

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Abstrakte Interpretation & Verifikation

Florian Schmaus, Simon Schuster

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
<https://www4.cs.fau.de>

25. Juni 2018





- TAns werden in der Übung verteilt
- $\geq 60\%$ Rückläuferquote $\leadsto$ Belohnung: **Semesterabschlussgrillen**
- Termin der Evaluation **30.06.2018**



1 Abstrakte Interpretation mit Astrée

2 Formale Verifikation

3 Aufgabenstellung



1 Abstrakte Interpretation mit Astrée

2 Formale Verifikation

3 Aufgabenstellung



- Ziel: Nachweis der Abwesenheit von Laufzeitfehlern
- Findet *alle* potentiellen Laufzeitfehler
- Leider auch *falsch-positive*
 - ↪ wegen **Gödelschem Unvollständigkeitstheorem** (1931)
„Jedes hinreichend mächtige formale System ist entweder widersprüchlich oder unvollständig.“

Programm ist korrekt, wenn

- Astrée keine Alarme meldet
- Oder für alle Alarme nachgewiesen, dass falsch-positiv



- Überschreitung von Array-Grenzen
- Ganzzahldivision durch Null
- Ungültige Dereferenzierung, arithmetische Überläufe
- Ungültige Gleitkommaoperationen
- Unerreichbarer Code
- Lesezugriff auf nicht initialisierte Variablen
- Verletzung benutzerdefinierter Zusicherungen
 $\leadsto$ `assert()`



- Rekursion prinzipiell erlaubt, wird aber nicht analysiert
 - ↪ für Rekursionsergebnis werden keine Einschränkungen ermittelt
- Auswertungsreihenfolge in C nicht vollständig spezifiziert
 - ↪ *eine* bestimmte Ordnung wird angenommen
 - stimmt nicht notwendigerweise mit Compiler überein
 - optionale Warnung durch Astrée
- Funktionen der Standard-C-Bibliothek werden nicht erkannt
 - ↪ mitgelieferte Stubs nutzen
- Dynamischer Speicher nicht erlaubt
 - ↪ kein `malloc()`
 - keine Einschränkung im sicherheitskritischen Echtzeitbereich



Astrée nimmt an, dass folgende semantische Regeln gelten:

1. Der C99-Standard
2. Implementierungsabhängiges Verhalten
 - Größe von Datentypen
 - Gleitkommastandard
 - ...
3. Benutzerdefinierte Einschränkungen
 - z. B. ob statische Variablen mit 0 initialisiert werden
4. Außerdem *benutzerspezifizierte Zusicherungen*
~> und da wird es interessant ☺



Benutzerdefinierte Zusicherungen

`__ASTREE_known_fact((B))`

- Analyzer nimmt an, dass B gegeben ist
- Analyzer warnt, falls B *nie wahr werden kann*
- `__ASTREE_known_range((V, [a; b]))` $\leadsto$ Wertebereich

`assert((B))/__ASTREE_assert((B))`

- Analyzer erzeugt Alarm, falls B *nicht immer wahr ist*
- B kann nicht von der Form `e1 ? e2 : e3` sein
- `__ASTREE_global_assert(( ))` $\leadsto$ gesamtes Programm

- B muss seiteneffektfrei sein
- *Doppelte Klammerung ist wichtig!*



```
1 #include <astree.h> // Astree-Makros ggf. abschalten
2
3 float filter(Alpha_State *s, float val) {
4     __ASTREE_known_fact((val == val)); // known_fact(!isnan(val))
5     __ASTREE_known_fact((-10.0f < val && val < 10.0f));
6     __ASTREE_known_fact((s->val == s->val));
7     __ASTREE_known_fact((FLT_MIN < s->val
8         && s->val < FLT_MAX));
9     __ASTREE_assert((0.0f < s->alpha));
10    __ASTREE_assert((s->alpha < 1.0f));
11
12    float residual = val - s->val;
13    s->val = s->val + s->alpha * residual;
14
15    __ASTREE_assert((s->val == s->val));
16    // ...
17    return s->val;
18 }
19
```



```
__ASTREE_modify((V1, ..., Vn))
```

- Zeigt an, dass Variablen V1 bis Vn unbekannten Wert haben

~> Braucht man um Stubs zu bauen

- Beispiel: Emulation von Sensoren

Beispiel

```
1 #ifdef __ASTREE__
2 __ASTREE_modify((x; full_range));
3 __ASTREE_known_fact((x >= 0));
4 #else
5 // ... Implementierung
6 #endif
7
```



Schleifen ausrollen

- Astrée versucht Aufwand zu vermeiden
- Schleifen/Verzweigungen werden nicht ausgerollt
- Konsequenz:

Beispiel

```
1 unsigned int i = 0;  
2 unsigned int j = 20;  
3 while (j > 0) { --j; ++i; }  
4
```

- Astrée kann nicht zeigen, daß die Schleife terminiert (☞ Satz von Rice, 1953)
- Annahme für weitere Analyse: i läuft über

Lösung:

```
1 __ASTREE_unroll((30))  
2 while (j > 0) { --j; ++i; }  
3
```



- Dito bei Verzweigungen
- Astrée betrachtet normalerweise nur den schlimmsten Zweig
- ~> Pessimistisches Ergebnis
- Lösung: Analyse vorübergehend aufspalten:

Verzweigungsanalyse

```
1 __ASTREE_partition_control
2 if (...) { ... }
3 else { ... }
4 ...
5 __ASTREE_partition_merge(());
6
```

- Auch für Schleifen, `switch` und Funktionsaufrufe
- ~> Blick ins Handbuch: es gibt noch weitere Tricks



Synchrone Programme

- Viele Echtzeitsysteme endlosschleifenbasiert ☹

→ Astrée modelliert dies

```
__ASTREE_max_clock((N))
```

beschränkt Schleifendurchläufe

```
__ASTREE_wait_for_clock(() )
```

wartet auf nächsten Tick

```
1 __ASTREE_max_clock((10)); // sonst evt. keine Schleifeninvariante
2 void main(void) {
3     while (1) {
4         // 1. 'volatile'-Werte lesen
5         // 2. Zustand und Ausgabe berechnen
6         // 3. Ausgabe schreiben
7         __ASTREE_wait_for_clock(() ); // auf naechsten Clock-Tick warten
8     }
9 }
10
```



- Astrée modelliert auch asynchrone Ausführung von Aufgaben
- Keine Annahmen über Scheduler oder Prioritäten
- automatic-shared-variables muss auf yes stehen

```
1  int x, y;
2  volatile int s;
3
4  void t1(void) {
5      x = 1; s = 1; x = 0;
6  }
7
8  void t2(void) {
9      if (s > 0) {
10         y = -1;
11     } else {
12         y = 1;
13     }
14 }
15
16 void main(void) {
17     x = y; // synchroner Teil
18     __ASTREE_asynchronous_loop((t1(), t2()));
19 }
20
```



`__ASTREE_volatile_input(V)`

- Zeigt an, dass V sich jederzeit ändern kann

↪ Modelliert Eingabe von außen

`__ASTREE_volatile_input((Vp, r))`

- p ist Pfad in der Variablen,
z. B. `V.a[3-4].b` ↪ Variable V, Arrayelemente `a[3]` und `a[4]`,
Struct-Element b
 - `[i]` ↪ Element i
 - `[i-j]` ↪ Elemente i bis j
 - `[]` ↪ alle Elemente
- r schränkt Wertebereich ein `[i, j]` ↪ von i bis j




```
__ASTREE_analysis_log()
```

- Gibt Zustand der Analyse an dieser Stelle aus

```
__ASTREE_log_vars((V1, ..., Vn))
```

- Zeigt Zustand der Analyse in Bezug auf einzelne Variablen an

```
__ASTREE_print("text")
```

- Gibt Text aus



- Astrée im CIP:

% /proj/i4ezs/tools/astree_c/bin/astreec

- Anmeldung mit Benutzername und Passwort

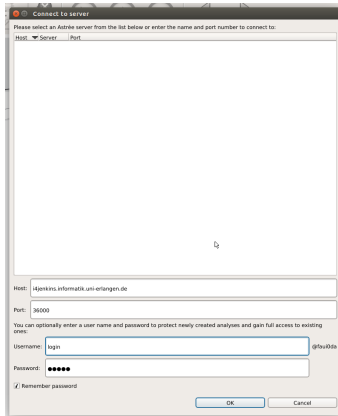
→ Passwort wird bei der ersten Anmeldung festgelegt

- Dokumentation

- HTML: /proj/i4ezs/tools/astree_c/share/astree_c/share/html/index.html

- PDF: /proj/i4ezs/tools/astree_c/help/astreec_QuickStart.pdf





Host i4jenkins.cs.fau.de

Port 36000

Kein Benutzername/Passwort notwendig.



Projekt anlegen

General
Specify the project name, the entry point of the analysis and the sources that you want to analyze.

Project name:

Description:

Analysis start:

Wrapper and stubs file:

Files to be used for the analysis

☐ Add already preprocessed files

☒ Add the source files and preprocess them with the built-in preprocessor

Number of files: 0

■ Quelldateien zum Projekt hinzufügen



Analysen konfigurieren I

Analyzer assumptions
Set the analysis options that configure the semantics used by the analyzer.

Application Binary Interface (ABI)
Analyzer default

Initial value for global and static

- ☒ Initialize static variables to 0
- ☒ Initialize global variables to 0

Allow the C volatile keyword to define

- ☒ [Volatile global variables](#)
- ☒ [Volatile local variables](#)

[Help browser](#)

How to get help

Position the mouse pointer over an Astrée option on the left side of this window to access the online documentation.

< Back Next > Finish Cancel

■ *Volatile global variables*

■ *Volatile local variables*



Coding rules

Configure checks of MISRA-C rules and coding guidelines



Rules Set

- ☐ CERT. SEI CERT C Coding Standard
- ☐ ISO. ISO/IEC TS 17961:2013 C secure coding rules
- ☐ M. MISRA-C:2004 Rules
- ☐ M2012. MISRA-C:2012 Rules
- ☐ M2012A1. MISRA-C:2012 Amendment 1
- ☐ S. Style Rules
- ☐ T. Metrics Rules

- ☐ Skip the run-time error analysis

< Back

Next >

Finish

Cancel



Analysen konfigurieren III

Analyzer verbosity

Activate or suppress some alarm types detected by the analyzer.



Raise alarms on integer overflows in

- ☒ Arithmetic operations excluding left shifts
- ☒ [Left shifts](#)
- ☒ [Explicit casts](#)
- ☒ Implicit casts

[Help browser](#)

How to get help

Position the mouse pointer over an Astrée option on the left side of this window to access the online documentation.

< Back

Next >

Finish

Cancel

- *Left shift*
- *Explicit casts*



Performance and precision

Choose the domains to be used and the program points to be observed.



Precision



Domains: all

Observability



Enable tooltips

Analysis complexity



Indicates the relative computation time and memory consumption of the analysis with the chosen initial precision and observability settings.

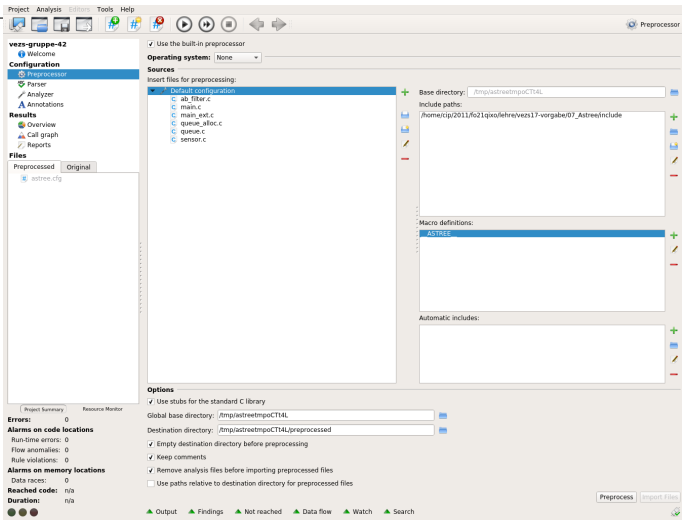
[Help browser](#)

How to get help

Position the mouse pointer over an Astrée option on the left side of this window to access the online documentation.

< Back Next > Finish Cancel



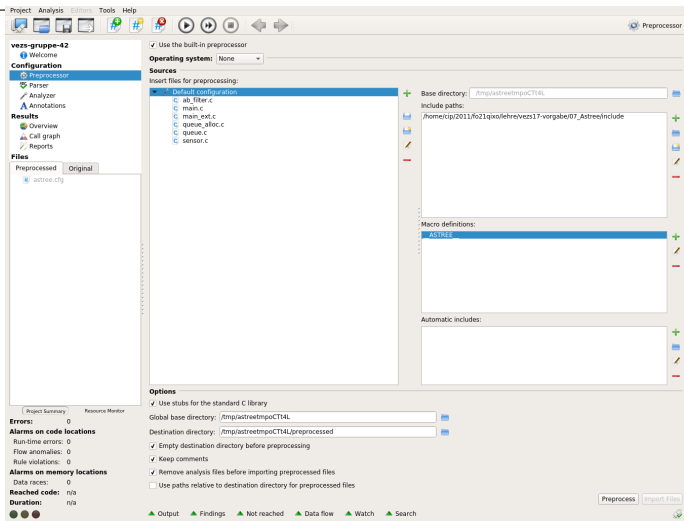


Libc soll nicht mit analysiert werden:

Use stubs for the standard C library

☞ Stubs für libc verwenden

Präprozessoreinstellungen



■ Include-Pfade festlegen

■ __ASTREE__ definieren

ABI festlegen

The screenshot shows the Astrée tool's configuration window for the ABI (Application Binary Interface). The window is titled 'veez-gruppe-42' and has tabs for 'Options', 'Rules', 'ABI', 'Parallelization', and 'Support'. The 'ABI' tab is currently selected.

The 'Reference Target' is set to 'Analyzer default'. The 'Option' column lists various ABI options, and the 'Value' and 'Key' columns show their current settings.

Size of scalar types (bytes)

Option	Value	Key
bool	1	sizeof_bool
short	2	sizeof_short
int	4	sizeof_int
long	4	sizeof_long
long long	8	sizeof_long_long
float	4	sizeof_float
double	8	sizeof_double
long double	12	sizeof_long_double
Pointer	4	sizeof_pointer
Function pointer	4	sizeof_function_pointer
Attributed pointer 1	4	sizeof_attributed_pointer_1
Attributed pointer 2	4	sizeof_attributed_pointer_2
Attributed pointer 3	4	sizeof_attributed_pointer_3

Alignment (bytes)

Option	Value	Key
bool	1	alignof_bool
short	2	alignof_short
int	4	alignof_int
long	4	alignof_long
long long	8	alignof_long_long
float	4	alignof_float
double	8	alignof_double
long double	16	alignof_long_double
Pointer	4	alignof_pointer
Function pointer	4	alignof_function_pointer
char array	1	alignof_char_array
Attributed pointer 1	4	alignof_attributed_pointer_1
Attributed pointer 2	4	alignof_attributed_pointer_2
Attributed pointer 3	4	alignof_attributed_pointer_3

Aligned access is atomic

Option	Value	Key
bool	no	atomic_bool
char	no	atomic_char
short	no	atomic_short

The 'Filter' field is empty, and the 'Findings' button is visible at the bottom.

Analyse starten

Start analysis

Start analysis & Preprocess

The screenshot shows the Astrée IDE interface. The 'Findings' window is active, displaying a table of rule violations. The 'Results' window shows a pie chart representing the distribution of findings. The 'Files' window lists the source files being analyzed.

Findings	Count	Name
12	12	Alarms
2	2	Overflow in conversion
10	10	Overflow in arithmetic

Alarms (12 findings)

100%

File	Count	Name
atashf	4	0
atashf	4	0
atashf	4	0
atashf	4	0
atashf	4	0
atashf	4	0
atashf	4	0
atashf	4	0
atashf	4	0
atashf	4	0

Result summary

```
/* Result summary */
Errors: 0
Alarms on code locations: 0
Run-time errors: 12
Flow anomalies: 0
Rule violations: 0
Alarms on memory locations: 0
Data races: 0
Reached code: 100 % (123/123 statements reached, 118/123 (95 %) statements proven)
Duration: 1.21 s (0 h 0 m 1 s)
```

*** Analysis of project vezs-gruppe-42 (s/a) with id 1578986432 revision 1 terminated successfully on 2018/06/28 at 17:12:41 ***

*** Starting postprocessing... ***

*** Postprocessing project vezs-gruppe-42 (s/a) with id 1578986432 revision 1 terminated successfully on 2018/06/28 at 17:12:42 ***

*** Analysis used not more than 101 MB (855 65 MB) of memory (total analysis time 0:01 minutes) ***

1 Abstrakte Interpretation mit Astrée

2 Formale Verifikation

3 Aufgabenstellung



Programme zur Softwareverifikation

- Astrée
- SeaHorn
- Frama-C
- VeriFast
- Dafny
- Coq
- Isabelle/HOL
- Agda
- Idris



```

method MergeSort(a: array<int>)
    requires a ≠ null ∧ sorted(a)
    ensures ∀ i, j. 0 ≤ i < j ≤ a.Length ⇒ (sorted(a[i..j], a[i])
    ∧ sorted(a[j+1..a.Length-1], a[j+1])
    ∧ sorted(a[i..a.Length-1], a[i]))
{
    var c, w, b, h, s, t, l, Length;
    while w < b
    {
        Invariant 0 ≤ c ≤ a ≤ b ≤ a.Length;
        Invariant 1 ≤ s ≤ t ≤ c + (a-b) ⇒ find
        (sorted(a[i..c], a[i]) ⇒ sorted(a[c+1..t], a[c+1])
        ∧ sorted(a[t+1..a.Length-1], a[t+1])
        ∧ sorted(a[i..a.Length-1], a[i]))
        {
            merge a[i..c], a[c+1..t], a[t+1..a.Length-1];
            case Red ⇒
                4?l? 4?l? ⇒ a?l?, a?l?;
                c, w = c + 1, w + 1;
            case Blue ⇒
                a = w + 1;
                c = Blue;
                w = 1;
        }
    }
}

```



...

P : wahr

```
S: int maximum(int a, int b) {  
    int result = INT_MIN;  
  
    if(a > b)  
        result = a;  
    else  
        result = b;  
  
    return INT_MAX;  
}
```

Q : $\text{result} \geq a \wedge \text{result} \geq b \wedge$
 $(\text{result} == a \vee \text{result} == b)$

```
const INT_MIN := -2147483648;  
const INT_MAX := 2147483647;
```

```
method maximum(a: int, b: int)  
    returns (max: int)  
    ensures max  $\geq$  a;  
    ensures max  $\geq$  b;  
    ensures max = a  $\vee$  max = b;  
    requires INT_MIN  $\leq$  a  $\leq$  INT_MAX;  
    requires INT_MIN  $\leq$  b  $\leq$  INT_MAX;  
{  
    var result := INT_MIN;  
    if (a > b) {  
        result := a;  
    } else {  
        result := b;  
    }  
    return INT_MAX;  
}
```



Beispiel: Maximum in Liste finden

$P : a \neq \text{NULL} \wedge \text{length} > 0$

```
S: int findMax(int *a, size_t length) {  
    int max = a[0];  
    for (size_t i = 0; i < length; i++) {  
        if (a[i] > max) {  
            max = a[i];  
        }  
    }  
    return max;  
}
```

$Q : \forall k. k \geq 0 \wedge k < \text{length}$
 $\Rightarrow a[k] \leq \text{result}$
 $\exists k. k \geq 0 \wedge k < \text{length}$
 $\Rightarrow a[k] == \text{result}$

```
method FindMax(a : array?<int>) returns (MaxValue: int)  
    requires a  $\neq$  null;  
    requires a.Length > 0;  
    ensures  $\forall k \bullet k \geq 0 \wedge k < \text{a.Length} \Rightarrow a[k] \leq \text{MaxValue}$ ;  
    ensures  $\exists k \bullet k \geq 0 \wedge k < \text{a.Length} \Rightarrow \text{old}(a[k]) = \text{MaxValue}$ ;  
{  
    var max := a[0];  
    var idx := 0;  
  
    var i := 1;  
  
    while (i < a.Length)  
        invariant  $\forall k \bullet k \geq 0 \wedge k < i \wedge k < \text{a.Length} \Rightarrow a[k] \leq \text{max}$ ;  
        invariant idx  $\geq 0 \wedge \text{idx} < i \wedge \text{idx} < \text{a.Length}$ ;  
        invariant a[idx] = max;  
        invariant  $\exists k \bullet k \geq 0 \wedge k < i \wedge k < \text{a.Length} \Rightarrow a[k] = \text{max}$ ;  
        {  
            if (max  $\leq$  a[i])  
            {  
                max := a[i];  
                idx := i;  
            }  
            i := i + 1;  
        }  
  
    return max;  
}
```



1 Abstrakte Interpretation mit Astrée

2 Formale Verifikation

3 Aufgabenstellung



- Sensorstubs implementieren
- Implementierung eines Ringpuffers zur Verwaltung der Sensorwerte
- System erstellen:
 - Sensoren und Filter initialisieren
 - Sensorwerte abrufen
 - Sensorwerte filtern
- Abwesenheit von Laufzeitfehlern nachweisen
 - ~> Astrée
- Numerische Stabilität des Filters nachweisen
 - ~> Astrée



■ Beispiel für dynamisch angelegten Ringpuffer

```
1 struct BndBuf{  
2     int* buf; // array of <size>  
3     int size;  
4     int read_pos;  
5     int write_pos;  
6 };
```

■ „Füllstandsanzeige”: Speichern von

- Belegten Plätzen
- Verfügbaren Plätzen

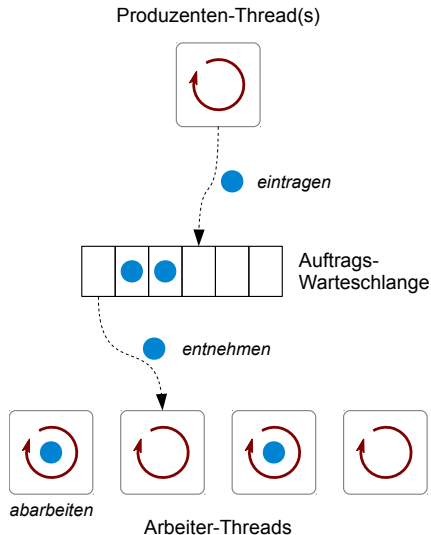
■ Mögliche Signalisierung falls

- Plätze frei wurden
- Plätze belegt wurde



Ringpuffer – Anwendungsszenario

- Verwaltung einer begrenzten Anzahl an Ressourcen
- Beispiel: Thread-Pool
- Feste Menge von Arbeiter-Threads:
 - laufen endlos
 - erhalten Aufträge zur Abarbeitung
- Verteilen der Aufträge mittels zentraler, synchronisierter Warteschlange (z. B. Ringpuffer)
- Vorteil: kein ständiges Erzeugen + Zerstören von Threads für Aufträge

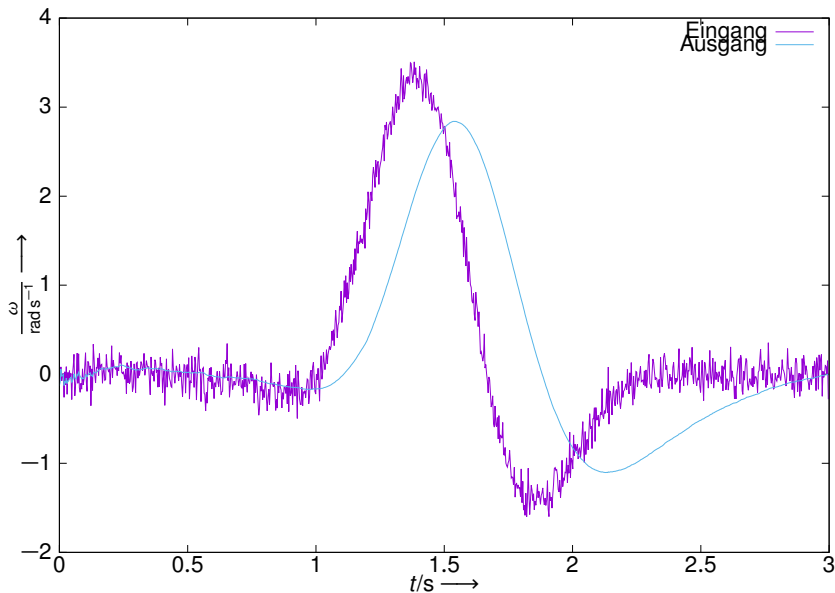




- Rauschunterdrückungsfilter
- Geeignet zur Schätzung von physikalischen Größen
 - mit Ableitung $\neq 0$
 - z. B. Position eines Flugzeugs, Lagewinkel ...
 - im I4Copter
 - liefern Sensoren häufiger Werte als diese später verarbeitet werden
 - ~> α - β -Filter für *Ratenwandlung* verwendet
 - ~> nutzt gewonnene Information vollständig



Beispiel α - β -Filterung



- Wird für jeden Messwert ausgeführt
- $y[\kappa]$: Eingabewert für Abtastschritt κ
- $\hat{x}[\kappa]$: Schätzung der Messgröße zum Abtastschritt κ
- T Abtastintervall, α , β Filterparameter
- Initialisierung: z. B. $\hat{x}[0] = \dot{\hat{x}}[0] = 0$

$$r[\kappa] = y[\kappa] - \hat{x}[\kappa - 1] \quad \leadsto \text{Schätzfehler} \quad (1)$$

$$\dot{\hat{x}}[\kappa] = \dot{\hat{x}}[\kappa - 1] + \frac{\beta}{T} \cdot r[\kappa] \quad \leadsto \text{1. Ableitung} \quad (2)$$

$$\hat{x}[\kappa] = \hat{x}[\kappa - 1] + T \cdot \dot{\hat{x}}[\kappa] + \alpha \cdot r[\kappa] \quad \leadsto \text{Schätzwert} \quad (3)$$

- Sinnvolle Werte für α und β ?
 - Literatur beschreibt viele Verfahren $\leadsto$ hier beispielhaft nur eines [5, 4]



T Abtastintervall

$\leadsto$ in welchem Zeitabstand gemessen wird

σ_w^2 Prozessvarianz

$\leadsto$ wie lebhaft der gemessene Prozess ist

σ_v^2 Rauschvarianz

$\leadsto$ wie verrauscht das Signal ist

$$\lambda = \frac{\sigma_w T^2}{\sigma_v} \quad \leadsto \text{Tracking Index} \quad (4)$$

$$\theta = \frac{4 + \lambda - \sqrt{8\lambda + \lambda^2}}{4} \quad \leadsto \text{Dämpfungsparameter} \quad (5)$$

$$\alpha = 1 - \theta^2 \quad \leadsto \text{Gewicht für Wert} \quad (6)$$

$$\beta = 2(2 - \alpha) - 4\sqrt{1 - \alpha} \quad \leadsto \text{Gewicht für Ableitung} \quad (7)$$



- Zu Beginn hat das Filter keinen gültigen Zustand
- ~ Einschwingphase, in der das Filter „lernt“
- n startet bei 1 und wächst in jeder Runde um 1

$$\alpha_n = \frac{2(2n+1)}{(n+2)(n+1)} \quad (8)$$

$$\beta_n = \frac{6}{(n+2)(n+1)} \quad (9)$$

- Einschwingphase endet, wenn $\alpha_n < \alpha$
- Wird der Filterzustand im Betrieb ungültig, wird eine neue Einschwingphase eingeleitet



- Filter ist nur dann korrekt, wenn es auch *stabil* ist
 - ↪ für wertbegrenzte Eingabe erfolgt wertbegrenzte Ausgabe [6]
 - ↪ für Eingabe 0 geht der Filterausgang asymptotisch gegen 0
- α - β -Filter stabil, wenn gilt

$$0 < \alpha \leq 1 \quad (10)$$

$$0 < \beta \leq 2 \quad (11)$$

$$0 < 4 - 2\alpha - \beta \quad (12)$$

- Außerdem: laut [2] Rauschunterdrückung nur dann, wenn

$$0 < \beta < 1 \quad (13)$$

- Stabilität muss auch in der Initialisierungsphase gegeben sein!



- Erfahrungen mit dem I4Copter haben gezeigt, dass sich die Parameter in folgenden Bereichen bewegen:

Abtastintervall $T \in (0 \dots 1]$

Prozessvarianz $\sigma_w^2 \in [0.5 \dots 30.0]$

Rauschvarianz $\sigma_v^2 \in [10^{-3} \dots 10^{-1}]$

Wert $y[k] \in [-10 \dots 10]$

→ Korrektheit mindestens für diese Wertebereiche zeigen!





AbsInt Angewandte Informatik GmbH.
The Static Analyzer Astrée, April 2012.



C. Frank Asquith.
Weight selection in first-order linear filters.
Technical report, Army Intertial Guidance and Control Laboratory Center, Redstone Arsenal, Alabama, 1969.



Eli Brookner.
Tracking and Kalman Filtering Made Easy.
Wiley-Interscience, 1st edition, 4 1998.



E. Gray, J. and W. Murray.
A derivation of an analytic expression for the tracking index for the alpha-beta-gamma filter.
IEEE Trans. on Aerospace and Electronic Systems, 29:1064–1065, 1993.



Paul R. Kalata.
The tracking index: A generalized parameter for α - β and α - β - γ target trackers.
IEEE Transactions on Aerospace and Electronic Systems, AES-20(2):174–181, mar 1984.



Richard G. Lyons.
Understanding Digital Signal Processing.
Prentice Hall, 3rd edition, 11 2010.