NTB
Interstaatliche Hochschule
für Technik Buchs
FHO Fachhochschule Ostschweiz

Neuer Sequencer für EEROS

Vertiefungsarbeit 1 2016/1017

von

Marcel Gehrig

Advisor:	Dr. Urs Graf
Abgabedatum:	8. Februar 2017

Inhaltsverzeichnis

1 Anforderungen an den Sequencer

1.1 Formulierung der Anforderungen

Im folgenden Kapitel werden die Anforderungen an den *Sequencer* beschrieben. Die Kapitel *Ziele*, *Nicht Ziele* und *Test Cases* beschreiben die Anforderungen auf verschiedene Arten. In *Ziele* und *Nicht Ziele* wird abstrakt beschrieben, welche Funktionen der neue *Sequencer* beinhalten, beziehungsweise nicht beinhalten muss. Im Kapitel *Test Cases* werden verschiedene Fälle beschrieben die mit dem neuen *Sequencer* möglichst elegant gelöst werden sollen.

1.2 Ziele

1.2.1 Einfaches Interface für den Applikationsentwickler

Mit dem bestehenden *Sequencer* sind vertiefte Programmierkenntnisse notwendig, um eine Sequenz zu erstellen, oder abzuändern. Sequenzen sind nicht intuitiv verständlich. Es werden Kenntnisse vom *Control System* und dem *Safety System* benötigt, um einen neuen Ablauf für den Roboter schreiben zu können. Zurzeit werden Sequenzen von den Steuerungsentwickler geschrieben.

Wenn der Roboter fertig entwickelt worden ist, soll er an einen Kunden übergeben werden können. Der Kunde, oder der Betreiber des Roboters, soll dann Änderungen im Ablauf des *Sequencers* vornehmen können, ohne dass er vertiefte Kenntnisse von der Programmiersprache C++ oder von der inneren Funktionsweise des Roboters haben muss. Dafür ist es notwendig, dass der Steuerungsentwickler den Roboter so abstrahiert, dass die Sequenzen aus logischen und verständlichen Schritten bestehen.

1.2.2 Flexibel einsetzbar für verschiedenste Arten von Roboter

Auch wenn die Sequenzen möglichst einfach aufgebaut werden sollen, muss der *Steuerungsentwickler* für alle möglichen Kategorien von Robotern Sequenzen bauen können. Verschiedene Arten von Roboter haben verschiedene Anforderungen. Das *Control System* von einem Roboterarm mit sechs Freiheitsgraden unterscheidet sich stark von einer Fertigungsstrassen mit mehreren Förderbändern. Trotz diesen Unterschieden soll es möglich sein, für beide Kategorien von Robotern sinnvolle Sequenzen zu erstellen. Das bedeutet, dass der *Steuerungsentwickler* möglichst viele Freiheiten beibehält, ohne dass er durch das *Framework* unnötig begrenzt wird.

1.2.3 Parallele und blockierende Sequenzen

Wie bereits im bestehenden Sequenzer verwirklicht wurde, sollen Sequenzen blockierend und nichtblockierend aufgerufen werden können. Diese Funktion von blockierenden und parallel ausgeführten Sequenzen soll auch im neuen *Sequencer* beibehalten werden.

1.2.4 Exception Handling

Eine *Exception* ist ein Ereignis, dass nicht immer auftritt, aber auftreten kann. Zu solchen *Exception*, oder Ausnahmen, gehören zum Beispiel:

1. Ein blockiertes Förderband
2. Ein Timeout
3. Der Roboter soll ein Paket abholen, dass nicht vorhanden ist

Solche Ausnahmen sollen im *Sequencer* erkennt werden können und flexibel darauf reagiert werden können. Eine solche Reaktion könnte eine spezielle Sequenz sein, die versucht, eine solche Ausnahme zu behandeln. Alternativ soll aber auch die aktuelle Sequenz abgebrochen, oder neu gestartet werden können.

1.2.5 Zugriff auf Control System

Der aktuelle *Sequencer* nutzt ein Pointer auf das *Control System* um Blöcke direkt auslesen und schreiben zu können. Wenn das *Control System* betrachtet wird, kann nicht festgestellt werden, welche Blöcke vom *Sequencer* ausgelesen oder geschrieben werden. Ein klares Interface zum *Sequencer* wäre wünschenswert, um das *Control System* übersichtlicher zu machen.

1.2.6 Safety System entlasten

Eine Analyse von bestehenden Implementationen des alten *Sequencers* hat gezeigt, dass das *Safety System* viele Aufgaben übernimmt, welche besser vom *Sequencer* übernommen werden sollten. Das *Safety System* sollte möglichst nur eingesetzt werden, um das System zu überwachen. Alle anderen Aufgaben sollen vom *Sequencer* oder vom *Control System* übernommen werden.

1.3 Nicht Ziele

1.3.1 Echtzeit

Das *Control System* und das *Safety System* laufen beide in einem Echtzeit-Task. Der *Sequencer* soll aber bewusst nur mit normaler Priorität laufen und besitzt keine Echtzeit Fähigkeit.

Da der *Sequencer* keine Regelung berechnet, benötigt er keine Echtzeit Fähigkeit. Eine niedrigere Priorität als das *Control System* und das *Safety System* ist notwendig, dass die beiden Systeme nicht vom *Sequencer* beeinträchtigt werden.

1.3.2 Pfadplanung

Die Pfadplanung ist nicht Teil des *Sequencers*, da sie den Rahmen dieser Arbeit sprengen würde.

1.4 Test Cases

1.4.1 Einleitung

Die Testfälle sind so aufgebaut, dass sie möglichst einfach und elementar sind. Jeder *Test Case* beschreibt eine andere Anforderung oder Spezialfall an den *Sequencer*. Alle in der Realität vorkommenden Situation sollte durch einen, oder einer Kombination von mehreren, *Test Cases* beschrieben werden können.

Eine Ausnahme dazu bildet *Test Case 8*. In diesem Testfall wurden möglichst viele verschiedene Situationen vereint.

Mit dem *Sequencer* sollen alle Testfälle sauber und elegant umgesetzt werden werden können.

1.4.2 Test Case 1: Achse einfach

System

- Eine Achse die sich nach links und rechts bewegen kann
- An beiden Enden befindet sich ein Endschalter
- Ein Taster *Taster links*, der die Achse nach links laufen lässt
- Ein Taster *Taster rechts*, der die Achse nach rechts laufen lässt

Aufgabe

- Solange *Taster links* gedrückt bleibt, fährt die Achse nach links
- Solange *Taster rechts* gedrückt bleibt, fährt die Achse nach rechts
- Die Achse hält an, wenn einer der beiden Endschalter erreicht wird

Herausforderungen

- Der *Sequencer* muss die Eingänge *Taster links*, *Taster rechts* und die beiden Endschalter permanent überwachen und auf eine Änderung reagieren.

1.4.3 Test Case 2: EEDURO Delta Roboter Maus

System

- EEDURO Delta Roboter mit Maus

Aufgabe

- Während dem Idle Zustand wird auf einen Input von der Maus gewartet
- Wenn sich die Maus für fünf Sekunden nicht bewegt, wird eine blockierende *Autor-Sort Sequenz* gestartet
- Bewegt sich die Maus innerhalb von fünf Sekunden, dann bewegen sich die Achsen entsprechend der Mausbewegung und der 5-Sekunden-Timer wird neu gestartet

Herausforderungen

- Timeout
- Blockierende Sequenz

1.4.4 Test Case 3: Rendezvous

System

- Greifer Zubringer
- Greifer Abholer
- Paket: Gegenstand, der übergeben wird
- Förderband Zubringer: Hält ständig ein neues Paket bereit für *Greifer Zubringer*
- Förderband Abholer: Transportiert ständig alle Pakete weg, welche vom *Greifer Abholer* abgelegt werden

Ablauf (Siehe Abbildung 1.1)

1. Der *Greifer Zubringer* holt ein neues Paket vom *Förderband Zubringer*
2. Der *Greifer Zubringer* bringt das Paket in Rendezvous-Position
3. Der *Greifer Abholer* übernimmt das Paket vom *Greifer Zubringer*
4. Der *Greifer Abholer* legt das Paket auf das *Förderband Abholer*

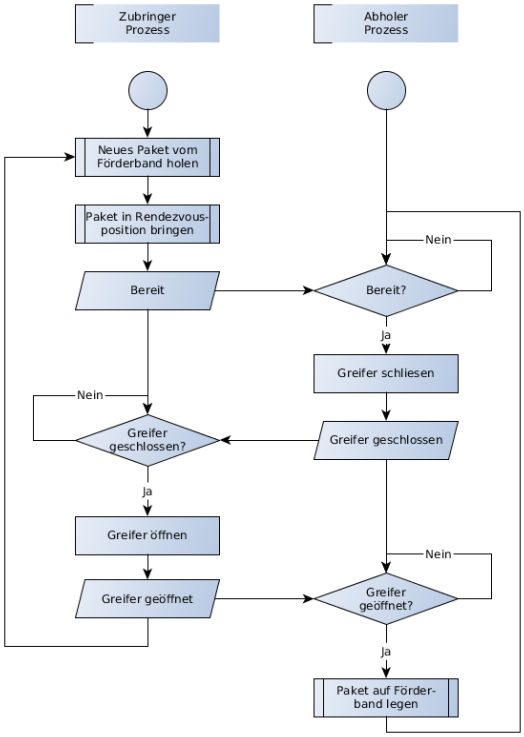


Abbildung 1.1: *Test Case 3*: Ablaufdiagramm vom

5. Die Sequenz beginnt wieder von Anfang an

Herausforderungen

- Synchronisation von zwei parallellaufenden Sequenzen
- Kommunikation zwischen zwei Sequenzen

1.4.5 Test Case 4: Sequenz pausieren

System

- Roboterarm
- Ein Taster *Taster Pause*, der die Sequenz pausiert

Ablauf

- Der Roboterarm führt eine sich wiederholende Sequenz endlos aus
- Wird *Taster Pause* gedrückt, pausiert die Sequenz
- Wird *Taster Pause* erneut gedrückt, wird die Sequenz fortgeführt

Herausforderungen

- Jede Sequenz muss jederzeit pausiert werden können
- Der Taster gibt nur einen Impuls, nicht eine bleibende Pegeländerung an einem Eingang
- Timeouts müssen pausiert werden

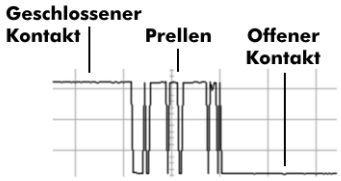


Abbildung 1.2: *Test Case 6*: Signalverlauf bei einem prellenden Schalter

1.4.6 Test Case 5: Zwei Roboterarme

System

- Roboterarm A
- Roboterarm B
- Ein Taster *Taster Pause*, der beide Sequenzen pausiert

Ablauf

- Beide Roboterarme führen sich wiederholende Sequenz endlos aus
- Wird *Taster Pause* gedrückt, pausieren beide Sequenzen

Herausforderungen

- Zwei parallellaufende Sequenzen müssen mit einem Taster pausiert werden können

1.4.7 Test Case 6: Prellender Taster

System

- Ein prellender Taster *Taster A*

Aufgabe

- Ein prellender Taster soll sauber eingelesen werden

Herausforderungen

- Der Taster muss entprellt werden
- Mehrmaliges drücken soll sauber registriert werden, auch wenn der Taster in schneller Abfolge gedrückt wird

1.4.8 Test Case 7: Menü

System

- Ein beliebige Anzahl Taster *Taster A*, *Taster B*, *Taster C*, ...

Aufgabe

- Wenn sich das System in einem *Idle* Zustand befindet, soll jeder Taster eine andere Sequenz starten

Herausforderungen

- Dieses *Menü* lässt sich nicht im klassischen Sinne als eine Sequenz, also eine Abfolge von Schritten, beschrieben werden. Trotzdem soll eine einfache Implementierung im *Sequencer* möglich sein

1.4.9 Test Case 8: Detailliertes Rendezvous

Dieser *Test Case* basiert auf auf dem *Test Case 3*, ist aber detaillierter ausgearbeitet. Im Anhang A ist das Ablaufdiagramm der Hauptsequenz *Sequenz Rendezvous* und der blockierenden Subsequenz *Sequenz PickUp* angehängt. Das Ziel von diesem *Test Case* war es nicht einen realistischen Fall abzubilden. Viel mehr dient er dazu, möglichst viele Fälle und Situationen abzubilden, die in einer Sequenz vorkommen können.

Das Ablaufdiagramm besteht aus folgenden Blöcken:

- Rechtecke: einzelne Steps
- Rechtecke mit doppelten, vertikalen Linien: Blockierende Subsequenzen
- Blaue Rauten: Entscheidungen
- Grüne Rauten: Blockierungen, bis Entscheidung wahr wird
- Parallelogramm: Statusvariable wird geändert
- Lange senkrechte Rechtecke: Bedingung, welche über längere Zeit überwacht wird

Quellenverzeichnis

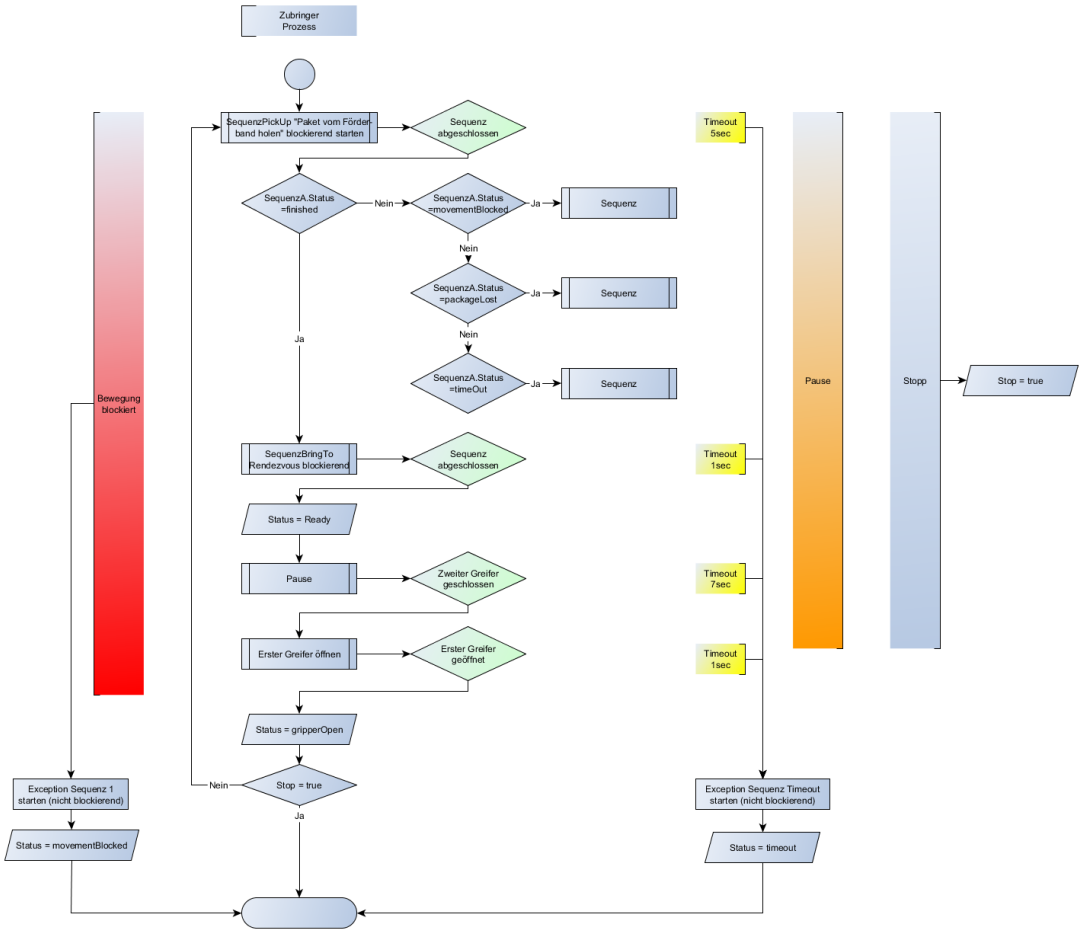
[EER-17] *Homepage: EEROS*
http://eeros.org
Stand vom 27.01.2017

[EEW-17] *Homepage: EEROS Wiki*
http://wiki.eeros.org
Stand vom 27.01.2017

Anhang

A Test Case 8

A.1 Sequenz Rendezvous



A.2 Sequenz PickUp