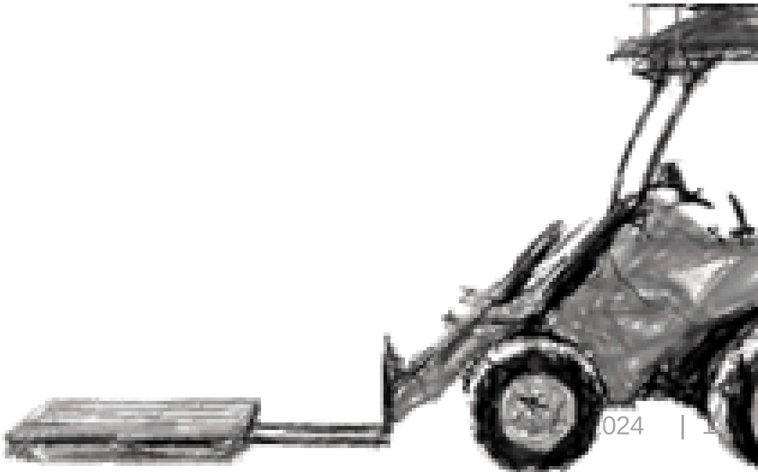
Tampere University

Introduction to and review of some general concepts

Reza Ghabcheloo

AUT 710

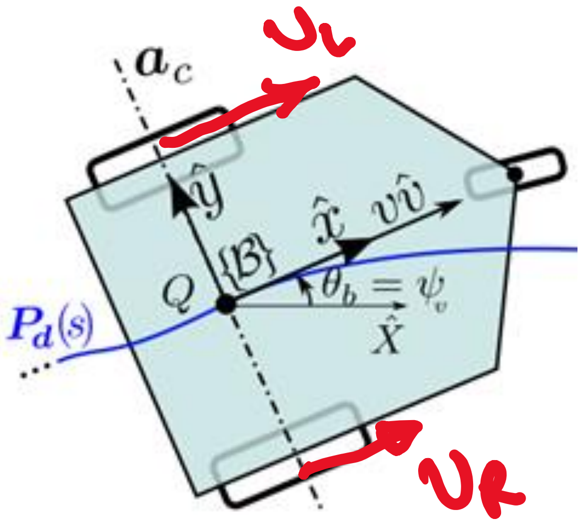


Concepts

- Coordinate frames
- Kinematics modelling
- Dynamics systems and state space
- **Probabilities and distributions**
- **Map, world model**

week 2

Relating body speeds to wheel speed



$$\dot{x}_B = v_{xB} \cos \psi$$
$$\dot{y}_B = v_{xB} \sin \psi$$
$$\dot{\psi} = \omega_{zB}$$

relation between body frame speeds (v_x, ω_z) and wheel angular speeds (w_R, w_L)

$$v_x = \frac{1}{2}(v_R + v_L)$$

$$\omega_z = \frac{1}{d}(v_R - v_L)$$

$$v_R = r w_R, v_L = r w_L$$

where r is the radius of the wheels, and d distance between wheels, and w_R and w_L are rotational speed of the wheels.

Odometry / dead-reckoning

Odometry / egomotion estimation/ dead-recking

Motion model:

$$\dot{x} = v\cos\psi$$
$$\dot{y} = v\sin\psi$$
$$\dot{\psi} = \omega$$

Odometry is the process of integration of above motion model given the speeds.

If the speeds come from wheels, it is called wheel odometry.

You could also have visual odometry.

Next we will discretize and rewrite them is some convenient form

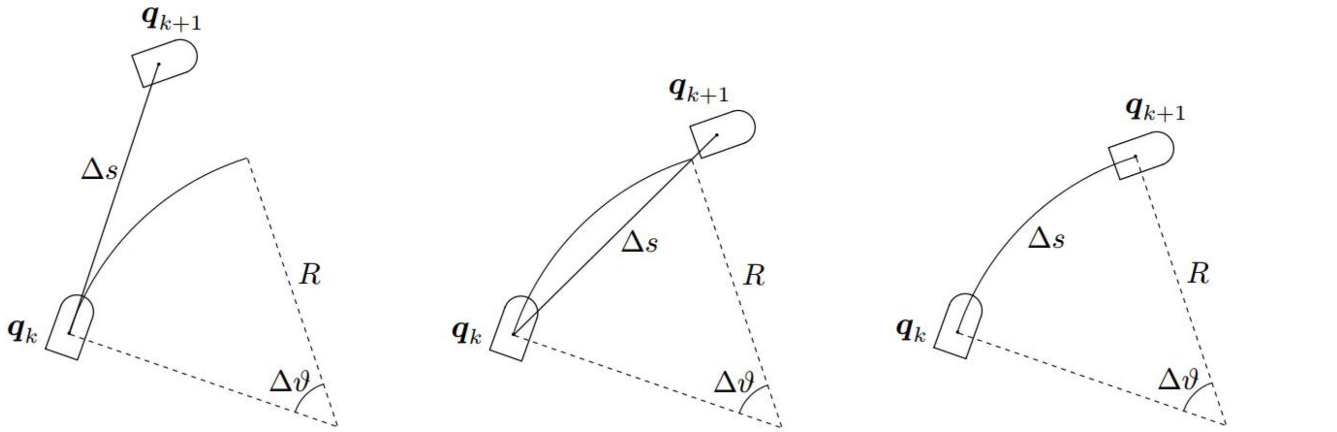


Fig. 11.21. Odometric localization for a unicycle moving along an elementary tract corresponding to an arc of circle; *left*: integration via Euler method, *centre*: integration via Runge–Kutta method, *right*: exact integration

Siciliano et al Motion planning and control, pp 514

Calculating odometry increment parameters from Sensor inputs

Sensor input:

- We start with wheel rotational speed (w_R, w_L); we then convert them to
- body linear and angular speeds $(v, \omega)^T$; we then again convert them to
- Translation and rotation increments $(\delta_{trans}, \delta_{rot})$. In simple models: $\delta_{trans} = v\Delta t$ and $\delta_{rot} = \omega\Delta t$
  - δ_{trans}: how many meters the robot moves forward in Δt second
  - δ_{rot}: how many radians the robot orientation changes in Δt second
- Here we assume (w_R, w_L) are **measured using wheel encoders** or we can also see then as **control inputs** send to the robot wheels to rotate with certain angular rate.

Odometry discrete model

Discrete model

$$x_t = x_{t-1} + v\Delta t\cos\psi_{\bar{t}}$$
$$y_t = y_{t-1} + v\Delta t\sin\psi_{\bar{t}}$$
$$\psi_t = \psi_{t-1} + \omega\Delta t$$

where Δt is the discretization time

Forward Euler method: $\psi_{\bar{t}} = \psi_{t-1}$

we use the current heading to predict the next position

Euler midpoint method: $\psi_{\bar{t}} = \psi_{t-1} + \frac{1}{2}\omega\Delta t$

we use the average of current and next heading to predict the next position

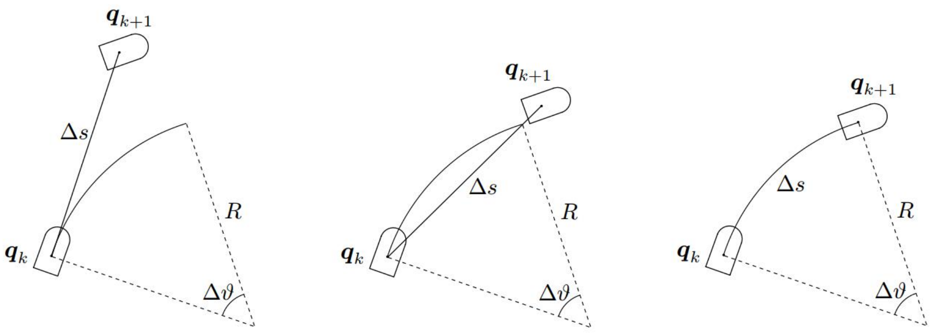


Fig. 11.21. Odometric localization for a unicycle moving along an elementary tract corresponding to an arc of circle; *left*: integration via Euler method, *centre*: integration via Runge–Kutta method, *right*: exact integration

Tampere University

Odometry model

- Euler integral approximation, midpoint method

$$x_t = x_{t-1} + v\Delta t \cos\left(\psi_{t-1} + \frac{1}{2}\omega\Delta t\right)$$

$$y_t = y_{t-1} + v\Delta t \sin\left(\psi_{t-1} + \frac{1}{2}\omega\Delta t\right)$$

$$\psi_t = \psi_{t-1} + \omega\Delta t$$

- Replacing the odometry increment parameters to get

$$x_t = x_{t-1} + \delta_{trans}\cos(\psi_{t-1}+\delta_{rot1})$$

$$y_t = y_{t-1} + \delta_{trans}\sin(\psi_{t-1}+\delta_{rot1})$$

$$\psi_t = \psi_{t-1} + \delta_{rot1} + \delta_{rot2}$$

For the purpose of this slide, you can assume $\delta_{rot1} = \delta_{rot2} = \frac{1}{2}\omega\Delta t$

We will use $\delta_{trans}, \delta_{rot1}$ and δ_{rot2} to estimate robot pose, but they include uncertainties

Uncertainty

Things that will create uncertainties in our system

- Environment
- Sensors
- Actuators
- Models
- Computation

- Probabilities are one of the most powerful tools to model uncertainties.
- Ability to cope with uncertainties is critical for successful robots
- I will follow *Probabilistic Robotics* book for this section on Uncertainty and probabilities.

Tampere University

Probabilities and distributions

Tampere University

Probabilities and distributions

- Let's start with random variable X, discrete space
- $P(X = x)$ is probablity of X taking on the value x (probability mass function)
- Dice example: $P(X = 1) = \cdots P(X = 6) = \frac{1}{6}$
- $\Sigma_x P(X = x) = 1$, $1 \geq P(X = x) \geq 0$
- For simplicity, we may omit X, and write $P(x)$
- We may draw a sample: $x \sim P(x)$

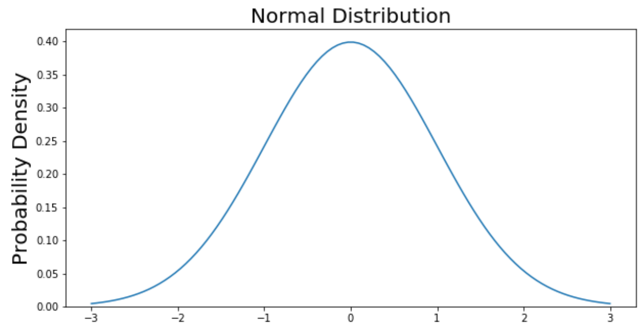
- Now, let's X be a random variable in continuous space
- They are described by probability density functions (PDFs)
- $\int p(x)dx = 1 , p(x) \geq 0$, however $p(x)$ is not buonded above by 1
- A Normal or Gaussian distribution, defined by density function $p(x) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\}$ for one dimensional variable x. See https://se.mathworks.com/help/stats/normal-distribution.html

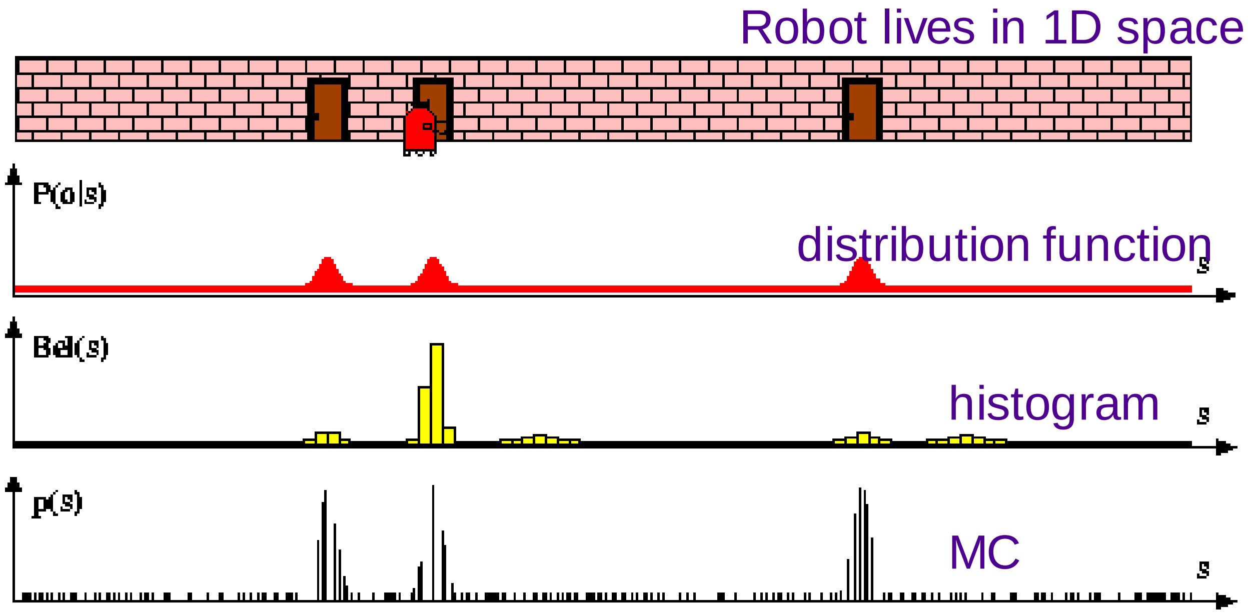
Robot pose

- We use probability distribution to describe robot pose.
- Distribution functions can be described/approximated by
  - Parameterized: Normal distribution (μ, σ^2)
  - Discretized: Histogram
  - Monte Carlo sampling: Particle filter
- Pose is a multi-dimensional vector $\mathrm{x} = (x, y, \psi)$ Normal distributions over vectors is called *multivariate*

$$p(\mathrm{x}) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp(-\frac{1}{2}(\mathrm{x}-\mu)^T\Sigma^{-1}(\mathrm{x}-\mu))$$

μ, Σ: the mean and the covariance matrix





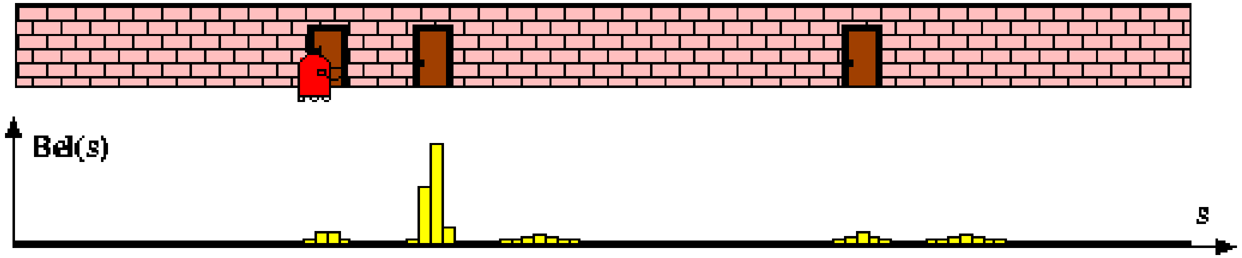
Some properties 1/4

- Joint probability $p(x, y) = p(X = x \text{ and } Y = y)$ $\quad p(RED = 5 \text{ and } GREEN = 3)$
- Conditional probability $p(x|y) = p(X = x|Y = y)$ $\quad p(RED = 5|GREEN = 3) = p(RED = 5)$
- $p(x, y) = p(x|y)p(y)$
- If X and Y are independent, then $p(x, y) = p(x)p(y)$, and $p(x|y) = p(x)$

<u>Robot localization on 1D world example</u>: X could be S (the state of the robot) and Y could be observation O (seeing a door). S can take values in $s \in \{1,2,\ldots 50\}$ and O can take values in {yes, no}.
So, if initially P(S = s)=1/50 for all values of s, after observation, we can ask / calculate P(S=15 | O=yes) and for the matter fact we can ask for all the values of s. We need one more thing to actually calculate this. Later on that.

You can ask P(S=15 and O=yes)

Obviously, S and O are not independent.



The values in the figure do not represent the example

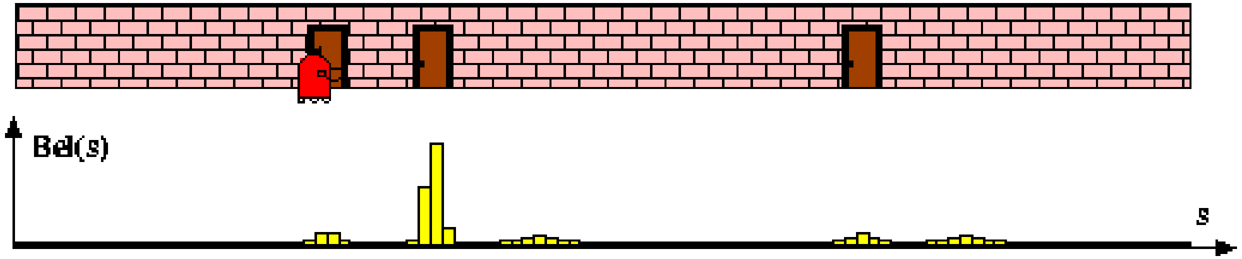
Some properties 2/4

- Joint probability $p(x, y) = p(X = x \text{ and } Y = y)$ $\quad p(RED = 5 \text{ and } GREEN = 3)$
- Conditional probability $p(x|y) = p(X = x|Y = y)$ $\quad p(RED = 5|GREEN = 3) = p(RED = 5)$,
- $p(x, y) = p(x|y)p(y)$;
- **Theory of total probability**
  - $P(x) = \Sigma_y P(x|y)P(y)$ the dice example is trivial: $\Sigma_y P(x|y)P(y)$=1/6*1/6 + 1/6*1/6 … = 1/6
  - $p(x) = \int p(x|y)p(y)\mathrm{d}y$
- Back to robot localization on 1D world example (this is not where we use total probability in localization, this is just for the sake of example): You can ask, what is $P(S = 15)$ after making an observation.
- You need to calculate $P(S = 15|O = o)$ for all o's, that is

$P(S = 15|O = yes)$ and $P(S = 15|O = no)$ and

You need to know the sensor model $P(O = o)$

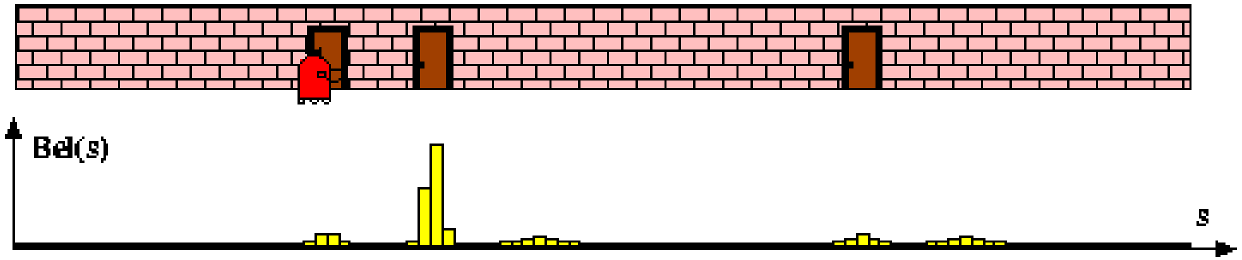
$$P(S = 15) = P(S = 15|O = yes)P(\mathrm{O} = \mathrm{yes}) + P(S = 15|O = no)P(\mathrm{O} = \mathrm{no})$$



The values in the figure do not represent the example

Some properties 3/4

- **Bayes rule** (relates $p(x|y)$ to its ”inverse”)
  - $p(x|y)= \frac{p(y|x)p(x)}{p(y)}$,
- Back to robot localization on 1D world example: Our sensor models are typically $p(O = o|S = s)$, the probability of seeing o if you are at state s. Bayes rule tells us how to caluculate $p(S = s|O = o)$ the probability of our state (our believe about the whereabout of our robot) after we have observed o (for example: O=yes)
- When we want to infer quantity x from y, then $p(x)$ is called prior probability, and y the data. For example, x is robot position, and y is the sensor measurement. $p(x|y)$ is called posterior probablity over X.
- belief distributions are posterior probabilities over robot/environment state conditioned on the available data $bel(x_t) = p(x_t|z_t, u_t)$, typically
  - z_t are sensor measurement
  - u_t are control inputs
  - you might also have other data (map of walls)



The values in the figure do not represent the example

Some properties 4/4

- Expectation
  - $E[X] = \Sigma_x x P(x)$ $E[RED] = \Sigma_x x P(x) =$ 1*1/6 + 2*1/6… = 3.5
  - $E[X] = \int xp(x)\mathrm{d}x$
- Covariance
  - $\mathrm{Cov}[X] = E\left[(X - E[X])^2\right]$ $\mathrm{Cov}[RED] = \Sigma_x (x - 3.5)^2 P(x) =$ 1*1/6 + 2*1/6… = 2.9
- $E[.]$ is a linear operator (a and b are not random variables)
  - $E[aX + b] = aE[X] + b$
- Bayes rule again
  - $p(x|y)= \frac{p(y|x)p(x)}{p(y)}$, where $p(y) = \Sigma_{x'} p(y|x')p(x')$
  - Since the deminator is not depend on x, we often write it as normalization variable, typically denoted η. That is, $p(x|y) = \eta\ p(y|x)p(x)$
  - η is calculated such that the resulting $p(x|y)$ sums up/integrates up to 1, instead of explicitly calculating $p(y) = \Sigma_{x'} p(y|x')p(x')$

Bayes filter

Measure and predict cycle:

- Initial belief $bel(x)$
- Predict (theory of total probability)

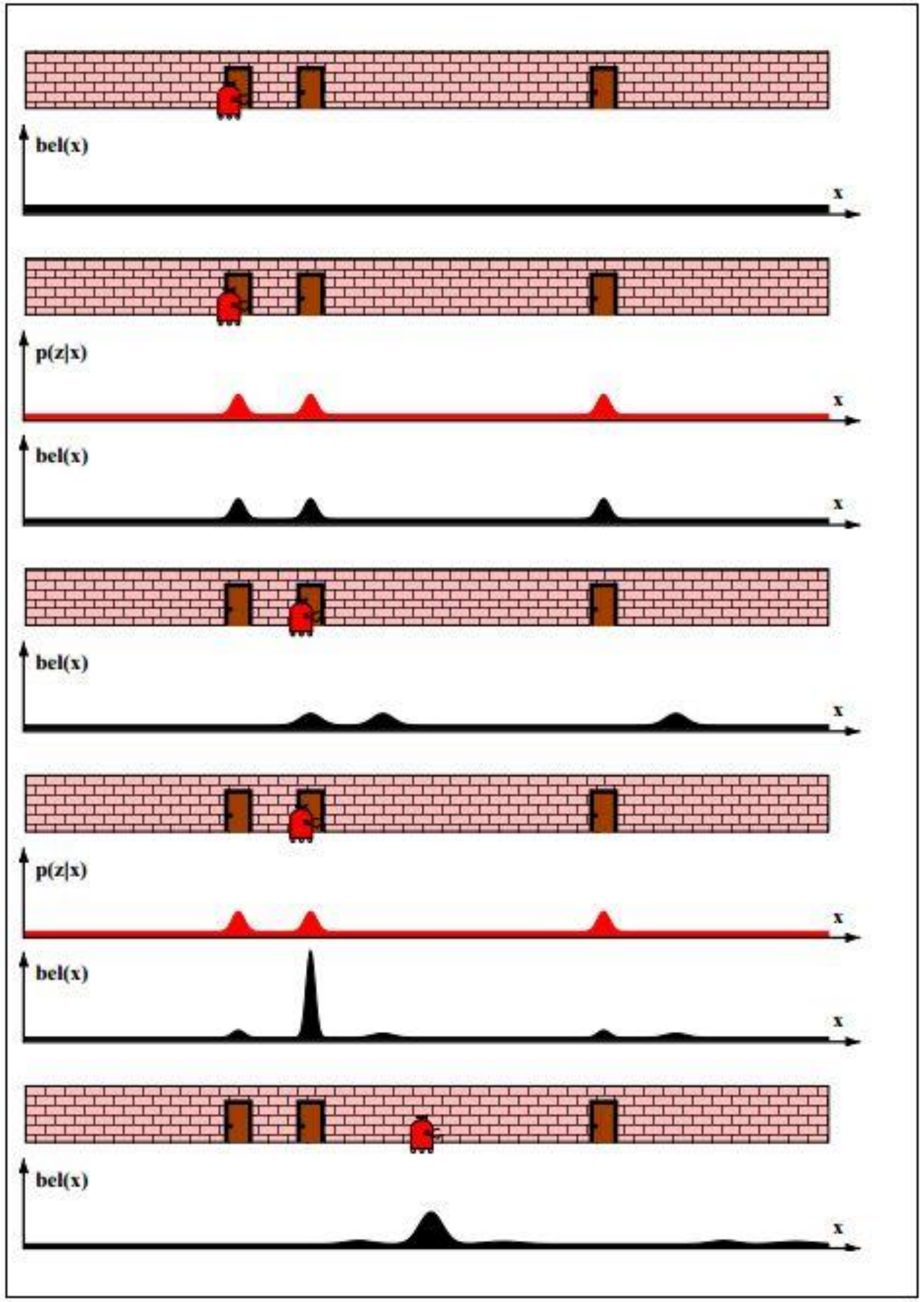
$$bel(x') = \sum_x p(x'|u,x)bel(x)$$

- Measurement (Bayes rule)

$$bel(x) = p(y|x')bel(x')$$

Uncertainty

- increases after prediction (dead-reckoning): bell function spread/flatten out
- decreases after measurement (information arrives)

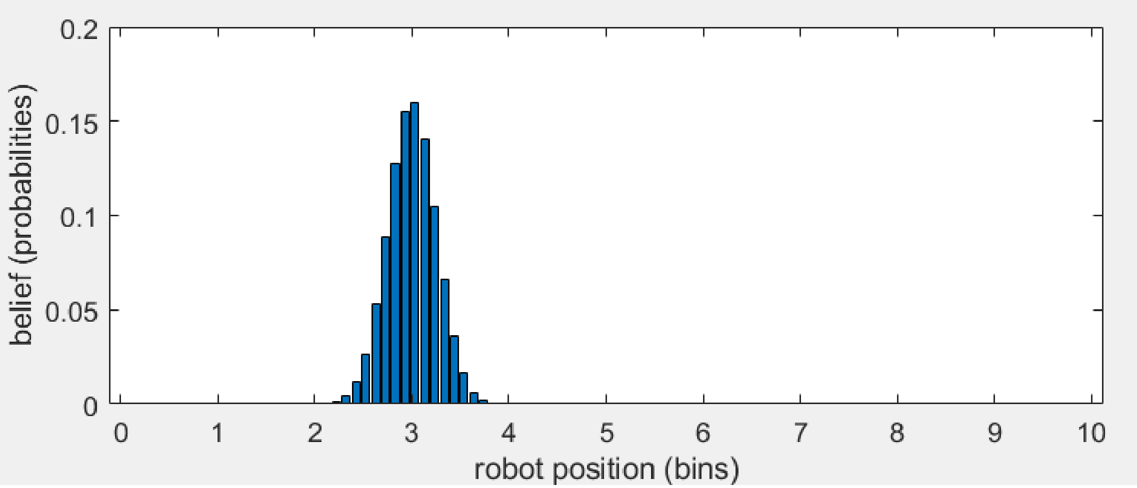


Numerical Example (overview)

Discrete case, Monte Carlo, Parametric distributions

(assistants will go through these in details)

Discrete case 1/3



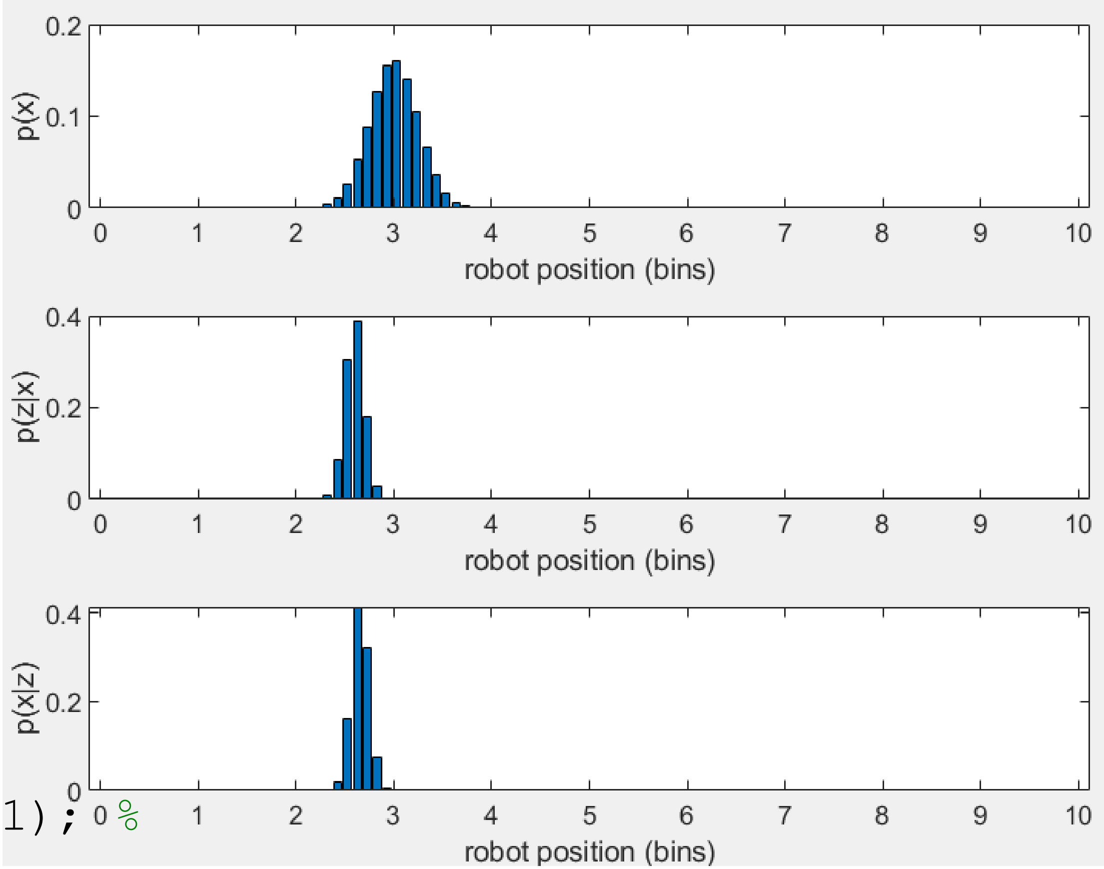

```
Robot lives in 1D space
x_min = 0; x_max = 10; %meter
x=linspace(x_min,x_max,100); % converted to
100 bins
pd_x = makedist('Normal','mu',3,'sigma',0.25);
% robot position belief
p_x = pdf(pd_x,x); % P(x)
eta = sum(p_x);
p_x = 1/eta*p_x; % now p sums up to 1
bar(x,p_x)

Ex = E(x,p_x)      % = 3.00
COVx = COV(x,p_x) % = 0.25^2
```

$$E[X] = \Sigma_x x P(x)$$
$$\mathrm{Cov}[X] = E\left[(X - E[X])^2\right]$$

```
function Ex = E(x,p)
Ex = sum(x.*p);
end

function COVx = COV(x,p)
Ex = E(x,p);
COVx = E((x-Ex).^2,p);
end
```

Discrete case 2/3

- Let's suppose robot has a sensor that can measure its distance to the left wall located at $x = 0$
- Sensor model: $z = x + noise$, where $noise \sim N(0{,}0.1^2)$ → $p(z|x) = N(x, 0.1^2)$
- Let's say the sensor measures $z = 2.6$
- Calculate $p(x|z) = \eta\, p(z|x)p(x)$
- Uncertainty of our believe about robot state decrease after measurement

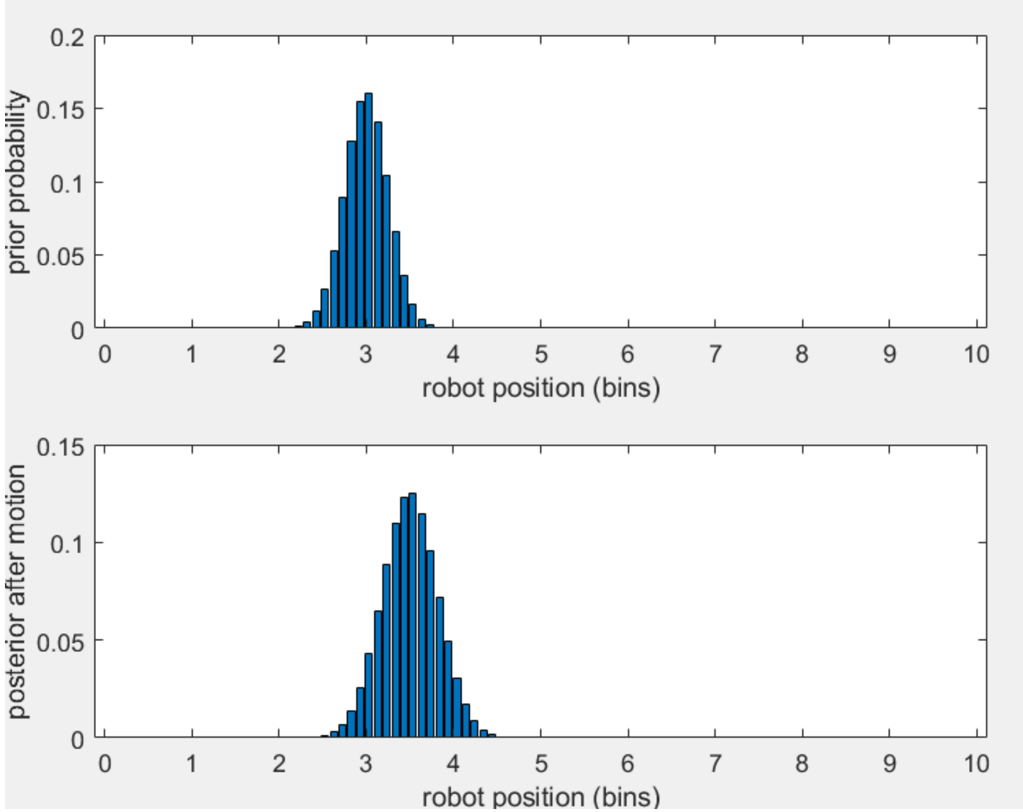



```
pd_noise = makedist('Normal','mu',0,'sigma',0.1); % sensor noise model

z_sensor = 2.6; % measurement value

pz_x = pdf(pd_noise, z_sensor - x); % p(z|x) for all x

px_z = pz_x.*p_x;

eta = sum(px_z);

px_z = 1/eta*px_z; % now p sums up to 1
```

Discrete case 3/3

- Let's suppose robot moves, and its motion is described by $x' = x + u + noise$, where $noise \sim N(0, 0.2^2)$. Then $p(x'|x,u) \sim N(x + u, 0.2^2)$, except at the boundries.
- Let's assume $u = 0.5$. Calculate belief after one step: $bel(x') = \Sigma_x p(x'|x,u) bel(x)$
- Uncertaitny of our believe about robot state increases after uncertain motion

```
pd_noise = makedist('Normal','mu',0,'sigma',0.2);
% sensor noise model
u = 0.5; % control action
for k = 1:100
    p_motion = pdf(pd_noise, x(k) - x - u); % p(x'|x,u)
    p_xpr(k) = sum(p_motion.*p_x);
end
eta = sum(p_xpr);
p_xpr = 1/eta*p_xpr; % now p sums up to 1
```



Note:
When `x(k) - x - u` is >10 or <0, we need to set `p_motion=0`.
This is neglected in the code, since the robot is sufficiently far from boundaries and Normal distribution `p_x` is almost zero there, and we will normalize our result at the end anyways.

Total probability says

$$bel(x') = \sum_x p(x'|u,x)\,bel(x)$$

Let's assume x,x' $\in S$. In our example, $S = \{0.0, 0.1, 0.2, \dots, 10.0\}$, and in the database they are indexed $k \in \{1,2,3, \dots, 100\}$. Notice then to calculate $bel(x')$ for all $x' \in S$, you need two for-loops, one for x one for x'. In other words, if you want to calculate the above equation for only one x', then you only need to loop over $x \in S$. The code below calculate $bel(x')$ for all the x', but one of the loops is explicit (line 4) and the other is implicit (line 6)

```
1  pd_noise=makedist('Normal','mu',0,'sigma',0.2);
2  % sensor noise model
3  u = 0.5; % control action
4  for k = 1:100
5   p_motion = pdf(pd_noise, x(k)-x-u);% p(x'|x,u)
6   p_xpr(k) = sum(p_motion.*p_x);
7  end
8  eta = sum(p_xpr);
9  p_xpr = 1/eta*p_xpr; % now p sums up to 1
```

In the code,

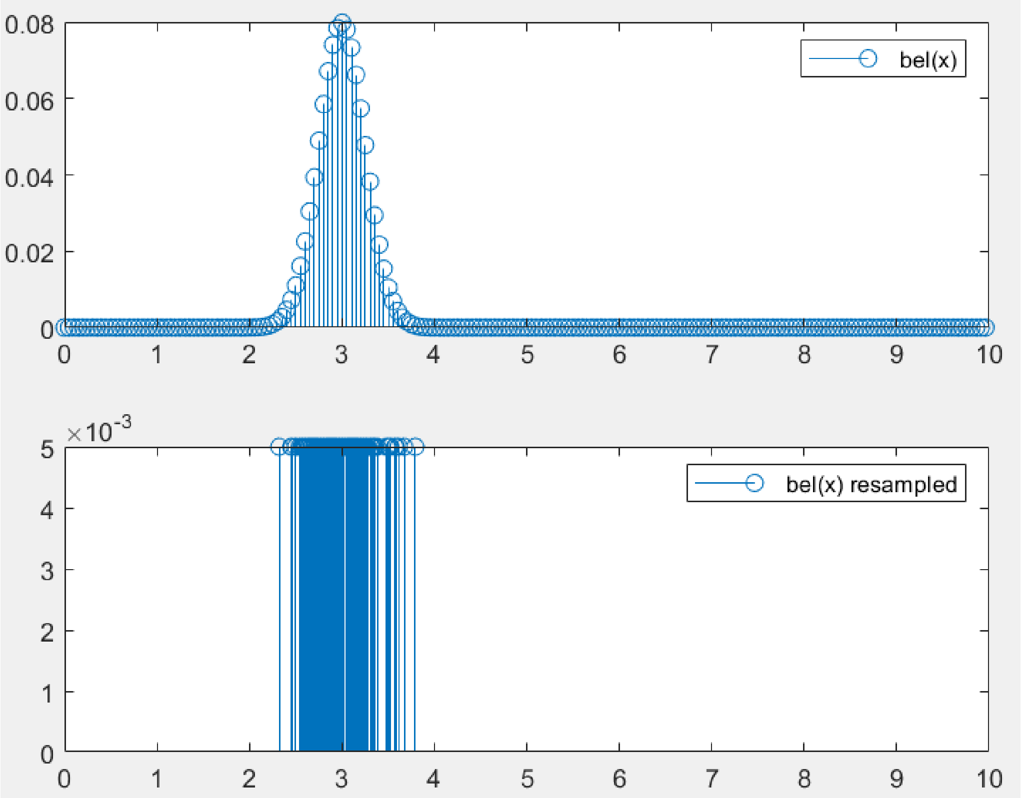
- Vector variable `x` stores set S. Each element in S is fetched by `x(k)`.
- Vector variable `p_x` stores $bel(x)$
- Vector variable `p_xpr` stores $bel(x')$

Line 5 calculates $p(x'|u,x)$ for one x' with `x(k)` and u `= u = 0.5`, and all x with variable `x`. This is done with vector calculation `x(k) - x - u`
So line 5 calculates `p_motion` (that is, $p(x'|u,x)$)

Therefore, elementwise product in `p_motion.*p_x` does it for all the states, and then is the normal sum, which will give you $bel(x')$ for one specific x' indexed k.

Monte Carlo case 1/2

- Full case requires some defintions that I would like to skip at this point.
- MC case relies on **sampling from a distribution**
- We represent **the distributions** with certain number of samples and their importance (weights), that is $M = \{(x_i, w_i)\}$
- The figure shows two possibilites: roughly speaking:
  - (upper) samples are uniformly distributed in which case wi's show their importance, or
  - (lower) importance sampling, in which case there are more samples in the areas with higher probabilities, and all samples have the same probability.



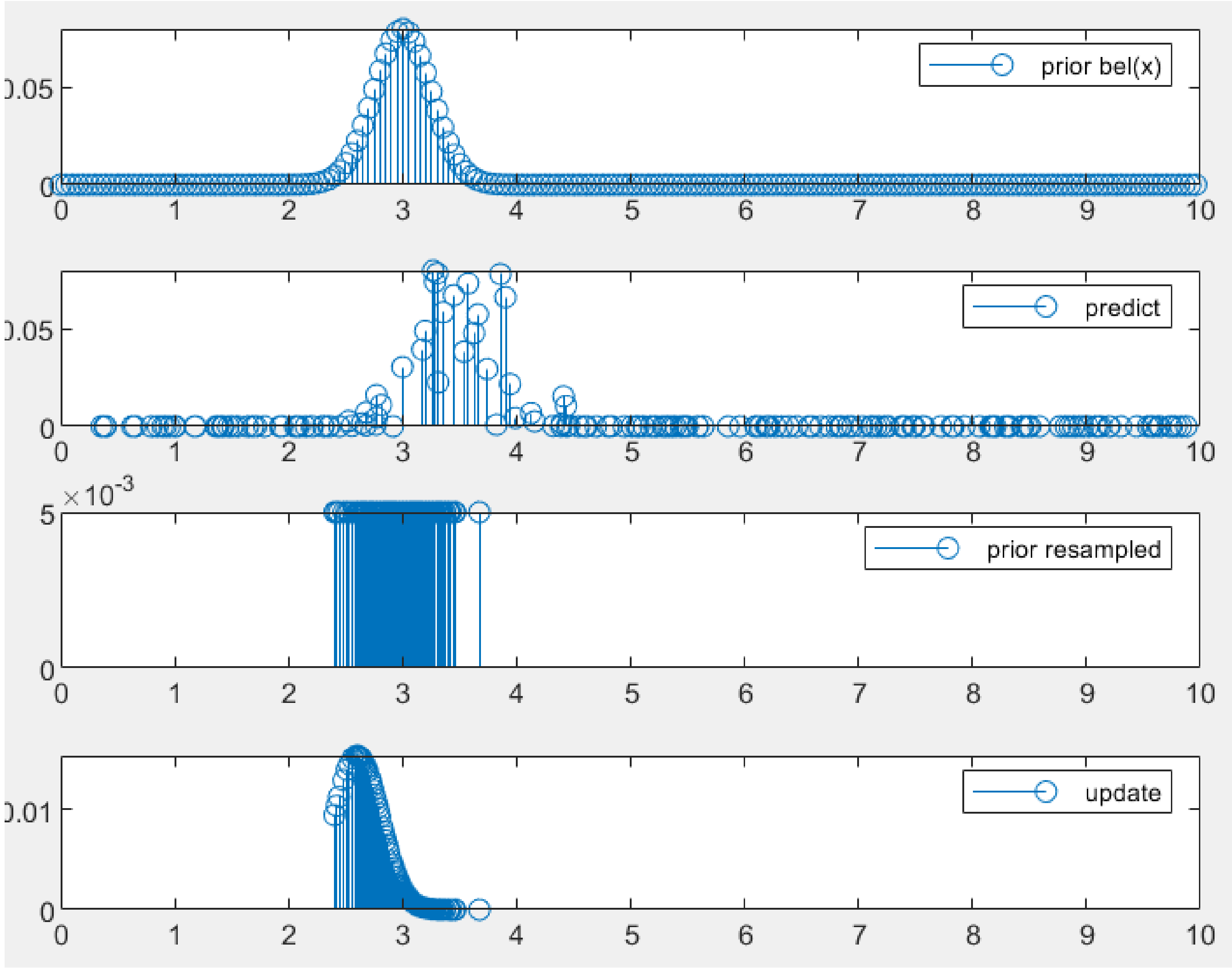

```
x0 = linspace(x_min,x_max,1000);
p_x0 = pdf(pd_x,x0);

% resample
xi_resampled = datasample(x0,100,…
                          'Weights', p_x0);
wi_resampled = 1/N*ones(1,100);
```

Monte Carlo case 2/2

Consider again the measurement and the motion examples.

Core idea: Imagine each sample as an instance of the robot

- `xi_predict = xi + u0 + random(pd_noise,size(xi)); % draw a sample from p(x'|x,u)`
- `xi_resampled = datasample(x0,100, 'Weights', p_x0);`
- `wi_update = pdf(pd_noise, z_sensor - xi_update); % importance, p(z|x)`



Parametric (Normal dist.) case 1/2

- Since **all the distributions** we defined in our examples are **Normal** and our motion model and measurement **model** are **linear** (neglecting the boundedness of the state space), we can do all these calculations also in parameter space.
  - Initial belief $bel(x) = N(3,0.25^2)$
  - Motion model $x' = x + u + noise$, where $noise \sim N(0,0.2^2)$,
  - Measurement model: $z = x + noise$, where $noise \sim N(0,0.1^2)$
- Since Normal function is defined by two parameters mean and covariance, we only need to calculate those to know the function.

We need some calculations beforehand.

Parametric case 2/3

Normal distributions and linear mapping

- Let $x \sim N(\hat{x}, P)$, and another random variable $\theta \sim N(0, Q)$ independent of x. Let's study the general vector form and multivariate case.
- Assume b and A are not random variables; Then Ax, $Ax + b$, $Ax + b + \theta$ are all Normal.
- All the following equalities you can calculate easily from rules defined on page 17, on expectation and covariance
- $x + b \sim N(\hat{x} + b, P)$
- $Ax \sim N\left(A\hat{x}, \mathrm{APA}^{\mathrm{T}}\right)$
- $x + \theta \sim N(\hat{x}, P + Q)$

These are the bases for **Kalman Filer**

Parametric case 3/3

- We need some more involved calculations before we are able to apply the ***measurement***.

- Let's assume we have two instances of measurement of the same variable, that is, $N(x_1, \sigma_1^2)$ and $N(x_2, \sigma_2^2)$. Our best unbiased estimate is given by $N(\hat{x}, \sigma^2)$, where $\frac{1}{\sigma^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}$, and $\frac{\hat{x}}{\sigma^2} = \frac{x_1}{\sigma_1^2} + \frac{x_2}{\sigma_2^2}$

- This equation gives us an estimate with minimum covariance σ^2. We will see the multivariate case, when we study Kalman filters.

- Our prior knowledge is $x \sim N(3, 0.25^2)$ and now measurement tells us that $x \sim N(2.6, 0.1^2)$. So, we can now calculate the optimal estimate, using above equations.

Comparing numerical results

Initial belief $bel(x) = N(3, 0.25^2)$

Motion model $x' = x + u + noise$, where $noise \sim N(0, 0.2^2)$, u=0.5

Parameric

- x_update = 2.6552
- sig_x_update = 0.0086
- x_predict = 3.5000
- sig_x_predict = 0.1025

Discretization

- Ex_z = 2.6552
- COVx_z = 0.0086
- Expr = 3.5000
- COVxpr = 0.1025

Monte Carlo

- meanMC_update = 2.7390
- meanMC_predict = 3.5359

Notation is from the previous numerical example codes.