framework (2SimRT), 2Simulate Model Control (2SimMC) und 2Simulate Control Center (2SimCC). Die erste Komponente 2SimRT stellt den Hauptbestandteil dar und sorgt für eine deterministische Planung und Regelung von Echtzeitprozessen. 2SimMC ist eine generische Modellschnittstelle für Modelle die aus einer Reihe von Modellierungssprachen entwickelt wurden. Beispielsweise MATLAB/Simulink oder 2Simulate Modelling Language (TSML).

Die letzte Komponente 2SimCC ist eine Bedienoberfläche für 2Simulate und wird zur Steuerung der laufenden Anwendungen verwendet [8].

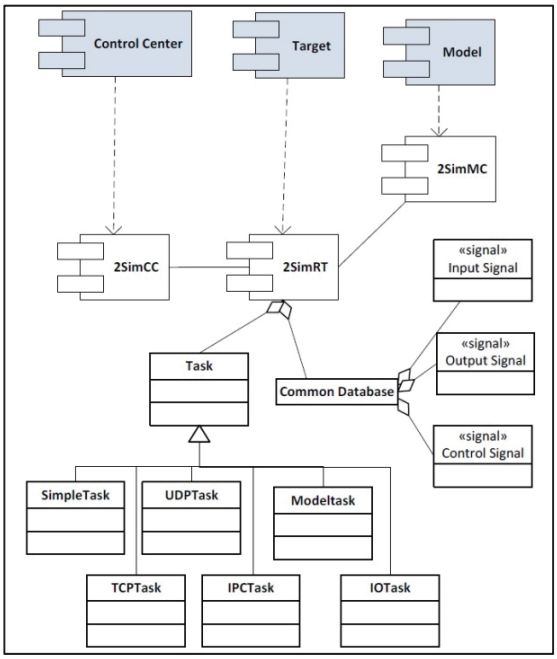

Abbildung 4 2Simulate Übersicht

Die Komponente 2SimRT stellt eine Anzahl an Prozessen in Form einer Klassenbibliothek zur Verfügung, siehe Abbildung 4. Der Nutzer kann die Prozesse durch eine Reihe von Callback Funktionen an seine Erfordernisse anpassen, die vor bzw. nach der Initialisierung sowie der Prozessierung ausgeführt werden. Beispielsweise kann ein Nutzer mit einem UDP-Task eine spezifische UDP-Verbindung betreiben oder mit einem TCP-Task eine spezifische TCP-Verbindung betreiben [8]. Die Verbindung zum Steuerkraftsystem wird über eine UDP Verbindung betrieben, siehe Kapitel 3.1. Ein Task wird durch die Instanziierung innerhalb einer 2Simulate-Anwendung zum Prozess und erfüllt für ein Nutzerprogramm eine spezifische Aufgabe.

Außerdem zählt eine generische Datenbank zu 2Simulate, die Ein-, Ausgangs- und Kontrollsignale für jede Anwendung in einer hierarchischen Struktur darstellt. Die Grundlage für die sogenannte *Common Database* sind SCD-Dateien, die in einem tabellarischen Format alle Signale auflisten. Beim Erstellen einer 2Simulate-Anwendung werden die Signale aus der SCD-Datei durch eine [externe] Anwendung in generische Strukturen umgewandelt. Die Strukturen werden über Headerdateien in eine Applikation eingebunden und können genutzt werden. Die Datenbank wird generell verwendet, um den Datenfluss durch interne und externe Schnittstellen handzuhaben [8].

3.1 Der WCLS-Task

Das aktive Steuerkraftsystem versendet Daten über eine UDP-Verbindung an die Simulationsinfrastruktur. Deswegen ist der *WCLS-Task* von dem UDP-Task abgeleitet und enthält entsprechende Methoden, um die UDP-Verbindung zu überprüfen, zu konfigurieren und zu verwenden. Zusätzlich enthält der *WCLS-Task* eigene Methoden zur Konfiguration des Steuerkraftsystems. Zum Beispiel eine Methode zum Hinzufügen von Achsen, die für jede zu verwendende Achse einmal aufgerufen werden muss. Dabei muss jeweils eine Struktur zum Empfangen und Versenden von Daten pro Achse übergeben werden.

4 WCLS-Beispielanwendung

Die Beispielanwendung wird mit einem Steuerkraftemulator der Fa. Wittenstein entwickelt. Er kann die Anbindung des Steuerkraftsystems an den Hubschraubersimulator nachbilden. Der Simulator stellt einen Großteil der Funktionen der realen Anlage zur Verfügung und ermöglicht es grundsätzliche Funktionen zu testen [5]. In den folgenden Kapiteln werden der Aufbau und die Funktionsweise der Beispielanwendung erklärt.

4.1 Die Hauptfunktion

Die Anwendung beginnt mit dem Hinzufügen von Headerdateien der verwendeten *2Simulate*-Prozesse und der *Common Database*. Darauf folgt die Hauptfunktion, sie wird teilweise in Listing 1 dargestellt. Die Funktion erstellt eine 2Simulate Root-Applikation und initiiert die Common Database. Innerhalb der Funktion wird ebenfalls eine Instanz des *WCLS-Task* erzeugt. Anschließend wird das UserCmd-Signal, über die setUserCtrl-Funktion gesetzt. Das UserCmd Signal schaltet den Zustand der Achsen, siehe auch Kapitel 4.2. Weiterhin werden innerhalb der Funktion Funktionszeiger für Callback-Funktionen an die 2Simulate Root-Applikation übergeben. Außerdem werden die Achsen des Steuerkraftsystems über eine AddAxis-Funktion hinzugefügt.

```
int main( int argc, char *argv[] )
{
//---- generate 2Simulate root ap-
plication ----
TSim* pTSim=new TSim("TUT06",
"2Simulate Tutorial6","1.0");

//---- setup common database ----
TSimulateTutorial6Database(
pTSim, &tSimulateTutorial6Common);
[...]
//---- WclsTask ----
TSimWclsTask* pWclsTask=new TSim-
WclsTask(pTSim,"WclsTask",
TASK_SCHED_RR, 30, 10*iMSECtoNSEC,
TRUE, TRUE, TRUE);
pWclsTask->setDesc("WclsTask for
CLS");
pWclsTask->setHost(
"127.0.0.1",3002,3001);
[...]
pWclsTask->setUserCtrl(
(int*)&com->c.wcls.UserCmd,
(int*)&com->c.wcls.State,
(int*)&com->c.wcls.AxisState);

//Callback implementation
pWclsTask->setPreProcCB(
(void(*)(TSim*,TSimRtTask*))
&pWclsTask_PreProcCB);

pWclsTask->setIntermediateProcCB(
(void(*)(TSim *,TSimUdpTask
*))&pWclsTask_IntermediateProcCB);

//Add axis
pWclsTask->addAxis(
(WCLSTASK_AXIS_DATA_RCV *)
&com->i.r.wcls.sp.PosPit,
(WCLSTASK_AXIS_DATA_SND *)
&com->o.r.wcls.sp.PosOffPit);
[...] }
```

Listing 1 Main-Funktion

In der Beispielanwendung werden ausschließlich die Callback-Funktion vor der Prozessierung (PreProcCB) und während der Prozessierung (IntermediateProcCB) genutzt.

4.2 Callback-Funktionen der Prozessierung

Innerhalb jedes Zyklus einer Prozessierung wird jede Callback-Funktionen aufgerufen [8]. Im Folgenden soll der wiederholte Ablauf der Prozessierung erläutert werden und anschließend die Funktionsweise der PreProcCB und der IntermediateProcCB in der Beispielanwendung erklärt werden.

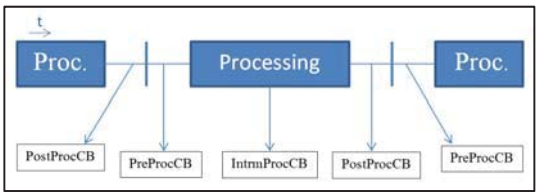

Abbildung 5 Abstrakter Prozessierungsablauf

Abbildung 5 zeigt den zeitlichen Verlauf mehrerer Prozessierungszyklen hintereinander. Dabei stellen die vertikalen Markierungen das Ende, bzw. den Anfang, eines Zyklus dar. Die PreProcCB wird vor der Prozessierung eines Zeitintervalls aufgerufen. Sie überprüft die UDP-Verbindung zwischen Simulator und aktivem Steuerkraftsystem, sowie den Betriebsstatus der Achsen. Während der Prozessierung wird der IntermediateProcCB aufgerufen. Dieser führt diverse Parameteränderungen durch, schaltet Zustände und verändert Kennlinien. Abbildung 5 stellt auch den PostProcCB im Prozessierungsablauf dar. Er wird nach der Prozessierung

eines Zyklus aufgerufen und kann zum Beispiel die berechneten Steuerkräfte kalibrieren.

Der PreProcessCB beginnt den Status der UPD-Verbindung zu überprüfen, sowie den Status der Achsen abzufragen. Listing 2 zeigt einen Ausschnitt aus der PreProcessCB-Funktion.

```
void pWclsTask_PreProcCB(
TSim *pAppl, TSimRtTask *pRtTask)
{
   [...]
   else if(
   iClsCmd==WCLSTASK_STATE_ACTIVE
   ||
   iClsCmd==WCLSTASK_STATE_ACTPEND
   )
   { com->c.WCLS.UserCmd =
   WCLSTASK_USERCMD_START; }

   else if(
   iClsCmd==WCLSTASK_STATE_STARTUP
   )
   {com->c.WCLS.UserCmd =
   WCLSTASK_USERCMD_INIT;}

   else if(
   iClsCmd==WCLSTASK_STATE_SETOFF
   )
   {com->c.WCLS.UserCmd =
   WCLSTASK_USERCMD_OFF;}
   [...]
}
```

Listing 2 PreProcess Callback-Funktion

Für die Achsenüberprüfung wird ein Signal namens UserCmd verwendet. Der Wert dieses Signals entscheidet über einen Wechsel der Achsenzustände, siehe Kapitel 4.1. Um das UserCmd zu beeinflussen, wird ein weiteres Signal aus der Datenbank genutzt, namentlich iClsCmd. Dieses wird durch eine bedingte Verzweigung in Listing 2 abgefragt und setzt dadurch den Wert der UserCmd-Variable.

Der IntermediateProcCB führt die Initialisierung von Parametern durch und vermeidet damit Fehler beim Ersten Programmdurchlauf. Eine Bedingung entscheidet, ob Veränderungen an einer Achse durchgeführt werden sollen. Dabei werden Integer-Werte von 0 bis 7 angenommen, welche entsprechend der acht unterschiedlichen Achsen Parameter verändern. Die Bedingung wird mit dem Trim-Release-Status als Beispiel in Listing 3 dargestellt.

```
void pWclsTask_IntermediateProcCB(
TSim *pAppl,TSimUdpTask *pUdpTask)
{
   [...]
   if ( com->c.WCLS.EditAxis >= 0
   && com->c.WCLS.EditAxis <= 7 ) {
      [...]
      //--set trim release status--
      fTrimRel[com->c.WCLS.EditAxis]
      = lcast(&com->c.WCLS.TrmRel[
      com->c.WCLS.EditAxis]);

      pWCLSTask->setTrimRel(
      com->c.WCLS.EditAxis,1,
      &fTrimRel[com-
      >c.WCLS.EditAxis]);
      }
   [...]
}
```

Listing 3 Parameteränderung aus IntermediateProcCB

Das Listing zeigt einen Ausschnitt aus dem verschachtelten Codeblock der oben genannten Bedingung. Die hier verwendete Funktion ändert den TrimRelease-Status indem sie der setTrimRel-Funktion einen Parameter übergibt, hier „0“ oder „1“. Für diese Funktion wird vorerst eine lokale Variable verwendet, um den Datentyp des Parameters zu casten. Dieser Cast ist notwendig, um die Variable als Argument zu übergeben. Anschließend wird die Speicheradresse des Parameters an die Funktion übergeben. Da die setTrimRel-Funktion Teil des WCLS-Tasks ist wird der Zugriffsoperator „->“ verwendet.

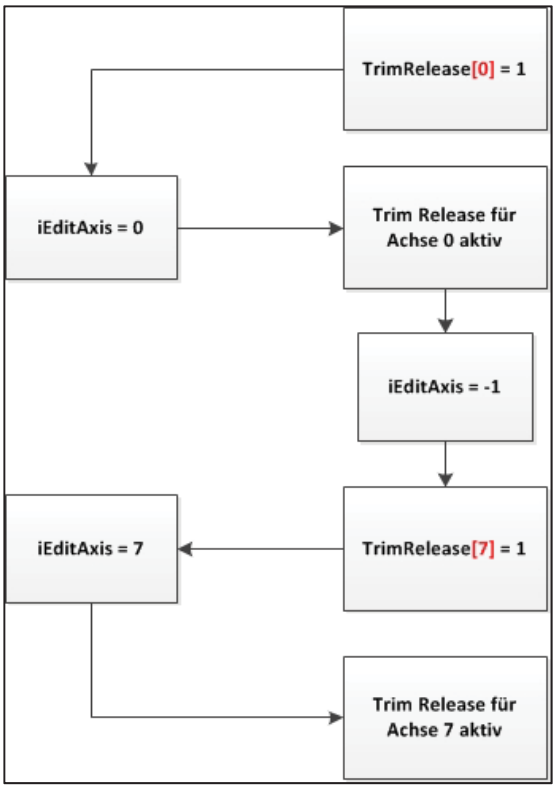

Abbildung 6 Beispiel der Parameterveränderung

Der oben beschriebene Funktionsablauf ist in Abbildung 6 als Ablaufschema dargestellt. Der genaue Ablauf beginnt damit eine achsenspezifische Variable eines Parameters auf einen gewünschten Wert zu setzen. Im Beispiel TrimRelease[0] = 1. Anschließend wählt man mit der iEditAxis-Variable

die zu verändernde Achse aus, wodurch die Bedienung aus Listing 3 verzweigt und die gewünschte Änderung ausgeführt wird. Abschließend wird die „iEditAxis“-Variable zurückgesetzt. Abbildung 6 verdeutlicht zusätzlich die Idee einer achsenspezifischen Variablen, indem der Ablauf der gleichen Änderung für eine andere Achse abgebildet ist.

Die Änderung der Kennlinien beinhaltet eine weitere Besonderheit. Bevor die Koordinaten einer Kennlinie übergeben werden, wird ein Programm zum Einlesen aufgerufen. Die eingelesenen Daten umfassen die Koordinaten, den Titel und weitere Bedingungen der Kennlinie. Für das Einlesen wird eine Textdatei verwendet, sie muss in einem definierten Format strukturiert sein.

5 Bewertung und Ausblick

Diese Publikation beschreibt das aktive Steuerkraftsystem der Fa. Wittenstein und geht auf die Fähigkeiten des *2Simulate*-Frameworks ein. Anhand einer Beispielanwendung wird eine mögliche Variante gezeigt, um Veränderungen am Steuerkraftsystem während einer *2Simulate*-Echtzeitanwendung durchzuführen.

Die graphische Benutzeroberfläche der Wittenstein Software, AktivToolkit, ist sehr umständlich bei Veränderungen an den Konfigurationen. Außerdem sind die Erklärungen der Handbücher nicht eindeutig und führen zu einer schweren Handhabung des Werkzeugs. Deswegen sollte der *WCLS-Task* eine graphische Benutzeroberfläche erhalten die die Übersichtlichkeit, besonders für Einsteiger, verbessert. Ebenso kann das Einlesen der Kennlinienkoordinaten vereinfacht werden, indem ein Anwenderfreundliches Programm wie Mircrosoft Excel zur Eingabe von Daten verwendet wird.

Im Umfang eines weiteren Projekts kann die erstellte Software an den Hubschraubersimulator angebunden werden. Damit können Nutzer von den Funktionen der Beispielanwendung in ihren eigenen Anwendungen Gebrauch machen.

6 Literaturverzeichnis

[1] von Grünhagen, Wolfgang, Müllhäuser, Mario et. Al.; „*Active Inceptors in FHS for Pilot Assistance Systems*“; 36th European Rotorcraft Forum; Paris, France; 7.-9.September 2010

[2] Duda, H., Gerlach, T., Advani S., Potter M.; „*Design of the DLR AVES Research Flight Simulator*“; AIAA Modeling and Simulation Technologies (MST) Conference, Boston, MA, 2013

[3] “*Wittenstein Control Loading Systems Manual*”; Wittenstein aerospace & simulation Inc; 17. Juli 2007

[4] “*Wittenstein AktivToolkit User Manual*”; Wittenstein aerospace & simulation Inc (2005)

[5] “*Wittenstein Control Loading Systems Protocol Simulator Instructions*”; Wittenstein aerospace & simulation Inc (2007)

[6] Allerton, David; “*Principles of Flight Simulation*”; John Wiley & Sons, United Kingdom; 2009

[7] Cowling, David; “*The Development of a new Range of Control Loading Systems*”; Royal Aeronautical Society Flight Simulation Group Meeting; United Kingdom, London; 3.-4. November 2004

[8] Gotschlich, J., Gerlach, T, Durak, U.; “*2Simulate: A Distributed Real-Time Simulation Framework*“; ASIM STS/GMMS Workshop 2014; Reutlingen, Germany; 2014

7 Abkürzungsverzeichnis

RHMT – Reconfigurable Helicopter Mission Trainer

AVES – Air VEhicle Simulator

DLR – Deutsches Zentrum für Luft- und Raumfahrt e. V.

SCM – System Control Module

SPS – System Power Supply

CAN-Bus – Controller Area Network

FbW – Fly-by-Wire

COTS – Commercial Off The Shelf

GUI – Graphische Benutzeroberfläche

2SimRT – 2Simulate Real-Time-Framework

2SimMC – 2Simulate Model Control

2SimCC – 2Simulate Control Center

SCD – Simulation Common Database

PreProcCB – Preprocessing Callback

IntrmProcCB – Intermediateprocessing Callback

PostProcCB – Postprocessing Callback