SENSOR FUSION

Signal and Systems Matlab Toolbox

Sensor Fusion

Fredrik Gustafsson
fredrik.gustafsson@liu.se

Gustaf Hendeby
gustaf.hendeby@liu.se

Linköping University

Background

Goals

1. Reproducible examples in theory and exercise books
2. Possibility to vary parameters in the examples
3. Possibility to extrapolate to similar use cases

It is really not a general purpose toolbox that covers all problems that can occur in sensor fusion.
However, many applications can be recast into the standard framework covered by the toolbox.

Model framework

- Sensor fusion as pattern recognition: rewrite problem into standard form

$$
\begin{aligned}
x(t+1) &= f(t, x(t), u(t), v(t); \theta), & v(t) &\in p_v(v) \\
y(t) &= h(t, x(t), u(t), e(t); \theta), & e(t) &\in p_e(e).
\end{aligned}
$$

- Toolbox use is also pattern recognition, with restriction to additive noises

$$
\begin{aligned}
x(t+1) &= f(t, x(t), u(t); \theta) + v(t), & v(t) &\in p_v(v) \\
y(t) &= h(t, x(t), u(t); \theta) + e(t), & e(t) &\in p_e(e).
\end{aligned}
$$

- Model specified by
  - the functions f, h,
  - the dimensions of the signals in alphabetical order (and order of appearance) $n = [n_x, n_u, n_y, n_\theta]$,
  - and the noise distributions p_v, p_e which are default Gaussian.

Main ideas

- The use of classes to define objects
  - `sensor` for sensor models and sensor networks $y(t) = h(t, x(t), u(t); \theta) + e(t)$
  - `nl`, `lss` for motion models with or without sensor $x(t+1) = f(t, x(t), u(t); \theta) + v(t)$
  - `pdfclass` for distributions such as p_v, p_e
  - `sig` for signal objects $\big(y(t), x(t), u(t)\big)$
- Try to reuse a few standard methods, *e.g.* `disp`, `simulate`, `ls/wls/ml`, `plot/xplot/xplot2`
- Plenty of standard models pre-defined in *e.g.* `exsensor`, `exnl`, `exmotion`

Installation

- Download the zip-file from `https://www.control.isy.liu.se/student/tsrt14/`
- Unzip the file structure and save it to a convenient folder
- Place Matlab in the same folder and run `initSigSys` to set the path to all subfolders

Sensor models

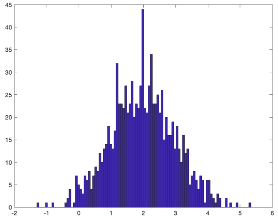
General form: $y(t) = h(x, u, \theta) + e$

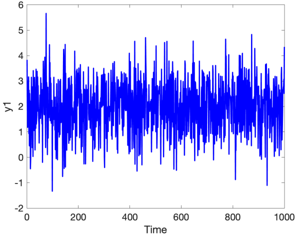
Dimensions: `n=[nx,nu,ny,nth]`

Used for simulation, calibration of θ, estimation of x or x, θ, and to create state space models

Simulation creates a SIG object

```
s=sensormod('x(1)^2+x(2)^2',[2 0 1 0]).      % Simplest
s=sensormod('x(1,:).^2+x(2,:).^2',[2 0 1 0])% Preferred
s.x0=[1;1];                  % Set x
data=simulate(s,1:3)         % Simulate y(1:3)
data.y                       % Display y(t)
s.pe=1                       % Set pe to N(0,1)
data=simulate(s,1:3)         % Simulate with noise
data.y
data=simulate(s,1:1000);     % Generate 1000 y
hist(data.y,100)             % Histogram
plot(data)                   % Plot y(t)
```

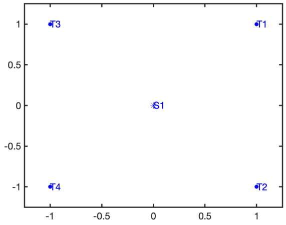


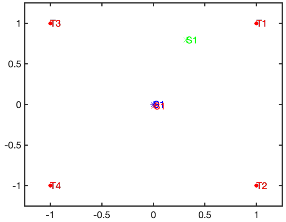


Sensor networks: calibration

Sensor networks is a special case of SENSORMOD.
The parameters θ used to represent sensor positions.
Many example networks in `exsensor(type,M,N)`:
TOA, TDOA, DOA, RSS, etc.
M targets and N sensors.

```
s=exsensor('toa',1,4);              % Standard TOA network
s.x0=[1 1 1 -1 -1 1 -1 -1];         % Set the 4 target pos
s.th=[0 0]; s.pe=0.001*eye(4);      % Sensor pos and dist
y=simulate(s,1);                    % Simulate one meas
s0=s; s0.th=s.th+0.5*randn(2,1);    % Perturb sensor pos
shat=calibrate(s0,y);               % Estimate sensor pos
plot(s,s0,shat)                     % Compare all three
```

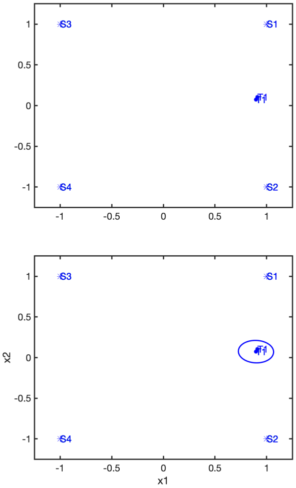




Sensor networks: estimation

Estimate x using `wls`, `ls`, `ml`.
Two outputs: SIG and SENSORMOD (incl. P_x, P_θ) objects.
The method `estimate` estimates any mix of x and θ

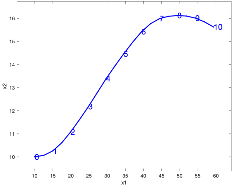
```
s=exsensor('toa',4,1)          % TOA now with 4 sensors
s.th=[1 1 1 -1 -1 1 -1 -1];    % Sensor positions
s.pe=0.01*eye(4);              % Sensor noise N(0,0.01)
y=simulate(s,1)                % Simulate one data point
[xhat,shat]=wls(s,y);          % Estimate target with WLS
                               % Output both SIG and SENSOR
plot(s,shat)                   % Plot both networks
hold on
xplot2(xhat,'conf',90)         % SIG provides covariance
```

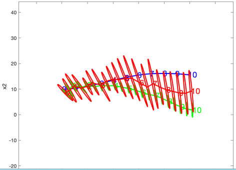


State space models

LSS denotes the class of linear models where the method `kalman` can be applied
NL is the more general class of nonlinear models
Many examples of motion and sensor models in `exnl`, `exsensor`
Motion models and sensor models can be mixed
Filters can be applied to simulated data e.g. `ekf`, `ukf`, `pf`, `crlb`

```
randn('state',2), rand('state',3)
fx=exmotion('ctcv2d')    % Motion model without sensor
hx=exsensor('radar',1)   % Radar sensor
ss=addsensor(fx,hx)      % Add sensor to motion model
yx=simulate(ss,10);      % Simulate 10 seconds
xplot2(yx)               % Plot state trajectory X,Y
xhat1=ekf(ss,yx);        % Apply EKF
ss.pv=2*ss.pv;           % Dithering noise since fx nonlin
ss.pe=2*ss.pe;           % Also hx nonlinear
xhat2=ekf(ss,yx);        % Second try
xplot2(yx,xhat1,xhat2,'conf',90)
axis('equal')            % Compare filters
```





Summary

- Sensor fusion has a very complex scope
- Strict model syntax

$$\begin{aligned} x(t+1) &= f(t, x(t), u(t); \theta) + v(t), & v(t) &\in p_v(v) \\ y(t) &= h(t, x(t), u(t); \theta) + e(t), & e(t) &\in p_e(e). \end{aligned}$$

- Start with working examples from `help`, book, exercises or manual to save time debugging syntax
- The goal with the toolbox is to get problem intuition
- Write your own code is also an important part of the learning process