Inhaltsverzeichnis

1 Einleitung

1.1 Vorwort

1.2 EEROS

EEROS (Easy, Elegant, Reliable, Open and Safe) ist ein open source Software Framework, welches an der NTB entwickelt wurde und auch immer noch weiter Entwicklung wird. Das Ziel von EEROS ist, möglichst einfach, zuverlässig und einfach in der Bedienung zu sein. Da das Framework besonders auch in industriellen Robotern zum Einsatz kommen soll, ist besonders auch die Zuverlässigkeit der Software ein wichtiger Punkt. Für die Software wird die objektorientierte Programmiersprache C++ verwendet.

EEROS kann in vier Hauptbereiche unterteilt werden.

1. Die HAL (Hardware Abstraction Layer) welche als Schnittstelle zur Hardware dient.
2. Das CS (Control System). Im CS wird die Regelung des Roboters aufgebaut. In diesem System wird aber nicht nur die Regelung gerechnet, sondern auch Aufgaben wie die Berechnung der Vorwärts- und inversen Kinematik werden hier erledigt.
3. Der Sequencer steuert den Ablauf des Roboters. Hier werden nicht nur Wegpunkte aufgelistet, sondern auch das allgemeine Verhalten definiert.
4. Im SS (Safety System) werden sicherheitsrelevante Parameter überwacht. Das SS arbeitet unabhängig vom CS und vom Sequencer. Es löst einen Not-Aus aus, wenn der Roboter ausserhalb der zulässigen Parameter operiert. Ein möglicher Grund für einen Not-Aus wäre zum Beispiel, wenn sich der Roboterarm in einem Sicherheitsbereich zu schnell bewegt.

1.3 Klarstellung der Benennungen

Mit den meisten Programmiersprachen werden in englisch codiert. Auch die offizielle Onlinedokumentation[1] von EEROS, und die Benennung von Komponenten und Konzepten, ist in Englisch. In diesem Dokument wird an vielen Stellen bewusst darauf verzichtet, englische Bezeichnungen auf Deutsch zu übersetzen. Dies kann zu Deutsch - Englischen Mischwörter führen. Solche Mischwörter sind zwar nicht elegant, können aber besser für die Verständlichkeit sein und werden deshalb mit Absicht verwendet. Auch einige Eigennamen, wie z.B. *Sequencer* anstelle von *Sequenzer* werden in diesem Dokument nicht auf Deutsch übersetzt.

In dieser Arbeit wird oft von drei verschiedenen Kategorien von Entwicklern gesprochen. Es wird zwischen EEROS-, Steuerungs-, und Applikationsentwickler unterschieden.

Der **EEROS-Entwickler** hat vertiefte Kenntnisse der Programmiersprache C++ und vom EEROS Framework. Seine Hauptaufgabe ist die Weiterentwicklung des Frameworks, welches vom Steuerungsentwickler verwendet wird.

Der **Steuerungsentwickler** hat ebenfalls gute C++ Kompetenzen und nutzt das Framework, um eine Steuerung für einen Roboter zu entwickeln. Dafür muss er seine Software speziell auf den Roboter anpassen. Er bereitet auch erste Sequenzen für den Applikationsentwickler vor. Oft wird die Entwicklung der Steuerung und der Applikation von der selben Person übernommen.

Um den Ablauf des Roboters anzupassen, kann der **Applikationsentwickler** bestehende Sequenzen einfach anpassen. Dazu werden nur grundlegende Programmierfähigkeiten benötigt. Mit etwas erweiterten Kenntnissen kann er auch neue Sequenzen erstellen.

[1] http://eeros.org/wordpress/

1.4 Aufgabenstellung

In der aktuellen Version von EEROS existiert bereits eine erste Version von einem Sequencer. Dieser ist in seiner Funktionalität und Übersichtlichkeit aber stark eingeschränkt. Oft musste auf Tricks zurück gegriffen werden, damit bestimmte Aufgaben mit dem bestehenden Sequencer gelöst werden konnten. Um eine bestehende Sequenz anpassen zu können, auch wenn der Ablauf nur geringfügig geändert werden soll, ist schon viel Fachwissen notwendig.

In dieser Vertiefungsarbeit sollen diese beide Probleme gelöst werden. Es soll ein neuer Sequencer entwickelt werden, der flexibel für verschiedenste Arten von Robotern eingesetzt werden kann. Die Sequenzen, welche den Ablauf des Roboters beschreiben, sollen dabei möglichst einfach und übersichtlich aufgebaut sein. Dank dem einfachen Aufbau soll es auch für einen Applikationsentwickler möglich sein, Sequenzen anzupassen und zu erstellen, auch wenn dieser Entwickler keine vertiefte Kenntnisse von C++ besitzt.

2 EEROS aktueller Stand

2.1 EEROS generell

2.2 Aktuelle Implementierung des Sequencers

Für EEROS besteht bereist ein rudimentärer *Sequencer*. Im folgenden Kapitel wird die bestehende Implementierung kurz erklärt. In der Onlinedokumentation[1] wird noch vertiefter in die Details des bestehenden *Sequencers* eingegangen, als in dieser Arbeit.

2.2.1 Sequencer

Die Grundlage bildet ein Sequencer-Objekt, das in einem Nicht-Realtime-Thread läuft. In so einem *Sequencer* können eine Serie von blockierenden *Sequences* aufgerufen werden. Wenn mehrere parallele *Sequences* aufgerufen werden sollen, muss für jede *Sequence* ein eigener *Sequencer* erstellt werden.

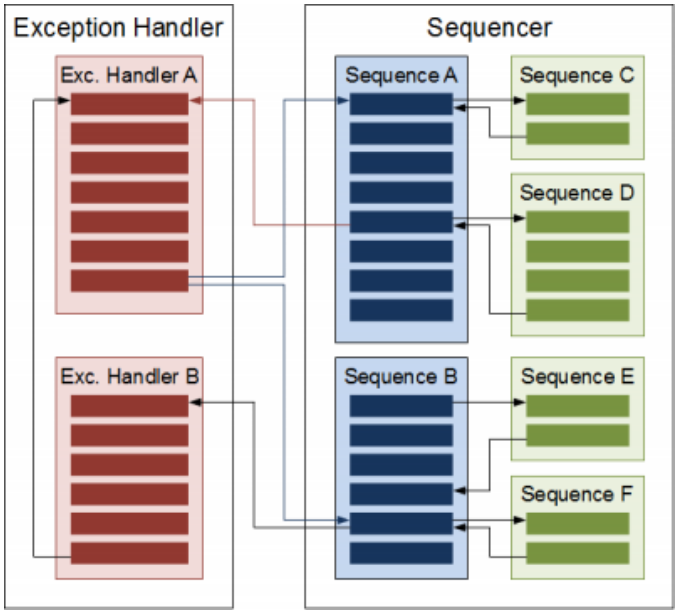

Abbildung 2.1: Schematische Darstellung des bestehenden Sequencers [2]

2.2.2 Sequence

Eine *Sequence* führt als erstes eine benutzerdefinierte Initialisierungsfunktion aus. Als nächstes wird eine *preCondition* überprüft. Fehlt die Überprüfung fehl, wird die *Sequence* abgebrochen. Bei einem positiven Ergebnis wird der Hauptteil, eine Abfolge von definierten *Steps* ausgeführt. Dem letzten *Step* folgt noch eine Überprüfung einer *postCondition*. Läuft der *Sequencer* im *stepping-mode*, wird die *Sequence* bei jeden *yield()* pausiert und wird erst fortgeführt, wenn der Befehl dazu gegeben wird.[3]

```
1  init();
2  yield();
3
4  if(!checkPreCondition())
5    return SequenceResult<void>(result::preConditionFailure);
6  yield();
7
8  run();
```

[1] http://wiki.eeros.org/eeros_architecture/sequencer/start
[3] http://wiki.eeros.org/eeros_architecture/sequencer/start

```
9  yield();
10
11 if(!checkPostCondition())
12   return SequenceResult<void>(result::postConditionFailure);
13 yield();
14
15 exit();
16 return SequenceResult<void>(result::success);
```

2.2.3 Step

Ein *Step* ist eine vom *Steuerungsentwickler* festgelegte Einheit, die von einem *yield()* Befehl unterteilt wird. *Steps* können im *stepping-mode* einzeln abgearbeitet werden.

2.2.4 Sub-Sequence

Subsequences sind sehr ähnlich wie normale *Sequenzen*. Sie können verwendet werden, wenn ein ähnlicher Ablauf mehrmals wiederholt werden soll. Durch eine Übergabe von Parameter an eine solche *Subsequence* kann sie sehr flexibel gestaltet werden. Wenn die *Subsequence* parallel, also nicht-blockierend, aufgerufen werden soll, muss sie einem neuen Sequencer übergeben werden.

```
1  class SequenceB : public Sequence<> {
2  public:
3    SequenceB(std::string name, Sequencer* seq, Robot& r) : Sequence<void,
         double>(name, seq), robot(r){ }
4
5    void run() {
6      robot.moveZ(5);
7      sleep(3);
8      yield();
9      robot.moveZ(0);
10   }
11
12 private:
13   Robot& robot;
14 };
```

2.2.5 Error-Handler

Der *Error-Handler* wird in der Onlinedokumentation zwar erwähnt, im Sourcecode von EEROS sind aber keine Spuren von der Implementierung zu finden. Der Dokumentation zu folge soll er *Exceptions* behandeln. Je nach *Exception* soll er flexibel reagieren, um Probleme zu beheben.

2.3 Fallbeispiel 'EEDURO Delta Roboter'

Der *EEDURO Delta Roboter* ist ein Roboter, dessen Hardware und auch Software an der NTB entwickelt wurde. Die Steuerung ist mit EEROS und dem bestehenden *Sequencer* aufgebaut. Im folgenden Kapitel wird der Quellcode der Steuerung analyisiert um herauszufinden, welche Aspekte des bestehenden *Sequencers* brauchbar sind, und wo er noch Lücken aufweist.

Der Quellcode der Steuerungssoftware ist auf folgendem Git-Repository zu finden:
https://github.com/ClaudiaVisentin/eeduro-platform

Die Software wurde vom Stand 10.10.2016 mit dem Hash *a25bcfa752516723f067f5a5166e8f09e60fc6e8* verwendet.

2.3.1 asdf

3 Anforderungen an den Sequencer

3.1 Nicht Teil des Sequencer

4 Aufbau des Sequencers

4.1 Caller Stack

Das unterste (älteste) Element ist die ID-Nummer der Hauptsequenz. Als nächstes folgt

5 Test des Sequencer

5.1 section

6 Fazit

6.1 section

Anhang