

# INTO THE ROS

## ADVANCED ROS NETWORK INTROSPECTION

Praxis der Softwareentwicklung  
Sommersemester 2014

### P f l i c h t e n h e f t



Auftraggeber

KIT - Karlsruher Institut für Technologie  
Fakultät für Informatik  
Institut für Anthropomatik und Robotik (IAR)  
Intelligente Prozessautomation und Robotik (IPR)

Betreuer: Andreas Bihlmaier  
andreas.bihlmaier@gmx.net

Auftragnehmer

Name	E-Mail-Adresse
Alex Weber	alex.weber3@gmx.net
Matthias Hadlich	matthiashadlich@yahoo.de
Matthias Klatte	matthias.klatte@go4more.de
Micha Wetzel	micha.wetzel@student.kit.edu
Sebastian Kneipp	sebastian.kneipp@gmx.net

Karlsruhe, 27.05.2014

# Inhaltsverzeichnis

<b>1</b>	<b>Zielbestimmung</b>	<b>4</b>
1.1	Musskriterien . . . . .	4
1.2	Wunschkriterien . . . . .	4
1.3	Abgrenzungskriterien . . . . .	5
<b>2</b>	<b>Produkteinsatz</b>	<b>6</b>
2.1	Anwendungsbereiche . . . . .	6
2.2	Zielgruppen . . . . .	6
2.3	Betriebsbedingungen . . . . .	6
<b>3</b>	<b>Produktumgebung</b>	<b>7</b>
3.1	Software . . . . .	7
3.1.1	Implementierung . . . . .	7
3.1.2	Dokumentation . . . . .	7
3.2	Hardware . . . . .	7
3.3	Lizenz . . . . .	7
<b>4</b>	<b>Funktionale Anforderungen</b>	<b>8</b>
4.1	Gesamtsystem . . . . .	8
4.1.1	Pflicht . . . . .	8
4.1.2	Optional . . . . .	8
4.2	Soll-Spezifikation . . . . .	9
4.2.1	Pflicht . . . . .	9
4.2.2	Optional . . . . .	9
4.3	API . . . . .	9
4.3.1	Pflicht . . . . .	9
4.3.2	Optional . . . . .	9
4.4	GUI . . . . .	10
4.4.1	Pflicht . . . . .	10
4.4.2	Optional . . . . .	10
<b>5</b>	<b>Nichtfunktionale Anforderungen</b>	<b>11</b>
5.1	Produktleistungen . . . . .	11

5.1.1	Pflicht . . . . .	11
5.1.2	Optional . . . . .	11
5.2	Qualitätsanforderungen . . . . .	11
5.2.1	Pflicht . . . . .	11
5.2.2	Optional . . . . .	12
<b>6</b>	<b>Produktdaten</b>	<b>13</b>
<b>7</b>	<b>Systemmodell</b>	<b>14</b>
7.1	Szenarien . . . . .	14
7.2	Anwendungsfälle . . . . .	15
<b>8</b>	<b>Benutzeroberfläche</b>	<b>16</b>
8.1	Skizzen bzw. Bilder . . . . .	17
8.1.1	name des bildes . . . . .	17
8.1.2	name des bildes . . . . .	18
<b>9</b>	<b>Qualitätsanforderungen</b>	<b>19</b>
9.1	Erweiterbarkeit . . . . .	19
9.2	Wartbarkeit . . . . .	19
9.3	Zuverlässigkeit . . . . .	19
<b>10</b>	<b>Testfälle</b>	<b>20</b>
<b>11</b>	<b>Glossar</b>	<b>21</b>
11.1	Allgemein . . . . .	21
11.2	OpenCV . . . . .	23

# 1 Zielbestimmung

Die zu entwickelnde Softwarelösung soll sich als Überwachungsinstrument in die Robot Operating System (ROS) Middleware einfügen. Dabei soll überwacht werden, ob jeder Prozess, der im ROS-Netzwerk enthalten ist, definierten Sollwerten entsprechend ordnungsgemäß Daten übermittelt.

Gerade durch die gegebene Problematik eines auf mehrere Hosts verteilten Systems, bereitet das Erfassen und Verstehen von auftretenden Problemen zum aktuellen Stand des ROS Schwierigkeiten. Das vorliegende Projekt erfasst den Ist-Zustand der Prozesse im ROS-Netzwerk ohne Leistungseinbußen kontinuierlich und visualisiert diesen im Vergleich zu definierten Soll-Werten. Auftretende Fehler und Abweichungen der von den Spezifikationen werden den Benutzer deutlich angezeigt. Es lassen sich Gegenmaßnahmen für verschiedene Fehler definieren.

## 1.1 Musskriterien

- Minimale Erweiterung bestehender Knoten zur Übermittlung von Metadaten
- Publizierung der Metadaten mit definierter Frequenz auf Topics
- Deaktivierung der Datenerfassung einzelner Knoten
- Definition von Soll-Zuständen
- Graphische Übersicht über Abweichungen von Soll- und Ist-Zustand der Knotendaten
- Statuserfassung der Host-Computer

## 1.2 Wunschkriterien

- Überwachung weiterer ROS Bestandteile: Service, Parameters
- Selbstkontrolle der Knoten
- Ist-Zustand als Soll-Zustand definieren
- Ergänzende Übersichten mit dem ROS Graphen und Plots über den zeitlichen Verlauf

## 1.3 Abgrenzungskriterien

- Das Projekt wird keine vollständige Automatisierung in Fehlerfällen umsetzen.
- Überwachung der Netzwerkinfrastruktur

## **2 Produkteinsatz**

### **2.1 Anwendungsbereiche**

Angewandt wird die Software zum Suchen von Fehlern und Überwachen kompletter ROS Netzwerke.

### **2.2 Zielgruppen**

Die Zielgruppe besteht aus Entwicklern und Administratoren von Umgebungen, in denen ROS eingesetzt wird. Es wird technisches Verständnis der Nutzer vorausgesetzt.

### **2.3 Betriebsbedingungen**

- Zur Fehlererkennung wird die Software gestartet, um eine genauere Ursache des Fehlers festzustellen.
- Während des Produktivbetriebs wird die Überwachung zugeschaltet, um Versagen in dem System zu erkennen und automatische Gegenmaßnahmen einzuleiten.

## 3 Produktumgebung

### 3.1 Software

#### 3.1.1 Implementierung

- GNU/Linux (Ubuntu 12.04)
- ROS Hydro oder neuer
- Python 2.7 oder neuer
- QT 5.0 oder neuer

#### 3.1.2 Dokumentation

- Latex
- Wiki im GitHub Repository

### 3.2 Hardware

Nach Möglichkeit alle Plattformen, die als Host für einen ROS-Knoten dienen können.

### 3.3 Lizenz

Da dies ein Opensource-Projekt ist, stehen jegliche Dokumentationen und der Quellcode unter freien Lizenzen.

## 4 Funktionale Anforderungen

### 4.1 Gesamtsystem

#### 4.1.1 Pflicht

**/FA0100/** Dezentrale Erfassung von Metadaten: Anzahl Publisher und Subscriber, Bandbreite, Frequenz, Latenz, Jitter

**/FA0200/** Monitoring Knoten zum zentralen Abgleich des Soll- und Ist-Zustandes

**/FA0210/** Warnungen und Fehlernachrichten bei maßgeblichen Abweichungen

**/FA0300/** Definition eines ROS Message Types für Metadaten

**/FA0400/** Eigenständiger Knoten zur Überwachung der Hardware des Host-Systems (CPU Auslastung, CPU Temperatur, RAM, Festplatten-Speicher)

#### 4.1.2 Optional

**/FA0500/** Überwachung weiterer ROS Komponenten wie Services und Parameters

**/FA0600/** Festlegen des Ist-Zustandes als Soll-Definition

**/FA0610/** Korrelationen zwischen Empfangs- und Sendeverhalten eines Knotens erkennen

**/FA0700/** Anpassung des Sendeverhaltens des Systems an Netzwerkgegebenheiten

**/FA0800/** Definition der Netzwerktopologie sowie Darstellung der Auslastung physischer Verbindungen

**/FA0900/** Integration mit roswtf



## 4.2 Soll-Spezifikation

### 4.2.1 Pflicht

**/FA1000/** Parametrisierung für Topics, Hosts und Knoten getrennt

**/FA1010/** Obere und untere Schranken für Metadaten definierbar

**/FA1100/** Laden der Soll-Spezifikation einmalig bei Start des Überwachungsknotens

**/FA1110/** Weitergabe der Soll-Spezifikation an den ROS Parameter Server bei Knotenstart

**/FA1200/** Möglichkeit nur Teile des Netzwerkes zu überwachen

**/FA1210/** Teilsysteme können sich überlappen

### 4.2.2 Optional

**/FA1300/** Ein Monitoringknoten verwaltet alle Teilsysteme

## 4.3 API

### 4.3.1 Pflicht

**/FA1400/** Die Metadaten werden durch Hinzufügen eines Funktionsaufrufes zu bestehenden Callbacks erfasst, alternativ durch eine geeignete Klasse, die von Subscriber erbt

**/FA1500/** Die Metadaten werden auf einem Topic mit definierter Frequenz publiziert

**/FA1600/** Die Metadatenerfassung lässt sich über Parameter deaktivieren

### 4.3.2 Optional

**/FA1700/** Knoten sind in der Lage, sich anhand ihrer Soll-Metadaten selber zu überwachen

**/FA1800/** Das System wird auch für C++ Knoten implementiert

## 4.4 GUI

### 4.4.1 Pflicht

**/FA1900/** Die grafische Benutzeroberfläche bietet eine Visualisierung des Soll-Ist-Vergleichs von sowohl Knoten als auch Hostsystemen

### 4.4.2 Optional

**/FA2000/** Der Soll-Ist-Vergleich wird in einer Visualisierung des ROS Graphen dargestellt

**/FA2100/** Geeignete Metadaten werden im ROS Graphen dargestellt

**/FA2200/** Es wird ein zeitlicher Verlauf der Metadaten aufgezeichnet und grafisch dargestellt

**/FA2300/** Knoten werden im Graphen nach ihrem Host gruppiert angezeigt

**/FA2400/** Knoten werden als Gruppen der unterschiedlichen Soll-Spezifikationen angezeigt

## 5 Nichtfunktionale Anforderungen

### 5.1 Produktleistungen

#### 5.1.1 Pflicht

**/NF0100/** Einfaches einbinden der Metadatenerfassung in bereits vorhandene Knoten

**/NF0200/** Geringen zusätzlichen Leistungsbedarf durch die Metadatenerfassung

**/NF0210/** Vernachlässigbarer Leistungsbedarf wenn die Erfassung deaktiviert ist

**/NF0300/** Stufenweise Selbstüberwachung des Gesamtsystems

**/NF0310/** Selbstüberwachung anhand Knoten und ihre Verbindungen

**/NF0320/** Selbstüberwachung durch die der Metadaten der Knoten

**/NF0330/** Selbstüberwachung unter Berücksichtigung von Hosts und Netzwerktopologie

#### 5.1.2 Optional

**/NF0400/** Flexibles GUI Layout

### 5.2 Qualitätsanforderungen

#### 5.2.1 Pflicht

**/NF0500/** Möglichkeit die Metadaten zu erweitern

**/NF0600/** Modularität der GUI

**/NF0700/** Dokumentationen in Englisch

**/NF0710/** Dokumentation der API

**/NF0720/** Dokumentation der GUI

**/NF0730/** Dokumentation des Python-Codes mittels docstrings

## **5.2.2 Optional**

**/NF0800/** Tutorial zur Nutzung des Programmes in Englisch

**/NF0900/** Weitgehende Abdeckung durch Tests

## 6 Produktdaten

beschreibung der inhalte der metadaten

angabe des config konzepts

## 7 Systemmodell

### 7.1 Szenarien

#### Darstellung von Fehlern

##### Fehlerfeststellung durch den Benutzer

- Einem Benutzer ist ein Fehler aufgefallen: Der Roboter verhält sich nicht mehr, wie erwartet wird.
- Er öffnet das PSE Introspection Plugin in der rqt GUI.
- Sofort sieht er, dass das Plugin den Fehler erfasst hat. Es zeigt ein Problem an.
- In einer Knotenübersicht fällt es dem Benutzer leicht, die farblich auffällig hinterlegten fehlerhaften Knoten auszumachen.
- Er öffnet die betroffenen Knoten in der Detailansicht und sieht, in welchen Punkten das Knotenverhalten von den Spezifikationen abweicht.

##### Fehlerfeststellung über die GUI

- Um den Status des ROS-Netzwerkes im Auge zu behalten, hat der benutzer das PSE Introspection Plugin in der rqt GUI geöffnet.
- Er stellt fest, dass die Statusanzeige nicht mehr in Grün volle Funktionalität signalisiert.
- Mit einem Klick öffnet er das Fehlerlog, in dem er Einträge findet, welche Knoten von ihren Spezifikationen abweichend arbeiten und die Fehlermeldung ausgelöst haben.
- Nun hat der Anwender die Möglichkeit, die Fehlerhaften Knoten aus einer Liste zu suchen und sie in einer Detailansicht zu öffnen, um sich weitere Details über die ausgefallenen Komponenten anzeigen zu lassen.

## Knotendetail-Ansicht

### Auswahl eines Knotens

- Der Benutzer möchte sich die Metadaten Informationen zu einen Knoten anzeigen.
- Er wählt sich den Knoten, der ihn interessiert aus einer Liste.
- Dazu besteht die Möglichkeit, diesen in einer Liste zu finden, welche sich auch über ein Suchfeld filtern lässt.
- Wählt er einen Knoten aus, öffnet sich ein kleines Fenster mit den Knoteninformationen und reiht sich in die Übersicht der geöffneten Detailfenster ein.

### Umschalten der Detailansicht am Beispiel der Topic Liste

- Das Detailfenster bietet für verschiedene Knotendaten Links, um die Ansicht zu erweitern.
- Der Benutzer bekommt ein Topic angezeigt, mit dem der Knoten in einem Publisher/Subscriber-Verhältnis steht.
- Klickt der Benutzer auf eine Schaltfläche neben der Anzeige, wird die Anzeige der sonstigen Metadaten durch eine vollständige Auflistung der relevanten Topics ersetzt.
- Intuitiv positioniert findet der Anwender eine Schaltfläche, um zur vorigen Ansicht zurückzukehren.

## Dynamische GUI

- Der Benutzer schließt ein rqt Plugin, das er nicht mehr benötigt.
- Daraufhin verschiebt sich das Layout in der rqt GUI und das PSE Introspection Plugin ändert seine Form und Größe.
- Das PSE Introspection Plugin rearrangiert seine Inhalte, um sie im neuen Layout anzuzeigen.

## 7.2 Anwendungsfälle

## 8 Benutzeroberfläche

hier kommen dann ein paar bilder und dazugehörige beschreibungenen unserer gui

- blasdgs
- sdfg
- was man halt alles so allgemeines reinschreiben kann



## 8.1 Skizzen bzw. Bilder

### 8.1.1 name des bildes



Abbildung 8.1: bildunterschrift

Ein ganz ganz langer text zum bild  
kdfnakgflkadshfkjsfkljsadfköadshfölkdf  
hsgkjöldhflkghsfdlkghödklfhglkdsjglödshgkdsfhögh  
sdfkengkjhdskföghadfsopfghaoifhaigtfuiaerhgiuag

### 8.1.2 name des bildes



Abbildung 8.2: bildunterschrift

Ein ganz ganz langer text zum bild  
kdfnakgflkadshfkjsfkljsadfjköadshfölkdf  
hsgkjöldhflkghsfdlkghödklfhglkdsjglödshgkdsfhögh  
sdflkgkjhdsköghadfsopfghaoifhaigtfuiaerhgiuag

## 9 Qualitätsanforderungen

Wartbarkeit, Zuverlässigkeit, Erweiterbarkeit etc. anpreisen

### 9.1 Erweiterbarkeit

Um in einer erweiternden Entwicklung den Umfang der übertragenen Metadaten zu erweitern, muss zum einen der Topic-Datentyp erweitert werden, mit dem die Daten zum Monitoringknoten übermittelt werden, und zum anderen muss die die Funktion zur Erfassung der Daten ergänzt werden.

### 9.2 Wartbarkeit

Aufgrund einer transparenten Abfolge in der Übertragung der Metadaten, können die einzelnen Stationen stückweise auf Fehler untersucht werden.

Da das Projekt mit dem ROS Publisher/Subscriber Prinzip die Metadaten auf Topics überträgt, ist es zudem ohne weiteres möglich, den Inhalt der übertragenen Daten mit bestehenden Mitteln einzusehen und zu kontrollieren.

### 9.3 Zuverlässigkeit

Die Funktionalität des ROS Netzwerkes wird durch unerheblichen Zusatzaufwand zur Verarbeitung der erhobenen Metadaten nicht weiter eingeschränkt.

## 10 Testfälle

**/TF0100/** Es wird festgestellt, dass ein Knoten keine Daten mehr sendet

**/TF0200/** Es wird festgestellt, wenn ein Knoten langsamer oder schneller als sonst Daten sendet.  
Es wird festgestellt, wenn ein Knoten maßgeblich von den Sollwerten abweicht.

**/TF0300/** Eingelesene Konfigurationsdaten lassen sich parsen und auslesen

**/TF0400/** Hostdaten werden übermittelt

**/TF0500/** rqt Plugin reagiert auf Größenänderung

## 11 Glossar

des ist jetzt mal der glossar von letztem jahr den können wir ja ganz schnell bearbeiten und unsere sachen einfügen

### 11.1 Allgemein

**API** Application Programming Interface. Eine Schnittstelle (s. Interface), über welche andere Programme auf der Quelltextebene auf dahinter verborgene Funktionalität zugreifen können

**Augmented Reality** „Erweiterung der Realität“ durch einen Computer, etwa bei der Einblendung von Informationen in ein Bild der Umgebung, das auf einem Smartphone angezeigt wird.

**Bug** Fehlerhaftes Verhalten eines Programmes.

**Binärform** Hier das Programm in für den Computer direkt verwendbarer Form im Gegensatz zum Quellcode. Anders als dort ist hier kaum sichtbar, wie das Programm genau arbeitet, weshalb bei OpenSource-Projekten gerade der Quelltext offen liegt (vgl. OpenSource).

**Datenstrom** Daten, die beispielsweise während der Ausführung eines Programms fließen, wobei das Ende dieses Flusses nicht absehbar ist.

**Debug-Modus** Modus, bei dem zusätzliche Informationen angezeigt werden, um dem Programmierer das Auffinden und Beheben von Bugs (kurz *Debugging* oder *Debuggen*), hier insbesondere Programmierfehlern, zu erleichtern. Vgl. Release-Modus.

**Debug-Visualisierung** Hier eine Visualisierung, die den Benutzer beim Debuggen unterstützt, indem sie relevante Daten zu den übergebenen Bildern anzeigt und diese damit leicht verständlich darstellt.

**FAQ** Engl. für „Häufig gestellte Fragen“ enthält sie viele Fragen, die besonders neue Benutzer sich stellen, wenn sie anfangen ein Projekt zu nutzen.

**Falschfarben** Verwendung von Farben in einem Bild, die sich von den natürlichen, erwarteten Farben stark unterscheiden, um etwa Details hervorzuheben.

**Filter** In der Bildverarbeitung die Veränderung eines Bildbereiches mithilfe eines bestimmten Algorithmus.

**GNU/Linux** Das GNU-Betriebssystem in Kombination mit einem Linux-Kern

**GNU-Projekt** Das Projekt zur Erstellung von GNU-Betriebssystem und Software.

**GUI** Graphical User Interface, zu deutsch Graphische Benutzeroberfläche. Stellt Funktionen graphisch dar, sodass der Benutzer beispielsweise per Mausklick damit interagieren kann; im Gegensatz zu textbasierten Benutzerschnittstellen (vgl. Interface).

**Kompilieren** Umwandlung eines Quellcodes in eine für den Computer verständliche Form, mithilfe eines Kompilers genannten Programms.

**Matches** Durch OpenCV erzeugte Verknüpfungen zwischen zwei Bildbereichen bzw. Bildpixeln, welche vom Benutzer an die API übergeben werden.

**Mouse over** Information über das Element einer GUI, auf dem der Mauszeiger ruht, wird angezeigt.

**Parallelrechner** Rechner, in dem mehrere Threads gleichzeitig nebeneinander ausgeführt werden können (vgl. Thread).

**OpenCV Test Framework** Stellt Funktionen zum Testen zur Verfügung, etwa Überprüfungen, ob zwei Matrizen gleich sind.

**Open Source** Software, bei welcher der Quellcode frei zugänglich gemacht wird. Dies erlaubt unter anderem die Weiterverwendung und -entwicklung durch andere.

**Overhead** Zusätzlicher Speicher- oder Zeitaufwand.

**proprietär** In diesem Zusammenhang Software, die nicht unter einer freien Lizenz steht.

**Release-Modus** Veröffentlichungs-Modus, in dem keine zusätzlichen Debug-Informationen angezeigt werden, also der Modus, in dem das fertige Programm läuft.

**Resource-Leak** Das Auftreten der Situation, dass Ressourcen irgendeiner Art (Speicher, Dateien, usw.) zwar alloziert werden, aber nach Verwendung nicht mehr an das System zurückgegeben werden.

**Rohdaten** Daten, welche direkt und ohne wirkliche Aufarbeitung, aus den vom Entwickler beim API-Aufruf übergebenen Datenstrukturen stammen.

**Speicherleck** *engl. memory leak* Vgl. Resource-Leak; die Ressource ist in diesem Fall Speicher.

**Stand-Alone-Programm** Programm, das für sich alleine funktioniert.

**Streaming** Hier das Weiterlaufen des Datenstroms.

**Tab** Hier ein registerkartenähnlicher Teil einer GUI.

**Thread** Ausführungsstrang in einem Programm. Der Begriff wird insbesondere im Zusammenhang mit Mehrfachkernsystemen, wo mehrere Threads gleichzeitig nebeneinander laufen können, oft verwendet.

**thread-lokal** Auf einen Thread beschränkt (vgl. Thread).

**Threadssafety** Es ist sichergestellt, dass mehrere Threads sich nicht gegenseitig stören, etwa beim Speichern von Daten.

**Translation Unit** Eine Einheit, die einzeln im Ganzen kompiliert wird (s. Kompilieren); ein Projekt teilt sich meist in mehrere solcher Einheiten auf.

**Undefiniertes Verhalten** Befehle, deren Verwendung dazu führt, dass der C++-Standard *keinerlei* Verhaltensgarantien irgendeiner Art für das gesamte Programm mehr gibt. Etwas Umgangssprachlich: Der Standard untersagt die Verwendung.

**View** Zusammengehörige Visualisierungen eines bestimmten OpenCV-Features (oder einer Featureart).

## 11.2 OpenCV

Weiterführende Informationen sind auf [docs.opencv.org](https://docs.opencv.org) zu finden.

**adaptiveThreshold** OpenCV-Methode, die mittels eines adaptiven Threshold (s. unten) Graustufenbilder in (u.U. invertierte) Binärbilder umwandeln kann.

**calcHist** Berechnet ein Histogramm.

**Canny** Kantenerkennung (mithilfe des Canny86-Algorithmus).

**Dilatation** Berechnung des Bereiches in einem Bild, der von einer Maske abgedeckt wird, wenn sich deren Bezugspunkt (oft der Mittelpunkt) durch den ganzen zu untersuchenden Bildbereich bewegt.

**DMatch** Klasse für das Matching (vgl. Matches).

**Erosion** Prüft, ob eine Maske, etwa eine geometrische Figur, vollständig in ein Bild bzw. einen Bildbereich passt und gibt u.U. ein Bild zurück, in dem nur die überdeckten Teile erhalten sind. Bildet zusammen mit der Dilatation zwei der grundlegenden Bildverarbeitungsmethoden.

**floodFill** Bei dieser Methode wird ein zusammenhängender Bereich des Bildes mit der übergebenen Farbe ausgefüllt.

**HoughCircles** Findet Kreise in einem Graustufenbild (unter Benutzung der Hough-Transformation).

**KeyPoint** Klasse, die Daten eines Punktes (Koordinaten, Nachbarschaftsgröße etc.) enthält.

**morphologyEx** diese Methode von OpenCV erlaubt fortgeschrittene morphologische Transformationen unter Benutzung und mehrfacher Anwendung von Dilatation und Erosion.

**ocl** Das OCL-Modul stellt Implementierungen von OpenCV-Funktionalität für Geräte, welche die Open Computing Language (kurz OpenCL), ein Interface über Parallelrechner, benutzen, zur Verfügung.

**Sobel-Operator** Ein Kantenerkennungs-Filter.

**Stitching** Zusammenfügen mehrerer Bilder mit zusammenhängenden Bereichen an den Rändern zu einem großen Bild, etwa von Einzelfotografien zu einem Panorama.

**threshold** Diese Methode eröffnet verschiedene Möglichkeiten, die Elemente eines Arrays auf ein bestimmtes Niveau zu trimmen, auf binäre Werte herunterzubrechen und Ähnliches.