Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Softwareentwurf am Beispiel

Phillip Raffeck, Tim Rheinfels, Simon Schuster, Peter Wägemann

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

Wintersemester 2022

Fragestellung

Wie komme ich von der Beschreibung zur Software?

Objektorientierter/Objektbasierter Entwurf [1]

1. Identifiziere **Objekte** und deren Attribute
2. Identifiziere **Operationen** jedes Objekts
3. Lege **Sichtbarkeit** fest
4. Lege **Objektschnittstellen** fest
5. **Implementiere** Objekte

Beispielbeschreibung – α-Filter

Ein α-Filter ist eine vereinfachte Form eines Beobachters, das zur Schätzung und Glättung verwendet werden kann. Es beobachtet genau einen Zustand $\hat{x}[\kappa]$. Die Schätzung des aktuellen Zustands ergibt sich hierbei aus dem Schätzwert der vorherigen Iteration, dem Filterparameter α und dem aktuellen Messwert $y[\kappa]$ in den folgenden zwei Schritten:

$$r[\kappa] = y[\kappa] - \hat{x}[\kappa - 1] \tag{1}$$

$$\hat{x}[\kappa] = \hat{x}[\kappa - 1] + \alpha \cdot r[\kappa] \tag{2}$$

Um den Filterparameter α optimal zu bestimmen, ist die Kenntnis von Prozessvarianz σ_w^2, Abtastintervall T sowie der Rauschvarianz σ_v^2 nötig. Die Rauschvarianz gibt hierbei die Genaugikeit der Messung an, die Prozessvarianz die Volatilität des beobachteten Systems. Sind diese Parameter bekannt, ergibt sich der optimale Wert für α wie folgt:

$$\lambda = \sigma_w \cdot T^2 / \sigma_v \tag{3}$$

$$\alpha = \frac{-\lambda^2 + \sqrt{\lambda^4 + 16\lambda^2}}{8} \tag{4}$$

Benutzerschnittstelle – α-Filter

Vor der ersten Benutzung muss das Filter unter Angabe der für die optimale Bestimmung von α benötigten Parameter Prozessvarianz, Rauschvarianz und Abtastintervall initialisiert werden. Es liegt in der Verantwortung des Benutzers sicherzustellen, dass das Filter vor der ersten Benutzung initialisiert wurde.
Nach erfolgreicher Initialisierung können einzelne Filterschritte unter Angabe des jeweils aktuellen Messwertes durchgeführt werden, sowie die aktuelle Schätzung des Zustands abgefragt werden.

1. Objekte und Attribute identifizieren

- Herangehensweise:
  - **Hauptwortextraktion** aus Anforderungsdokument
  - Identifikation der relevanten **Hauptwörter**
  - für kleinere Probleme: *Intuition*

Beispielbeschreibung – α-Filter

Ein **α-Filter** ist eine vereinfachte Form eines Beobachters, das zur Schätzung und Glättung verwendet werden kann. Es beobachtet genau einen **Zustand** $\hat{x}[\kappa]$. Die **Schätzung** des aktuellen **Zustands** ergibt sich hierbei aus dem **Schätzwert der vorherigen Iteration**, dem **Filterparameter** α und dem **aktuellen Messwert** $y[\kappa]$ in den folgenden zwei Schritten:

... (1/2)

Um den **Filterparameter** α optimal zu bestimmen, ist die Kenntnis von **Prozessvarianz** σ_w^2, **Abtastintervall** T sowie der **Rauschvarianz** σ_v^2 nötig. Die Rauschvarianz gibt hierbei die Genaugikeit der Messung an, die Prozessvarianz die Volatilität des beobachteten Systems. Sind diese Parameter bekannt, ergibt sich der optimale Wert für α wie folgt:

... (3/4)

1. Objekte und Attribute identifizieren

- Was ist das Objekt? $\leadsto$ **Filter**
- Attribute? Welche Information brauche ich für jeden Filterschritt?
  - Schätzung aus der Vorrunde $\hat{x}[\kappa - 1]$
  - Filterparameter α
  - aktuellen Messwert $y[\kappa]$ $\leadsto$ kein Zustand, kommt von aussen
- Attribute? Welche Information brauche ich für den Filterparameter?
  - Rauschvarianz
  - Prozessvarianz
  - Abtastintervall

Vorläufige Objektschablone

```
1 typedef struct _Alpha_Filter {
2   AF_Value_t x;
3   AF_Value_t alpha;
4 } Alpha_Filter;
```

2. Operationen identifizieren

- Herangehensweise:
  - **Verbenextraktion**
  - Identifikation der relevanten **Verben**
  - für kleinere Probleme: *Intuition*

Benutzerschnittstelle – α-Filter

Vor der ersten Benutzung muss das Filter unter Angabe der für die optimale Bestimmung von α benötigten Parameter Prozessvarianz, Rauschvarianz und Abtastintervall initialisiert werden. Es liegt in der Verantwortung des Benutzers sicherzustellen, dass das Filter vor der ersten Benutzung initialisiert wurde.

Nach erfolgreicher Initialisierung können unter Angabe des jeweils aktuellen Messwertes einzelne Filterschritte durchgeführt werden, sowie die aktuelle Schätzung des Zustands abgefragt werden.

2. Operationen identifizieren

- Leben eines Objekts:
  1. Initialisierung $\leadsto$ Betriebsmittel anfordern
  2. Verwendung
  3. Beseitigung $\leadsto$ Betriebsmittel freigeben
- Was möchten Benutzer mit dem Filter machen?
  - Filter initialisieren
  - Filterschritt ausführen
  - Schätzwert erfragen
  - Betriebsmittelfreigabe nicht notwendig

3. Sichtbarkeit festlegen

- In modernen Programmiersprachen `private`, `public`, ...
- In C nur eingeschränkt möglich
  - modulintern vs. modulextern
- Leitfaden: möglichst wenig sichtbar machen

  ⇝ öffentliche Schnittstelle bedeutet Verpflichtung
- Was soll bei unserem Filter öffentlich sein?
  - Initialisierung
  - Filterschritt
  - Schätzung abfragen
- Alle anderen Operationen modulintern

  ⇝ Hilfsfunktionen `static`

4. Schnittstelle festlegen

- Zwischen Modul und Außenwelt
- Statische Semantik

Schnittstelle

```
1  void afilter_init(Alpha_Filter *filter,
2                    AF_Value_t process_variance,
3                    AF_Value_t noise_variance,
4                    AF_Value_t sampling_interval);
5
6  void afilter_step(Alpha_Filter *filter,
7                    AF_Value_t measurement);
8
9  AF_Value_t afilter_get_estimate(Alpha_Filter *filter);
```

5. Implementierung – Header

alpha_filter.h

```
1  #ifndef ALPHA_FILTER_H_INCLUDED
2  #define ALPHA_FILTER_H_INCLUDED
3
4  typedef float AF_Value_t;
5  typedef struct _Alpha_Filter {
6    AF_Value_t x;
7    AF_Value_t alpha;
8  } Alpha_Filter;
9
10 void afilter_init(Alpha_Filter *filter,
11                   AF_Value_t process_variance,
12                   AF_Value_t noise_variance,
13                   AF_Value_t sampling_interval);
14
15 void afilter_step(Alpha_Filter *filter,
16                   AF_Value_t measurement);
17
18 AF_Value_t afilter_get_estimate(Alpha_Filter *filter);
19 #endif // ALPHA_FILTER_H_INCLUDED
```

5. Implementierung – Initialisierung

$$\lambda = \sigma_w \cdot T^2 / \sigma_v \tag{3}$$

$$\alpha = \left(-\lambda^2 + \sqrt{\lambda^4 + 16\lambda^2}\right)/8 \tag{4}$$

$$\hat{x}[0] = 0 \tag{5}$$

alpha_filter.c

```
1  void afilter_init(Alpha_Filter *filter,
2                    AF_Value_t process_variance,
3                    AF_Value_t noise_variance,
4                    AF_Value_t sampling_interval) {
5    filter->x = 0;
6    AF_Value_t l = sqrt(process_variance)
7      * sampling_interval * sampling_interval
8      / sqrt(noise_variance);
9    filter->alpha = (-l*l
10     + sqrtf(l*l*l*l + 16.0f*l*l)) / 8.0f;
11 }
```

5. Implementierung – Filterschritt

$$r[\kappa] = y[\kappa] - \hat{x}[\kappa - 1] \tag{1}$$

$$\hat{x}[\kappa] = \hat{x}[\kappa - 1] + \alpha \cdot r[\kappa] \tag{2}$$

alpha_filter.c

```
1  void afilter_step(Alpha_Filter *filter,
2                    AF_Value_t measurement) {
3    AF_Value_t  r = measurement - filter->x;
4    filter->x = filter->x + filter->alpha * r;
5  }
6
7  AF_Value_t afilter_get_estimate(Alpha_Filter *filter) {
8    return filter->x;
9  }
```

Zwei Prinzipien für die Übung

KISS – Keep it Small and Simple!

- Kleine Softwaremodule mit geringer Kopplung
- *Eine* (C-)Funktion löst *eine* Aufgabe

☞ Bessere Wartbarkeit, Testbarkeit, Verifizierbarkeit

Literatur I

Grady Booch.
Software Engineering with Ada.
The Benjamin/Cummings Publishing Company, Inc., 2nd edition, 1987.