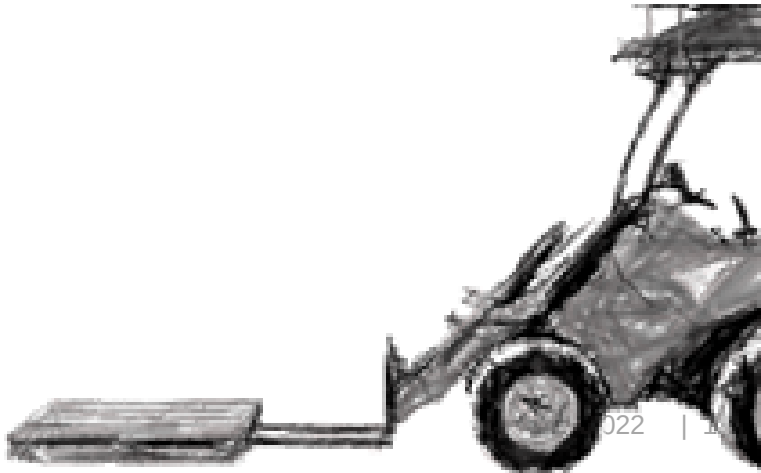
Tampere University

Introduction to and review of some general concepts

Reza Ghabcheloo

AUT 710

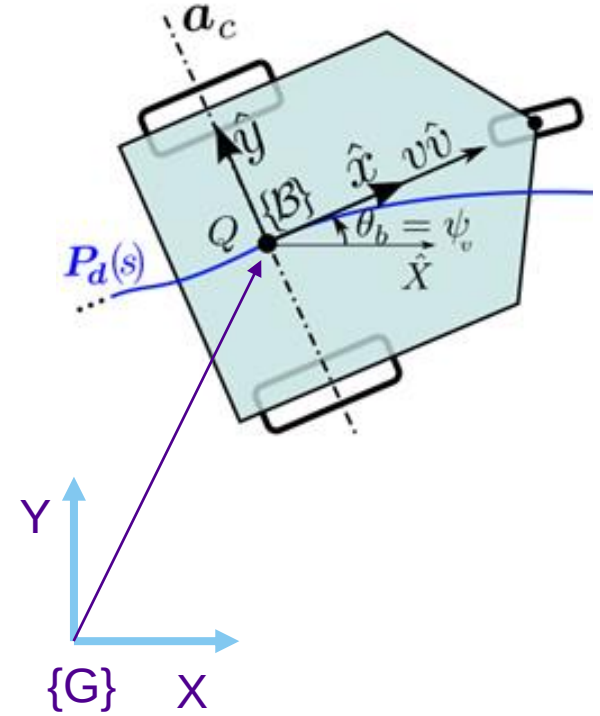


Concepts

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

Part 2

Differential drive kinematics (unicycle model)



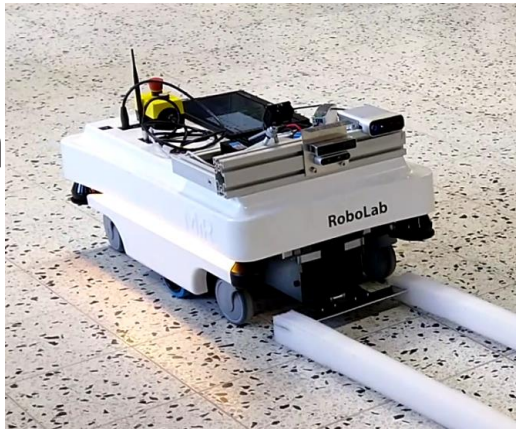
Equations of motion is given by

$$\dot{x} = v_{xB}\cos\psi$$
$$\dot{y} = v_{xB}\sin\psi$$
$$\dot{\psi} = \omega_z$$

where (v_x, ω_z) are independent inputs, defined in body frame {B}.

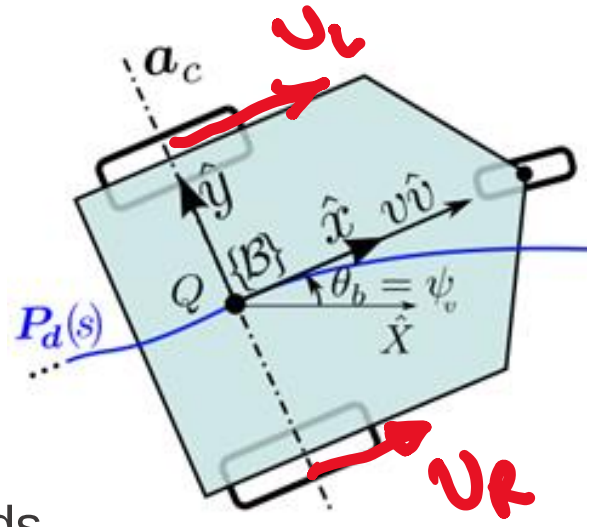
We will use these equations for both **control** and **localization**

- Control: v_x and ω_z are robot control commands (twist cmd)
- Localization: knowing v_x and ω_z and initial conditions $x_B(0), y_B(0), \psi(0)$, integrate the kinematic equations to get $x_B(t), y_B(t), \psi(t)$ (*dead-reckoning*)



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)

$$v_x = \frac{1}{2}(v_R + v_L)$$
$$\omega_z = \frac{1}{d}(v_R - v_L)$$
$$v_R = rw_R, v_L = rw_L$$

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 integration: dead-recking

differential drive / unicycle model

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

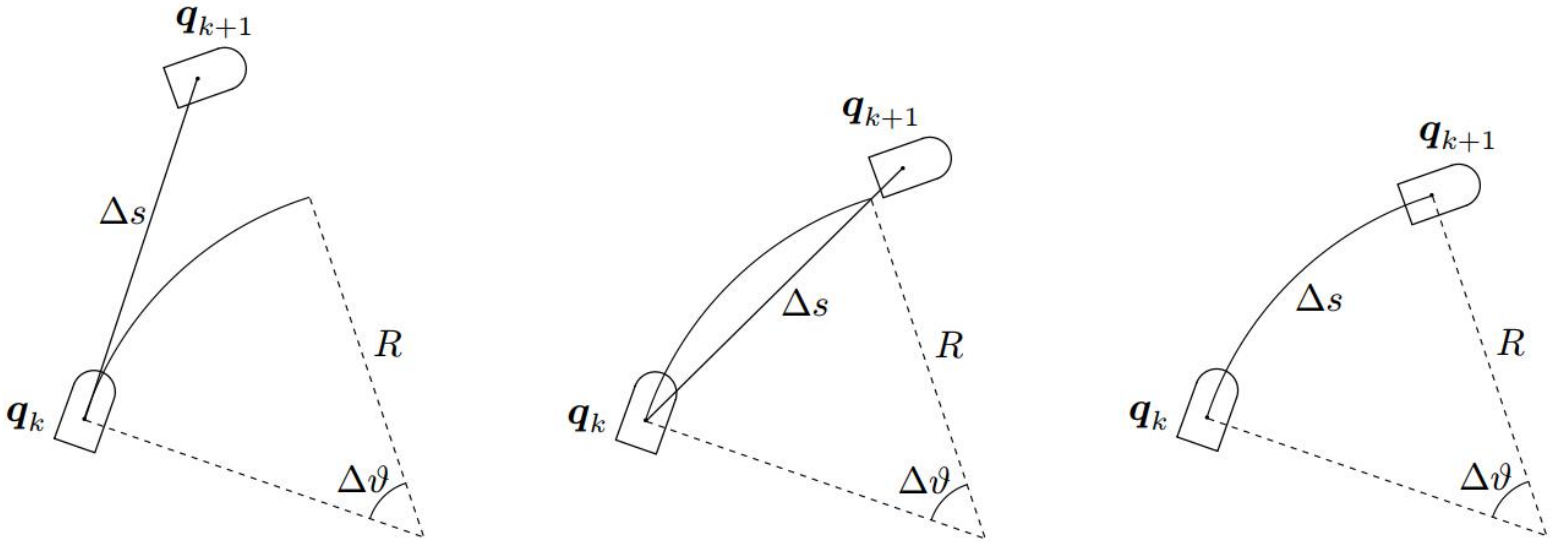


Fig. 11.21. Odometric localization for a unicycle moving along an elementary tract corresponding to an arc of circle; *left*: integration via Euler method, *centre*: integration via Runge–Kutta method, *right*: exact integration

Siciliano et al Motion planning and control, pp 514

Sensor inputs to odometry increment parameters

Sensor input:

- Wheel rotational speed (w_R, w_L); converted to:
- Body linear and angular speeds $(v, \omega)^T$; converted to:
- Translation and rotation increments $(\delta_{trans}, \delta_{rot})$

In simple models: $\delta_{trans} = v\Delta t$ and $\delta_{rot} = \omega\Delta t$

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: $\psi_{\bar{t}} = \psi_{t-1}$

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

Odometry model

- Euler integral approximation, midpoint method

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

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

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

- Similarly using odometry increment parameters

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

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

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

Q: How to use these to build probabilistic models? Where sources of 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 (recall our discussion on “AI Robotics vs Industrial robots”)

I will follow *Probabilistic Robotics* book for this section on Uncertainty and probabilities.

Probablities and destributions

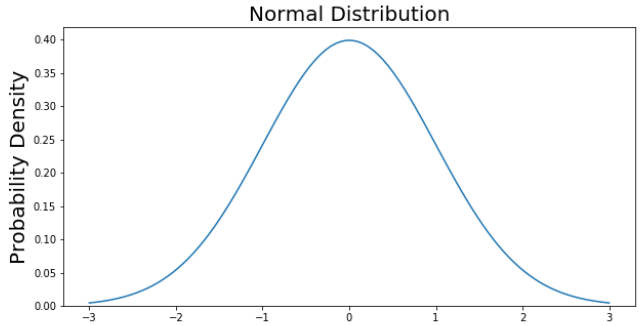
- Let's start with random variable X, discrete space
- $P(X = x)$ is probablity of X taking on the value x (probability mass function)
- Dice example: $P(X = 1) = \cdots P(X = 6) = \frac{1}{6}$
- $\Sigma_x P(X = x) = 1$, $1 \geq P(X = x) \geq 0$
- We may draw a sample: $x \sim P(x)$
- For simplicity,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

- 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

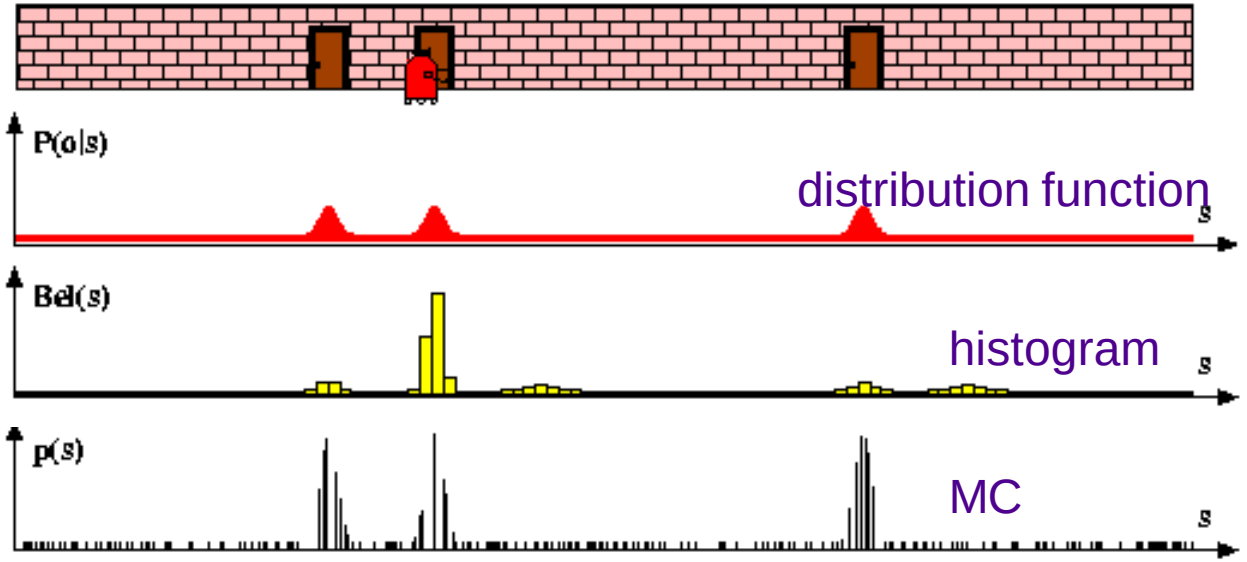
Robot pose

- 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*

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



Robot lives in 1D space



Some properties 1/2

- Joint probability $p(x, y) = p(X = x \text{ and } Y = y)$
- Conditional probability $p(x|y) = p(X = x|Y = y)$
- $p(x, y) = p(x|y)p(y)$
- If X and Y are independent, then $p(x, y) = p(x)p(y)$, and $p(x|y) = p(x)$
- Theory of total probability
  - $P(x) = \Sigma_y P(x|y)P(y)$
  - $p(x) = \int p(x|y)p(y)\mathrm{d}y$
- Bayes rule (relates $p(x|y)$ to its ”inverse”)
  - $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$,
- 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 $bel(x_t) = p(x_t|z_t, u_t)$

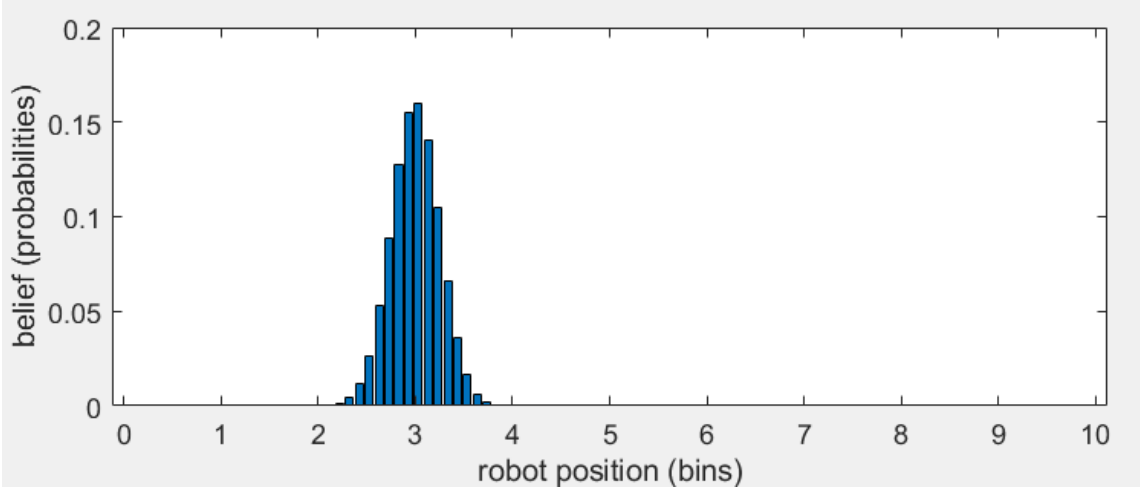
Some properties 2/2

- Expectation
  - $E[X] = \Sigma_x x P(x)$
  - $E[X] = \int x p(x) \mathrm{d}x$
- Covariance
  - $\mathrm{Cov}[X] = E\ [(X - E[X])^2]$
- $E[.]$ 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.
  - $p(x|y) = \eta\ p(y|x)p(x)$
  - η is calculated such that the resulting $p(x|y)$ sums up/integrates up to 1.

Matlab example

Discrete case, Monte Carlo, Parametric distributions

Discrete case 1/3



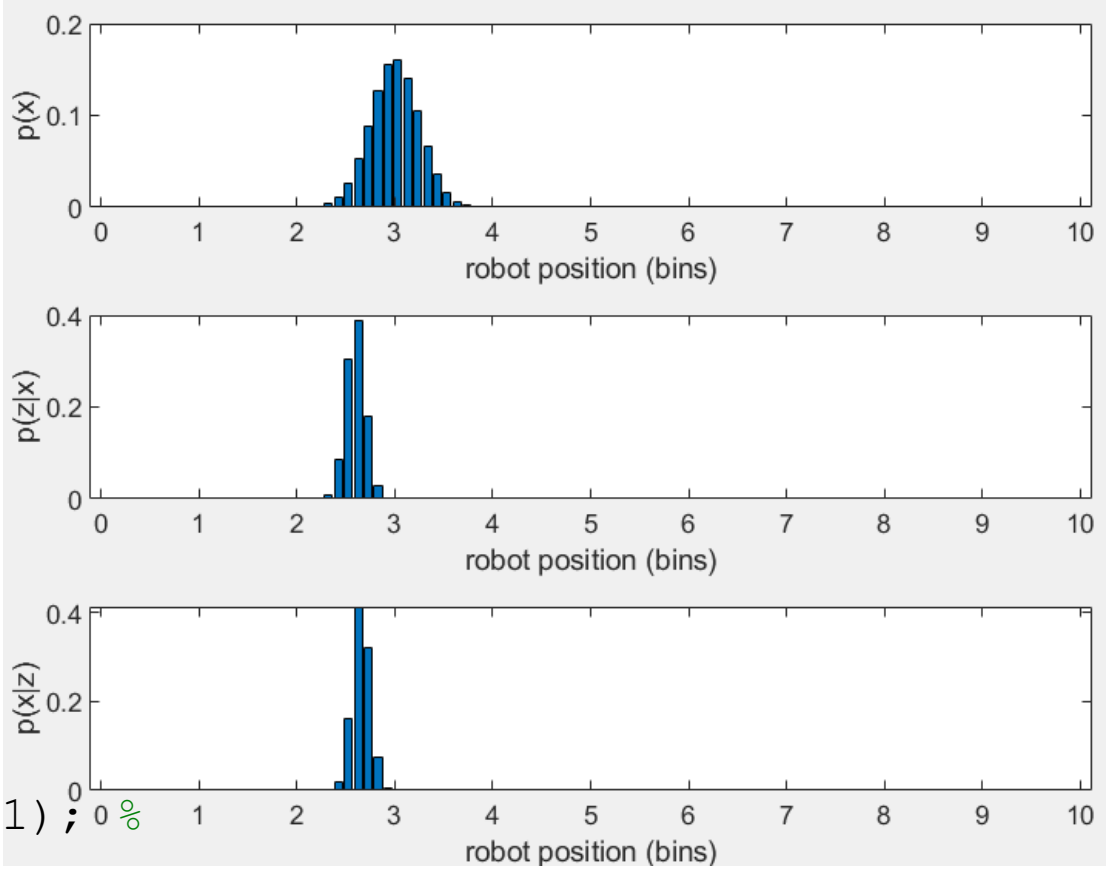

```
Robot lives in 1D space
x_min = 0; x_max = 10; %meter
x=linspace(x_min,x_max,100); % converted to
100 bins
pd_x = makedist('Normal','mu',3,'sigma',0.25);
% robot position belief
p_x = pdf(pd_x,x); % P(x)
eta = sum(p_x);
p_x = 1/eta*p_x; % now p sums up to 1
bar(x,p_x)

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

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

```
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 $p(x|z) = \eta\, p(z|x)p(x)$

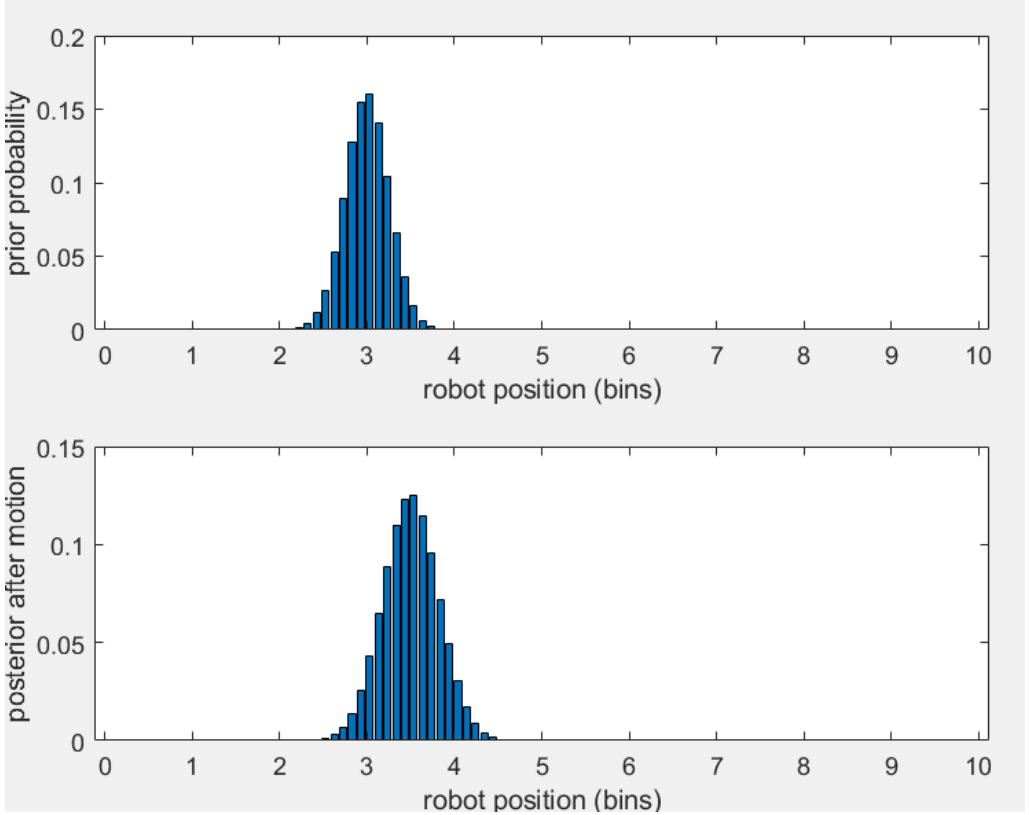



```
pd_noise = makedist('Normal','mu',0,'sigma',0.1);
sensor noise model

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: $\overline{bel}(x') = \Sigma_x p(x'|x,u) bel(x)$

```
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
```

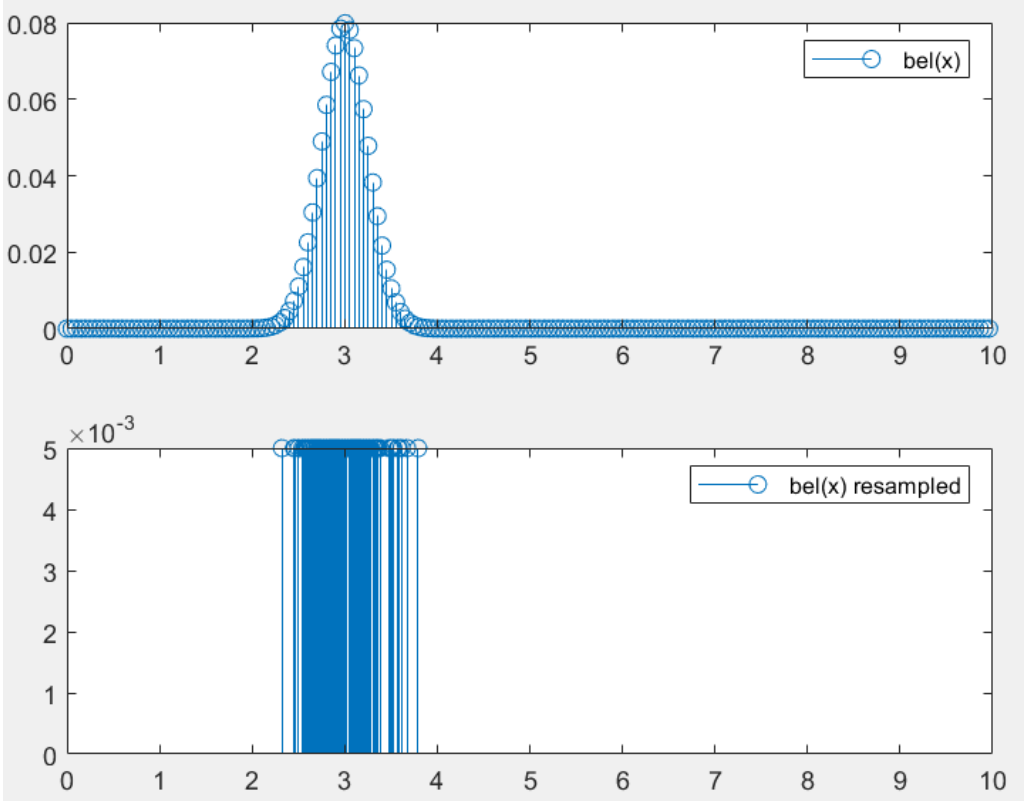


Note:
When `x(k) - x - u` is >10 or <0, we need to set `p_motion=0`.
This is neglected in the code, since the robot is sufficiently far from boundaries and Normal distribution `p_x` is almost zero there, and we will normalize our result at the end anyways.

Lecture 2 ended at here (19.1.)

Monte Carlo case 1/2

- Full case requires some defintions that I would like to skip at this point.
- We represent the distributions with certain number of samples and their importance (weights), that is $M = \{(x_i, w_i)\}$
- The figure shows two cases: 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.



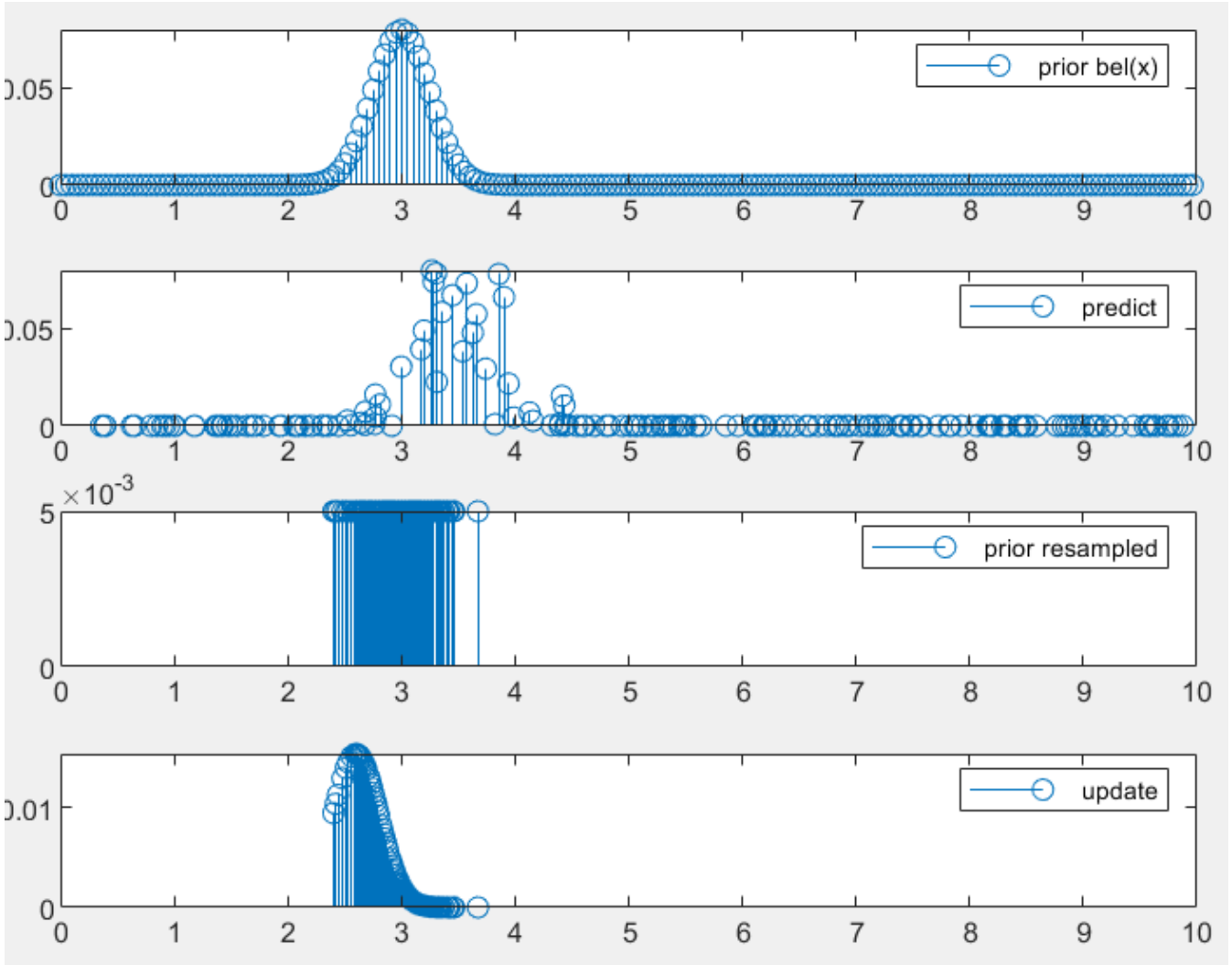

```
x0 = linspace(x_min,x_max,1000);
p_x0 = pdf(pd_x,x0);

% 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 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 $\mathrm{A}x$, $\mathrm{A}x + b$, $\mathrm{A}x + b + \theta$ are all Normal.
- $x + b \sim N(\hat{x} + b, P)$
- $Ax \sim N\left(A\hat{x}, A\mathrm{P}\mathrm{A}^{\mathrm{T}}\right)$
- $x + \theta \sim N(\hat{x}, P + Q)$

```
x_hat = 3;
sig_x = 0.25^2;
sig_motionNoise = 0.2^2; % zero mean
sig_measurNoise = 0.1^2; % zero mean
xpr_hat = x_hat + u0;
sig_xpr = sig_x + sig_motionNoise;  ← This shows that every step uncertainty increases!
```

Motion example:

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

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

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 value, 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 filter.
- 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

(x_hat = 3, sig_x = 0.0625)

Parameric

- x_hat_update = 2.6552
- sig_x_update = 0.0086
- x_predict_hat = 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

Bayes filter

Measure and predict cycle:

- Initial belief $bel(x)$
- Predict

$$bel(x) = \sum_{x'} p(x|u, x')bel(x')$$

- Measurement

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

Uncertainty

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