Datenfusion

Parameter	Kursinformationen
Veranstaltung:	`Robotik Projekt`
Semester	`Wintersemester 2024/25`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	`Grundlagen der Datenfusion für redundanter Sensoren`
Link auf GitHub:	https://github.com/TUBAF-IfI-LiaScript/VL_SoftwareprojektRobotik/blob/master/08_Datenfusion/08_Datenfusion.md
Autoren	Sebastian Zug & Georg Jäger

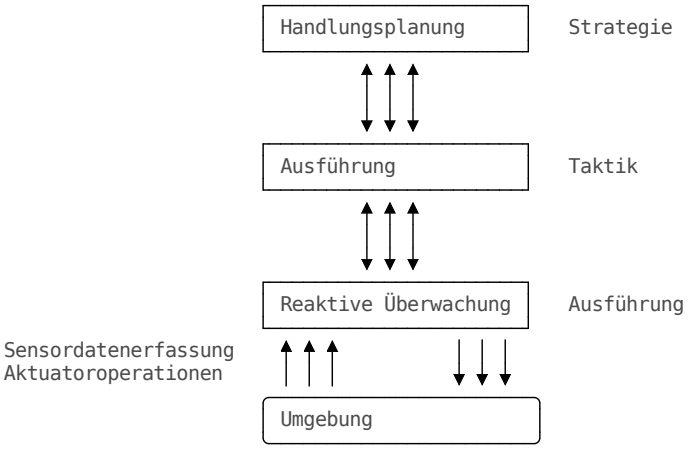
Zielstellung der heutigen Veranstaltung

- Vermittlung eines Grundverständnisses für die Datenfusion
- Einführung eines Bayes basierten Schätzers als diskreten Filter

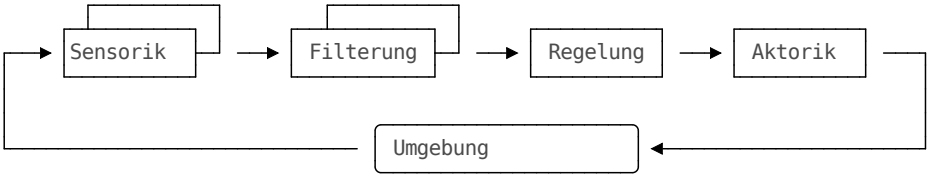
---

Wie weit waren wir gekommen?

... wir generieren ein "rohes" Distanzmessignal und haben es gefiltert.



Im weiteren Verlauf der Veranstaltung werden wir uns auf den letzte Ebene fokussieren und die elementare Verarbeitungskette verschiedener Sensorsysteme analysieren.



Grundlagen und Konzepte

Zielstellung: Übergreifende Abbildung der Ergebnisse einer multimodalen Umgebungserfassung

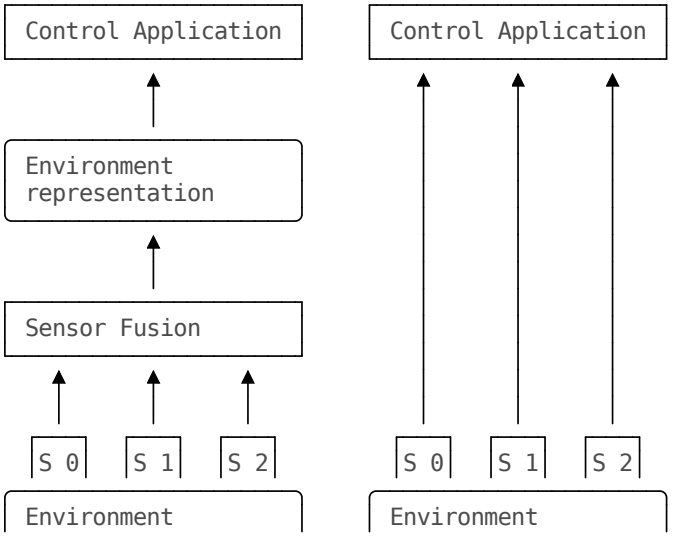
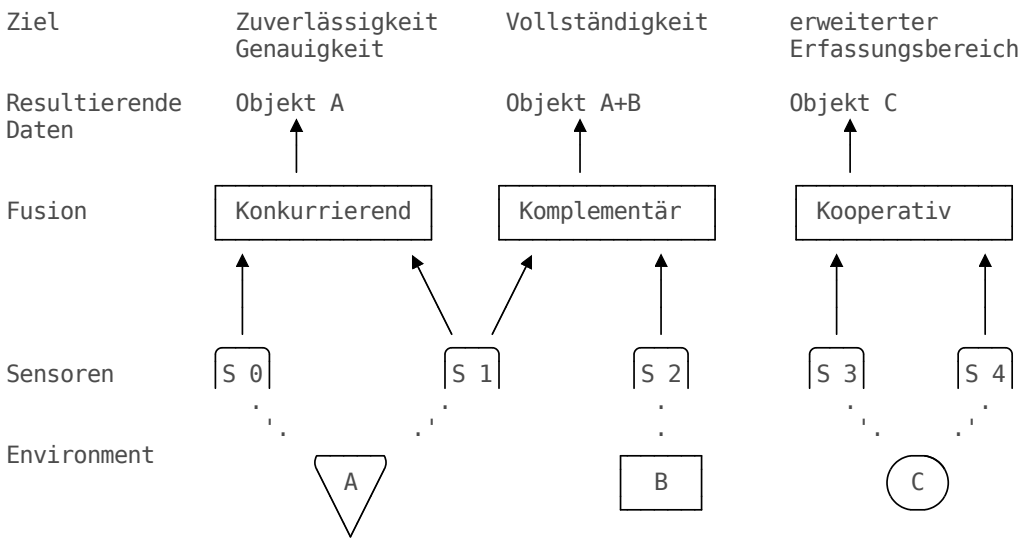


Abbildung motiviert nach Wilfried Elmenreich, An Introduction to Sensor Fusion, Research Report 47/2001, TU Wien

Herausforderungen für die Fusion der Messdaten:

- unterschiedliche Messraten (→ Synchronisation)
- verschiedene räumliche Abdeckungen/ Auflösungen (→ Kalibrierung)
- unterschiedliche (variable) Fehlermodelle
- ...

Je nach Zielstellung verfolgt die Fusion unterschiedliche Ziele:



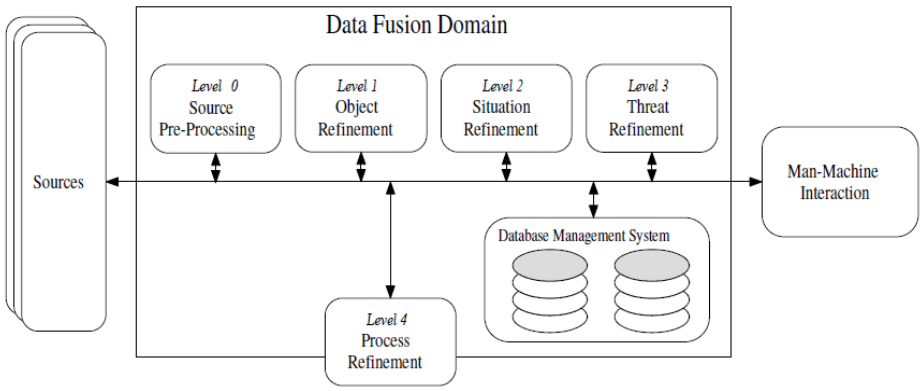
Unterschiedliche Fusionsansätze analog zu Brooks und Iyengar (1997)

- Bei der **konkurrierenden Fusion** erfassen Sensoren gleichzeitig denselben Sichtbereich und liefern Daten gleicher Art. Die (oft gewichtete) Verknüpfung solcher, "konkurrierender" Daten kann die Genauigkeit des Gesamtsystems erhöhen.
- Eine **komplementäre Fusion** hat das Ziel, die Vollständigkeit der Daten zu erhöhen. Unabhängige Sensoren betrachten hierfür unterschiedliche Sichtbereiche und Phänomene oder messen zu unterschiedlichen Zeiten.
- Reale Sensoren erbringen die gewünschten Informationen oft nicht allein. So ergibt sich beispielsweise die benötigte Information erst aus dem Zusammensetzen der verschiedenen Ausgabedaten. Eine solche Fusion wird als **kooperative Fusion** bezeichnet.

Was aber wird fusioniert? Ein einfaches Distanzmaß, die Positioninformation eines Roboters oder die Aufenthaltswahrscheinlichkeit in einem bestimmten Raum ... Hall & Llinas (1997) unterscheiden dafür drei Ebenen der Sensordatenfusion, die die Datenkategorien - Raw Data, Feature und Decicion - referenzieren:

- Bei der **data fusion** werden die rohen Sensordaten vor weiteren Signalverarbeitungsschritten miteinander verschmolzen.
- Bei der **feature fusion** erfolgt vor der Verschmelzung eine Extraktion eindeutiger Merkmale. Die neu kombinierten Merkmalsvektoren werden im Anschluss weiterverarbeitet.
- Bei der **decision fusion** erfolgt die Zusammenführung erst nachdem alle Signalverarbeitungs- und Mustererkennungsschritte durchgeführt wurden.

Ein immer wieder zitiertes Architekturkonzept für die Abbildung dieser Ansätze ist das *Joint Directors of Laboratories* (JDL) Modell.



Sensor and Data Fusion: Concepts and Application, second ed., vol. TT14SPIE Press (1999)

1. Vorverarbeitung : zeitliche und räumliche Registrierung der Daten sowie Vorverarbeitungsmaßnahmen auf Signal- oder Pixelebene.
2. Objekterkennung und Extraktion Schätzung und Vorhersage von kontinuierlichen oder diskreten Objektmerkmalen.
3. Situationsanalyse: alle detektierten Objekte werden in einen größeren Kontext gebracht und Objektbeziehungen analysiert werden.
4. Bedrohungsanalyse: Situationsabhängig werden, im Sinne einer Risikominimierung, unterschiedliche Handlungsoptionen evaluiert.
5. Prozessoptimierung: Verwaltung und Adaption der Ressourcen

Einführung in den diskreten Bayes Filter

Das Beispiel basiert auf der hervorragenden Einführung in die Mathematik und Anwendung von Fusionsansätzen des

https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/02-Discrete-Bayes.ipynb

Das Anwendungsbeispiel zielt auf den Schwimmroboter, der sich auf der Freiberger Mulde bewegt, lässt sich aber analog auf beliebige andere 1D, 2D oder 3D Szenarien übertragen.

Anstatt eine kontinuierliche Messung zu realisieren bilden wir unsere Position auf diskrete Streckenabschnitte ab. Im Bild oben werden diese mit 0-9 indiziert.

```
  Streckensegmente der Freiberger Mulde

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
======O===SB======O===O===B===S=======S== ⟶
                                        Strömung
```

Abbildung des Streckenmodels (O = Orange Warnschilder am Ufer, S = Starke Strömung, B = Brücken)

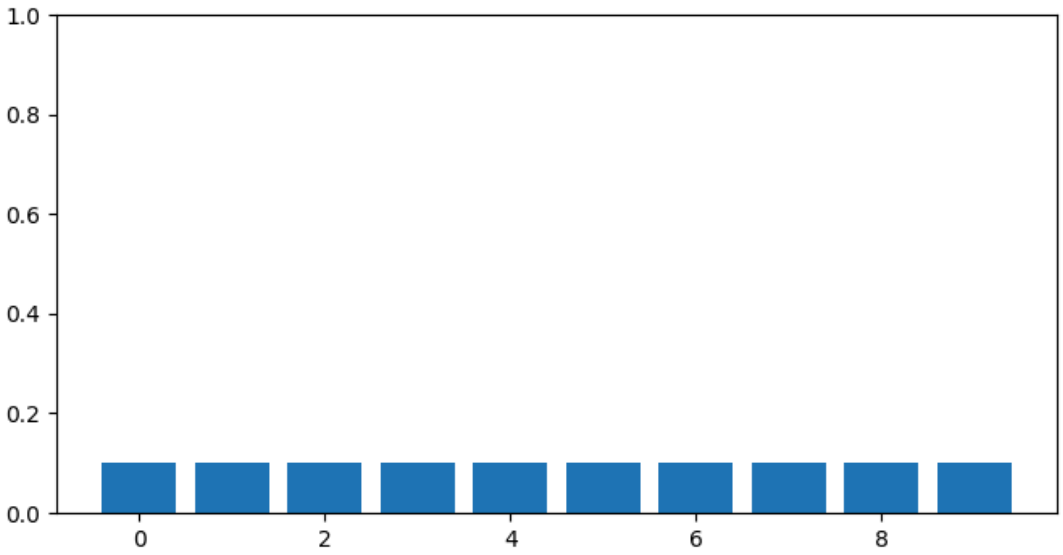
Welche Anfangskenntnis zur Position haben wir zunächst? Die Aufenthaltswahrscheinlichkeit ist für alle 10 Streckensegmente gleich und entsprechend $p_i = 0.1$ für alle $0 \leq i < 10$

generateBelief.py

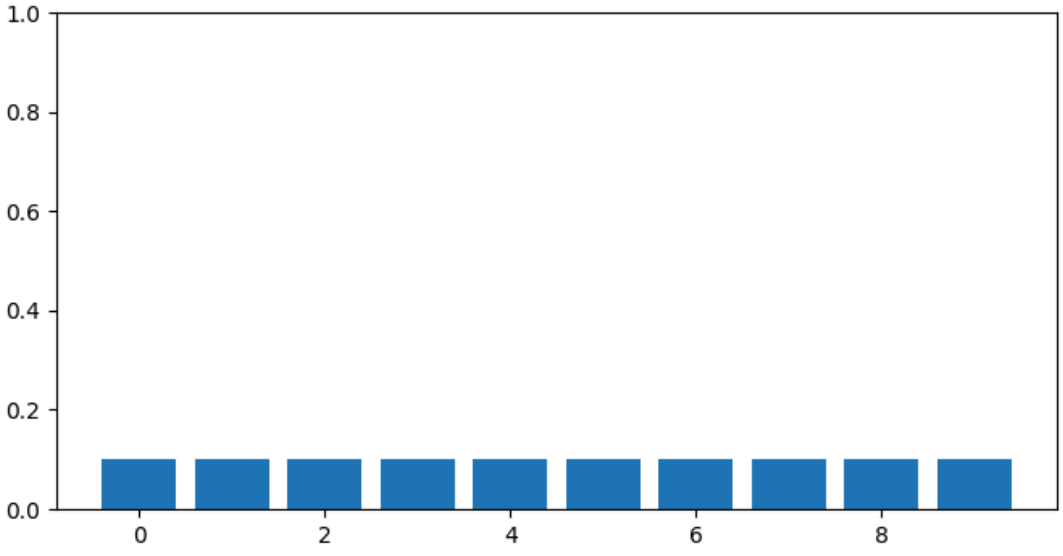
```python
import numpy as np
import matplotlib.pyplot as plt
belief = np.array([1./10]*10)
print(belief)

fig, ax = plt.subplots(figsize=(8,4))
ax.bar(np.arange(len(belief)), belief)
plt.ylim(0, 1)
#plt.show()
plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript
```

```
[0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]
[0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]
```

foo.png



foo.png



Es wird deutlich, dass wir aktuell noch kein Wissen um die Position des Bootes haben. Man spricht vom "apriori-"Wissen.

Sensoren

Unser Roboter ist mit verschiedenen Sensorsystemen ausgestattet.

1. Im Frontbereich des Roboters befindet sich ein Kamerasystem, dass aus Bilddaten Features zu erkennen versucht. Dabei konzentrierte man sich auf orange Warnschilder und Brücken über die Mulde.
2. Ein Beschleunigungssensor bestimmt die Bewegungen des Bootes. Damit lassen sich zum Beispiel starke Strömungen gut erkennen.
3. An einige Stellen können mit dem GNSS-System Messdaten generiert werden. Die umgebenden Höhenzüge behindern dies.

Alle drei Messmethoden sind mit Fehlern überlagert. Welche können das sein?

Bildsensor

Nun nutzen wir den ersten Sensormodus, die Erkennung der orangen Zeichen. Wir gehen davon aus, dass der Sensor nur in seltenen Fällen eine falsche Klassifikation (Schild/kein Schild) vornimmt.

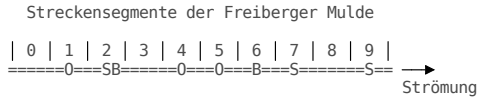


Abbildung des Streckenmodels (O = Orange Warnschilder am Ufer, S = Starke Strömung, B = Brücken)

Im folgenden sprechen wir vom tatsächlichen Zustand x und der Messung z. Für unsere Messmethode haben wir verschiedene Laborexperiemente gemacht und eine Vierfelder-Tafel erstellt.

		Schild im Segement: ja	Schild im Segement: nein	
Schild erkannt	ja	10	2	12
	nein	1	7	8
		11	9	20

Vierfeldertafel unseres "Schilderkenners"

Abgebildet auf Wahrscheinlichkeiten bedeutet das ja:

		Schild im Segement: ja	Schild im Segement: nein	
Schild erkannt	ja	0.5	0.1	0.6
	nein	0.05	0.35	0.4
		0.55	0.45	1.0

Wie wahrscheinlich ist es also, dass wir uns tatsächlich an einem orangen Schild befinden, wenn wir eine entsprechende Messung vorliegen haben?

$$p(x|z) = \frac{0.5}{0.5 + 0.1} = 0.83$$

Wie wahrscheinlich ist eine Messung eines orangen Schildes, wenn wir gar keines erreicht haben?

$$p(x|z) = \frac{0.1}{0.5 + 0.1} = 0.17$$

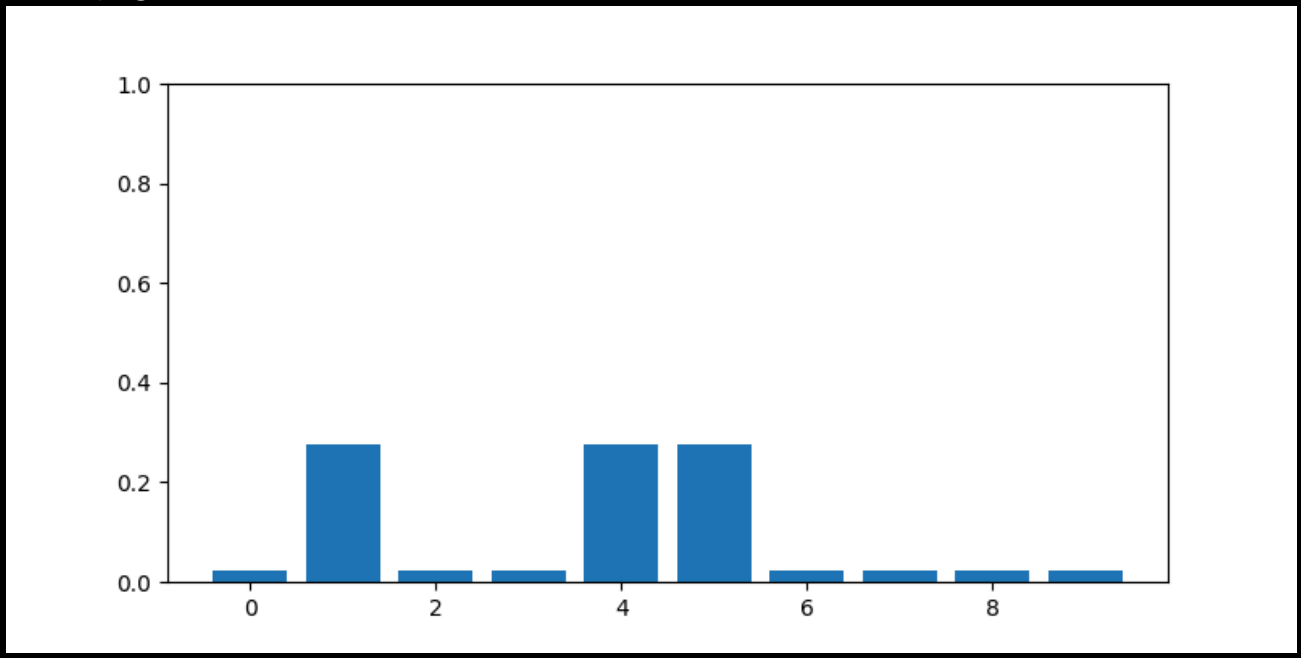
Wie können wir diese Sensorcharakteristik nun für unser Roboterbeispiel verwenden? Unsere Klassifikation wird durch die verschiedenen Möglichkeiten, an denen sich der Roboter aufhalten kann, „verwässert“. Wenn wir nur ein Segment mit einem Schild hätten müssten wir diesem eine Aufenthaltswahrscheinlichkeit von $p = 0.83$ zuordnen. Da es aber mehrere Möglichkeiten gibt, splittet sich diese auf.

generateMeasurements.py

```python
import numpy as np
import matplotlib.pyplot as plt

markers = np.array([0, 1., 0, 0, 1., 1., 0, 0, 0, 0])
truePositive = 0.83
belief = markers / sum(markers)*truePositive
belief[markers == 0] = (1-truePositive)/np.count_nonzero(markers==0)
print(belief)

fig, ax = plt.subplots(figsize=(8,4))
ax.bar(np.arange(len(belief)), belief)
plt.ylim(0, 1)
#plt.show()
plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript
```

```
[0.02428571 0.27666667 0.02428571 0.02428571 0.27666667 0.27666667
 0.02428571 0.02428571 0.02428571 0.02428571]
```

foo.png




```
[0.02428571 0.27666667 0.02428571 0.02428571 0.27666667 0.27666667
 0.02428571 0.02428571 0.02428571 0.02428571]
```

foo.png

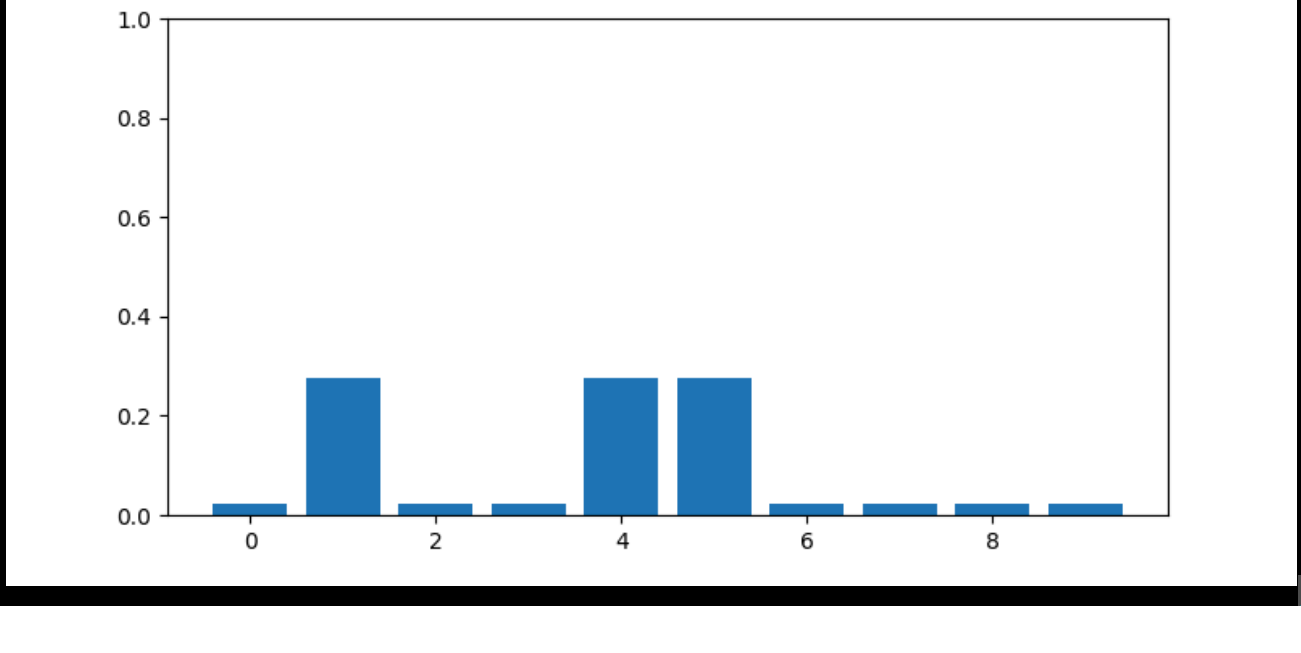


Abbildung auf apriori Wissen

Was bedeutet dies aber im Hinblick auf die Positionsbestimmung unter Berücksichtigung des apriori-Wissens?

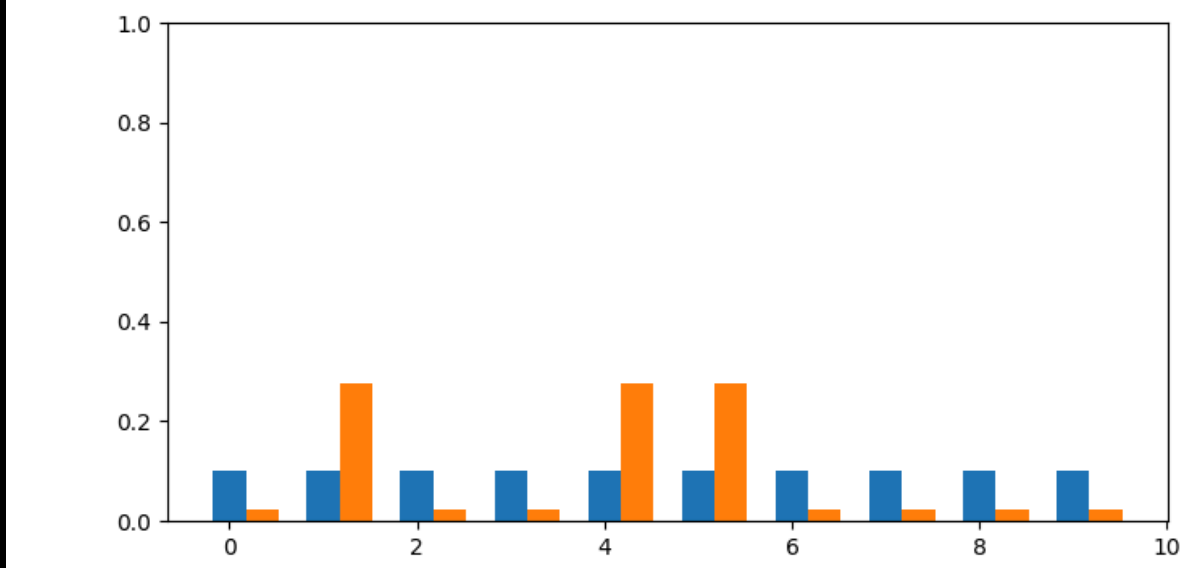
Wir haben zwei Vektoren mit Wahrscheinlichkeiten bezüglich unseres Aufenthaltes. Die Kombination draus ist eine einfaches Produkt gefolgt von einer Normalisierung.

$$posteriori = \frac{belief \cdot prior}{normalization}$$

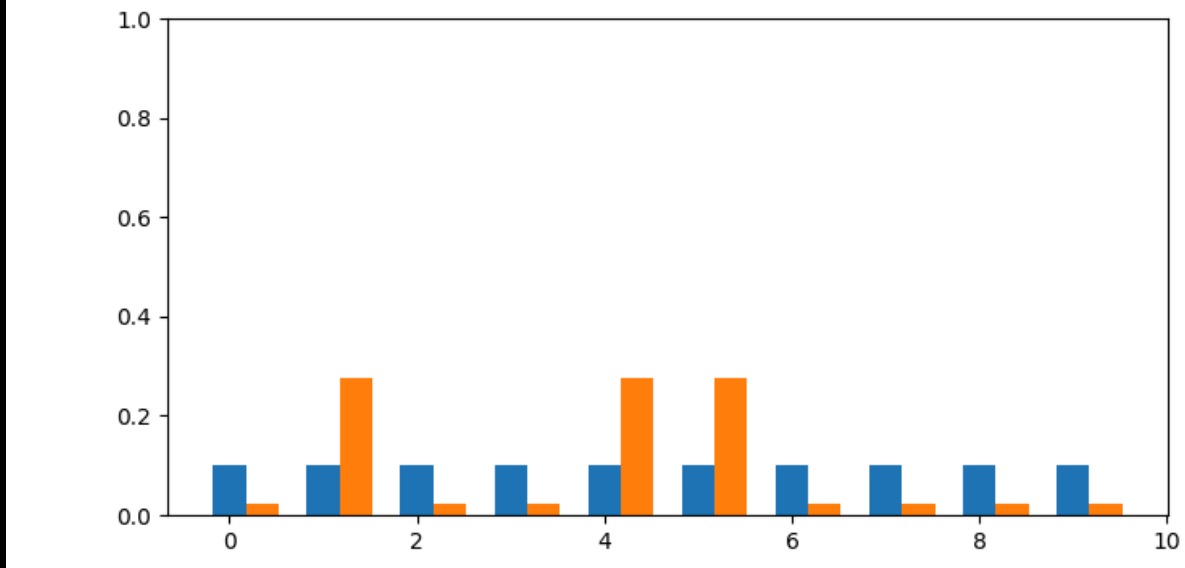
generateBelief.py

```python
import numpy as np
import matplotlib.pyplot as plt

# Vorwissen der Position
apriori = np.array([1./10]*10)
# Messungen
markers = np.array([0, 1., 0, 0, 1., 1., 0, 0, 0, 0])
truePositive = 0.83
belief = markers / sum(markers)*truePositive
belief[markers == 0] = (1-truePositive)/np.count_nonzero(markers==0)
posteriori = (apriori * belief) / sum(apriori*belief)

fig, ax = plt.subplots(figsize=(8,4))
width = 0.35
ax.bar(np.arange(len(apriori)), apriori, width)
ax.bar(np.arange(len(posteriori)) + width, posteriori, width)
plt.ylim(0, 1)
#plt.show()
plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript
```

foo.png



foo.png



Die Positionen der orangen Schilder heben sich deutlich ab, obwohl wir bei der konkreten Erkennung unsicher sind. Welche Veränderung erwarten Sie, wenn wir die Qualität der Sensormessungen erhöhen?

Wie können wir unsere Positionsschätzung verbessern:

1. Verbesserung der Qualität der Sensoren
2. Einbettung weiterer Sensoren
3. Wiederholung der Messungen (sofern wir von statistisch unabhängigen Messungen ausgehen)

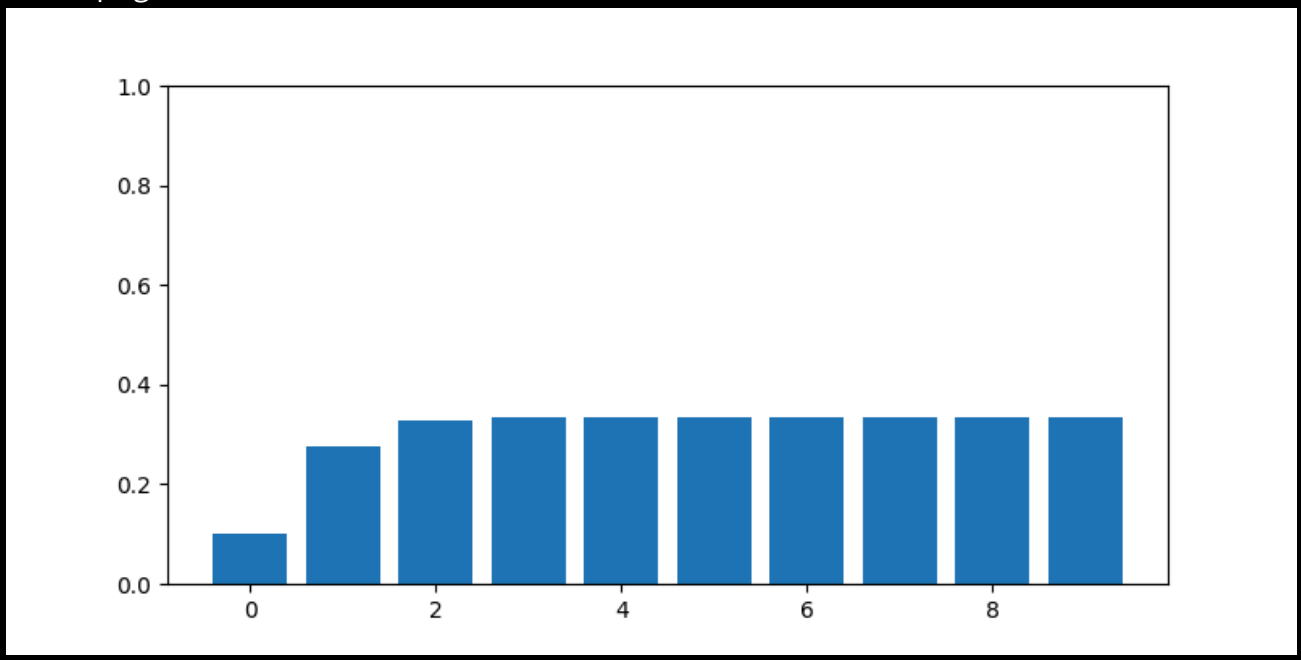
Lassen Sie uns die Messung einige Male wiederholen. Das folgende Diagramm zeigt die Schätzung an der Position 1, nachdem mehrere Messungen fusioniert wurden.

constantPosition.py

```python
import numpy as np
import matplotlib.pyplot as plt

apriori = np.array([1./10]*10)
markers = np.array([0, 1., 0, 0, 1., 1., 0, 0, 0, 0])
truePositive = 0.83
belief = markers / sum(markers)*truePositive
belief[markers == 0] = (1-truePositive)/np.count_nonzero(markers==0)
p_1 = []; p_1.append(apriori[1])
posteriori = apriori
for i in range(1, 10):
  posteriori = (posteriori * belief) / sum(posteriori*belief)
  p_1.append(posteriori[1])
print(p_1)

fig, ax = plt.subplots(figsize=(8,4))
ax.bar(np.arange(len(p_1)), p_1)
plt.ylim(0, 1)
#plt.show()
plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript
```

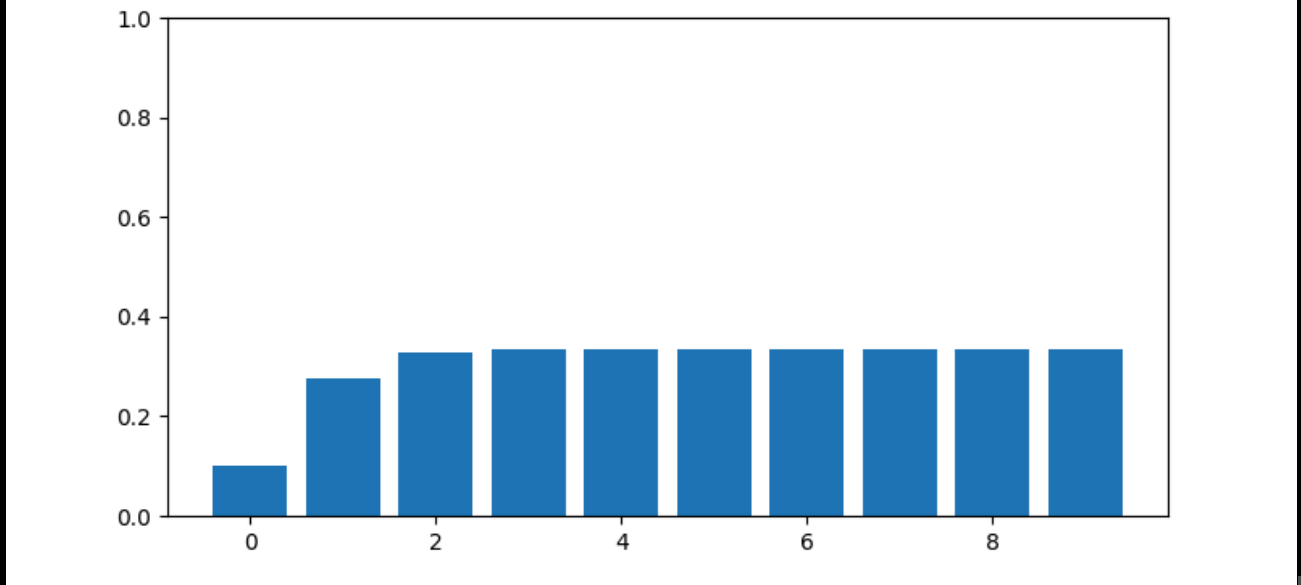
```
[np.float64(0.1), np.float64(0.2766666666666666),
np.float64(0.3274461872750729), np.float64(0.33280809936741945),
np.float64(0.33328716209257464), np.float64(0.3333292799239552),
np.float64(0.33333297752236696), np.float64(0.33333330210032647),
np.float64(0.3333333305917094), np.float64(0.3333333330926743)]
```

```
foo.png
```




```
[np.float64(0.1), np.float64(0.2766666666666666),
np.float64(0.3274461872750729), np.float64(0.33280809936741945),
np.float64(0.33328716209257464), np.float64(0.3333292799239552),
np.float64(0.33333297752236696), np.float64(0.33333330210032647),
np.float64(0.3333333305917094), np.float64(0.3333333330926743)]
```

```
foo.png
```

Unsere Positionsschätzung nähert sich der belief-Verteilung unsere Messung (3 von 10 Positionen sind mit einem orangen Schild ausgestattet) an. Der Einfluß des Anfangswissens geht zurück. Je häufiger ich messe, desto mehr vertraue ich den Messungen, sofern diese statistisch unabhängig sind.

Bewegung

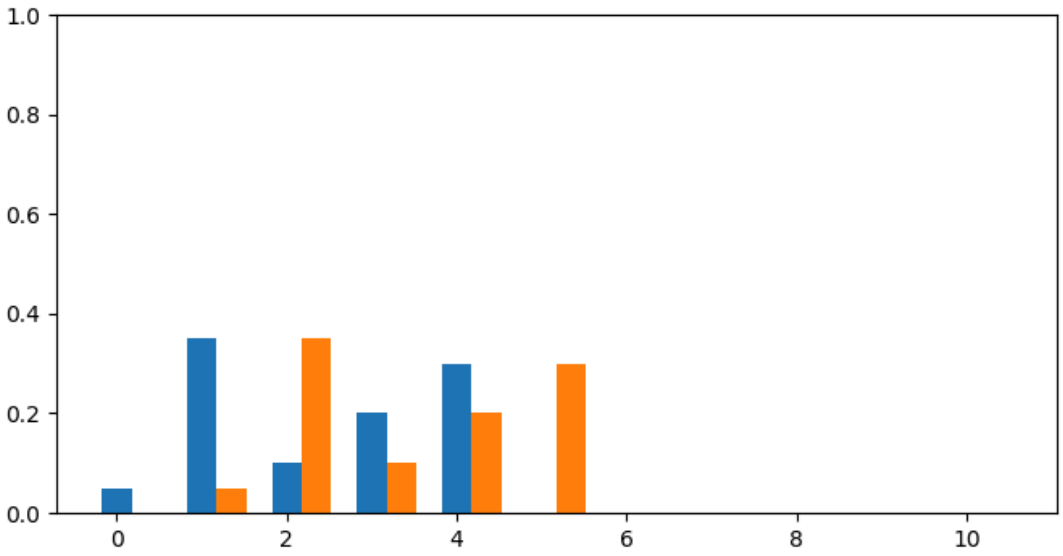
Die über der Zeit gestiegene Qualität der Positionsschätzung lässt sich bisher nur realisieren, wenn keine Bewegung erfolgt. Wie aber kann diese abgebildet werden?

Nehmen wir wiederum eine Aufenthaltwahrscheinlichkeit über unseren Segmenten an, die durch eine Bewegung u verändert wird. u ist dabei die Kontrollfunktion, mit der wir unserer Umgebung/Roboter manipulieren Der Einfachheit halber berücksichtigen wir zunächst eine ungestörte Abbildung. Für $u = v = 1$ verschiebt sich unser Roboter um eine Einheit.

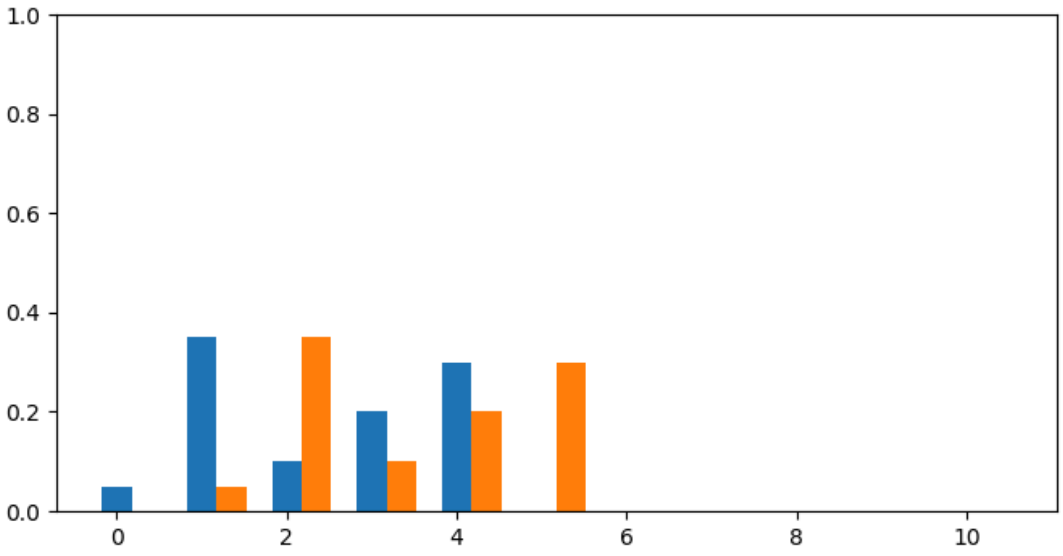
movements.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  def perfect_predict(belief, u):
5      n = len(belief)
6      result = np.zeros(n)
7      for i in range(n):
8          result[i] = belief[(i-u) % n]
9      return result
10
11 belief = np.array([0.05, .35, .1, .2, .3, 0, 0, 0, 0, 0, 0])
12 result = perfect_predict(belief, 1)
13
14 fig, ax = plt.subplots(figsize=(8,4))
15 width = 0.35
16 ax.bar(np.arange(len(belief)), belief, width)
17 ax.bar(np.arange(len(result)) + width, result, width)
18 plt.ylim(0, 1)
19 #plt.show()
20 plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript
```

foo.png



foo.png



Welche Probleme sehen Sie?

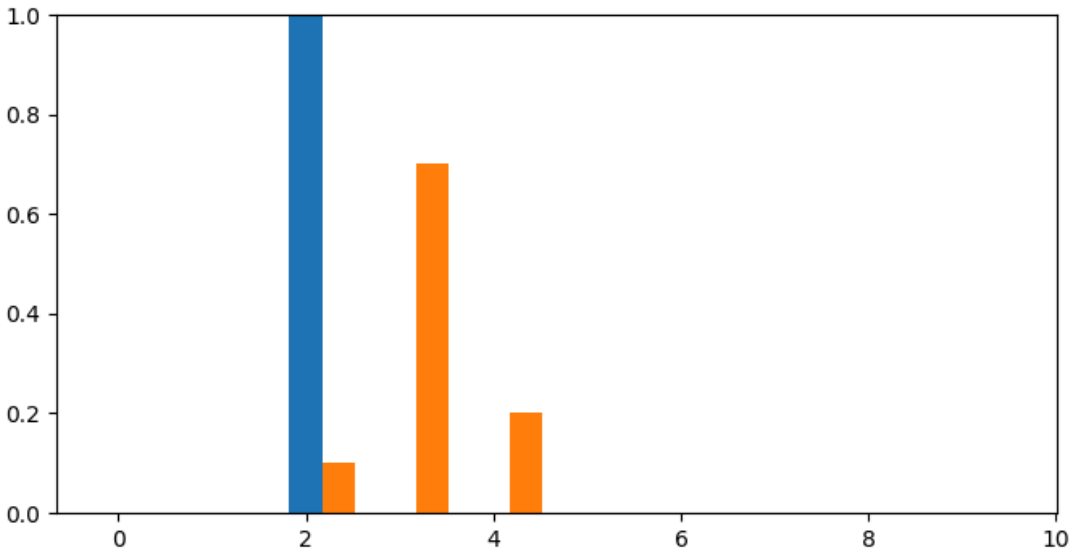
Allerdings ist es völlig unrealistisch, dass sich unser System perfekt verhält. Strömungen und Dynamikeinflüsse generieren situative Verstärkungen oder Abschwächungen der Geschwindigkeit.

Zunächst berücksichtigen nur die beiden benachbarten Felder und ein perfektes apriori Wissen. Durch die Strömung bedingt verlässt unser Roboter die Position 2 trotz einer Beschleunigung stromab nicht, möglicherweise bewegt er sich aber auch zwei Segmente weiter.

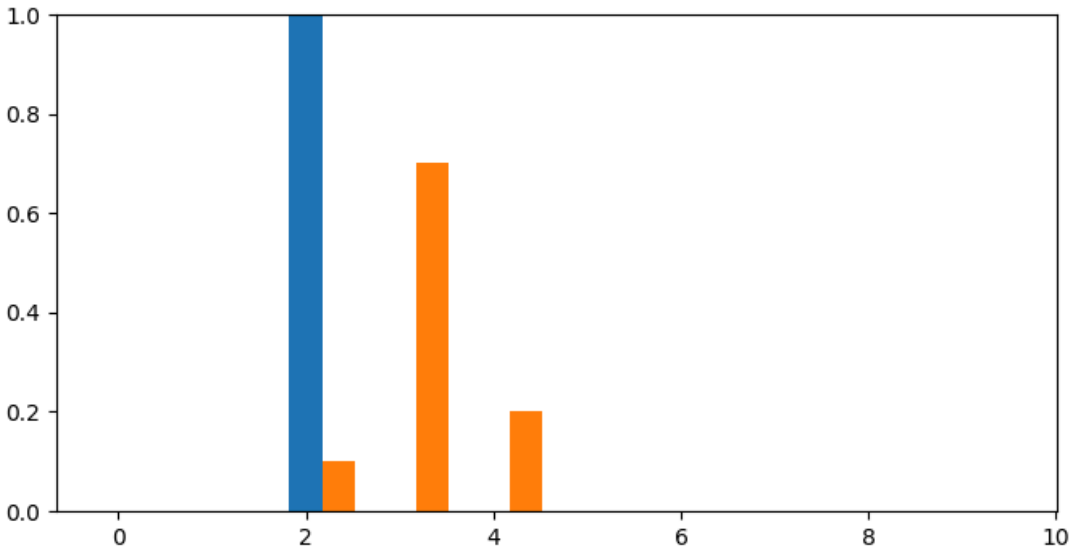
uncertainmovements.py

```python
import numpy as np
import matplotlib.pyplot as plt

def predict_move(belief, move, p_under, p_correct, p_over):
    n = len(belief)
    prior = np.zeros(n)
    for i in range(n):
        prior[i] = (
            belief[(i-move) % n]   * p_correct +
            belief[(i-move-1) % n] * p_over +
            belief[(i-move+1) % n] * p_under)
    return prior

belief = [0., 0., 1. , 0. , 0., 0., 0., 0., 0., 0.]
#belief = [0., 0.,  .4,  .6, 0., 0., 0., 0., 0., 0.]
result = predict_move(belief, 1, .1, .7, .2)

fig, ax = plt.subplots(figsize=(8,4))
width = 0.35
ax.bar(np.arange(len(belief)), belief, width)
ax.bar(np.arange(len(result)) + width, result, width)
plt.ylim(0, 1)
#plt.show()
plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript
```

foo.png



foo.png



Was aber geschieht, wenn wir von einem unsicheren priori Wissen ausgehen?

```
//          0   1    2   3   4   5   6   7   8   9
belief = [0., 0., .4, .6,  0., 0., 0., 0., 0., 0.]
```

Wie groß ist die Wahrscheinlichkeit, dass wir Segment x erreichen? Dabei berücksichtigen wir nun die verschiedenen Startpunkte, in dem wir die Abbildungsfunktion von u auf alle Kombinationen anwenden.

Segment	Rechnung	Ergebnis
0		nicht erreichbar
1		nicht erreichbar
2	$0.4 \cdot 0.1$	0.04
3	$0.4 \cdot 0.7 + 0.6 \cdot 0.1$	0.34
4	$0.4 \cdot 0.2 + 0.6 \cdot 0.7$	0.5
5	$0.6 \cdot 0.2$	0.12
6		nicht erreichbar
7		...

Grafisch dargestellt ergibt sich damit folgendes Bild:

```
             |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9  |

    priori      0     0    0.4   0.6    0     0     0     0     0     0

Reihenfolge   [0.2   0.7   0.1]  An die Stelle i=2 kann ich von i=0 (p=0.2)
invertiert                0.04  und i=1 (p=0.7) gelangen.
                          ====

                     [0.2   0.8   0.1]
                                0.34
                                ====

                           [0.2   0.8   0.1]
                                        0.5
                                        ====
```

Abbildung des Streckenmodels (O = Orange Warnschilder am Ufer, S = Starke Strömung, B = Brücken)

Damit bilden wir eine diskrete Faltungsoperation über unserer Aufenthaltswahrscheinlichkeit ab. Die Defintion der Faltung ergibt sich zu

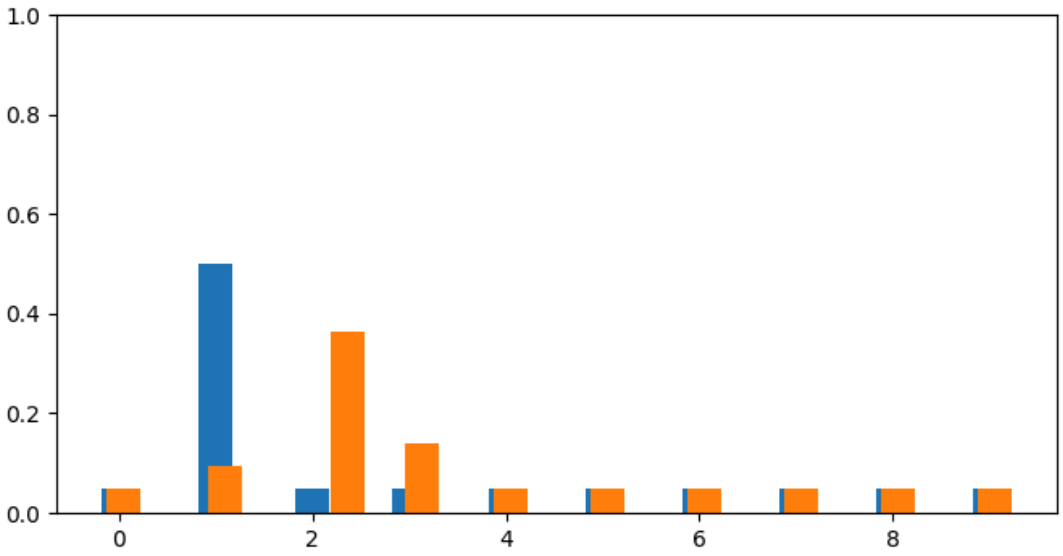
$$(f * g)(n) = \sum_{k \in D} f(k) g(n - k)$$

Wenn wir die Rechung also verallgemeinern können wir auf eine bestehende Implementierung zurückgreifen. Die `scipy` Bibliothek hält eine Funktion `convolve` bereit Link

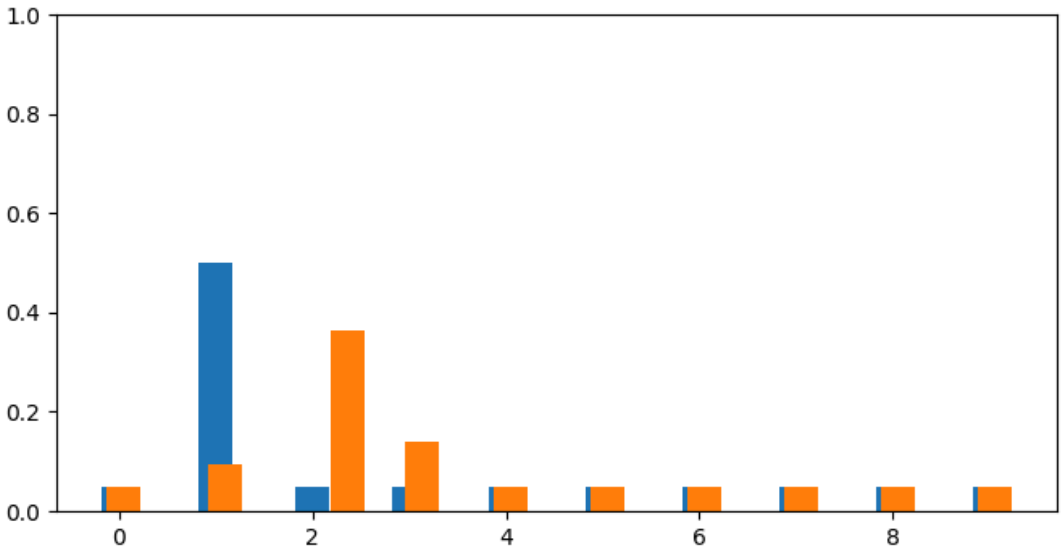
uncertainmovements.py

```
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt

belief = [.05, .5, .05, .05, .05, .05, .05, .05, .05, .05]
kernel = [.1, 0.7, 0.2]
prior = ndimage.convolve(np.roll(belief, int(len(kernel) / 2)), kerne
  mode='wrap')

#belief = prior   # Multiple movements
#prior = ndimage.convolve(np.roll(belief, int(len(kernel) / 2)), kern
  mode='wrap')

fig, ax = plt.subplots(figsize=(8,4))
width = 0.35
ax.bar(np.arange(len(belief)), belief, width)
ax.bar(np.arange(len(prior)) + prior, prior, width)
plt.ylim(0, 1)
#plt.show()
plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript
```

foo.png



foo.png

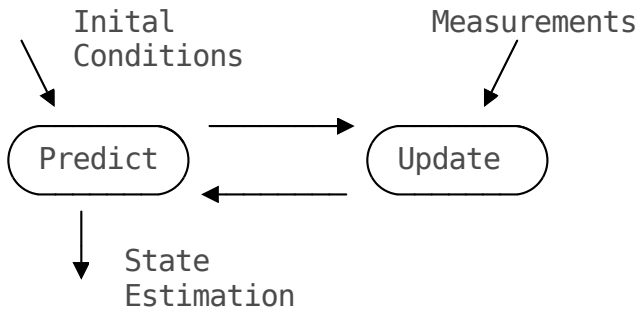


Mit jedem Prädiktionsschritt fächert die Breite der Unsicherheit entsprechend auf.

Und jetzt alles zusammen

Fassen wir nun beide Aspekte, die Vorhersage des Systemverhaltens und die Korrektur anhand der Messdaten zusammen. Aus dieser Konstallation wird deutlich, dass wir einen iterativen Prozess realisieren, in dessen Ablauf eine Zustandsvariable, hier unser Positionsindex "verfolgt" wird.

Zustand	Bedeutung
Initialisierung	Definition einer Anfangsschätzung
Vorhersage	Auf der Grundlage des unsicheren Systemverhaltens den Zustand für den nächsten Zeitschritt vorhersagen
Aktualisierung	Eintreffen einer Messung, Abbildung auf die Wahrscheinlichkeit eines Systemzustandes

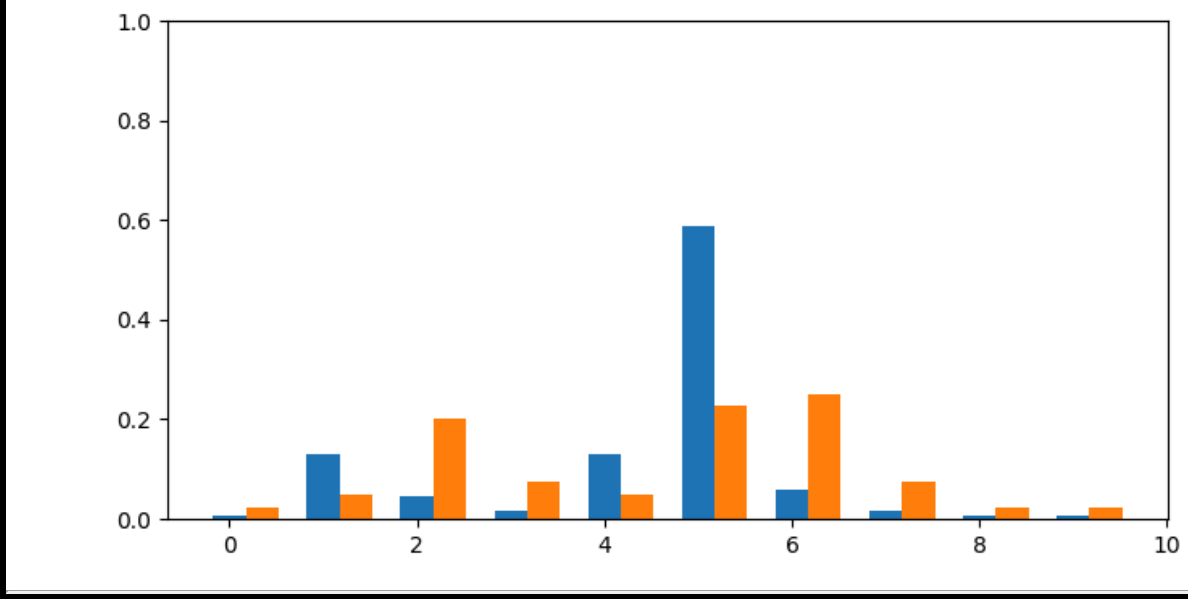


Das folgende Codefragment bildet zwei Iterationen für unser Beispiel ab. Im ersten Durchlauf ändert die Prediktionsphase den intertialen Wissensstand nicht. Die Faltung des Kernels ändert die Aufenthaltwahrscheinlichkeit nicht. Eine Präzisierung erfährt diese mit der ersten Messung durch den Schildersensor.

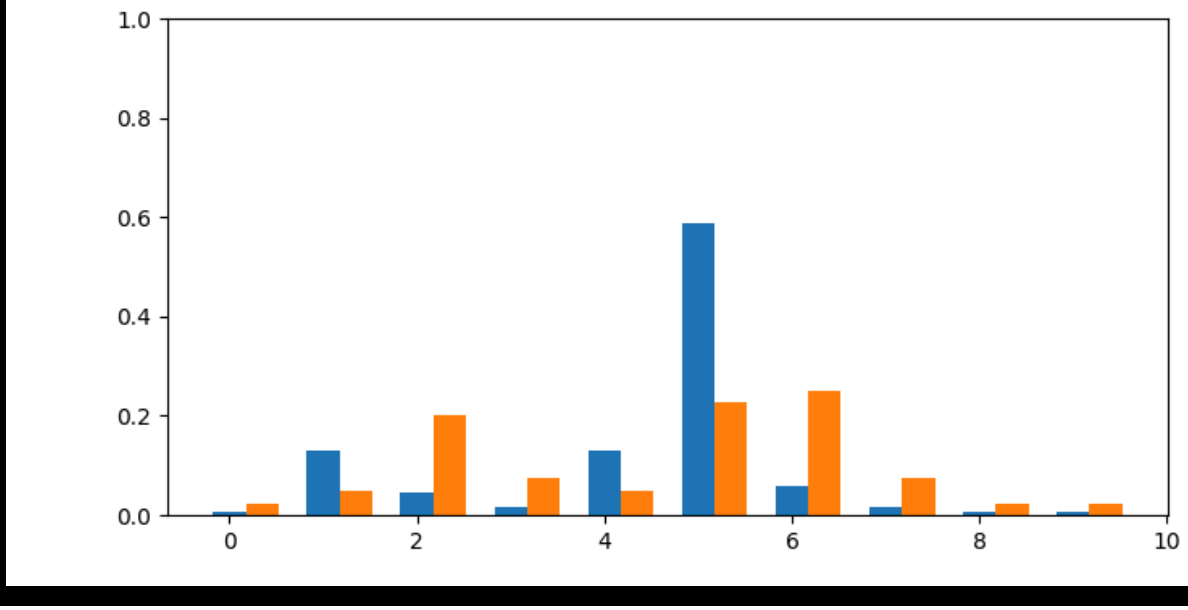
BayesFilter.py

```
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt

priori = np.array([1./10]*10)
kernel = [.1, 0.7, 0.2]
markers = np.array([0, 1., 0, 0, 1., 1., 0, 0, 0, 0])
truePositive = 0.83
belief = markers / sum(markers)*truePositive
belief[markers == 0] = (1-truePositive)/np.count_nonzero(markers==0)

# FIRST LOOP - prediction
posteriori = ndimage.convolve(np.roll(priori, int(len(kernel) / 2)),
  , mode='wrap')
# FIRST LOOP - update
priori = posteriori * belief / sum(posteriori * belief )

# SECOND LOOP - prediction
posteriori = ndimage.convolve(np.roll(priori, int(len(kernel) / 2)),
  , mode='wrap')
# FIRST LOOP - update
priori = posteriori * belief / sum(posteriori * belief )

fig, ax = plt.subplots(figsize=(8,4))
width = 0.35
ax.bar(np.arange(len(priori)), priori, width)
ax.bar(np.arange(len(posteriori)) + width, posteriori, width)
plt.ylim(0, 1)
#plt.show()
plt.savefig('foo.png') # notwendig für die Ausgabe in LiaScript
```

foo.png

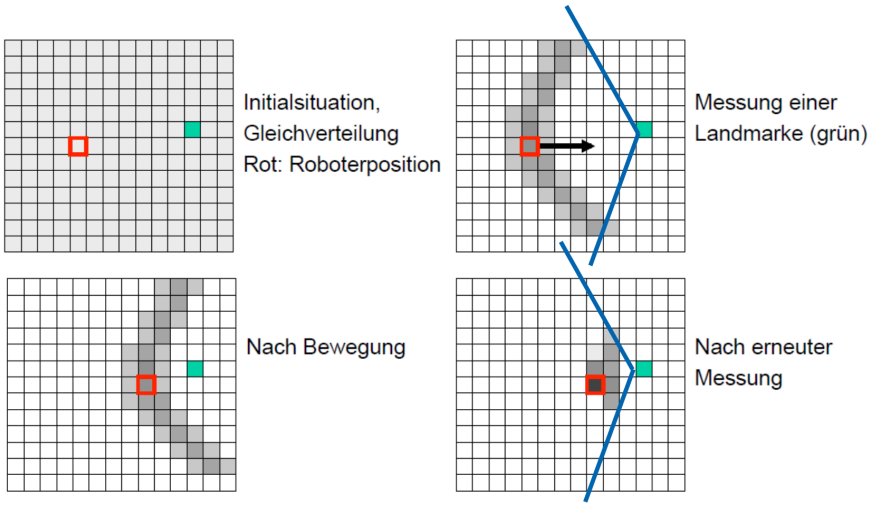


foo.png



Welche Einschränkungen sehen Sie in dem Beispiel?

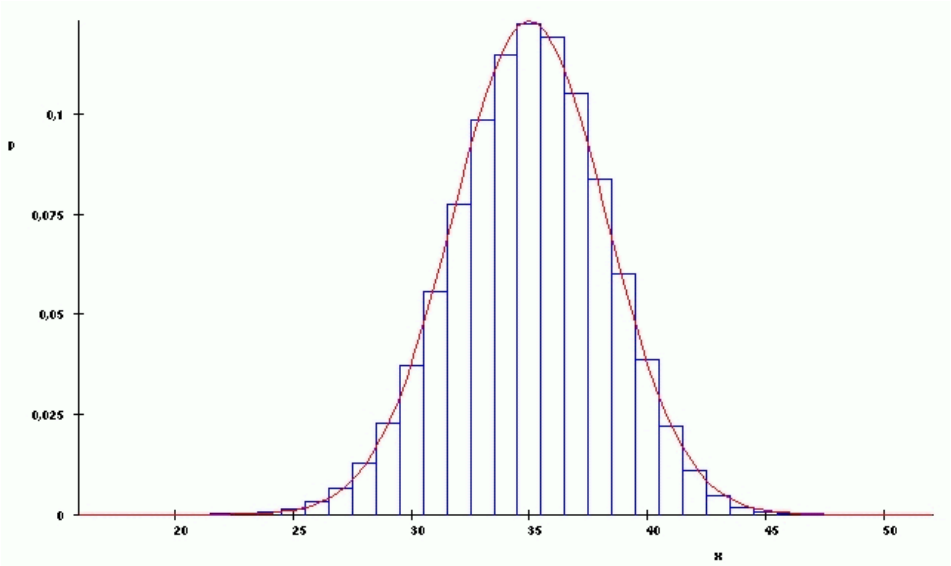
- Wir bilden nur eine Zustandsvariable ab. Bereits die Umsetzung eines 2D oder 3D Beispiels würde eine erhebliche Anpassung notwendig machen. Damit würde dann aber auch die Komplexität und die Berechnungsdauer entsprechend ansteigen.



- Diese Überlegung ist in starkem Maße mit der Frage nach der Auflösung unserer Diskretisierung verbunden. Ein 100x100m große Fabrikhallen, die mit einem 10cm Grid überzogen werden soll, bedeutet, dass wir jeweils 1 Million Kacheln evaluieren müssen.
- Die Abbildung der Sensorunsicherheit ist hier stark vereinfacht. Das Fehlermodell allein auf die Klassifikationsgüte abzubilden genügt in der Regel nicht. Die Ausgabe unseres Kamerasystems wird als statt einer konstanten Abbildungsfunktion der Messungen auf die Zustände eher ein variables Qualitätsattribut realisieren.
- Die Modalität des Sensors wurde so gewählt, dass dessen Daten einfach zu integrieren sind. Wie würden Sie die Informationen des Beschleunigungssensors berücksichtigen?

Übertragung auf kontinuierliche Systeme

- Im Allgemeinen ist der Zustandsraum so groß, dass der Bayesscher Filter-Algorithmus nicht direkt implementiert werden kann.
- Wahrscheinlichkeiten können durch Wahrscheinlichkeitsgitter approximiert werden (grid-Verfahren). Gitter können dann aber immer noch unpraktikabel groß werden.
- Falls Wahrscheinlichkeiten als Normalverteilungen angenommen werden, dann ergibt sich der Kalman-Filter.



Merke: Alle Größen sind normalverteilt: Vorteil: Radikal vereinfachte Berechnung!

Kalman-Filter Grundlagen

Der Kalman-Filter basiert auf zwei grundsätzlichen Gleichungen, die das Systemverhalten beschreiben - der Systemgleichung und der Messgleichung.

$$x_t = a \cdot x_{t-1} + b \cdot u_t + \epsilon$$
$$z_t = c \cdot x_t + \delta$$

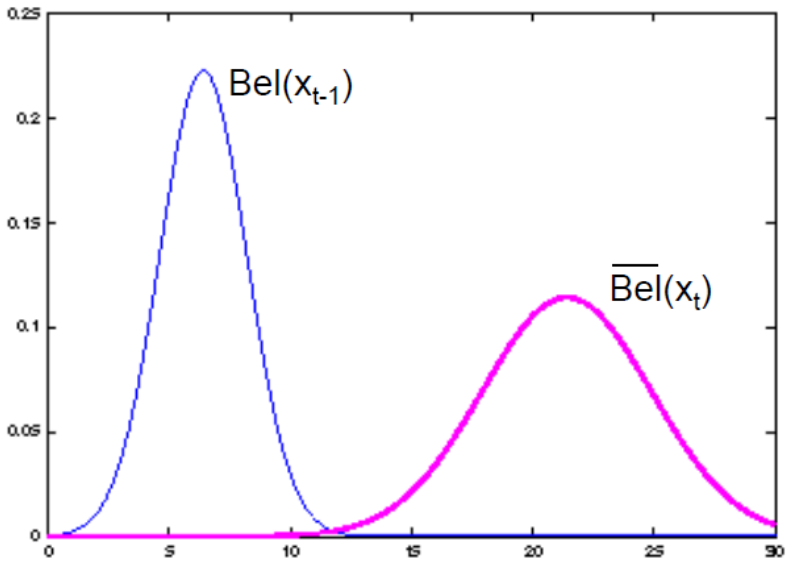
Zustand x_t, Steuerbefehl u_t und Sensorwert z_t sind eindimensional.

- a repräsentiert die Transformationsbedingung des Zustandes und b die Eingabematrix
- ϵ und δ sind Weißes Gauß'sches Rauschen (d.h. normalverteilt mit Mittelwert 0)

Kalman-Filter als Prozess

1. Vorhersage-Schritt (prediction step):

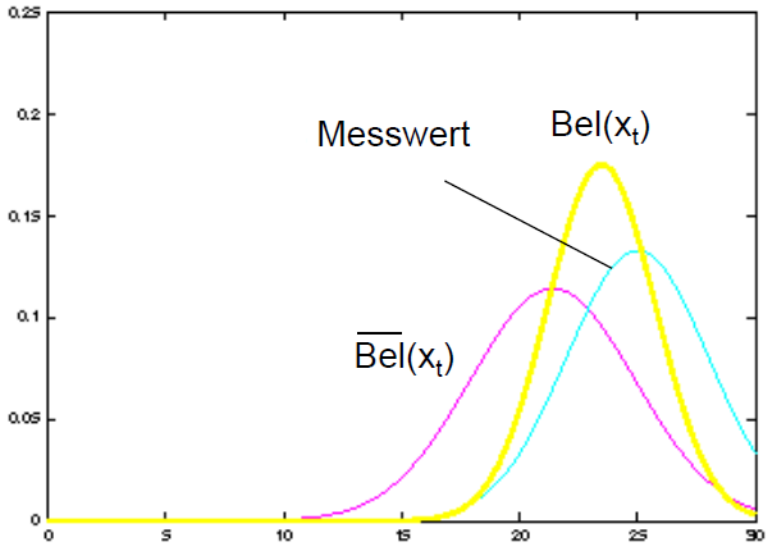
Mit Hilfe eines Systemmodells (z.B. Bewegungsmodell) wird aus dem alten Zustand x_{t-1} und einem Steuerbefehl u_t der neue Zustand x_t geschätzt.



$$\overline{\mu_t} = a\mu_{t-1} + bu_t$$
$$\overline{\sigma_t^2} = a\sigma_{t-1}^2 + \sigma_\epsilon^2$$

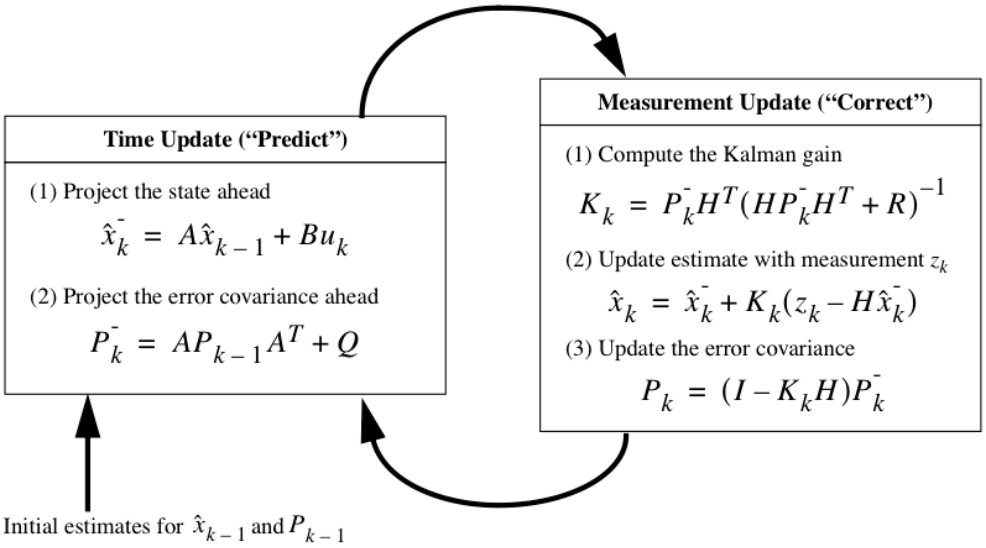
2. Korrektur-Schritt (correction step; measurement update):

Aus dem vorhergesagtem Zustand μ_t und der Messgleichung lässt sich ein Messwert $c\mu_t$ schätzen. Aus der Differenz des tatsächlichen Messwertes z_t und des geschätzten Messwerts lässt sich Zustand $\overline{\mu_t}$ korrigieren:



$$\mu_t = \overline{\mu_t} + k_t(z_t - c\mu_t)$$
$$k_t = \frac{c^2\overline{\sigma_t^2}}{c^2\overline{\sigma_t^2} + \sigma_\delta^2}$$

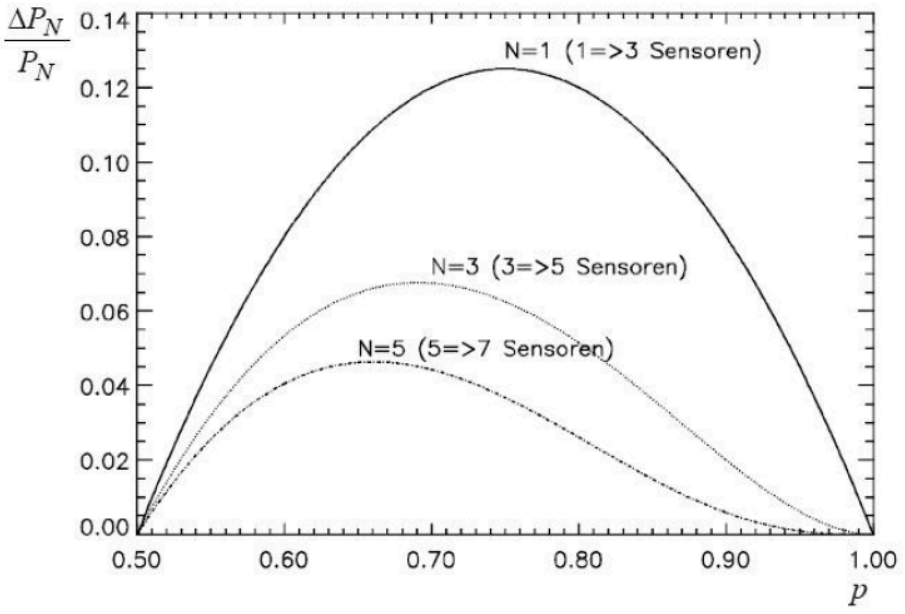
Wenn wir den Ablauf auf ein n-dimensionales Problem anwenden, ergibt sich folgendes Standardschaubild zum Ablauf.



S. Maybeck, Stochastic models, estimation, and control, 1977

Ausblick

Merke: Datenfusion generiert einen erheblichen Aufwand und erfordert Annahmen zur Umgebung des Systems. Gleichzeitig ist sie kein Garant für ein funktionsfähiges System!



Nahin, John L., Can Two Plus Two Equal Five? 1980

Aufgaben

- Variieren Sie die Parameter der Beispiele der Implementierung und evaluieren Sie den Einfluss unterschiedlicher Bewegungsmodelle