DR. MICHAEL P. HAYES

ENMT482

AUTONOMOUS ROBOTICS LECTURE NOTES

V109

ELECTRICAL AND COMPUTER ENGINEERING

Contents

Glossary and definitions

Notation

In general, a uppercase letter denotes a random variable[1] with its lowercase symbol denoting a specific value. A bold letter denotes a vector or matrix. Where possible a bold lowercase letter denotes a vector and a bold uppercase letter denotes a matrix. However, a random vector is denoted by a bold uppercase letter.

[1] In signal processing it is common to denote a random process with a lowercase letter, e.g., $x(t)$, since expectations are mostly used instead of densities.

X random variable

x specific value of random variable X

$\mu_X = \mathrm{E}[X]$ expected value (mean) of X

$\sigma_X^2 = \mathrm{Var}[X] = \mathrm{E}\left[(X - \mathrm{E}[X])^2\right]$ variance of X

$\sigma_X = \mathrm{Std}[X]$ standard deviation of X

$F_X(x)$ cumulative probability density function of X

$f_X(x)$ probability density function[2] of X

[2] Some authors use $p(x)$.

$f_{X,Y}(x,y)$ joint probability density function of X and Y

$X|Y$ conditional random variable

$f_{X|Y}(x|y)$ conditional probability density function for random variable X given random variable Y

$\rho = \rho_{XY}$ correlation coefficient between random variables X and Y

$X(\theta)$ random variable parameterised by θ

$f_X(x;\theta)$ probability density function for random variable $X(\theta)$ parameterised[3] by θ

[3] Some authors use $f_X(x|\theta)$ but with this notation one cannot determine if θ is a specific value of a random variable or a parameter.

$\mathbf{X}$ random vector

$\mathbf{x}$ specific vector of random vector $\mathbf{X}$

$\mu_{\mathbf{X}} = \mathrm{E}[\mathbf{X}]$ mean of random vector $\mathbf{X}$

$\Sigma_{\mathbf{X}} = \mathrm{E}\left[(\mathbf{X} - \mathrm{E}[\mathbf{X}])(\mathbf{X} - \mathrm{E}[\mathbf{X}])^{\mathrm{T}}\right]$ covariance matrix of random vector $\mathbf{X}$

$\hat{X}$ estimator of X (random variable)

$\hat{x}$ estimate of X (specific value)

$L_{X|Z}(x|z)$ likelihood function for $X = x$ given $Z = z$

$\mathrm{E}\{x(t)\}$ expected value of a realisation of an ergodic random process (long-time average)

Glossary

$a^{(m)}$ Weight of m^{th} particle

A State transition matrix

B Input matrix

C Output matrix

D Feed-through matrix

d Pose change decomposition vector

D Pose change decomposition random vector

g General motion model

g' Derivative of general motion model

h General sensor model

h' Derivative of general sensor model

I Identity matrix

J Jacobian matrix

K Kalman gain vector

M Random map (grid of random variables M_{ij})

$\hat{\mathbf{m}}$ Estimated map

m Specific map

m_{ij} Specific map cell

P^- State prior estimate variance

P^+ State posterior estimate variance

$\mathbf{P}^-$ State prior estimate covariance matrix

$\mathbf{P}^+$ State posterior estimate covariance matrix

R Range random variable

r Range

u Control action scalar

$\mathbf{u}$ Control action vector

v_d Desired linear speed

V Measurement noise random variable

$\mathbf{V}$ Measurement noise random vector

$\mathbf{V}$ Speed random vector

V Linear speed random variable

v Linear speed

$\mathbf{W}$ Process noise random vector

W Process noise random variable

$\mathbf{w}$ Weight vector

w Weighting

X Belief of state (random variable)

X^- Prior belief of state (random variable)

X^+ Posterior belief of state (random variable)

x Specific state

$\mathbf{X}$ Random state vector

$\mathbf{X}^-$ Prior belief of state vector

$\mathbf{X}^+$ Posterior belief of state vector

$\mathbf{x}$ Specific state vector

$\hat{\mathbf{x}}$ Estimate of random state vector

$\hat{\mathbf{x}}^-$ Estimate of prior random state vector

$\hat{\mathbf{x}}^+$ Estimate of posterior random state vector

$\mathbf{x}^{(m)}$ State vector of m^{th} particle

$\check{\mathbf{x}}$ Local pose vector

$\mathbf{Z}$ Measurement random vector

$\mathbf{z}$ Measurement vector

Z Measurement random variable

z Measurement value

Δt Sampling period

$\mathbf{\Sigma}_{\mathbf{X}^-}$ Prior belief covariance matrix

$\mathbf{\Sigma_{X^+}}$ Posterior belief covariance matrix

$\mathbf{\Sigma_W}$ Measurement noise covariance matrix

$\mathbf{\Sigma_V}$ Process noise covariance matrix

Θ Heading angle random variable

θ Heading angle

Ω Angular speed random variable

ω Angular speed

ω_d Desired angular speed

Definitions

A priori before the measurement.

A posteriori after the measurement.

Dynamics concerns with the study of forces and torques and their effect on motion (as opposed to kinematics, which studies the motion of objects without reference to its causes).

Holonomic drives have three degrees of freedom and provide motion in any direction with independent rotation. In comparison, differential drives have only two degrees of freedom.

Kinematics is the study of classical mechanics which describes the motion of points, bodies, and systems of bodies without consideration of the causes of motion[4].

[4] From the Greek kinema for movement or motion.

Markov (complete state) assumption says the current state can be determined from just the previous state and the current control input.

Odometry is the measurement of distance.

Parameter identication This is an estimation problem where the goal is to the find the unknown parameters of a known equaton.

Pose describes the location and orientation. For a robot it is a subset of its configuration (which also describes the angles of joints. etc. of manipulators).

System identication This is an estimation problem where the goal is to the find a model of a system and its parameters.

Initialisms and acronyms

BLUE Best Linear Unbiased Estimate (Estimator).

EKF Extended Kalman Filter.

IID Independent and Identically Distributed.

MLE Maximum Likelihood Estimate (Estimator).

MVUE Minimum Variance Unbiased Estimate (Estimator).

ROS Robot Operating System.

SDF Simulator Description Format. This is an XML format for representing a robot model. This can be used by the Gazebo simulator.

SLAM Simultaneous Localisation and Mapping.

UKF Unscented Kalman Filter.

URDF Unified Robot Description Format. This is an XML format for representing a robot model. There is a Solidworks add-in for producing it. This can be used by the Gazebo simulator although there is a better preferred Simulator Description Format (SDF).

0
Introduction

Mobile robotics is difficult since it requires mastery of many disciplines, including:

- Kinematics
- Dynamics
- Locomotion
- Sensors
- Actuators
- Electronics
- Power electronics
- Power management
- Control theory
- Machine vision
- Perception
- Computer hardware
- Computer software
- Computer networking
- Probability and statistics
- . . .

0.1 Probabilistic robotics

Perhaps surprisingly, one of the key technologies for autonomous mobile robotics is statistics, in particular the application of Bayes' theorem. The main reason is that the environments of robots are inherently unpredictable[1], especially for mobile robots. Moreover, sensors are subject to physical limitations and their measurements are noisy and sometimes ambiguous. Similarly, actuators are also unpredictable; wheels slip, gears have backlash, motors can fail. Furthermore, the complexity of robot environments requires approximate models[2] that can be analysed in real-time. These algorithmic approximations introduce further uncertainty.

Probabilistic algorithms have an advantage in that they can handle multiple hypotheses. Instead of relying on a single best guess, probabilistic algorithms represent information by probability distributions.

[1] Plants grow, snow covers things, people move.

[2] Consider a robot environment with 20 objects each having 10 states. The number of permutations (with replacement) is 10^{20}.

0.2 Summary

1. Robot environments are unpredictable
2. Robot environments are complicated and approximate models are required
3. Sensor information is ambiguous and noisy
4. Actuators are unreliable and noisy
5. Probabilistic models can support multiple hypotheses
6. Heuristics are required to handle combinatorial explosion

0.3 Further reading

- http://www.probabilistic-robotics.org/
- "Probabilistic Robotics", S. Thrun, W. Burgard, and D. Fox. MIT Press, 2005. *This is the book!*
- "Autonomous Mobile Robots", R. Siegwart and I. Nourbaksh, MIT Press, 2004. *This is simpler but with more arm waving.*

1
Sensors

Sensor fusion is commonly used in mobile robotics to combine (fuse) measurements from different sensors. The resulting estimate has a lower variance than the individual measurements. An example is the fusing of measurements from an inertial measurement unit[1] (IMU) with a global positioning system (GPS). To understand how a better estimate is obtained, it is necessary to consider some probability and statistics, in particular, estimation theory.

[1] These contain accelerometers, gyroscopes, and magnetometers.

1.1 A little problem

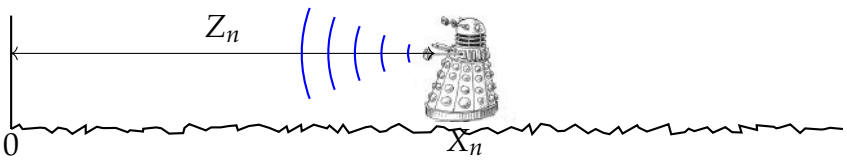


Figure 1.1: Robot position estimation problem.

Consider a robot that has been instructed to move[2] at a speed of 3 m/s for 3 s. When it has stopped, an ultrasonic range sensor says that it moved 8 m, however, a laser range finder says that it moved 11 m. So how do we decide how far the robot moved?

[2] With open-loop control.

1. Do we ignore the sensors and say that the robot moved 9 m since that is what we told it to do?
2. Do we choose the laser measurement of 11 m since laser range sensors are usually more accurate than ultrasonic range sensors?
3. Do we choose the ultrasonic measurement of 8 m since it is closest to what we expect?
4. Do we average the two measurements? If so, do we give each measurement equal weighting?

To find the best solution requires a little estimation theory, but first let's consider some sensor models.

1.2 *Sensor models*

Sensors produce a measurement, z, of an unknown state, x. If the sensor is *linear* and noiseless,

$$z = cx + d, \tag{1.1}$$

where c is a constant scale factor and d is a constant offset.

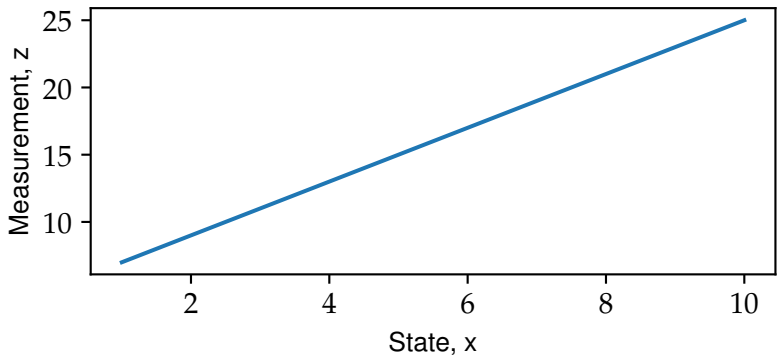


Figure 1.2: Example relationship of a linear sensor: $z = 2x + 5$.

Some sensors have a non-linear relationship between the state and measurement. For example, an IR triangulating range sensor has an approximate model

$$z = \frac{k_1}{k_2 + x}, \tag{1.2}$$

where k_1 and k_2 are constant parameters.

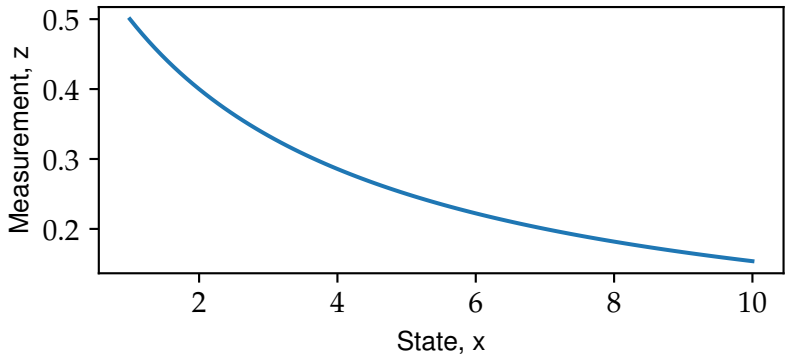


Figure 1.3: Example relationship of a non-linear sensor: $z = 2/(3 + x)$.

In general, a noiseless non-linear sensor be modelled as

$$z = h(x). \tag{1.3}$$

1.3 Uncertainty

All sensors have uncertainty due to electronic noise. This is modelled by considering the measurement as a random variable[3]. Usually, the noise is additive and thus a sensor can be modelled by

$$Z = h(x) + V, \tag{1.4}$$

where V is a random variable describing the additive noise.

[3] A random variable, Z, can be considered a placeholder for the outcome of a measurement. The actual measured value is denoted z.

1.3.1 Gaussian noise

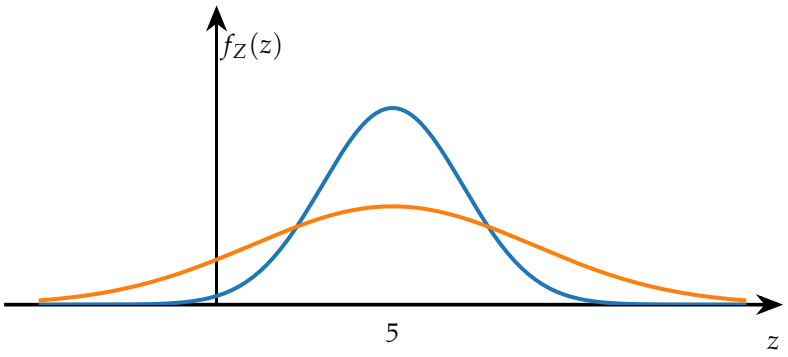


Figure 1.4: Gaussian distribution with mean, $\mu_V = \mathrm{E}[V] = 5$, and variance, $\sigma_V^2 = \mathrm{Var}[V] = 2$ and 4.

Random variables have an associated probability density function (PDF). This is denoted[4],

$$V \sim f_V(v). \tag{1.5}$$

The PDF of electronic noise is a Gaussian (normal) distribution[5], denoted

$$f_V(v) = \mathcal{N}\left(\mu_V, \sigma_V^2\right). \tag{1.6}$$

This has two parameters:

Mean $\mu_V = \mathrm{E}[V]$,

Variance $\sigma_V^2 = \mathrm{Var}[V]$.

An example of the Gaussian distribution is shown in Figure 1.4. In general, it is described by

$$f_V(x) = \frac{1}{\sqrt{2\pi\sigma_V^2}} \exp\left(-\frac{1}{2}\frac{(x-\mu_V)^2}{\sigma_V^2}\right). \tag{1.7}$$

All PDFs integrate to unity and so a Gaussian distribution with a smaller variance has a higher peak.

[4] The $\sim$ symbol is read "is distributed as" or "has a probability density".

[5] The Gaussian distribution has some interesting properties: the product of two Gaussians is a Gaussian and so is the convolution of two Gaussians.

1.3.2 *Measurement mean and variance*

Consider the sensor model with additive noise

$$Z = h(x) + V. \tag{1.8}$$

The sensor noise is usually zero mean[6], i.e.,

$$\mathrm{E}[V] = 0, \tag{1.9}$$

and so the expected measurement (the mean value) is

$$\mathrm{E}[Z] = h(x). \tag{1.10}$$

The noise has a variance

$$\mathrm{Var}[V] = \sigma_V^2, \tag{1.11}$$

and so the measurement variance is

$$\mathrm{Var}[Z] = \mathrm{Var}[V] = \sigma_V^2. \tag{1.12}$$

[6] Assuming the sensor is calibrated.

1.4 *Time averaging*

Consider N measurements from a sensor with additive noise. These can be modelled by

$$Z_n = h(x) + V_n. \tag{1.13}$$

This assumes that the unknown state x is not changing. The average of the measurements is

$$\overline{Z} = \frac{1}{N}\sum_{n=1}^{N} Z_n. \tag{1.14}$$

This is also a random variable.

If the parameters of the sensor noise are unchanging, the random noise process $\{V_n\}$ is identically distributed, i.e.,

$$\mathrm{E}[V_n] = \mathrm{E}[V], \tag{1.15}$$

$$\mathrm{Var}[V_n] = \sigma_V^2. \tag{1.16}$$

If each of the noise random variables are mutually independent, the random noise process is termed IID (independent and identically distributed). With this assumption, the expected value of the average is

$$\mathrm{E}\left[\overline{Z}\right] = h(x), \tag{1.17}$$

and the variance is

$$\mathrm{Var}\left[\overline{Z}\right] = \frac{1}{N}\sigma_V^2. \tag{1.18}$$

This is an important result: averaging reduces the uncertainty. Note, if the noise random process is Gaussian distributed, the average is also Gaussian distributed.

1.5 Sensor model considerations

The modelling of sensors is not straightforward, since the noise can vary with the state and with time, the sensor can be ambiguous, and the sensor can produce erroneous results (outliers).

1.5.1 State varying variance

The variance of a range sensor increases with distance. This is because the returned signal becomes smaller[7], the farther it has to travel. This effect can be modelled by making the noise a function of the state, for example,

$$Z = h(x) + V(x). \tag{1.19}$$

Here the random noise is parameterised by x. The corresponding PDF is denoted[8] $f_V(v; x)$.

[7] Due to spreading and absorption effects.

[8] The variable after the semicolon is a parameter.

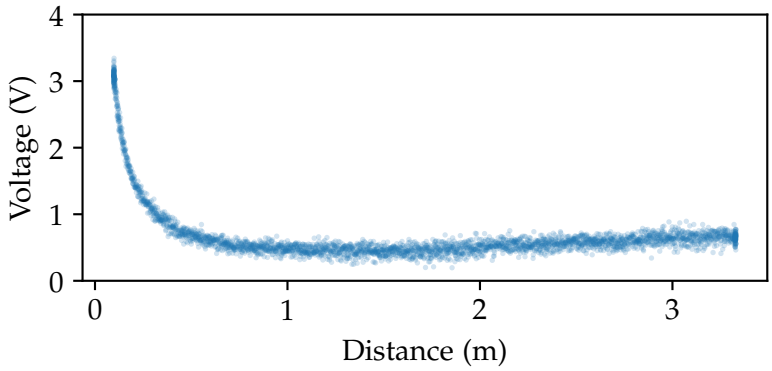


Figure 1.5: Example of range varying variance for triangulating IR sensor. This sensor gives an output voltage. Note, the non-linear relationship between distance and output voltage.

1.5.2 Ambiguities

Some sensors produce ambiguous measurements[9] For example, a ultrasonic range sensor might produce ambiguous measurements due to reflections from walls. The resulting PDF is multimodal, see Figure 1.6.

[9] We have all been fooled by optical illusions and robots are no different.

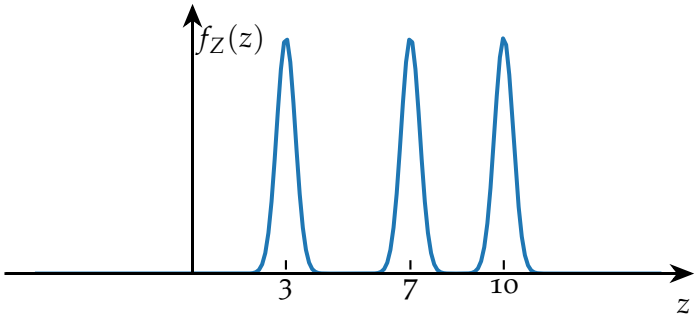


Figure 1.6: Multimodal PDF.

1.5.3 *System identification*

The parameters of the sensor model, including the parameters of the noise, can be found using system identification techniques. For example, $h(x)$ can be guessed and least squares employed to determine the parameters. Unfortunately, the result is biased by outliers and *robust regression*[10] techniques are required.

[10] For example, iteratively reweighted least squares (IRLS).

1.5.4 *Outliers*

Sensors can sometimes produce weird measurements, often due to electrical interference[11]. In this case, the simple additive Gaussian noise model is invalid. If the outliers are due to interference, then the statistics of the noise will vary with time.

[11] This might be from a robot's motors.

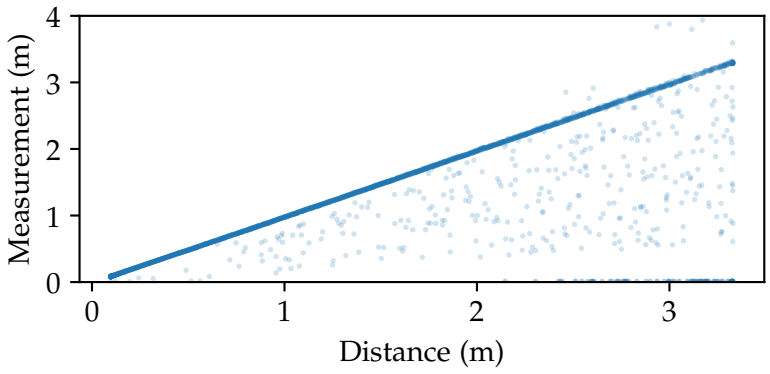


Figure 1.7: Ultrasonic range sensor measurements showing outliers. This sensor gives an output in metres.

Outlier rejection is difficult. One approach uses Iglewicz and Hoaglin's modified Z-score[12]:

$$M_i = 0.6745 \frac{(\epsilon_i - \text{median}(\boldsymbol{\epsilon}))}{\text{median}(|\boldsymbol{\epsilon}|)}, \qquad (1.20)$$

where $\boldsymbol{\epsilon}$ is a vector of errors. Values of ϵ_i where $|M_i| > 3.5$ are considered outliers. The effect of this is shown in Figure 1.8.

[12] The flaw with this approach is that the errors cannot be determined without a model!

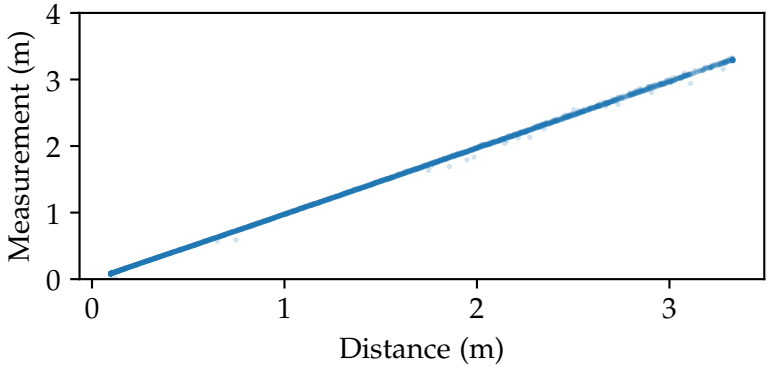


Figure 1.8: Ultrasonic range sensor measurements with outlier rejection.

1.6 Random variable revision

1.6.1 Linear transformation

Consider a random variable X that is scaled by a factor c and offset by d,

$$Y = cX + d. \tag{1.21}$$

The result Y is another random variable. It has an expected value

$$\mathrm{E}[Y] = \mathrm{E}[cX + d] = c\,\mathrm{E}[X] + d, \tag{1.22}$$

and a variance

$$\mathrm{Var}[Y] = \mathrm{Var}[cX + d] = c^2\,\mathrm{Var}[X]. \tag{1.23}$$

Note, if X has a Gaussian PDF then Y also has a Gaussian PDF.

1.6.2 Sum of two random variables

The sum of two random variables is also a random variable,

$$Z = X + Y. \tag{1.24}$$

This has an expected value,

$$\mathrm{E}[Z] = \mathrm{E}[X + Y] = \mathrm{E}[X] + \mathrm{E}[Y], \tag{1.25}$$

and a variance

$$\mathrm{Var}[Z] = \mathrm{Var}[X + Y] = \mathrm{Var}[X] + \mathrm{Var}[Y] + \sqrt{\mathrm{Var}[X]\,\mathrm{Var}[Y]}\rho_{XY}. \tag{1.26}$$

The result depends on the correlation ρ_{XY} between X and Y. Usually, X and Y are independent and thus the correlation is zero. With this assumption,

$$\mathrm{Var}[Z] = \mathrm{Var}[X + Y] = \mathrm{Var}[X] + \mathrm{Var}[Y]. \tag{1.27}$$

If X and Y are both Gaussian distributed, Z is also Gaussian distributed[13].

[13] In general, the PDF of Z is the convolution of the PDF of X with the PDF of Y.

1.6.3 Average of two random variables

The average of two random variables is

$$A = \frac{1}{2}(X + Y). \tag{1.28}$$

The expected value of the average is

$$\mathrm{E}[A] = \frac{1}{2}\mathrm{E}[X+Y] = \frac{1}{2}\left(\mathrm{E}[X] + \mathrm{E}[Y]\right), \tag{1.29}$$

and assuming X and Y are independent,

$$\mathrm{Var}[A] = \frac{1}{4}\left(\mathrm{Var}[X] + \mathrm{Var}[Y]\right). \tag{1.30}$$

Thus if X and Y have the same variance, the variance of the average is halved. This is an important result for improving sensor measurements.

1.7 Summary

1. Sensor measurements are corrupted by noise.
2. Sensor noise is modelled with random variables.
3. A continuous random variable has a probability density function[14] (PDF).
4. The variance of a range sensor increases with range.
5. Averaging reduces the variance of sensor measurements.
6. Sensors can sometimes produce outliers.
7. Sensors can sometimes be ambiguous.

[14] A discrete random variable has a probability mass function (PMF).

1.8 Exercises

1. An ultrasonic sensor with a variance of $0.01\,\mathrm{m}^2$ measures the range to a wall to be $1.2\,\mathrm{m}$. A lidar with a variance $0.01\,\mathrm{m}^2$ measures the range to the wall to be $1.1\,\mathrm{m}$.
   (a) If the measurements are averaged, what is the expected value?
   (b) If the measurements are averaged, what is the variance? State any assumptions.
2. An ultrasonic sensor with a variance of $0.04\,\mathrm{m}^2$ measures the range to a wall to be $1.2\,\mathrm{m}$. A lidar with a variance $0.01\,\mathrm{m}^2$ measures the range to the wall to be $1.1\,\mathrm{m}$.
   (a) If the measurements are averaged, what is the expected value?

(b) If the measurements are averaged, what is the variance? State any assumptions.

(c) Can you think of a better way of combining the measurements?

3. Consider two range sensors: one that samples at 1 Hz with a variance of 0.01 m^2 and another that samples at 10 Hz with a variance of 0.05 m^2. Which would you choose?

4. If X is random distance variable what are the units for its variance?

5. Emily's UFO has a altitude sensor where the output voltage can be modelled by a random variable Z, where

$$Z = 2x + 1 + V,$$

for an unknown altitude x (in km) with additive zero-mean Gaussian random noise, represented by the random variable V with a standard deviation $\sigma_V = 2$ V.

(a) If the expected value for the altitude is 1 km determine the expected value of Z.

(b) Given that the altitude is 1 km determine the variance of Z.

6. Sam's UFO has an altitude sensor where the output voltage can be modelled by a random variable Z, where

$$Z = 3x^2 + V,$$

x is an unknown altitude (in km), and V is a zero-mean Gaussian random variable with a standard deviation $\sigma_V = 3$ V.

(a) If the expected value for the altitude is 1 km determine the expected value of Z.

(b) Given that the altitude is 1 km determine the variance of Z.

7. Evelyn's UFO has an ACME altitude sensor where the output voltage can be modelled by a random variable Z, where

$$Z = 2x + V(x),$$

x is an unknown altitude (in km), and V is a zero-mean Gaussian random variable with a standard deviation,

$$V(x) = 0.01x. \tag{1.31}$$

(a) If the expected value for the altitude is 1 km determine the expected value of Z.

(b) Given that the altitude is 1 km determine the variance of Z.

2
Estimation

The goal of estimation is to determine unknown parameters from measurements. Without any prior information, the best possible estimator is called a minimum variance unbiased estimator (MVUE)[1].

The minimum variance unbiased estimator (MVUE) may not exist or may be hard to determine for certain estimation problems. In practice, a maximum likelihood estimator (MLE) is used since it asymptotically becomes a MVUE with a large number of measurements.

[1] This has a variance given by the Cramér-Rao lower bound (CRLB). No unbiased estimators can achieve a lower variance than the CRLB.

2.1 *The likelihood function*

Consider the simple sensor model[2], linear in the unknown parameter x, with additive uncertainty:

$$Z = x + V. \tag{2.1}$$

[2] Also called an observation model.

Let's assume that V is zero mean with a variance of $0.01\,\mathrm{m}^2$. If we measured a value $z = 2$ we would say that a value for $x = 2$ is likely but that a value $x = 5$ is unlikely.

How likely the parameter is, given a measurement, is quantified with a likelihood function. Since the noise is additive, the likelihood function for (2.1) is

$$L(x|z) = f_V(z - x). \tag{2.2}$$

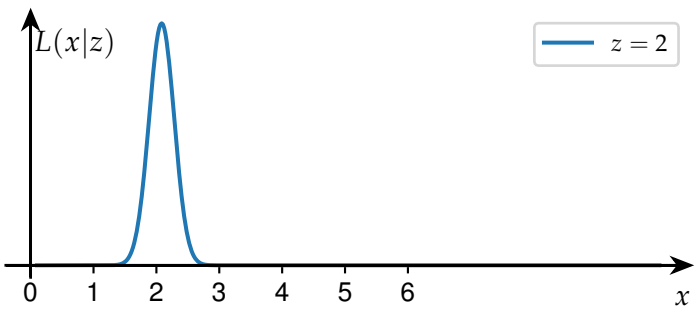


Figure 2.1: Likelihood function for a sensor with additive zero-mean Gaussian noise. In this example, $\sigma_V = 0.2$.

2.2 *Maximum likelihood estimation (MLE)*

The maximum likelihood estimate is found by searching for the peak of the likelihood function. This can be denoted as an optimisation:

$$\hat{x} = \arg\max_x L(x|z). \tag{2.3}$$

The hat on x denotes an estimate[3].

[3] Sometimes this is denoted $\hat{x}_{\text{ML}}$ where the subscript denotes the type of estimator.

2.2.1 *Example: MLE for simple model with additive noise*

Consider the sensor model:

$$Z = x + V, \tag{2.4}$$

where the sensor uncertainty is described by

$$V \sim f_V(v). \tag{2.5}$$

The maximum likelihood estimate for the parameter, $\hat{x}$ is found as the value of x that maximises the likelihood, i.e,

$$\hat{x} = \arg\max_x f_V(z - x). \tag{2.6}$$

For example, if the measurement noise is Gaussian distributed with zero mean and variance 0.5, then

$$f_V(x) = \mathcal{N}(0, 0.5). \tag{2.7}$$

From the measurement model (2.4), the measurement likelihood is also Gaussian distributed

$$L(x|z) = \mathcal{N}(z - x, 0.5). \tag{2.8}$$

The peak occurs when $z = x$ and thus the maximum likelihood estimate is $\hat{x} = z$.

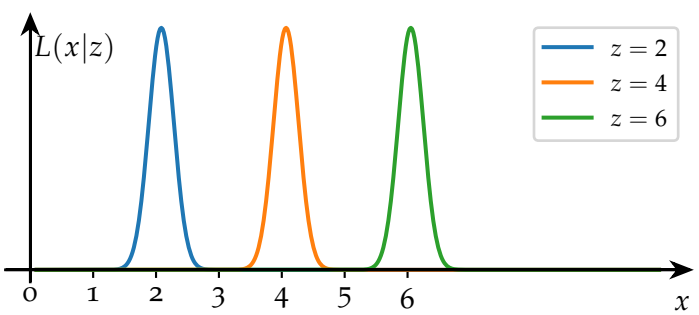


Figure 2.2: Likelihood function for a sensor with additive zero-mean Gaussian noise. In this example, $\sigma_V = 0.2$.

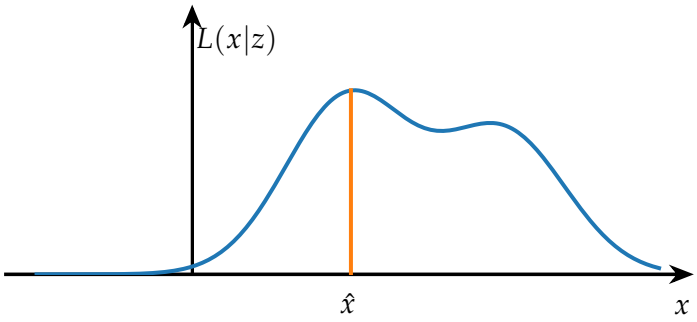


Figure 2.3: Maximum likelihood estimation.

2.2.2 *Example: MLE for non-linear sensor*

In general, for a sensor model

$$Z = h(x) + V, \tag{2.9}$$

the likelihood function is

$$L(x|z) = f_V(z - h(x)). \tag{2.10}$$

It is worth noting that if $h(x)$ is a non-linear function of x, then the likelihood function may not be symmetrical or unimodal, even for Gaussian noise. For example, see Figure 2.4 where

$$h(x) = \frac{10}{x}. \tag{2.11}$$

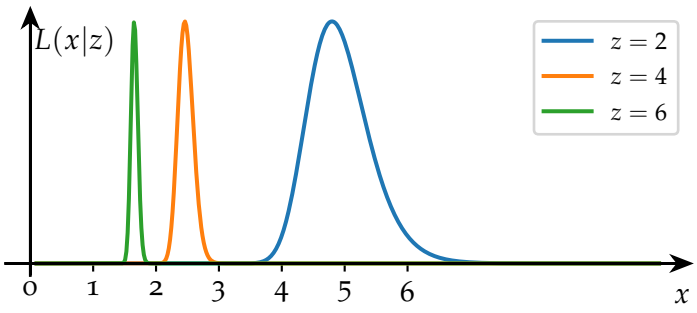


Figure 2.4: Likelihood function for a non-linear sensor where $h(x) = 10/x$ and where the addtitive noise is zero-mean Gaussian with a constant variance. The eagle-eyed will notice that when $z = 2$. the peak is not exactly at $x = 5$.

2.2.3 *Example: MLE for distance sensor*

With a distance sensor, the uncertainty increases with distance, so for a simple linear sensor

$$Z = x + V(x). \tag{2.12}$$

For example, if the noise is zero-mean and Gaussian distributed

$$V(x) \sim \mathcal{N}(0, \sigma_{V(x)}), \tag{2.13}$$

where $\sigma_{V(x)}$ is the distance varying standard deviation (usually estimated from calibration data).

The likelihood function is

$$L(x|z) = \mathcal{N}(z - x, \sigma_{V(x)}). \tag{2.14}$$

This is shown in Figure 2.5. Note that the maximum likelihood estimate is not quite z; there is a small bias due to the distance varying variance.

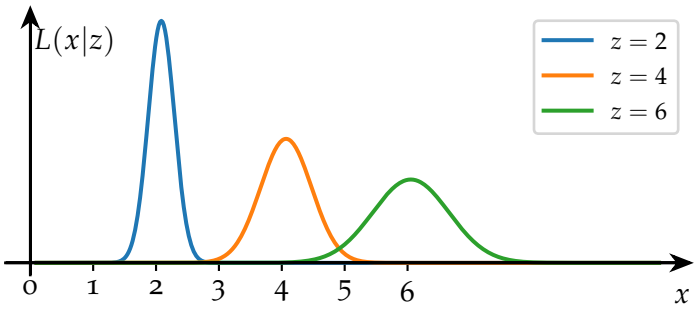


Figure 2.5: Likelihood function for a distance sensor where the noise is zero-mean Gaussian with a variance proportional to distance. In this example, $\sigma^2_{V(x)} = 0.1x$.

2.2.4 *ML estimator for linear model with additive Gaussian noise*

Consider a linear model with additive zero-mean Gaussian noise,

$$Z = cx + d + V. \tag{2.15}$$

If the measurements are unknown, the likelihood function is

$$L(x|Z) = \mathcal{N}\left(Z - cx - d, \sigma_V^2\right). \tag{2.16}$$

This has a maximum value at $\hat{X}$ called the *maximum likelihood estimator*[4]. A capital X to denote that $\hat{X}$ is a random variable since it is a function of the random variable Z.

It can be shown that the ML estimator for a linear model is equivalent[5] to

$$\hat{X} = \frac{Z - d}{c}. \tag{2.17}$$

This has an expected value

$$\mathrm{E}\left[\hat{X}\right] = \mathrm{E}\left[x + V\right] = x, \tag{2.18}$$

and thus the estimator is unbiased. The estimator has a variance

$$\mathrm{Var}\left[\hat{X}\right] = \frac{1}{c^2}\mathrm{Var}\left[Z\right] = \frac{1}{c^2}\mathrm{Var}\left[V\right]. \tag{2.19}$$

[4] Warning, potential notation confusion. $\hat{x}$ is called an estimate. It depends on a measured value z. $\hat{X}$ is called an estimator. It depends on a random variable value Z.

[5] By differentiating the log-likelihood with respect to the parameter.

2.3 Exercises

1. Emily's UFO has a altitude sensor where the output voltage (in V) can be modelled by a random variable Z, where
$$Z = 2x + 1 + V,$$
for an unknown altitude x (in km) with additive zero-mean Gaussian random noise, represented by the random variable V with a standard deviation $\sigma_V = 2\,\text{V}$.

   (a) If the sensor outputs a value $z = 9$, sketch the likelihood function for the sensor.

   (b) If the sensor outputs a value $z = 9$, determine the maximum likelihood estimate of x.

2. Sam's UFO has an altitude sensor where the output voltage (in V) can be modelled by a random variable Z, where
$$Z = 3x^2 + V,$$
x is an unknown altitude (in km), and V is a zero-mean Gaussian random variable with a standard deviation $\sigma_V = 3\,\text{V}$.

   (a) If the sensor outputs a value $z = 3$, determine an expression for the sensor's likelihood function.

   (b) If the sensor outputs a value $z = 3$, determine the maximum likelihood estimate for the altitude?

   (c) If a measurement of Z is $z = 3\,\text{V}$, and a prior estimate for x is 1 km, determine the variance of the ML altitude inferred from the measurement. Hint, consider the estimator.

2.4 Advanced exercises

1. Show that the ML estimator for the model $Z = cx + d + V$ is $\hat{X} = (Z - d)/c$ if V is a zero mean Gaussian random variable. Hint, differentiate the log of the likelihood function with respect to X to find where the maximum is.

2. Determine the ML estimator for the model $Z = k/x + V$ if V is a zero mean Gaussian random variable.

3
Sensor fusion

Sensor fusion is commonly used in mobile robotics to combine (fuse) measurements from different sensors. The resulting estimate has a lower variance than the individual measurements. An example, is the fusing of measurements from an inertial measurement unit[1] (IMU) with a global positioning system (GPS). To understand how a better estimate is obtained, it is necessary to consider some probability and statistics, in particular, estimation theory.

[1] These contain accelerometers, gyroscopes, and magnetometers.

3.1 MLE for multiple measurements

Consider two sensors, with linear models and additive noise,

$$Z_1 = x + V_1, \tag{3.1}$$
$$Z_2 = x + V_2. \tag{3.2}$$

We can evaluate the likelihood of the unknown state, x, using $Z_1 = z_1$ and $Z_2 = z_2$. This is described by the joint likelihood function, $L(x|z_1, z_2)$. If the *errors are independent*, then this is simply the product of the likelihood functions for each measurement,

$$L(x|z_1, z_2) = L(x|z_1)L(x|z_2). \tag{3.3}$$

Note, multiplying likelihood functions together produces a narrower likelihood function and thus a better estimator.

3.2 Best linear unbiased estimate (BLUE)

If both sensors have Gaussian distributed zero mean noise, the joint likelihood function is

$$L(x|z_1, z_2) = \mathcal{N}\left(x - z_1, \mathrm{Var}\left[V_1\right]\right)\mathcal{N}\left(x - z_2, \mathrm{Var}\left[V_2\right]\right). \tag{3.4}$$

The product of two Gaussians is also a Gaussian and it can be shown that the joint maximum likelihood estimate is the weighted sum of the maximum likelihood estimate of each sensor,

$$\hat{X} = w_1 \hat{X}_1 + w_2 \hat{X}_2. \tag{3.5}$$

Here w_1 and w_2 are weighting factors. To ensure the estimator is unbiased, the weights must sum to unity, and thus

$$\hat{X} = w_1 \hat{X}_1 + (1 - w_1) \hat{X}_2. \tag{3.6}$$

If the sensor errors are uncorrelated, the variance of the estimator can be shown to be

$$\operatorname{Var}\left[\hat{X}\right] = w_1^2 \operatorname{Var}\left[V_1\right] + (1 - w_1)^2 \operatorname{Var}\left[V_2\right], \tag{3.7}$$

and thus the best linear unbiased (BLU) estimator[2] can be found to occur where the weights are chosen inversely proportional to the variances[3]. This yields

$$\hat{X}_{\text{BLU}} = \frac{\frac{1}{\operatorname{Var}[V_1]} \hat{X}_1 + \frac{1}{\operatorname{Var}[V_2]} \hat{X}_2}{\frac{1}{\operatorname{Var}[V_1]} + \frac{1}{\operatorname{Var}[V_2]}}, \tag{3.8}$$

with a variance [4]

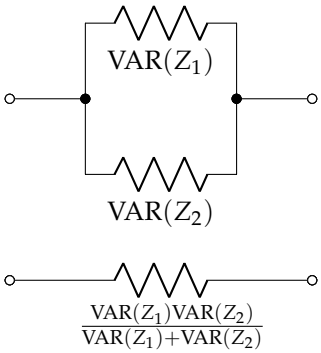
$$\operatorname{Var}\left[\hat{X}_{\text{BLU}}\right] = \frac{1}{\frac{1}{\operatorname{Var}[V_1]} + \frac{1}{\operatorname{Var}[V_2]}}, \tag{3.9}$$

$$= \frac{\operatorname{Var}\left[V_1\right] \operatorname{Var}\left[V_2\right]}{\operatorname{Var}\left[V_1\right] + \operatorname{Var}\left[V_2\right]}. \tag{3.10}$$

Note that this is smaller than $\operatorname{Var}[V_1]$ and $\operatorname{Var}[V_2]$ (see Figure 3.1).

[2] This is also called the minimum variance linear unbiased (MVLU) estimator.
[3] Provided the measurements are uncorrelated.

[4] This result is analogous to two resistors in parallel.



3.2.1 *BLUE sensor fusion example*

Let's recap the sensor fusion problem introduced in Section 1.1. The observation model for this problem is

$$Z_1 = x + V_1, \tag{3.11}$$

$$Z_2 = x + V_2, \tag{3.12}$$

where:

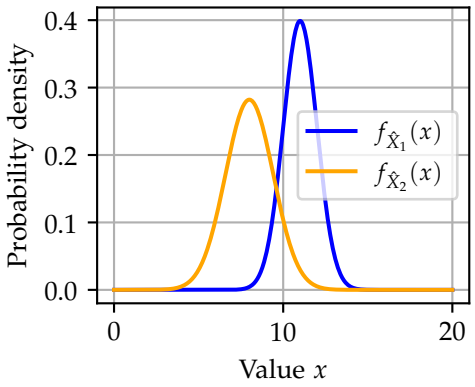
x is the constant but unknown position of the robot to estimate,

Z_1 is a random variable describing the laser sensor measurement,

V_1 is a random variable describing the laser sensor uncertainty,

Z_2 is a random variable describing the ultrasound sensor measurement,

V_2 is a random variable describing the ultrasound sensor uncertainty.



It is assumed that the sensors are calibrated so that V_1 and V_2 are zero mean. It is also assumed that the sensor errors are uncorrelated.

Since both sensors are linear in x, the maximum likelihood estimators are

$$\hat{X}_1 = Z_1, \tag{3.13}$$
$$\hat{X}_2 = Z_2. \tag{3.14}$$

Let's say that the laser measures distance with a standard deviation $\sigma_{V_1} = 1\,\text{m}$ and the ultrasonic sensor measures distance with a standard deviation $\sigma_{V_2} = 2\,\text{m}$? The BLU estimator using these sensors is

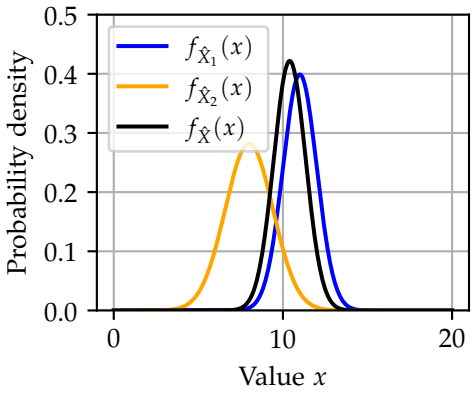
$$\hat{X}_{\text{BLU}} = \frac{\frac{1}{\text{Var}[V_1]}\hat{X}_1 + \frac{1}{\text{Var}[V_2]}\hat{X}_2}{\frac{1}{\text{Var}[V_1]} + \frac{1}{\text{Var}[V_2]}}, \tag{3.15}$$
$$= \frac{4\hat{X}_1 + \hat{X}_2}{5}, \tag{3.16}$$
$$= 0.8\hat{X}_1 + 0.2\hat{X}_2, \tag{3.17}$$

with a variance

$$\text{Var}\left[\hat{X}_{\text{BLU}}\right] = \frac{1}{\frac{1}{\text{Var}[V_1]} + \frac{1}{\text{Var}[V_2]}}, \tag{3.18}$$
$$= \frac{1}{\frac{1}{1} + \frac{1}{4}}, \tag{3.19}$$
$$= 0.8, \tag{3.20}$$

and a standard deviation

$$\sigma_{\hat{X}_{\text{BLU}}} = \sqrt{\text{Var}\left[\hat{X}_{\text{BLU}}\right]}, \tag{3.21}$$
$$= \sqrt{0.8}, \tag{3.22}$$
$$= 0.89. \tag{3.23}$$



Note, this is smaller than both σ_{V_1} and σ_{V_2} but there is not a large improvement.

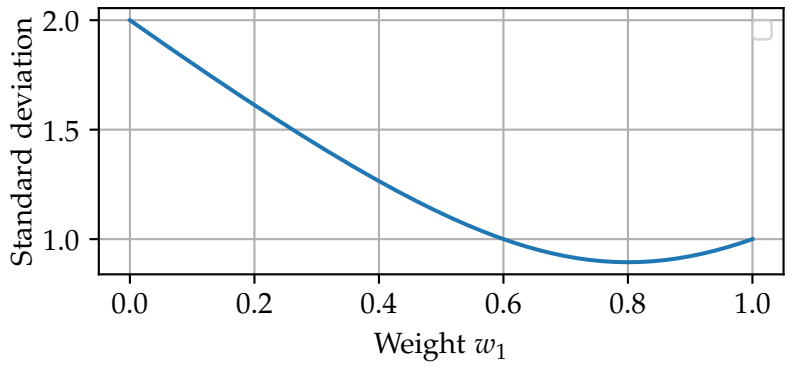


Figure 3.1: Standard deviation of weighted sum as a function of weight w_1.

So with a measured value $z_1 = 11$ for Z_1 and $z_2 = 8$ for Z_2, then the BLU estimate is

$$\hat{x}_{\text{BLU}} = \frac{\frac{1}{\text{Var}[V_1]}z_1 + \frac{1}{\text{Var}[V_2]}z_2}{\frac{1}{\text{Var}[V_1]} + \frac{1}{\text{Var}[V_2]}}, \tag{3.24}$$

$$= 0.8 \times 11 + 0.2 \times 8, \tag{3.25}$$

$$= 10.4\,\text{m}. \tag{3.26}$$

Note, the estimate may result in a poorer value than one of the measured values but on average it will be better.

σ_{V_1}	σ_{V_2}	w_1	w_2	$\sigma_{\hat{X}}$
1.00	1.00	0.50	0.50	0.71
1.00	2.00	0.80	0.20	0.89
1.00	3.00	0.90	0.10	0.95
1.00	4.00	0.94	0.06	0.97
1.00	5.00	0.96	0.04	0.98

Table 3.1: Optimal weights for the linear weighted sum of two random variables. Note, the resulting standard deviation, $\sigma_{\hat{X}}$, is smaller than the standard deviations of the two random variables but there is diminishing returns.

3.2.2 *General result*

In general, if N independent random variables are added together, the best linear unbiased estimator, $\hat{Z}_{\text{BLU}}$, is found from

$$\hat{Z}_{\text{BLU}} = \sum_{i=1}^{N} w_i Z_i, \tag{3.27}$$

$$= \frac{\sum_{i=1}^{N} \frac{1}{\text{Var}[Z_i]} Z_i}{\sum_{i=1}^{N} \frac{1}{\text{Var}[Z_i]}}, \tag{3.28}$$

with a variance

$$\text{Var}\left[\hat{Z}_{\text{BLU}}\right] = \frac{1}{\sum_{i=1}^{N} \frac{1}{\text{Var}[Z_i]}}. \tag{3.29}$$

3.3 Summary

1. Sensor fusion combines measurements from disparate sources to improve the estimate (for example, GPS and IMU).
2. Ideally we would like a MVU estimator but this is not always possible to determine.
3. A BLU estimator assumes that the sensor noise is additive, zero-mean (i.e., the sensors are calibrated), and uncorrelated between sensors.
4. The weights for a BLU estimator are inversely proportional to the variances and normalised so they sum to unity.
5. If the sensor model is linear and the noise is additive with a Gaussian PDF, then the BLU estimator is a MVU estimator.
6. See `http://132.181.50.21:8000` for interactive demos.

3.4 Exercises

1. An ultrasonic sensor with a variance of $0.01\,\mathrm{m}^2$ measures the range to a wall to be 1.2 m. A lidar with a variance $0.01\,\mathrm{m}^2$ measures the range to the wall to be 1.1 m.
   (a) If the measurements are to be fused using a linear weighted sum, what should the weightings be?
   (b) If the measurements are to be fused using a linear weighted sum, what is the resulting variance?
2. An ultrasonic sensor with a variance of $0.04\,\mathrm{m}^2$ measures the range to a wall to be 1.2 m. A lidar with a variance $0.01\,\mathrm{m}^2$ measures the range to the wall to be 1.1 m.
   (a) If the measurements are to be fused using a linear weighted sum, what should the weightings be?
   (b) If the measurements are to be fused using a linear weighted sum, what is the resulting variance?
3. Consider two range sensors: one that samples at 1 Hz with a variance of $0.01\,\mathrm{m}^2$ and another that samples at 10 Hz with a variance of $0.05\,\mathrm{m}^2$. Which would you choose?

4
Prior information and dead-reckoning

With estimation problems we often have some prior information about the state being estimated. For example, if a robot starting at the origin moves at 3 m/s for 3 s, we would expect that it would move 9 m. Of course, something may go wrong. The bateries may go flat, the wheels may slip, it may crash into something, or it may be kidnapped. So there is always some uncertainty.

Prior information improves the estimate. In some situations, it simply can be fused with the measurements using a weighted average. The trick is to represent the unknown parameter, x, being estimated as a random variable X[1].

[1] At the expense of some notational confusion!

In probabilistic robotics, it is common to refer to X as the belief of the robot's state. Using a random variable allows the belief of the prior information to be represented as a PDF.

4.1 Beliefs

Most of the time, beliefs have a Gaussian distribution since this only requires two parameters: mean and variance. However, there are many other possible distributions.

4.1.1 Uniform distribution

If the state is known be uniformly distributed between $x_{\min}$ and $x_{\max}$ then X is a uniform random variable, with a PDF,

$$f_X(x) = \begin{cases} 0 & x < x_{\min}, \\ \frac{1}{x_{\max} - x_{\min}} & x_{\min} \leq x \leq x_{\max}, \\ 0 & x > x_{\max}. \end{cases} \tag{4.1}$$

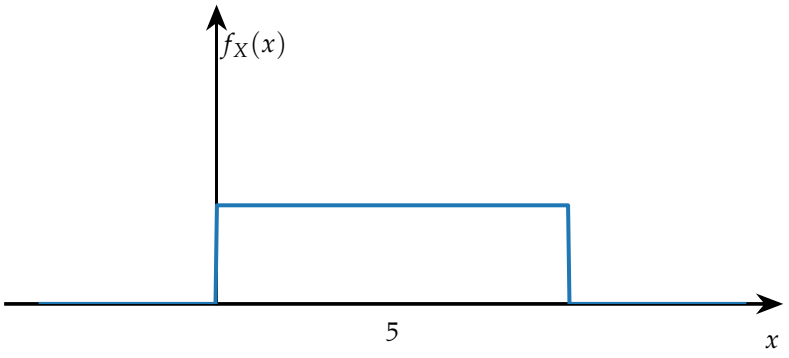


Figure 4.1: Uniform distribution.

The shorthand notation that a random variable, X, is uniformly distributed between $x_{\min}$ and $x_{\max}$ is

$$X \sim \mathcal{U}\left(x_{\min}, x_{\max}\right). \tag{4.2}$$

The uniform distribution can be roughly approximated by a Gaussian PDF by only considering the first two moments: the mean and variance,

$$\mu_X = \frac{x_{\min} + x_{\max}}{2}, \tag{4.3}$$

$$\sigma_X^2 = \frac{\left(x_{\max} - x_{\min}\right)^2}{12}. \tag{4.4}$$

4.1.2 *Multiple hypotheses*

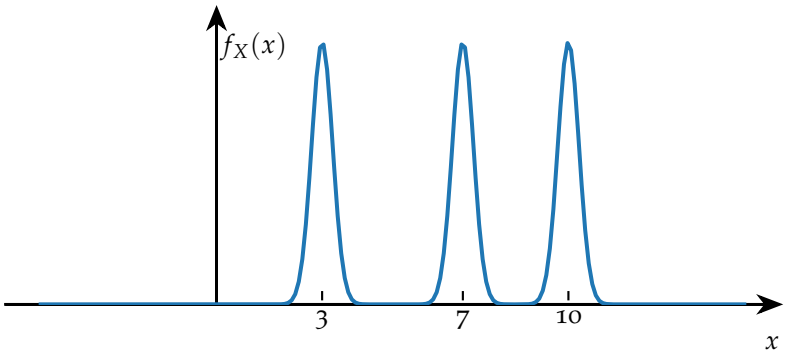


Figure 4.2: Multimodal distribution.

A belief with multiple hypotheses has as a mode for each hypothesis. For example, consider a robot passing through a door into a hallway with three doors. The position of the robot along the hallway is either one of the three doorways. One way of modelling multiple hypotheses is a weighted sum of Gaussian distributions.

4.2 Dead reckoning

When sensor measurements are unavailable, the belief of the state of a robot can be updated using dead reckoning. This requires a motion model[2] for the robot and an initial belief of the state. Unfortunately, errors propagate with time and so the belief becomes less certain.

[2] Also known as a transition or system model.

4.2.1 Linear motion model with additive noise

A linear 1-D motion model can be used to predict the position of a robot constrained to a rail. Denoting the robot's unknown position at time-step n by X_n, and the speed commanded to the robot's motor by u_n, then

$$X_n = X_{n-1} + u_{n-1}\Delta t + W_n. \tag{4.5}$$

Here Δt is the duration of a time-step and W_n models the process noise, i.e., the uncertainty in the model due to wheel slippage or other disturbances.

In general, a linear 1-D motion model with additive zero-mean process noise has the form[3],

$$X_n = aX_{n-1} + bu_{n-1} + W_n, \tag{4.6}$$

where a and b are constants. This assumes a 1-D control u_n.

[3] Note, (4.5) can be seen to have the form of (4.6) with $a = 1$ and $b = \Delta t$.

4.2.2 Propagation of uncertainty

The motion model is a form of prediction and the effect of the process noise increases the uncertainty of the prediction. If X_{n-1} and W_n are independent, then

$$\operatorname{Var}[X_n] = a^2 \operatorname{Var}[X_{n-1}] + \operatorname{Var}[W_n]. \tag{4.7}$$

Thus for every iteration of the model, the uncertainty increases[4]. This is depicted in Figure 4.3 to Figure 4.6.

[4] This is the disadvantage of dead-reckoning where measurements are not used.

4.2.3 General motion models

In general, a motion model with additive process noise has the form,

$$X_n = g(X_{n-1}, u_{n-1}) + W_n, \tag{4.8}$$

where g is a function of the previous state, X_{n-1}, and the known previous control, u_{n-1}, and W_n is the zero-mean random variable that represents the uncertainty in the motion model (the process noise). The motion model is found by fitting to a histogram obtained from many repeated measurements.

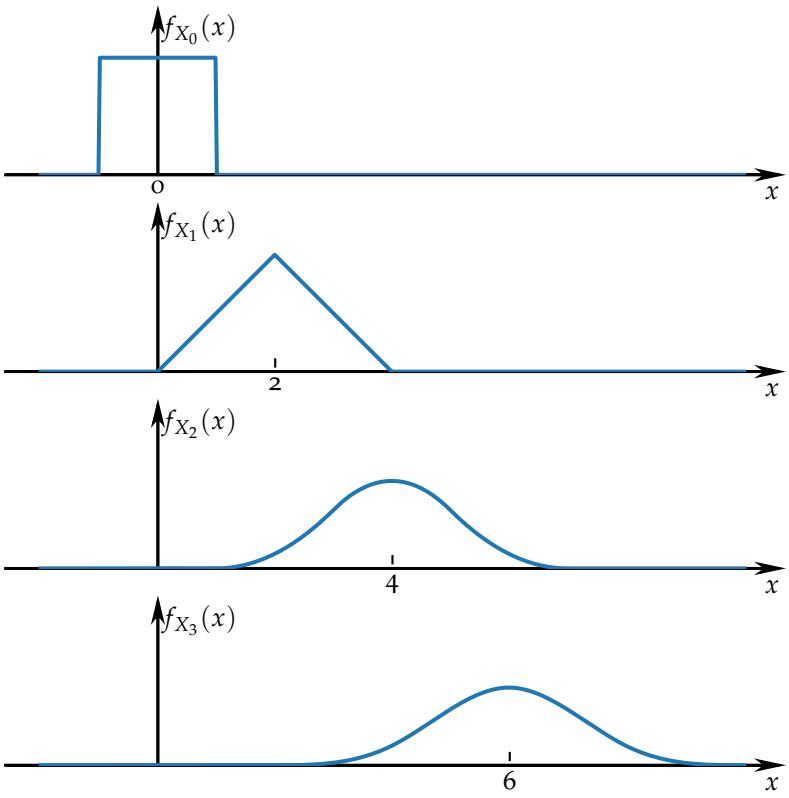


Figure 4.3: Propagation of state belief with no measurements: uniform initial belief distribution and linear motion model with additive uniform noise.

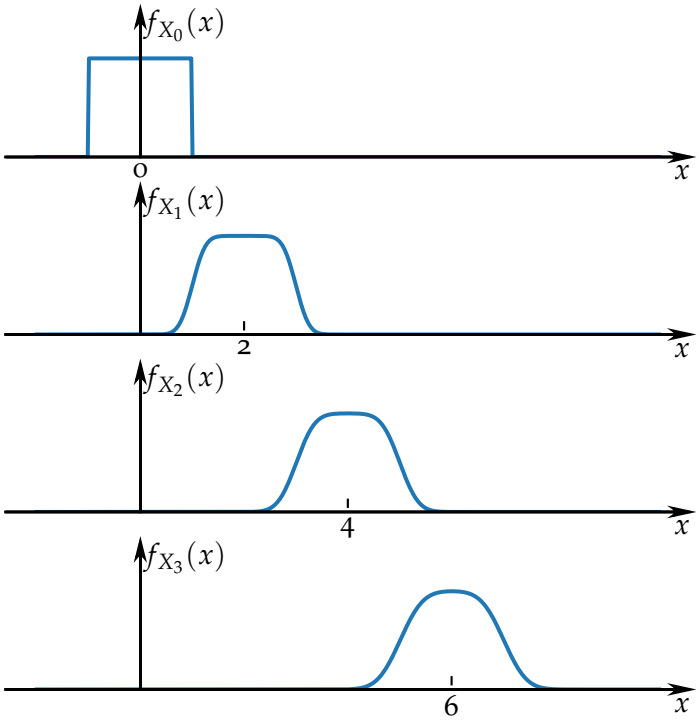


Figure 4.4: Propagation of state belief with no measurements: uniform initial belief distribution and linear motion model with additive Gaussian noise.

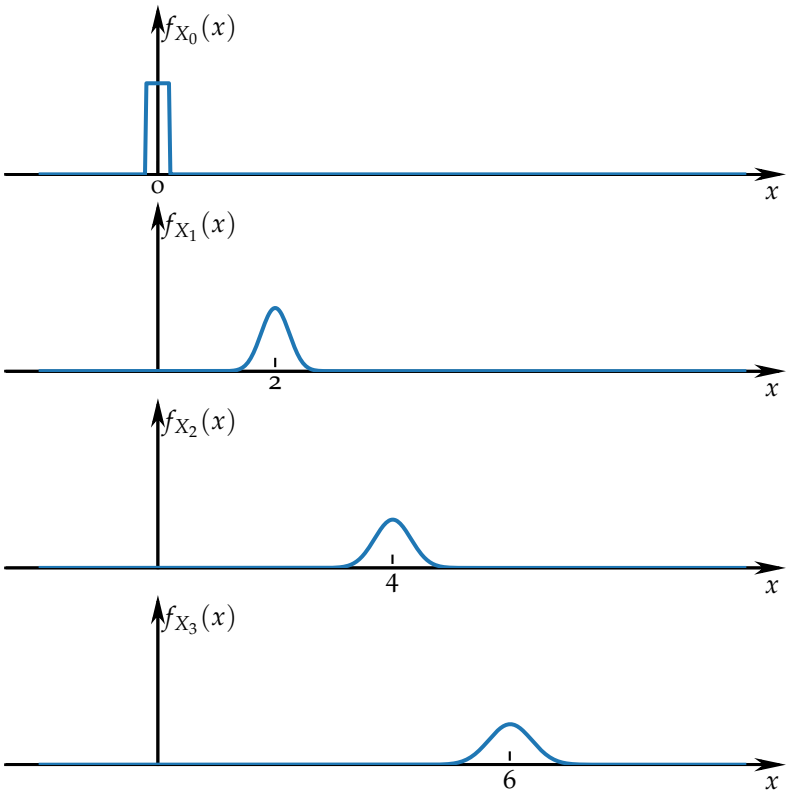


Figure 4.5: Propagation of state belief with no measurements: uniform initial belief distribution and linear motion model with additive Gaussian noise.

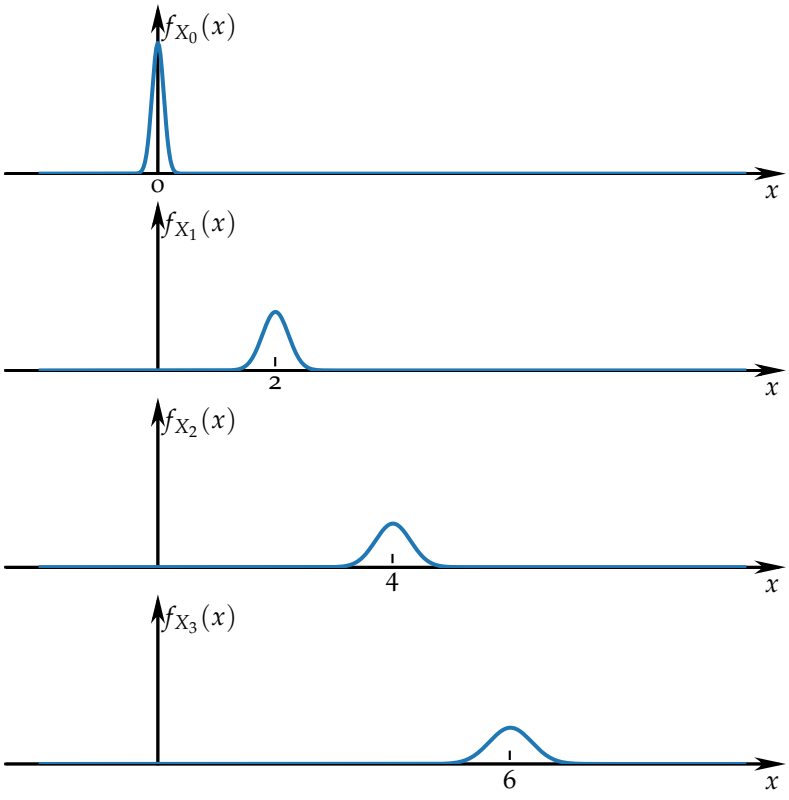


Figure 4.6: Propagation of state belief with no measurements: Gaussian initial belief distribution and linear motion model with additive Gaussian noise.

4.3 MLE with prior information

For this simple 1-D robot problem, the position of the robot's position can be inferred from the measurements. But we also have some prior information; the robot started at position 0 and was told to move 9 m. This prior information can be fused with the position estimated from the measurements using a weighted sum[5],

[5] This is the core of a Kalman filter.

$$X|Z = K\hat{X} + (1 - K)\, X, \tag{4.9}$$

where

X is the *prior belief*[6],
$\hat{X} = \hat{X}(Z)$ is the *measurement estimator*,
$X|Z$ is the *posterior belief*[7], and
K is a weighting factor.

[6] Before the measurement. From Latin *a priori*: from what is before.
[7] After the measurement. From Latin *a posteriori*: from later.

Note the weights sum to unity as required for an unbiased estimator. For a BLU estimator, the weights should be inversely proportional to the variances of the measured and prior estimators, so

$$K = \frac{\frac{1}{\mathrm{Var}\left[\hat{X}\right]}}{\frac{1}{\mathrm{Var}\left[\hat{X}\right]} + \frac{1}{\mathrm{Var}[X]}}. \tag{4.10}$$

The variance of the resulting estimate is

$$\mathrm{Var}\,[X|Z] = \frac{1}{\frac{1}{\mathrm{Var}\left[\hat{X}\right]} + \frac{1}{\mathrm{Var}[X]}}. \tag{4.11}$$

For example, from fusing the measurements,

$$\hat{x} = 10.4\,\mathrm{m}, \tag{4.12}$$
$$\mathrm{Var}\left[\hat{X}\right] = 0.8\,\mathrm{m}^2. \tag{4.13}$$

We also know that the robot was told to move 9 m so,

$$\mathrm{E}\,[X] = 9. \tag{4.14}$$

But how sure are we that the robot moved 9 m? Perhaps the wheels slipped? To handle this uncertainty, we need the variance of the prior. Let's say that

$$\mathrm{Var}\,[X] = 1.2\,\mathrm{m}^2, \tag{4.15}$$

and so from (4.10)

$$K = \frac{\frac{1}{0.8}}{\frac{1}{0.8} + \frac{1}{1.2}}, \tag{4.16}$$
$$= 0.6. \tag{4.17}$$

Now using (4.9),

$$\mathrm{E}[X|Z=z] = 0.6 \times 1.4 + (1-0.6) \times 9, \tag{4.18}$$
$$= 9.94\,\mathrm{m}, \tag{4.19}$$

with a variance calculated from (4.11),

$$\mathrm{Var}[X|Z] = \frac{1}{\frac{1}{0.8} + \frac{1}{1.2}}, \tag{4.20}$$
$$= 0.47\,\mathrm{m}^2. \tag{4.21}$$

Note, this is smaller than both the variances of the measured estimate and prior estimate.

4.4 Summary

1. The belief of a state is represented by a random variable X, with a PDF $f_X(x)$.
2. The belief of a state can be predicted with a motion model from a previous belief (dead-reckoning).
3. Dead-reckoning causes the variance of the belief to increase.
4. The PDF of the sum of two independent random variables is given by a convolution of their PDFs:

$$f_{X+Y}(z) = \int_{-\infty}^{\infty} f_X(x) f_Y(z-x)\mathrm{d}x \quad X \text{ and } Y \text{ independent.} \tag{4.22}$$

5. The sum of two independent Gaussian random variables is a Gaussian random variable[8],

$$X + Y \sim \mathcal{N}\left(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2\right). \tag{4.23}$$

6. Under certain conditions, prior information can be included using a weighted average.

[8] A Gaussian PDF convolved with a Gaussian PDF produces a Gaussian PDF with a larger variance.

4.5 Exercises

1. Sketch the PDF of a uniform distribution with mean 5 and range 4.
2. Sketch the PDF of a uniform distribution with mean 5 and range 8.
3. Determine the variance of a uniform distribution with mean 5 and range 4.
4. Sketch the PDF of a Gaussian distribution with mean 5 and variance 1.
5. Sketch the PDF of a Gaussian distribution with mean 5 and variance 4.
6. Consider the sum of two independent random variables X and Y. Sketch the PDF of the result if X a uniform distribution with mean 5 and range 4 and Y has a uniform distribution with mean 5 and range 8.
7. Consider the average of two independent random variables X and Y. Sketch the PDF of the result if X a uniform distribution with mean 5 and range 4 and Y has a uniform distribution with mean 5 and range 8.
8. Sketch the approximate PDF of the average of ten independent random variables, each with a uniform distribution of mean 5 and range 4.
9. If X is a random variable described by a Gaussian distribution with mean 5 and variance 4, sketch the PDF of the random variable $Z = X/5$.
10. A initial belief of a robot's position at $n = 0$ can be described by a uniform distribution with a mean of 1 and a range of 2. If the robot's speed is a constant 1 m/s, sketch the belief of the robot's position for four time-steps assuming no process noise.
11. A initial belief of a robot's position at $n = 0$ can be described by a uniform distribution with a mean of 1 and a range of 2. If the robot's speed is a constant 1 m/s, roughly sketch the belief of the robot's position for four time-steps assuming zero-mean additive Gaussian process noise with a standard deviation of 0.5 m/s.
12. Consider Figure 4.3 and determine the mean speed of the robot.

13. Consider Figure 4.3 and sketch the conditional PDF for the motion model.

14. Consider Figure 4.6. If the standard deviation of the initial belief, X_0, is 0.1 and the standard deviation of the process noise is 0.2, determine the variance for X_n.

5
Bayesian estimation

Traditional estimation only uses measurements; the most common technique being maximum likelihood estimation. Bayesian estimation also includes prior information regarding the belief of the state.

The BLU estimator used in a Kalman filter is a Bayesian estimator but only if the beliefs are Gaussian. When the beliefs are not Gaussian, Bayes' theorem needs to be applied to correctly determine the posterior PDF.

5.1 Bayes' theorem

Bayes' theorem provides a more general method for fusing measurements with prior knowledge to obtain a posterior PDF. For a continuous random variable, X, Bayes' theorem states:

$$f_{X|Z}(x|z) = \frac{L_{X|Z}(x|z) f_X(x)}{f_Z(z)}. \tag{5.1}$$

This specifies the posterior PDF for $X = x$ given $Z = z$, where Z is the measurement random variable and z is a specific measurement.

There are four different quantities involved:

$f_{X|Z}(x|z)$ is the *posterior PDF* of X,

$L_{X|Z}(x|z)$ is the *measurement likelihood function*,

$f_X(x)$ is the *prior PDF* of X, and

$f_Z(z)$ is the *evidence PDF*.

The evidence, $f_Z(z)$, is not known. However, since it is independent of x, it can be replaced by a normalising factor, $1/\eta$, to ensure that the posterior PDF integrates to one,

$$f_{X|Z}(x|z) = \eta\, f_X(x) L_{X|Z}(x|z). \tag{5.2}$$

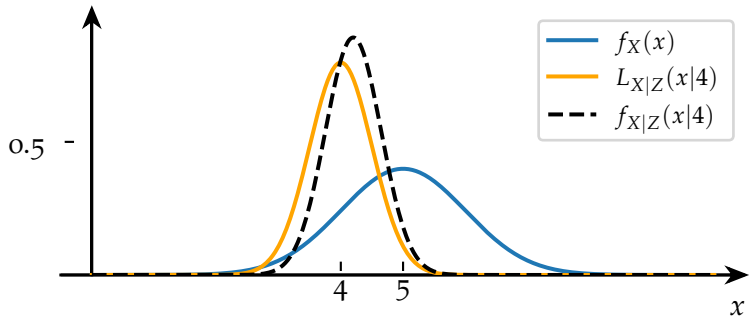
The scale factor, η, is calculated using

$$\eta = \frac{1}{\int_{-\infty}^{\infty} f_X(x) L_{X|Z}(x|z) \mathrm{d}x}. \tag{5.3}$$

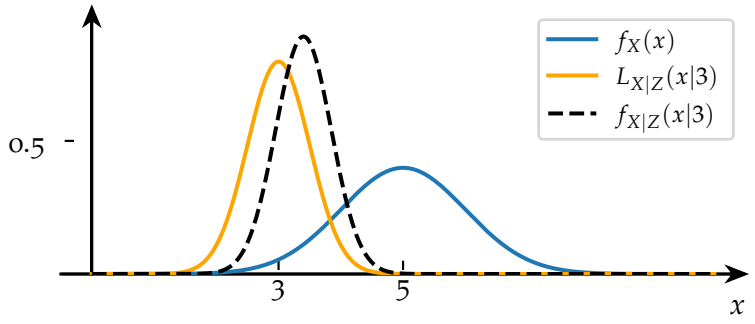
5.1.1 *Bayes' theorem example*

Figure 5.1 shows an example of applying Bayes' theorem. The blue curves show the prior PDF and the dashed black curves shows the posterior beliefs[1]. Note, these depend on the specific measurement value (the centre of the red curves).

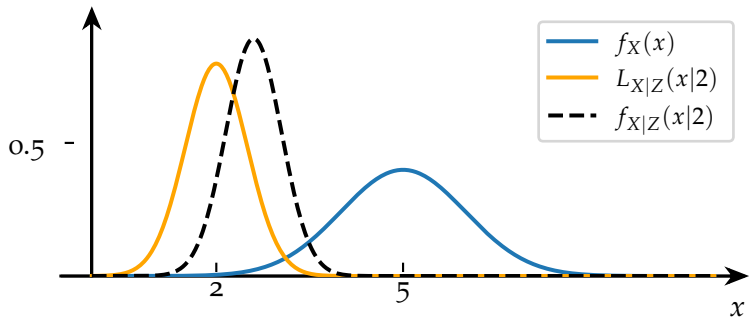
[1] The product of a Gaussian with a Gaussian is a narrower Gaussian.



(a)



(b)



(c)

Figure 5.1: Bayes' theorem example 1: (a) $z = 4$, (b) $z = 3$, (c) $z = 2$. In each case the posterior distribution, $f_{X|Z}(x|z)$, has the smallest variance. Note, the measured values only affect the mean of the posterior and not the variance.

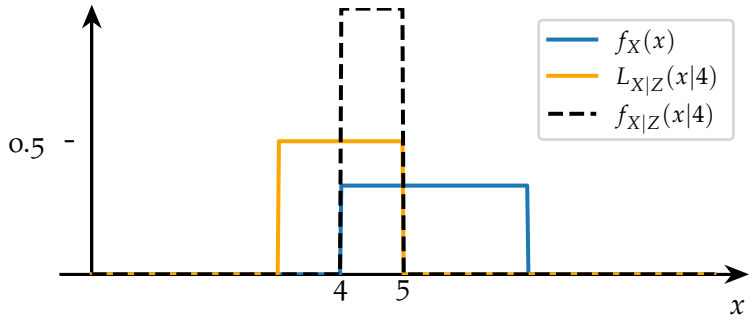


Figure 5.2: Bayes' theorem example 2: $z = 4$.

5.2 Bayesian estimates

The posterior PDF provides the probability density for each possible value of X. From this information we have to choose a best estimate. There are two common estimators for this: MAP and MMSE.

The *maximum a posteriori* (MAP) estimate is the mode of the posterior PDF:

$$\hat{x}_{\text{MAP}} = \arg\max_x f_{X|Z}(x|z). \tag{5.4}$$

Note, this is similar to the maximum likelihood (ML) estimate but modified by prior information.

The *minimum mean squared error* (MMSE) estimate is the mean of the posterior PDF:

$$\hat{x}_{\text{MMSE}} = \text{E}\left[X|Z=z\right] = \int_{-\infty}^{\infty} x f_{X|Z}(x|z)\text{d}x. \tag{5.5}$$

Note, if the posterior PDF is Gaussian, MAP and MMSE give the same estimate[2].

[2] This is what the Kalman filter tracks.

5.3 Robotics PDF notation

In robotics literature the state transition conditional probability density is often denoted as

$$p(x_n|x_{n-1}, z_n, u_{n-1}). \tag{5.6}$$

What they mean is

$$f_{X_n|X_{n-1},Z_n}(x_n|x_{n-1}, z_n; u_{n-1}). \tag{5.7}$$

Here x_n is a value of the random variable X_n conditioned by X_{n-1} and the measurement Z_n. u_{n-1} is a known parameter and not a random variable.

5.4 Bayes' theorem for probabilities

Bayes' theorem[3] is usually stated in probability form as

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}, \tag{5.8}$$

since

$$P(A \cap B) = P(A|B)\,P(B) = P(B|A)\,P(A). \tag{5.9}$$

Here $P(A|B)$ is the *conditional probability*[4] for A given B and $P(B|A)$ is the *conditional probability* for B given A.

[3] Bayes' rule is Bayes' theorem in odds form.

[4] A classic example of using conditional probabilities is the Monty Hall game show problem: *Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others a donkey. You pick a door, say No.1, and the host, who knows what's behind the doors, opens another door, say No.3, which has a donkey. He then says to you, "Do you want to pick door No.2?" Is it to your advantage to switch your choice?*

5.5 Summary

1. The PDF before a measurement is called the prior PDF.
2. The PDF after a measurement is called the posterior PDF.
3. The posterior PDF is computed from a prior PDF using Bayes' theorem and the likelihood function.
4. The posterior PDF has a smaller variance than the prior PDF.
5. Bayes' theorem for continuous random variables is

$$f_{X|Z}(x|z) = \frac{L_{X|Z}(x|z) f_X(x)}{f_Z(z)}. \tag{5.10}$$

6. Bayes' theorem for continuous random variables simplifies to:

$$f_{X|Z}(x|z) = \eta\, L_{X|Z}(x|z) f_X(x), \tag{5.11}$$

   where η is a scale factor to ensure the integral of the result is 1.

7. If both $f_X(x)$ and $L_{X|Z}(x|z)$ are Gaussian, then the posterior PDF, $f_{X|Z}(x|z)$, is also Gaussian[5],

$$f_{X|Z}(x|z) = \mathcal{N}\left(\frac{\text{Var}\left[Z|X\right]\text{E}\left[X\right] + \text{Var}\left[X\right]z}{\text{Var}\left[Z|X\right] + \text{Var}\left[X\right]}, \frac{\text{Var}\left[Z|X\right]\text{Var}\left[X\right]}{\text{Var}\left[Z|X\right] + \text{Var}\left[X\right]}\right). \tag{5.12}$$

[5] A Gaussian multiplied by a Gaussian produces a Gaussian.

6
Kalman filter

Dead-reckoning using a motion model works well for a while but becomes progressively less accurate. Measurements are required to improve the accuracy of the belief of the state. This lecture considers an iterative algorithm called a Kalman filter[1] that fuses measurements and predictions to estimate the state of a system, for example, the position of a robot.

[1] These were invented around 1960 and were used on the Apollo lunar lander.

6.1 *Overview*

The Kalman filter is a recursive[2] algorithm for estimating state on the fly. At its heart is a weighted average:

$$\hat{x}^+ = K\hat{x} + (1 - K)\,\hat{x}^-, \tag{6.1}$$

[2] This aspect leads to a notational nightmare that masks the simplicity of a Kalman filter.

where

$\hat{x}$ is the maximum likelihood estimate using the measurement z,

$\hat{x}^-$ is the prior estimate,

$\hat{x}^+$ is the posterior estimate,

K is the Kalman gain chosen to achieve a BLU estimate.

The Kalman filter requires:

1. A linear sensor model with additive Gaussian noise[3].
2. A Gaussian initial belief.
3. A linear motion model.
4. The complete state assumption[4].

[3] Thus the likelihood function is a Gaussian.

[4] This allows recursion by only tracking the mean and variance of the belief.

6.2 *Operation*

After initialisation, the Kalman filter iterates between predict and update steps.

6.2.1 *Initialisation*

The initial belief is a Gaussian distribution centred on the initial estimate $\hat{x}_0$:

$$X_0 \sim \mathcal{N}\left(\hat{x}_0, P_0\right). \tag{6.2}$$

This has two parameters:

$$\hat{x}_0 = \mathrm{E}\left[X_0\right], \tag{6.3}$$
$$P_0 = \mathrm{Var}\left[X_0\right]. \tag{6.4}$$

6.2.2 *Predict*

The initial belief is used to predict the belief at the next time-step using a linear motion model with additive process noise:

$$X_1^- = aX_0 + b + W_1. \tag{6.5}$$

Here X_1^- is the prior belief. This also has a Gaussian distribution

$$X_1^- \sim \mathcal{N}\left(\hat{x}_1^-, P_1^-\right), \tag{6.6}$$

where[5]

$$\hat{x}_1^- = a\hat{x}_0^- + b, \tag{6.7}$$
$$P_1^- = a^2 P_0 + \mathrm{Var}\left[W_1\right]. \tag{6.8}$$

[5] Note, the uncertainty increases.

6.2.3 *Update*

Measurements are made and the maximum likelihood estimate is found, $\hat{x}_1$, with a variance P_1. Assuming a single linear sensor with additive Gaussian noise,

$$Z_1 = cX_1 + d + V_1, \tag{6.9}$$

the maximum likelihood estimate for a measurement $Z_1 = z_1$ is

$$\hat{x}_1 = \frac{z_1 - d}{c}, \tag{6.10}$$

with a variance

$$P_n = \frac{\mathrm{Var}\left[V_n\right]}{c^2}. \tag{6.11}$$

Bayes' theorem is applied to fuse the maximum likelihood estimate with the prior estimate,

$$\hat{x}_1^+ = K_1\hat{x}_1 + \left(1 - K_1\right)\hat{x}_1^-. \tag{6.12}$$

This reduces the variance to,

$$P_1^+ = \frac{1}{\frac{1}{P_1^-} + \frac{1}{P_1}}. \tag{6.13}$$

The Kalman gain is chosen to minimise this variance:

$$K_1 = \frac{\frac{1}{P_1}}{\frac{1}{P_1^-} + \frac{1}{P_1}}. \tag{6.14}$$

The update step of the Kalman filter is shown in Figure 6.1. It can also be expressed[6] and in Figure 6.2.

[6] $Ka + (1-K)b = b + K(a-b)$. The factor in parentheses is called the *innovation*.

6.2.4 *Recursion*

For the next iteration, the posterior belief is used as the initial belief.

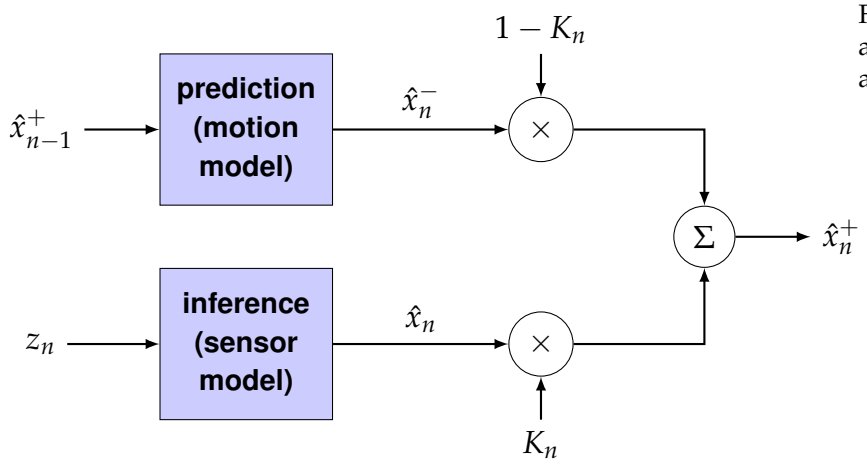


Figure 6.1: Block diagram of a Kalman filter as a weighted average.

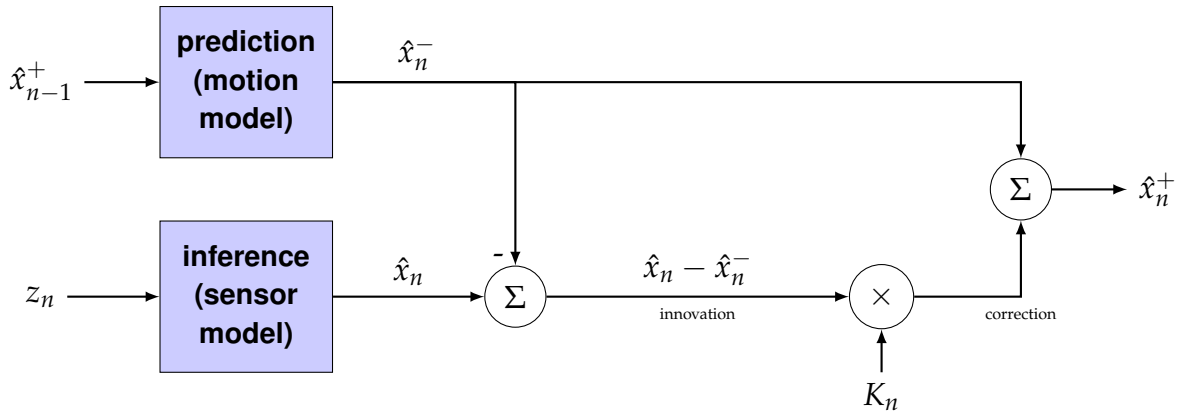


Figure 6.2: Block diagram variant of a Kalman filter as a weighted average.

6.3 Kalman filter implementation

The 1-D Kalman filter is simple to implement[7]. Since the beliefs are Gaussian, it needs only to track the mean and variance.

[7] See Listing 6.1.

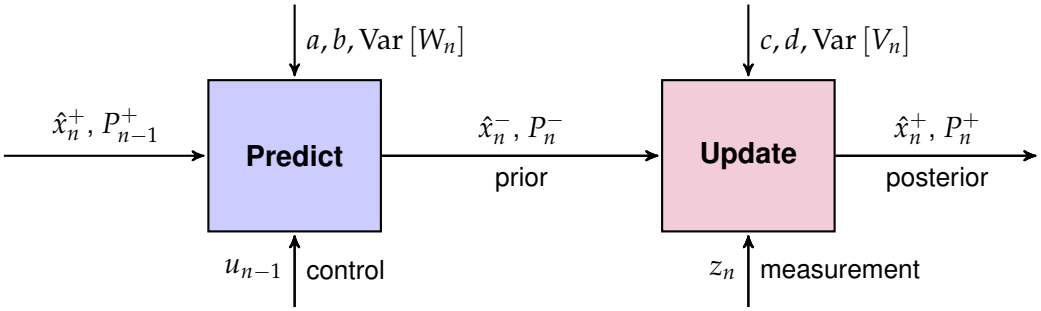


Figure 6.3: Kalman filter predict and update steps.

6.3.1 Comments

1. The Kalman filter requires:
   (a) A linear motion model with additive Gaussian noise.
   (b) A linear sensor model with additive Gaussian noise.
   (c) A Gaussian initial belief.
2. The Kalman filter is iterative and only needs to maintain the mean of current state belief, $\hat{x}_n^+$, and its variance P_n^+. All the previous measurements (and control actions) do not need to be stored.
3. The Kalman filter dynamically adjusts the Kalman gain depending on the variances of the state belief, process noise, and measurement noise.
4. If the process and measurement noise variances are constant then the Kalman gain will also converge to a constant called the *steady state Kalman gain*.
5. The Kalman gain is not affected by the measured values.

6.4 Worked example 1

Let's consider a 1-D tracking problem where a robot starts at a known position; 20 m from a wall. At time zero, the robot moves towards the wall at 1 m/s. It has an ultrasound sensor that measures the distance to

the wall. The goal of a Kalman filter is to estimate the robot's position, with respect to the starting position.

Let's assume that the sensor variance[8] is $\text{Var}[V_n] = 0.01\,\text{m}^2$, the process noise variance is $\text{Var}[W_n] = 0.04\,\text{m}^2$, and the initial belief variance is $\text{Var}[X_0] = 0.01\,\text{m}^2$. To keep the numbers simple, let's assume a time step, $\Delta t = 1\,\text{s}$.

[8] In practice, the range sensor will get more accurate as the robot gets closer to the wall.

The sensor model is

$$Z_n = 20 - X + V_n, \tag{6.15}$$

the motion model is

$$X_n = X_{n-1} + 1 + W_n, \tag{6.16}$$

and the initial belief is

$$X_0 = \mathcal{N}(0, 0.01). \tag{6.17}$$

Thus the initial estimate and its variance is

$$\hat{x}_0 = 0\,\text{m}, \tag{6.18}$$
$$P_0 = 0.01\,\text{m}^2. \tag{6.19}$$

Using the motion model, at the next time step the predicted belief is

$$X_1^- = X_0 + 1 + W_1. \tag{6.20}$$

Thus

$$\hat{x}_1^- = \hat{x}_0 + 1 = 1\,\text{m}, \tag{6.21}$$
$$P_1^- = 0.01 + 0.04 = 0.05\,\text{m}^2. \tag{6.22}$$

Note, the variance increases.

Let's say the sensor measures $z_1 = 19.1$. Using the sensor model, the maximum likelihood estimate of the robot's position is

$$\hat{x}_1 = 20 - 19.1 = 0.9\,\text{m}, \tag{6.23}$$

with a variance $P_1 = \text{Var}\left[\hat{X}_1\right] = 0.01\,\text{m}^2$.

The Kalman gain is

$$K_1 = \frac{\frac{1}{P_1}}{\frac{1}{P_1^-} + \frac{1}{P_1}}, \tag{6.24}$$
$$= \frac{\frac{1}{0.01}}{\frac{1}{0.05} + \frac{1}{0.01}}, \tag{6.25}$$
$$= 0.833. \tag{6.26}$$

Using this value, the posterior state estimate is

$$\hat{x}_1^+ = K_1\hat{x}_1 + (1 - K_1)\,\hat{x}_1^-, \tag{6.27}$$
$$= 0.833 \times 0.9 + 0.166 \times 1, \tag{6.28}$$
$$= 0.916\,\text{m}, \tag{6.29}$$

with a variance

$$P_1^+ = \frac{1}{\frac{1}{P_1^-} + \frac{1}{P_1}}, \tag{6.30}$$
$$= \frac{1}{\frac{1}{0.05} + \frac{1}{0.01}}, \tag{6.31}$$
$$= 0.0083\,\text{m}^2. \tag{6.32}$$

Note, how there has been a small improvement in the variance of the estimate.

6.5 Worked example 2

Let's apply a Kalman filter to the problem in Section 1.1, where the known information is[9]:

v	3
Δt	3
$z_{1,1}$	8
$\text{Var}\left[Z_{1,1}\right]$	1
$z_{2,1}$	11
$\text{Var}\left[Z_{2,1}\right]$	4

[9] $Z_{1,1}$ is a random variable for the laser range finder at time-step 1, $Z_{2,1}$ is a random variable for the ultrasound sensor at time-step 1. The lower case equivalents denote specific measurements.

Using the sensor model, the robot's position at time-step 1 can be estimated using:

$$\hat{x}_1 = \frac{\frac{1}{\text{Var}[Z_{1,1}]}z_{1,1} + \frac{1}{\text{Var}[Z_{2,1}]}z_{2,1}}{\frac{1}{\text{Var}[Z_{1,1}]} + \frac{1}{\text{Var}[Z_{2,1}]}}, \tag{6.33}$$
$$= \frac{\frac{1}{1} \times 8 + \frac{1}{4} \times 11}{\frac{1}{1} + \frac{1}{4}}, \tag{6.34}$$
$$= 0.8 \times 8 + 0.2 \times 11, \tag{6.35}$$
$$= 8.6\,\text{m}. \tag{6.36}$$

The variance of this estimate[10] is

[10] This does not depend on any realization.

$$\text{Var}\left[\hat{x}_1\right] = \frac{1}{\frac{1}{\text{Var}[Z_{1,1}]} + \frac{1}{\text{Var}[Z_{2,1}]}}, \tag{6.37}$$
$$= \frac{1}{\frac{1}{1} + \frac{1}{4}}, \tag{6.38}$$
$$= 0.8. \tag{6.39}$$

Let's assume that the initial position estimate is $\hat{x}_0 = 0$ with a variance $P_0 = 2\,\mathrm{m}^2$ and that the variance of the process noise is $\mathrm{Var}[W_1] = 0.4\,\mathrm{m}^2$. The motion model predicts that the robot moves to

$$\begin{aligned}\hat{x}_1^- &= \hat{x}_0 + v\Delta t, && (6.40)\\ &= 0 + 3 \times 3, && (6.41)\\ &= 9\,\mathrm{m}, && (6.42)\end{aligned}$$

with a variance

$$\begin{aligned}P_1^- &= P_0 + \mathrm{Var}[W_1], && (6.43)\\ &= 2 + 0.4, && (6.44)\\ &= 2.4\,\mathrm{m}^2. && (6.45)\end{aligned}$$

Combining the predicted position with the measurement position using a Kalman filter, the posterior estimate of the position is

$$\begin{aligned}\hat{x}_1^+ &= \frac{\frac{1}{P_1}\hat{x}_1 + \frac{1}{P_1^-}\hat{x}_1^-}{\frac{1}{P_1} + \frac{1}{P_1^-}}, && (6.46)\\ &= \frac{\frac{1}{0.8} \times 8.6 + \frac{1}{2.4} \times 9}{\frac{1}{0.8} + \frac{1}{2.4}}, && (6.47)\\ &= 0.75 \times 8.6 + 0.25 \times 9, && (6.48)\\ &= 8.7\,\mathrm{m}, && (6.49)\end{aligned}$$

with a variance

$$\begin{aligned}P_1^+ &= \frac{1}{\frac{1}{P_1} + \frac{1}{P_1^-}}, && (6.50)\\ &= \frac{1}{\frac{1}{0.8} + \frac{1}{2.4}}, && (6.51)\\ &= 0.6\,\mathrm{m}^2. && (6.52)\end{aligned}$$

Here the Kalman gain is

$$\begin{aligned}K_1 &= \frac{\frac{1}{P_1}}{\frac{1}{P_1} + \frac{1}{P_1^-}}, && (6.53)\\ &= \frac{\frac{1}{0.8}}{\frac{1}{0.8} + \frac{1}{2.4}}, && (6.54)\\ &= 0.75. && (6.55)\end{aligned}$$

Note, since the prior estimate of the robot's position is poor, a higher weighting is placed on the sensor's measurements.

6.6 Summary

1. The Kalman filter is an on-the-fly (online) iterative state estimator.
2. The Kalman gain dynamically weights the contribution of each new measurement.
3. The Kalman filter stores only the estimated mean state and the estimated variance[11].
4. The Kalman filter is only applicable for linear systems.
5. The Kalman filter requires the noise to be additive and Gaussian.

[11] Or covariance matrix for multidimensional problems.

6.7 Exercises

1. What is the difference between prior and posterior beliefs?
2. What is a sensor model used for?
3. What is a motion model used for?
4. What should the initial Kalman gain be? State any assumptions.
5. If the Kalman gain is zero, does a Kalman filter trust the model or the measurement?
6. What does a Kalman gain of unity imply?
7. Consider a system with a single state variable and a single measurement sensor. If the measurement variance is constant, sketch how the Kalman gain varies with increasing time-step.
8. If you require a robot to respond quickly to new measurements, what should the Kalman gain be close to?
9. The prior of a robot's position is that it is uniformly distributed between 0 and 10 m. What is the mean and the variance of the prior?
10. A robot that has been instructed to move at a speed of 2 m/s for 3 s. When it has stopped, an ultrasonic range sensor says that it moved 8 m, however, a laser range finder says that it moved 11 m. Determine the posterior estimate of the robot's position, and its variance, assuming that the ultrasonic range sensor has a variance of 0.02, the laser range sensor has a variance of 0.01, the initial estimate of the robot's position is 0 with a variance 0.02, and the process noise has a variance 0.002.
11. Consider a flying robot coming into land where it drops 2% of its altitude between time-steps. Is the motion model linear? Can the Kalman filter be used?
12. What value for the Kalman gain corresponds to dead reckoning?

```
# Kalman filter demonstration program to estimate
# position of a Dalek moving at constant speed.
# M.P. Hayes UCECE 2015
from numpy.random import randn
from numpy import zeros

# Initial position
x = 0
# Speed
v = 0.2

# Number of steps and time-step interval
Nsteps = 200
dt = 0.1

# Process and sensor noise standard deviations
std_W = 0.2 * dt
std_V = 0.1

# Process and measurement noise variances
var_W = std_W ** 2
var_V = std_V ** 2

# Simulate Dalek
x = zeros(Nsteps)
z = zeros(Nsteps)
for n in range(1, Nsteps):
    # Update simulated Dalek position
    x[n] = x[n - 1] + v * dt + randn(1) * std_W
    # Simulate measured range
    z[n] = x[n] + randn(1) * std_V

# Start with a poor initial estimate of Dalek's position
mean_X_post = 10
var_X_post = 10 ** 2

for n in range(1, Nsteps):
    # Calculate prior estimate of position and its variance (using motion model)
    mean_X_prior = mean_X_post + v * dt
    var_X_prior = var_X_post + var_W

    # Estimate position from measurement (using sensor model)
    x_infer = z[n]

    # Calculate Kalman gain
    K = var_X_prior / (var_V + var_X_prior)

    # Calculate posterior estimate of position and its variance
    mean_X_post = mean_X_prior + K * (x_infer - mean_X_prior)
    var_X_post = (1 - K) * var_X_prior
```

Listing 6.1: Python implementation of a Kalman filter for the robot position estimation problem.

7
Kalman filters for non-linear systems

Kalman filters represent the belief as a Gaussian distribution. This is numerically efficient since a Gaussian belief only requires two parameters[1]: the mean and variance.

Unfortunately, many motion and sensor models are non-linear. This distorts the belief so that it is no-longer Gaussian. One approach to overcome this difficulty it to linearise the motion (and/or sensor) model around the current estimate. This is used by the extended Kalman filter (EKF) and the unscented Kalman filter (UKF).

[1] A multivariate Gaussian distribution can be modelled by a vector of means and a covariance matrix, see Section 10.3.

7.1 Kalman filter for linear systems

The Kalman filter requires:

1. An initial Gaussian belief.
2. A linear motion model with additive zero-mean Gaussian distributed process noise.
3. A linear sensor model with additive zero-mean Gaussian distributed sensor noise.
4. The complete state assumption[2].

If all these conditions are met, then the PDF of the belief is always Gaussian[3].

A Kalman filter is not applicable to a non-linear system since the mapping of a Gaussian random variable by a non-linear function produces a random variable with a non-Gaussian PDF (see Figure 7.2). In general, the resulting PDF cannot be parameterised by a mean and a variance.

[2] So that the belief can be estimated recursively.

[3] A linear mapping of a Gaussian random variable is also a Gaussian random variable but with a possibly different mean and variance, see Figure 7.1.

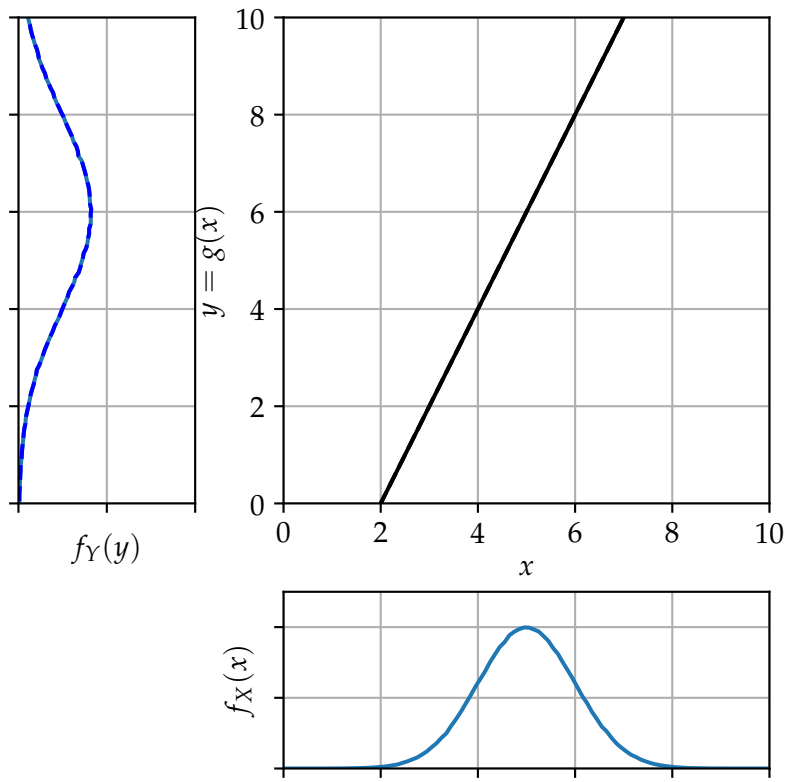


Figure 7.1: Error propagation with a linear function, $Y = g(X) = 2X - 4$. Note how a Gaussian distribution for X becomes a Gaussian distribution for Y.

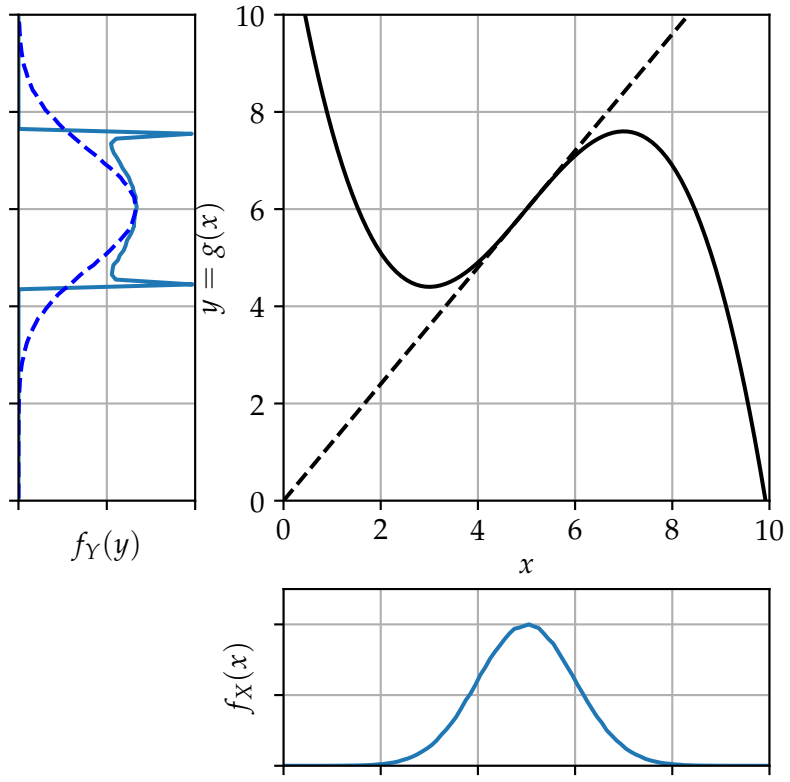


Figure 7.2: Error propagation with a non-linear function, $Y = g(X) = 1.2X - 0.1(X - 5)^3$. The dashed line is a linear approximation around $E[X] = 5$.

7.2 Non-linear system models

A non-linear motion model with additive process noise can be represented as[4],

$$X_n = g(X_{n-1}, u_{n-1}) + W_n. \tag{7.1}$$

[4] In practice, X_{n-1} would be the previous posterior belief, X^+_{n-1}, and X_n would be the prior belief, X^-_n.

Similarly, a non-linear sensor model with additive process noise can be represented as

$$Z_n = h(X_n, u_{n-1}) + V_n. \tag{7.2}$$

7.3 Extended Kalman filter (EKF)

The EKF linearises the non-linear motion and sensor models[5] at the mean of the current belief. Thus the resulting beliefs remain Gaussian.

The principle is simple:

[5] By considering the first two terms of a Taylor series expansion. See http://home.wlu.edu/~levys/kalman_nutorial/ for an example.

$$g(t) \approx g(t_0) + g'(t_0)(t - t_0), \tag{7.3}$$
$$\approx g'(t_0)t + g(t_0) - g'(t_0)t_0, \tag{7.4}$$

where g' is the derivative of g. But the notation gets messy...

7.3.1 Linearised motion model

The linearised motion model[6] is

[6] Note that parameters of the model change with each time step.

$$X_n \approx C_n X_{n-1} + D_n + W_n, \tag{7.5}$$

where

$$C_n = g'\left(\hat{x}_{n-1}, u_{n-1}\right), \tag{7.6}$$
$$D_n = g\left(\hat{x}_{n-1}, u_{n-1}\right) - g'\left(\hat{x}_{n-1}, u_{n-1}\right)\hat{x}_{n-1}, \tag{7.7}$$

and where

$$g'(\hat{x}_{n-1}, u_{n-1}) = \left.\frac{\partial g(X_{n-1}, u_{n-1})}{\partial X_{n-1}}\right|_{X_{n-1}=\hat{x}_{n-1}}. \tag{7.8}$$

Here $\hat{x}_{n-1}$ denotes the expected belief, $\mathrm{E}\left[X_{n-1}\right]$.

7.3.2 Linearised sensor model

The linearised sensor model is

$$Z_n \approx A_n X_n + B_n + V_n, \tag{7.9}$$

where

$$A_n = h'\left(\hat{x}_n, u_{n-1}\right), \tag{7.10}$$
$$B_n = h\left(\hat{x}_n, u_{n-1}\right) - h'\left(\hat{x}_n, u_{n-1}\right)\hat{x}_n, \tag{7.11}$$

and where

$$h'(\hat{x}_n, u_{n-1}) = \left.\frac{\partial h(X_n, u_{n-1})}{\partial X_n}\right|_{X_n=\hat{x}_n}. \tag{7.12}$$

Here $\hat{x}_n$ denotes $\mathrm{E}[X_n]$.

7.4 Unscented Kalman filter (UKF)

The extended Kalman filter gives poor performance for highly non-linear systems. It also requires explicit calculation of derivatives[7].

[7] Jacobians for multidimensional systems.

As an alternative, the unscented Kalman filter (UKF) models the Gaussian distribution by a set of *sigma points* around the mean. These sigma points are then mapped through the non-linear system to characterise the resulting PDF, again approximated as a Gaussian.

7.5 Multiple hypothesis tracking

A Gaussian PDF has a single mode and thus can only track a single hypothesis. For multiple hypotheses, a mixture (weighted sum) of Gaussians can be employed; one for each hypothesis. The resultant PDF is

$$f_X(x) = \sum_i \frac{w_i}{\sqrt{2\pi\sigma_{X_i}^2}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_{X_i}}{\sigma_{X_i}}\right)^2\right). \tag{7.13}$$

An example is shown in Figure 7.3.

Using a mixture of Gaussians, a multiple hypothesis Kalman filter (MHKF) or a multiple hypothesis extended Kalman filter (MHEKF) can be created. The latter linearises around the mean of each hypothesis.

7.6 Example: IR range sensor model

IR range sensors that measure distance, r, using triangulation output a voltage, z, where

$$z = \frac{k_1}{k_2 + r}, \tag{7.14}$$

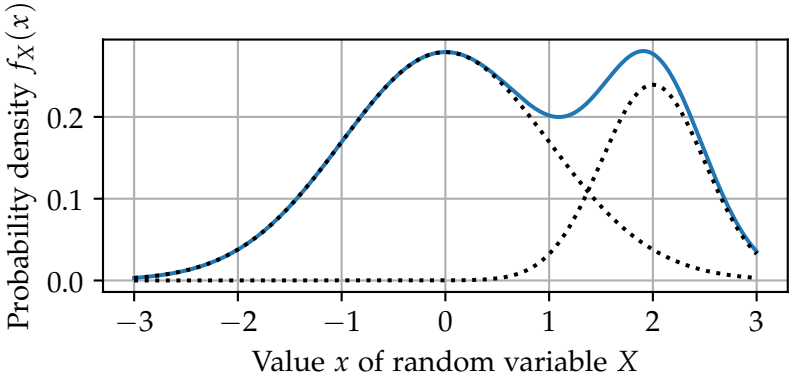


Figure 7.3: A weighted sum of two Gaussian distributions: $\mu_1 = 0$, $\sigma_1^2 = 1$, $w_1 = 0.3$, $\mu_2 = 2$, $\sigma_2^2 = 0.5$, $w_2 = 0.7$.

and where k_1 and k_2 are constants[8]. This equation only applies for $r > r_{\min}$, where $r_{\min}$ is a minimum range. In practice, there is some uncertainty of where the received light falls on the position sensitive detector and, in addition, there will be additive voltage noise. Thus if Z is a random variable denoting the output voltage, then

$$Z = \frac{k_1}{k_2 + R} + V, \tag{7.15}$$

where V is a random variable denoting measurement noise. This is likely to be Gaussian distributed, however, the model is non-linear and so the likelihood function for R will not be Gaussian distributed.

For example, let's denote the robot's state, X, as its distance from a wall. Thus the sensor model is

$$Z_n = \frac{k_1}{k_2 + X_n} + V_n, \tag{7.16}$$

and the measurement likelihood is

$$L_{X_n|Z_n}(x|z) = f_{V_n}\left(z - \frac{k_1}{k_2 + x}\right). \tag{7.17}$$

Even though V_n has a Gaussian distribution, the likelihood does not since it is a function of x_n.

If the sensor model is linearised, then the likelihood function has a Gaussian shape[9]. To see this, from (7.16), the gradient of Z_n with respect to X_n is

$$\frac{\partial Z_n}{\partial X_n} = -\frac{k_1}{(k_2 + Z_n)^2}, \tag{7.18}$$

and thus the linearised version of (7.16) around $X_n = \hat{x}_n^-$ is

$$Z_n \approx -\frac{k_1}{\left(k_2 + \hat{x}_n^-\right)^2}\left(X_n - \hat{x}_n^-\right) + \frac{k_1}{k_2 + \hat{x}_n^-} + V_n. \tag{7.19}$$

[8] To a good approximation $k_2 = 0$.

[9] Unlike a PDF, the likelihood function does not necessarily integrate to 1.

This is plotted as the dashed line in Figure 7.4. Note, the likelihood function for the linearised model has a Gaussian shape that approximates the true likelihood function. Also note, in both cases the maximum likelihood estimate is biased[10]. The maximum likelihood estimate is the mode of the likelihood function and is calculated using

$$\hat{x}_n = \arg\max_x L_{X_n|Z_n}(x|z_n). \tag{7.20}$$

[10] This is typical with a maximum likelihood estimate.

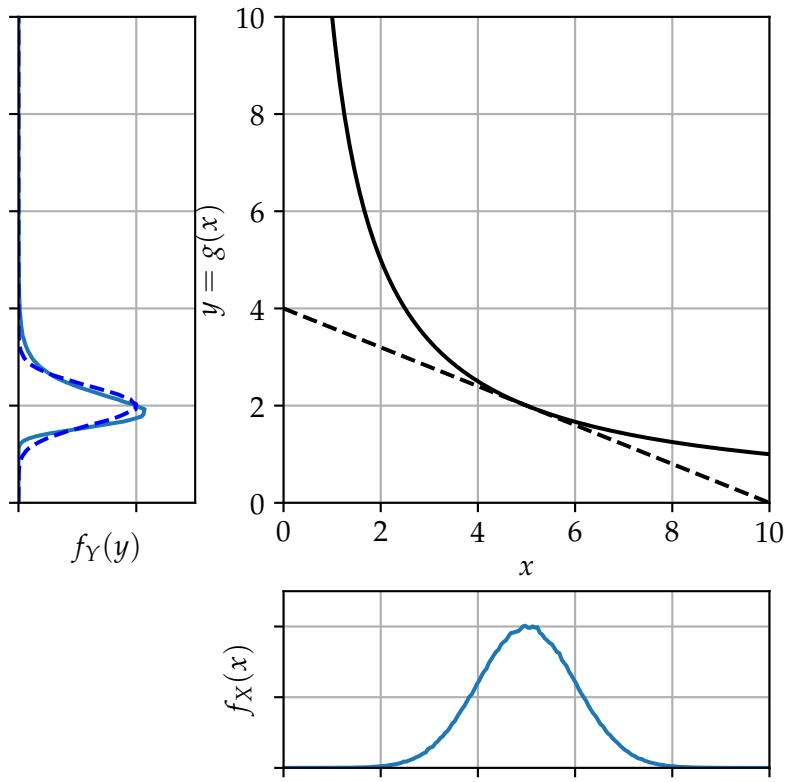


Figure 7.4: Error propagation with a non-linear function, $Y = g(X) = 10/X$.

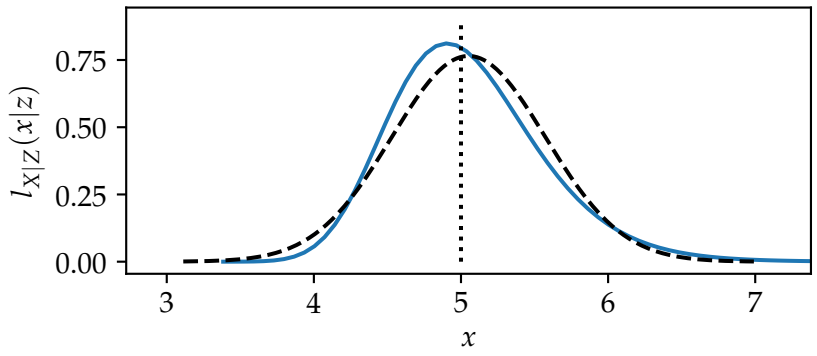


Figure 7.5: The likelihood function, $L_{X|Z}(x|z)$, for the IR sensor using (7.16) compared with the likelihood function for the linearised sensor model (7.19). Note for both approaches, the maximum likelihood estimate has a bias since the true value is 5.

7.7 Measurement likelihood for non-linear sensor model

If the noise is additive, the measurement likelihood is related to the sensor noise PDF by

$$L_{X_n|Z_n}(x|z) = f_{V_n}\left(z - h(x, u_{n-1})\right). \tag{7.21}$$

This is found from solving (7.2) for V_n.

Note, if the noise is not additive, an inverse model is required,

$$L_{X_n|Z_n}(x|z) = f_{V_n}\left(h^{-1}(x, z, u_{n-1})\right). \tag{7.22}$$

7.8 Summary

1. Kalman filters track only the mean and covariance of the estimated state and thus assume a Gaussian PDF.
2. When the system is non-linear, the posterior PDF is no longer Gaussian.
3. A non-linear system can be linearised around the mean of the belief with a Taylor series; this is the basis of the extended Kalman filter.
4. The unscented Kalman filter uses a different linearisation based on sigma points.
5. Multiple hypotheses can be tracked using a weighted sum of Gaussians (one per hypothesis).

7.9 Exercises

1. A random variable X is mapped to a variable Y by $Y = 3X + 2$. If the mean of X is 1 and the standard-deviation is 2, determine the mean and standard deviation of Y. Are any assumptions required?
2. A random variable X is mapped to a random variable Y by $Y = X^2 + 2$. If the mean of X is 1 and the standard-deviation is 2, determine the mean and standard deviation of Y. Are any assumptions required?
3. A Gaussian random variable X with mean 2 and variance 3 is mapped to a random variable Y by $Y = 2X - 1$. What is the PDF of Y?

4. A Gaussian random variable X with mean 2 and variance 3 is mapped to a random variable Y by $Y = 2X^2 - 1$. Is the PDF of Y Gaussian?

8
Bayes filters

Kalman filters are a special class of filter called a Gaussian Bayes filter and represent the posterior belief as a Gaussian distribution. There are other more generic Bayes filters that work with arbitrary posterior distributions, such as the histogram and particle filters.

All Bayes filters update the belief of the state using Bayes' theorem, starting from an initial belief, using probabilistic sensor and motion models[1]. With the complete state assumption, Bayes filters are recursive[2].

[1] Conceptually, a Bayes filter is straightforward but notationally are confusing.

[2] In a signal processing context, a recursive filter is defined in terms of a recurrence relation. In computer science, recursion is where a function calls itself.

8.1 Complete state (Markov) assumption

At each step of a Bayes filter, an initial belief is propagated through a motion model to find the prior belief. This depends on all the previous controls and previous measurements (but not including the current measurement). The posterior belief is then determined from the prior belief and the current measurement.

With the complete state assumption, it is not necessary to track all the previous controls and measurements. Only the previous belief is required. This considerably simplifies the computation and storage.

In general the prior belief is

$$X_n^- = X_n | X_{0:n-1}(u_{0:n-1}), Z_{0:n-1} = z_{0:n-1}. \tag{8.1}$$

The posterior belief also depends on the current measurement,

$$X_n^+ = X_n | X_{0:n-1}(u_{0:n-1}), Z_{0:n} = z_{0:n}. \tag{8.2}$$

With the complete state assumption the prior belief simplifies to

$$X_n^- = X_n | X_{n-1}(u_{n-1}), \tag{8.3}$$

and the posterior belief simplifies to

$$X_n^+ = X_n | X_{n-1}(u_{n-1}), Z_n = z_n. \tag{8.4}$$

8.2 Recursive Bayes filter

Bayes filters have two steps at each time-step: predict[3] and update[4]. Using the *complete state assumption*, Bayes filters can be implemented recursively.

[3] Sometimes called propagate.
[4] Sometimes called correct.

8.2.1 Predict step

The predict step requires finding the probability density of the prior. It is determined using

$$f_{X_n^-}(x_n) = \int_{-\infty}^{\infty} f_{X_n|X_{n-1}}(x_n|x_{n-1};u_{n-1})\, f_{X_{n-1}^+}(x_{n-1})\mathrm{d}x_{n-1}. \tag{8.5}$$

This considers all the possible ways a previous state, x_{n-1}, can become a current state, x_n.

8.2.2 Update step

The update step applies Bayes theorem to calculate the posterior PDF

$$f_{X_n^+}(x_n) = \eta\, L_{X_n|Z_n}(x_n|z_n) f_{X_n^-}(x_n). \tag{8.6}$$

8.2.3 Complete state assumption

A recursive Bayes filter requires the complete state (Markov) assumption. With this assumption

$$f_{X_n|X_{0:n-1}}(x_n|x_{0:n-1};u_{0:n-1}) = f_{X_n|X_{n-1}}(x_n|x_{n-1};u_{n-1}), \tag{8.7}$$

and

$$L_{X_{0:n}|Z_n}(x_{0:n}|z_n) = L_{X_n|Z_n}(x_n|z_n). \tag{8.8}$$

Without this assumption, computation of the belief becomes extremely complicated since all previous states, not just the last one, need to be considered.

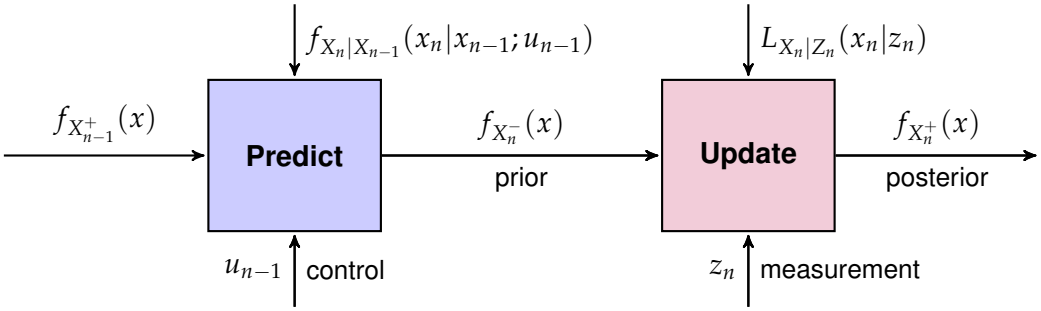


Figure 8.1: Recursive Bayes filter predict and update steps.

8.3 Bayes filter example

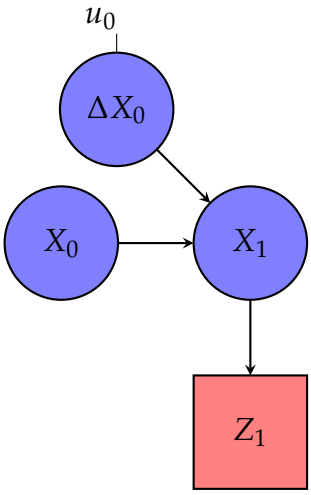


Figure 8.2: Bayesian network of a hidden Markov model (HMM) with controls. The random variable ΔX_0 denotes the probabilistic motion model parameterised by the desired speed u_0. The circles denote hidden random variables and the squares denote observable random variables.

Let's consider the estimation of a 1-D state using a single measurement and control output at each time-step. Given an initial belief, X_0, and the specified control, u_0, the prior belief, X_1, can be predicted using[5]

$$f_{X_1}(x_1; u_0) = \int_{-\infty}^{\infty} f_{X_1|X_0}(x_1|x_0; u_0)\, f_{X_0}(x_0)\mathrm{d}x_0. \tag{8.9}$$

[5] With a linear transition model, this integral is equivalent to a convolution and the result is a Gaussian if both functions being convolved are Gaussians.

The conditional PDF, $f_{X_1|X_0}(x_1|x_0; u_0)$, is derived from the transition (motion) model. For example, with the constant speed model,

$$X_1 = X_0 + u_0\Delta t + W_1, \tag{8.10}$$

then

$$f_{X_1|X_0}(x_1|x_0; u_0) = f_{W_1}(x_1 - x_0 - u_0\Delta t). \tag{8.11}$$

Then given a measurement z_1, the posterior belief, $X_1|Z_1 = z_1$, can be determined using Bayes' theorem,

$$f_{X_1|Z_1}(x_1|z_1; u_0) = \eta_1\, f_{Z_1|X_1}(z_1|x_1) f_{X_1}(x_1; u_0). \tag{8.12}$$

Here η_1 is chosen to normalise the integral of the result and the conditional PDF, $f_{Z_1|X_1}(z_1|x_1)$, is derived from the measurement (sensor) model.

At the next time-step, the prior belief can be found[6] in a similar manner to (8.9),

[6] Using the complete state assumption. $u_{0:1}$ denotes the set of controls u_0, u_1.

$$f_{X_2|Z_1}(x_2|z_1; u_{0:1}) = \int_{-\infty}^{\infty} f_{X_2|X_1}(x_2|x_1; u_1)\, f_{X_1|Z_1}(x_1|z_1; u_0)\mathrm{d}x_1. \tag{8.13}$$

Then given a measurement z_2, the posterior belief can be found using

$$f_{X_2|Z_{1:2}}(x_2|z_{1:2}; u_{0:1}) = \eta_2\, f_{Z_2|X_2}(z_2|x_2) f_{X_2|Z_1}(x_2|z_1; u_{0:1}). \tag{8.14}$$

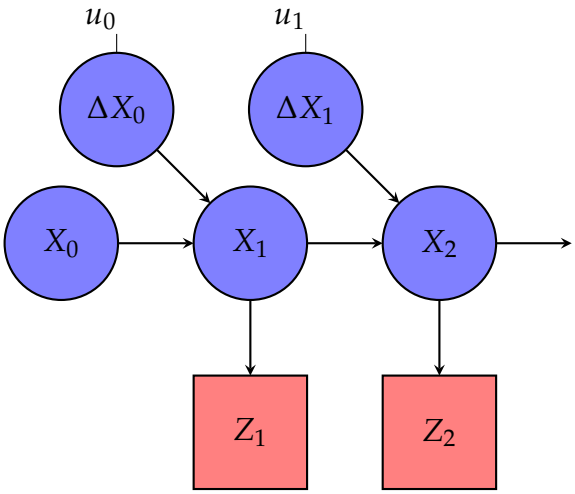


Figure 8.3: Bayesian network of a hidden Markov model (HMM) for X_2.

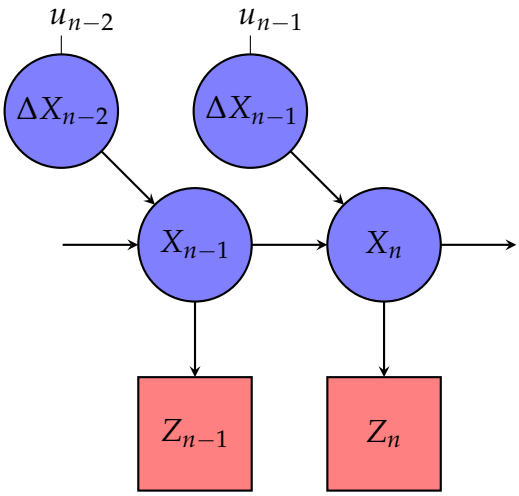


Figure 8.4: Bayesian network of a hidden Markov model (HMM) for X_n.

8.4 Bayes filter variants

Gaussian Bayes (Kalman) filters These assume the posterior PDF has a Gaussian shape so they only need to track the mean and variance. Multimodal PDFs can be represented using a weighted sum of Gaussians.

Histogram filters These approximate the posterior PDF with a histogram.

Particle filters These approximate the posterior PDF with point masses (particles).

Both histogram and particle filters are non-parametric and thus require more memory to represent the posterior PDF.

8.4.1 Kalman filters

Kalman filters assume a Gaussian distribution for the posterior PDF and can represent this efficiently using a parametric model[7].

[7] Using mean and variance as parameters.

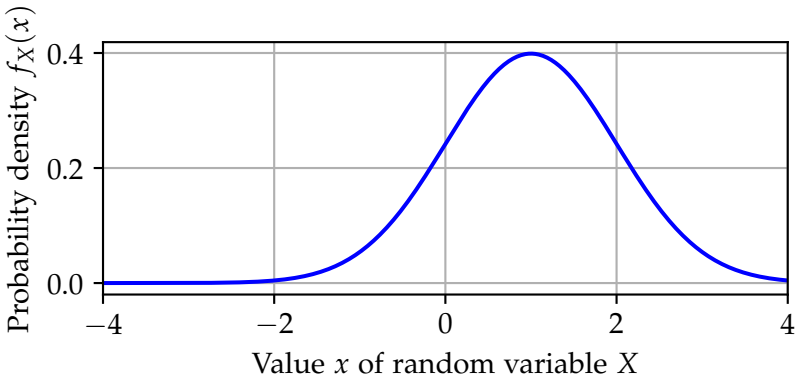


Figure 8.5: Representing a Gaussian distribution using parametric model.

Multimodal distributions can be approximated with a weighted sum of Gaussians[8]:

[8] As used by the MHKF and the MHEKF.

$$f_X(x) \approx \sum_{i=0}^{I-1} \frac{a_i}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_i}{\sigma_i}\right)^2\right). \qquad (8.15)$$

However, many posterior distributions are difficult to parameterise with Gaussians and it is difficult to determine μ_i, σ_i, a_i and I given $f_X(x)$.

8.4.2 *Histogram filters*

The histogram filter approximates the belief of the state with a histogram and so is equivalent to a discrete Bayes filter. The predict step requires a discrete convolution[9].

[9] This requires $O(M^2)$ multiplications for M histogram bins.

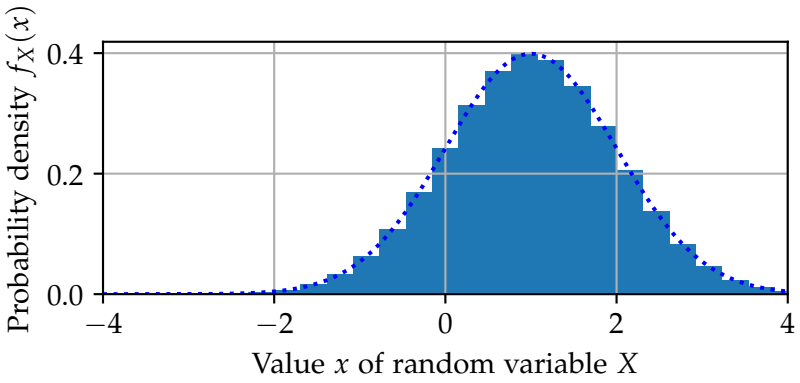


Figure 8.6: Approximation a Gaussian distribution using a histogram.

8.4.3 *Particle filters*

Particle filters represent the belief of the state with particles using the concept of importance sampling. This uses a pseudorandom number generator to randomly choose the particles. Importance sampling ensures that most of the particles are chosen near the modes in the posterior distribution.

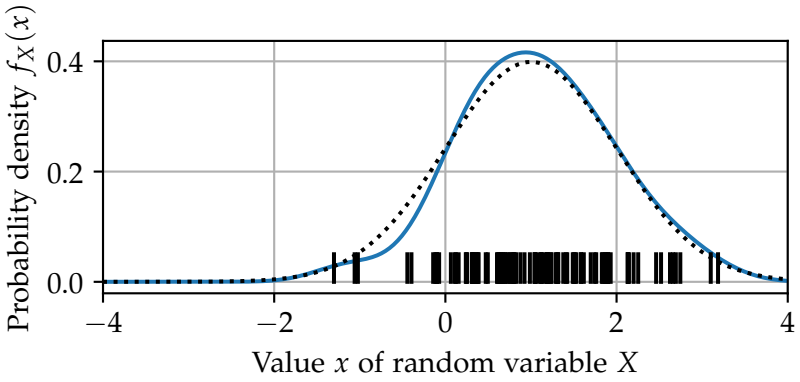


Figure 8.7: Approximation a Gaussian distribution using particles.

Examples of using particles to approximate a Gaussian distribution are shown in Figure 8.8 and Figure 8.9 for 25 and 250 particles. Notice how the random sampling selects more particles around the mode and fewer in the tails of the distribution. Also note how more particles better approximate the desired distribution.

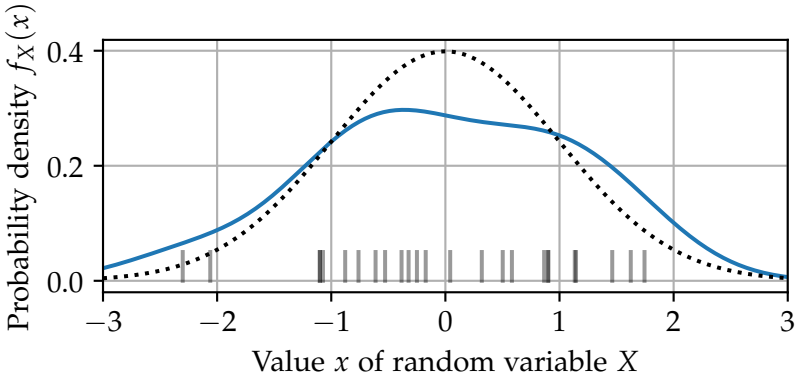


Figure 8.8: Approximation of a Gaussian distribution with 25 uniformly weighted particles.

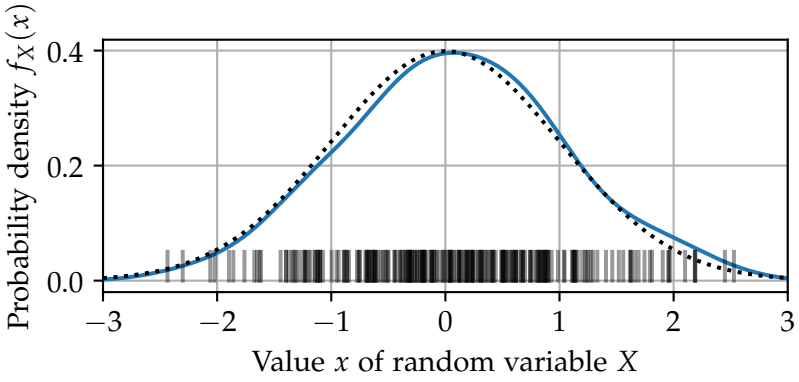


Figure 8.9: Approximation of a Gaussian distribution with 250 uniformly weighted particles.

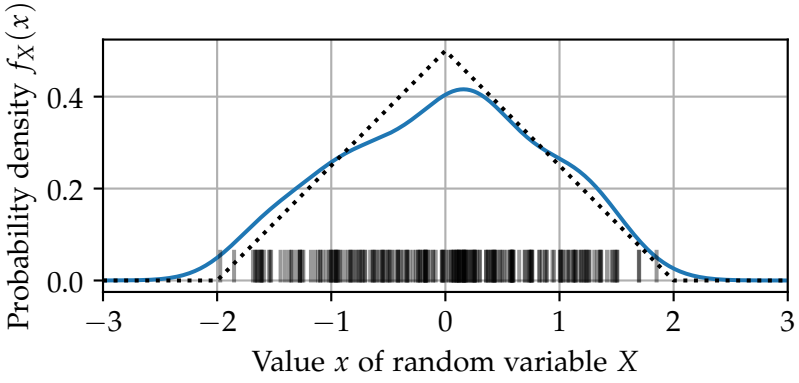


Figure 8.10: Approximation of a triangle distribution with 250 uniformly weighted particles.

8.5 Summary

1. Bayes filters repeatedly use prediction followed by application of Bayes' theorem.
2. Bayes filters can be implemented recursively if the complete state assumption applies.
3. The Kalman filter is a recursive Bayes filter that assumes linear motion and sensor models, a Gaussian prior, and additive Gaussian sensor and process noise.
4. Histogram filters approximate the posterior density with a histogram[10].
5. Particle filters approximate the posterior density with randomly chosen particles (samples).
6. Kalman, histogram, and particle filters assume the complete state (Markov) approximation.

[10] The probability density function is replaced with a probability mass function.

8.6 Exercises

1. What should a Bayes filter do at time-step n if there is no measurement available?
2. Which is the most efficient way to represent a Gaussian PDF?
3. Describe three ways that a posterior belief can be represented.
4. How does the complete state assumption simplify a Bayes filter.

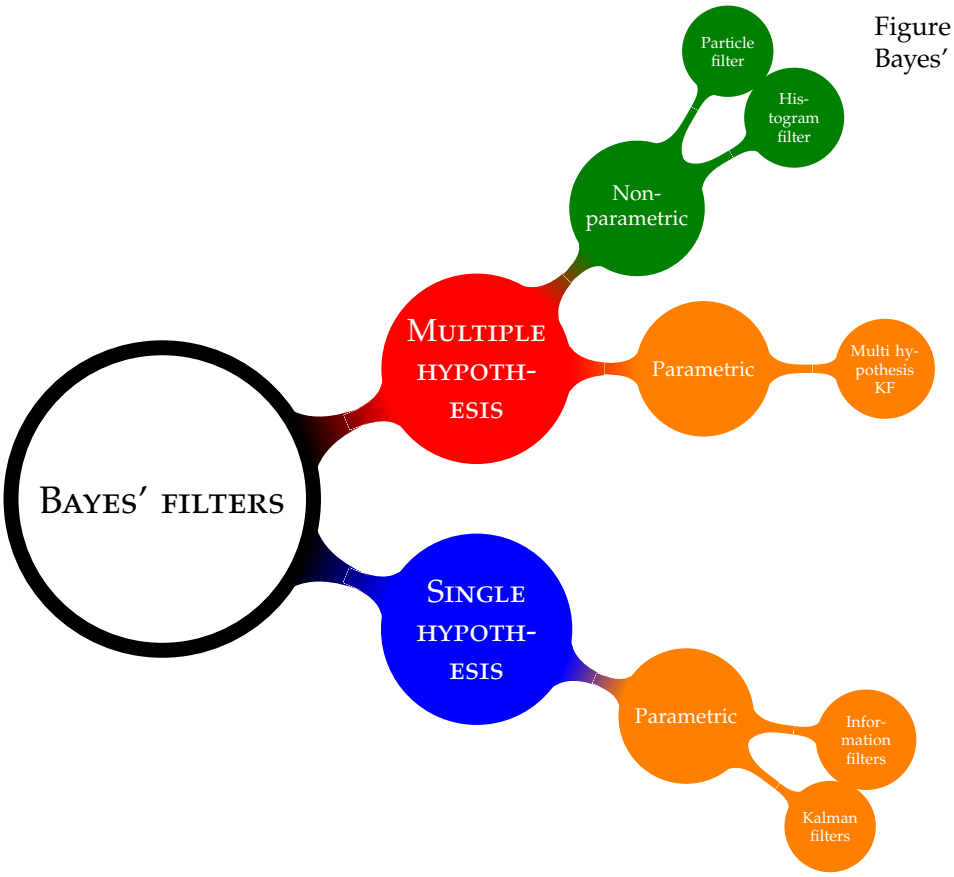


Figure 8.11: Categories of Bayes' filters.

9
Particle filters

Particle filters represent the belief of the state with particles using the concept of importance sampling. The particles are spaced more closely near modes in the posterior belief. At each time-step the particles are moved according to a motion model and are weighted according to a sensor model and sensor measurement. Particles with small weights can be repositioned using resampling.

9.1 Particle filter algorithm

Each particle[1] has a state x and a weight a.

[1] Subscripts and superscripts denoting the time-step and particle number have been dropped for clarity.

9.1.1 Initialisation

For each particle:
state x is sampled from initial belief $f_{X_0}(x)$
weight $a = 1$

9.1.2 Predict

For each particle:
w is sampled from $f_W(w)$
$x \leftarrow x + g(x, u) + w$

Here $f_W(w)$ is the PDF of the process noise, assumed additive, and $g(x, u)$ is the motion model.

9.1.3 Update

For each particle:
$a = a \times L_{X|Z}(x|z)$

Here z is the current measurement and $L_{X|Z}(x|z)$ is the likelihood function of the sensor.

9.2 *Inverse transform sampling*

Inverse transform sampling is a method for generating pseudorandom samples from any probability density function (PDF), $f_X(x)$. There are three steps:

1. Integrate the probability density function, $f_X(x)$, to get the cumulative distribution function (CDF), $F_X(x)$.
2. Generate a random number u from the standard uniform distribution in the interval $[0, 1]$.
3. Determine the value x such that $F_X(x) = u$, i.e., $x = F_X^{-1}(u)$.

In general, obtaining analytic expressions for F_X^{-1} is difficult. In practice, interpolation is used with a discrete approximation to $F_X(x)$.

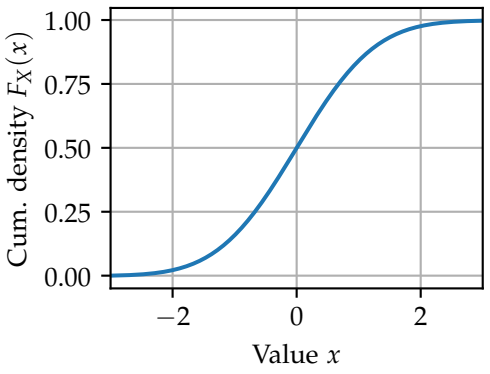


Figure 9.1: Gaussian CDF $F_X(x)$.

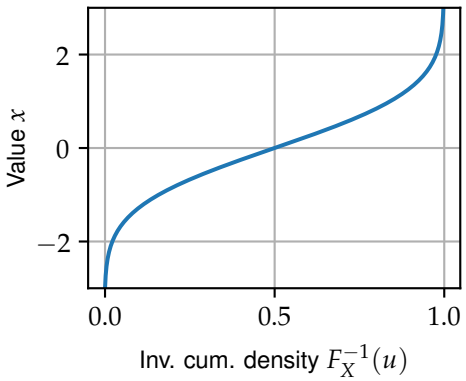


Figure 9.2: Inverse Gaussian CDF $F_X^{-1}(u)$.

9.3 *Resampling*

The weight of a particle in the wrong place can quickly become zero and thus does not contribute to the posterior PDF. So to reduce the number of particles, they are repositioned in areas of high belief by sampling from the posterior belief. This process is called *importance sampling* or *resampling*.

9.3.1 *What can go wrong?*

1. Have too few particles to accurately represent the belief posterior[2]. But even with a large number of particles, none may occur in the vicinity of the correct state. This is called the *particle deprivation problem*. It can occur as the result of variance in the random sampling—an unlucky sequence wipes out the particles near the true state.
2. Have poor motion and sensor models.
3. Choose 'bad' particles in the resampling process[3]. Any sampling variance is amplified during resampling. In practice, the resampling is not performed every iteration.
4. Have an ill-conditioned system, say due to insufficient or ambiguous measurements.

[2] There are techniques to reduce the number of particles, such as Rao-Blackwellisation to partition the belief. This is used by the gmapping SLAM algorithm.

[3] It is better to sample from the estimated continuous posterior, this is called regularized resampling. In this case no two particles will be the same.

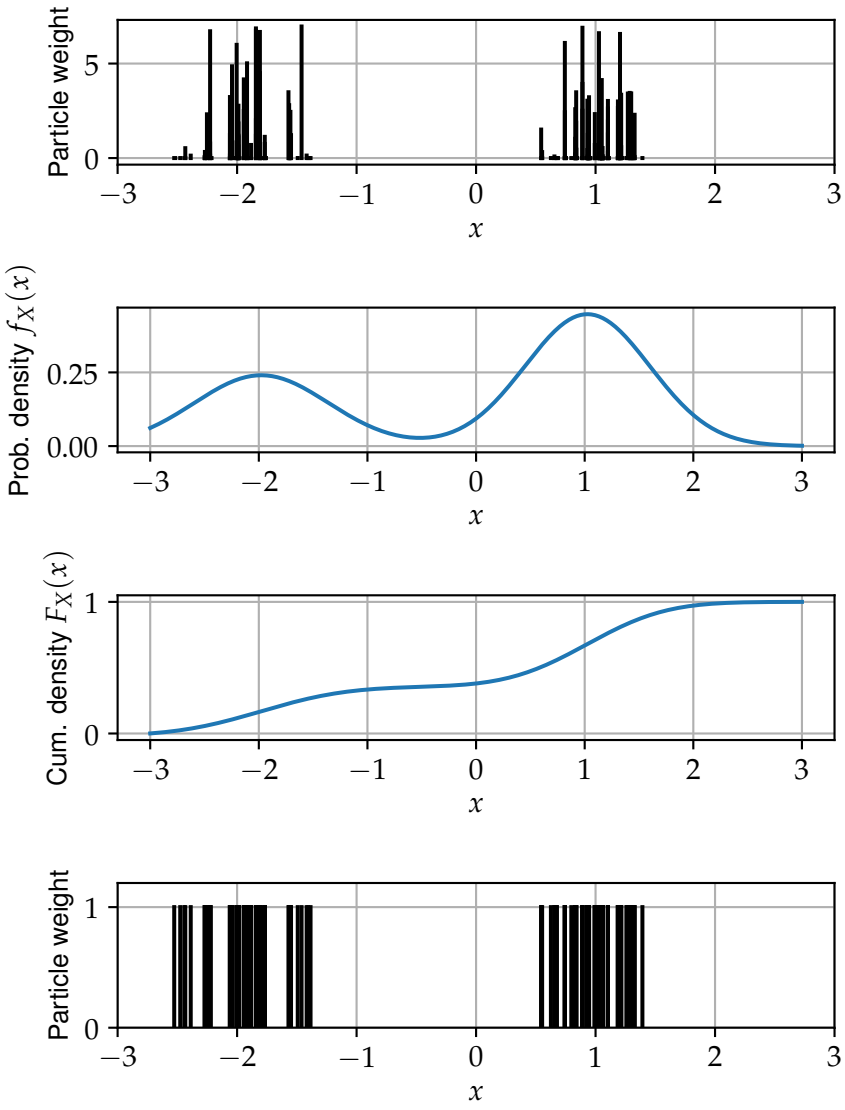


Figure 9.3: Demonstration of particle resampling. Note, many of the particles in the top plot have negligible weights and do not contribute to the belief.

9.4 *Particle filter density extraction*

Density extraction concerns estimation of the posterior PDF from the particles. There is no simple solution, especially when the state space has many dimensions. There are a number of approaches:

1. Determine the (weighted) mean and variance of the particles and construct a Gaussian PDF. This forces the posterior to have a single mode.
2. Approximate the posterior with a histogram.
3. Use kernel density estimation[4] where each particle 'location' is replaced by a Gaussian distribution; these are summed to form a continuous estimate of the posterior. Examples are shown in Figure 9.4 and Figure 9.5.
4. Use k-means but this requires an estimate of the number of modes.

[4] http://en.wikipedia.org/wiki/Kernel_density_estimation

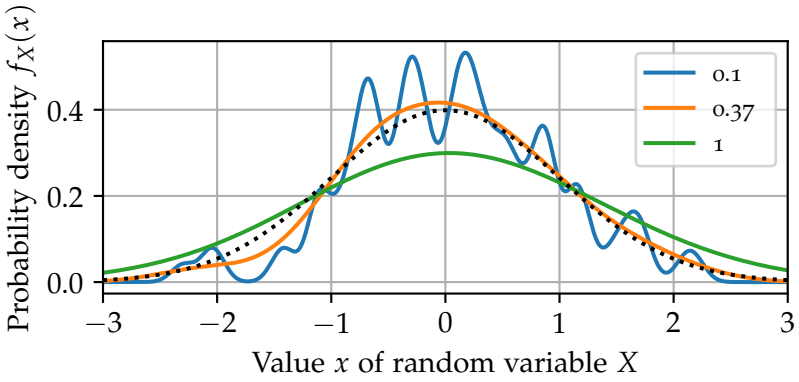


Figure 9.4: The effect of σ on kernel density estimation with 100 particles. 0.37 is the value for σ derived using Silverman's rule of thumb.

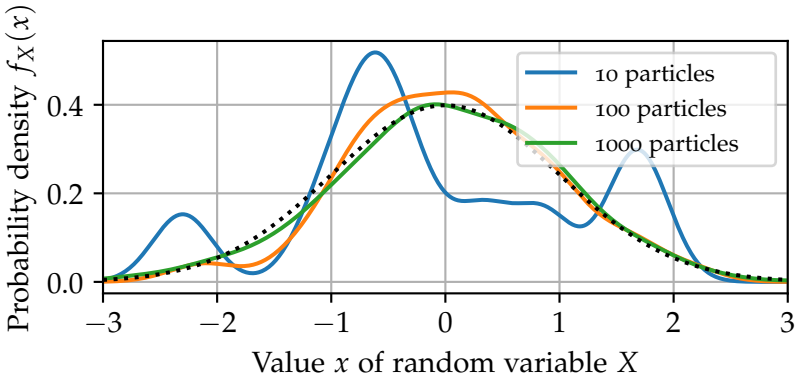


Figure 9.5: Kernel density estimation using Silverman's rule of thumb with 10, 100, 1000 particles.

9.5 Particle filter variants

There are many types of particle filter. The common ones are:

Sequential importance sampling (SIS) This normalises the weights to sum to one but does not perform resampling. Thus it can lead to a degeneracy problem where many particles have zero weight.

Sampling importance resampling (SIR) This performs resampling at each time-step but suffers from the *sample impoverishment problem* since the new particles are only chosen from the existing particles; particles with a high weight may be chosen multiple times.

Regularized particle filter This estimates the posterior PDF using kernel density estimation (KDE) and resamples from this continuous density. No two new particles will be identical so it does not suffer from the sample impoverishment problem.

9.6 Robot tracking example

The best way to understand a particle filter is through a simple example, in this case a robot moving along a rail at a constant speed, v, with a motion model

$$X_n = X_{n-1} + v\Delta t + W_n, \tag{9.1}$$

where W_n denotes process noise.

The robot has a simple range sensor that measures the distance from the starting point and thus the sensor model is

$$Z_n = X_n + V_n, \tag{9.2}$$

where V_n denotes sensor noise.

9.6.1 Initial belief

Let's say that the robot is known to start precisely[5] at $x = 0$, thus the initial belief is a Dirac delta:

$$f_{X_0}(x) = \delta(x). \tag{9.3}$$

We start by choosing a number of particles, M, and since the starting point is known[6] we define

$$x_0^{(m)} = 0 \quad \text{for } m = 0 \cdots M-1. \tag{9.4}$$

Each of the particles is initialised with unity weight,

$$a_0^{(m)} = 1 \quad \text{for } m = 0 \cdots M-1. \tag{9.5}$$

[5] Yeah right!

[6] If the starting point is unknown, we would sample from the initial belief.

9.6.2 *Predict step*

The prior belief of the robot's position can be described by a linear motion model,

$$X_1 = X_0 + v\Delta t + W_1. \tag{9.6}$$

A particle filter implements this by moving each particle according to

$$x_1^{(m)} = x_0^{(m)} + v\Delta t + w_1^{(m)}. \tag{9.7}$$

Here each particle is perturbed by $w_1^{(m)}$. This is a random number sampled[7] from the distribution $f_{W_1}(w)$.

[7] Using an appropriate random number generator or inverse transform sampling.

9.6.3 *Update step*

The update step uses a measurement, z_1, to update the particle weights according to

$$a_1^{(m)} = a_0^{(m)} L_{X_1|Z_1}(x_1^{(m)}|z_1). \tag{9.8}$$

Since the sensor model is

$$Z_1 = X_1 + V_1, \tag{9.9}$$

the likelihood function is

$$L_{X_1|Z_1}(x_1^{(m)}|z_1) = f_{V_1}(z_1 - x_1^{(m)}). \tag{9.10}$$

In other words, the particles with a position, $x_1^{(m)}$, close to the measurement, z_1, get a higher weight.

9.6.4 *Estimate*

A popular point estimate for a particle filter is the MMSE estimate[8]. This is the mean of the posterior belief and can be found using a weighted average of the particle's positions:

$$\hat{x}_1^+ = \frac{\sum_{m=0}^{M-1} a^{(m)} x_1^{(m)}}{\sum_{m=0}^{M-1} a^{(m)}}. \tag{9.11}$$

[8] The MAP estimate is not so simple to find but is more appropriate for multimodal beliefs.

The variance can be found in a similar manner. To determine the PDF requires a technique known as kernel density estimation (KDE).

9.7 UAV over a fjord example

Another example is a localisation problem with a non-linear sensor model. Consider a UAV[9] flying a straight path at a constant altitude and constant speed over a fjord. It has a single sensor that measures its height directly above the ground (or sea) below. The UAV has lost its position along its path but has a map of the expected heights along the path. From this map, and the height measurement, a particle filter can be used to estimate the UAV's position.

This problem has the same motion model as the robot tracking problem but the sensor model is non-linear and the initial position is unknown.

[9] Unmanned aerial vehicle, a.k.a., drone.

9.7.1 Preliminaries

Let's consider the state of the UAV as its position, x, along the flight track and let's denote its altitude above sea level by l, its speed by v, and its height measurement by z. Using a simple, linear motion model with additive process noise, the random variable describing its state at time-step n is

$$X_n = X_{n-1} + v\Delta t + W_n. \tag{9.12}$$

Note, the control output u_{n-1} is equal to the speed v.

If the random variable, W_n, describing the process noise is zero-mean Gaussian, then its PDF is

$$f_{W_n}(w) = \mathcal{N}(0, \sigma^2_{W_n}). \tag{9.13}$$

Since the motion model is linear, the conditional probability for the motion model is also Gaussian,

$$f_{X_n|X_{n-1}}(x_n|x_{n-1}; u_{n-1}) = f_{W_n}(x_n - x_{n-1} - v\Delta t). \tag{9.14}$$

The random variable, Z_n, describing height measurement can be expressed as

$$Z_n = l - h(x_n) + V_n, \tag{9.15}$$

where V_n denotes the sensor noise[10]. Assuming zero mean Gaussian sensor noise, then

[10] In practice, the variance of this will increase with measurement distance $l - h(x_n)$.

$$f_{V_n}(v_n) = \mathcal{N}(0, \sigma^2_{V_n}). \tag{9.16}$$

Thus the likelihood function for the sensor model is

$$L_{X_n|Z_n}(x_n|z_n) = f_{V_n}(z_n + h(x_n) - l). \tag{9.17}$$

9.7.2 *Particle filter algorithm*

1. Let's say that the only initial knowledge of the AUV's position is that it is between position a and b with equal probability. Thus the initial belief can be described by a uniform distribution,

$$f_{X_0}(x_0) = \mathcal{U}(a, b). \tag{9.18}$$

   We thus need to choose the positions, $x_0^{(m)}$, of the M particles uniformly spread[11] between a and b.

2. The particle weights are set to 1,

$$a_0^{(m)} = 1. \tag{9.19}$$

3. The particles are moved[12] using a motion model. The simplest thing to do here is to add $v\Delta t$ plus some Gaussian random noise[13],

$$x_1^{(m)} = x_0^{(m)} + v\Delta t + w_1^{(m)}. \tag{9.20}$$

4. A measurement is made, z_1, and the weights of the particles are updated using the likelihood function for the sensor model,

$$a_1^{(m)} = a_0^{(m)} f_{X_1|Z_1}(x_1^{(m)}|z_1). \tag{9.21}$$

5. If many of the weights become negligible, the particles are resampled from the current posterior PDF, $f_{X_1}(x_1|z_1)$, and their weights are reset to 1.
6. The algorithm repeats at step 3 (but with the subscripts incremented).

[11] For other distributions it easier to randomly sample from the distribution.

[12] For more complex motion models, we would sample from the conditional probability density $f_{X_1|X_0}(x_1|x_0; u_0)$.

[13] Sampled from the PDF of the process noise random variable W_1.

9.8 *Particle filter algorithm summary*

1. Generate initial distribution of particles, each with unit weight, by sampling from the initial belief, $f_{X_0}(x)$.
2. Transition each particle using state transition model (motion model)[14].
3. Weight each particle using evidence likelihood (sensor model) $L_{X_n|Z_n}(x_n|z_n)$.
4. Resample particles, by sampling from the posterior[15] belief, and reset weights to unity.

[14] Usually by sampling from $f_{X_n|X_{n-1}}(x_n|x_{n-1}; u_{n-1})$.

[15] Represented by the particles' values and weights.

9.9 Summary

1. Particles are randomly sampled from the initial belief distribution using inverse transform sampling.
2. Inverse transform sampling generates pseudorandom numbers from a desired PDF.
3. Particle filters assume the complete state (Markov) approximation.
4. Bayes filtering of the particles modifies the particle weights to update the posterior distribution.
5. The particles are resampled (based on importance sampling using the particle weights) and the weights are set back to unity.
6. Particle filters are simple to program[16].

[16] The tricky aspect is developing the sensor and motion models.

9.10 Further reading

1. Simple explanation of a particle filter for a non-linear 1-D system, https://www.youtube.com/watch?v=aUkBa1zMKv4

9.11 Exercises

1. What is the point of resampling?
2. What can go wrong with resampling?
3. How should the particles be initially distributed if you have some prior information?
4. What is the advantage of a particle filter over a Kalman filter?
5. Are there any restrictions on the state transition model for a particle filter?
6. Do the measurements have to have Gaussian noise for a particle filter?
7. How can the posterior PDF be determined from the particles?
8. How does a particle filter support multiple hypotheses?

10 Multivariate beliefs

So far we have considered univariate PDFs. However, if we have multiple sensor measurements we need to consider joint PDFs. These are multidimensional. Unless the measurements are independent we need to consider the correlations between the measurements.

Similarly, if we wish to estimate multiple state variables, say the x and y positions of a robot, we need to represent the belief of the 2-D position with a multivariate PDF.

10.1 Joint random variables

Let's say we wish to track the 2-D position of a robot on a flat surface. We can describe this with a pair of random variables, X, Y. Each of these random variables has a mean and variance[1]. In addition, there may be a correlation between them[2]. To handle the correlation we require a joint PDF, $f_{X,Y}(x, y)$.

[1] And perhaps other higher order moments.

[2] For example, consider a robot constrained to a rail at an angle of 45 degrees to the x-axis.

10.2 Bivariate Gaussian PDF

If both X and Y are Gaussian distributed we can describe their joint PDF with a bivariate Gaussian distribution. This can be parameterised by a mean vector and a covariance matrix for the random vector

$$\mathbf{X} = [X, Y]^{\mathrm{T}}. \tag{10.1}$$

The mean vector of $\mathbf{X}$ is

$$\mu_{\mathbf{X}} = \mathrm{E}[\mathbf{X}] = [\mu_X, \mu_Y]^{\mathrm{T}}, \tag{10.2}$$

and the covariance matrix of $\mathbf{X}$ is

$$\boldsymbol{\Sigma}_{\mathbf{X}} = \mathrm{Covar}[\mathbf{X}] = \mathrm{E}\left[(\mathbf{X} - \mu_{\mathbf{X}})(\mathbf{X} - \mu_{\mathbf{X}})^{\mathrm{T}}\right]. \tag{10.3}$$

This has a symmetrical form,

$$\mathbf{\Sigma_X} = \begin{bmatrix} \sigma_X^2 & \rho\sigma_X\sigma_Y \\ \rho\sigma_X\sigma_Y & \sigma_Y^2 \end{bmatrix}, \tag{10.4}$$

where

σ_X^2 is the variance of X,

σ_Y^2 is the variance of Y, and

$\rho = \rho_{XY}$ is the correlation coefficient between X and Y. When $\rho = 0$ then X and Y are uncorrelated[3]. When $\rho = 1$ then X and Y are perfectly correlated[4].

[3] $f_{X,Y}(x,y) = f_X(x)f_Y(y)$.
[4] $f_{X,Y}(x,y) = f_X(x)\delta(y-x)$.

The bivariate[5] Gaussian PDF is described by

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \det(\mathbf{\Sigma_X})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_{\mathbf{X}})^{\mathrm{T}} \mathbf{\Sigma}_{\mathbf{X}}^{-1} (\mathbf{x}-\mu_{\mathbf{X}})\right). \tag{10.5}$$

[5] Note, in the 1-D case, this collapses to $f_X(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left(-0.5(x-\mu_X)^2/\sigma_X^2\right)$.

A number of examples are plotted in Figure 10.1 to Figure 10.4.

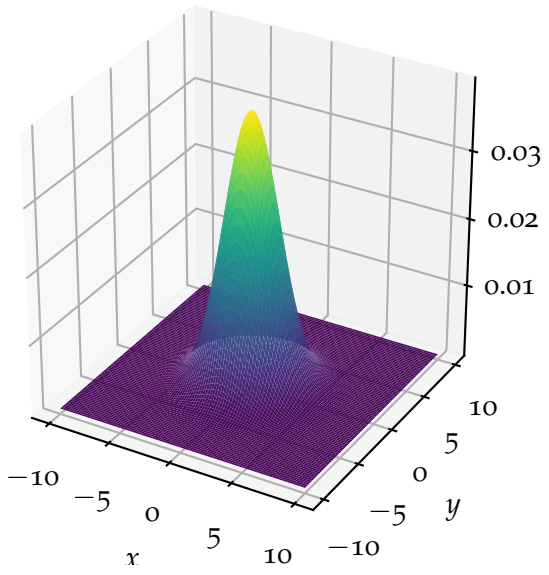


Figure 10.1: 2-D Gaussian PDF where $\mu_X = 0$, $\mu_Y = 0$, $\sigma_X = 2$, $\sigma_Y = 2$, $\rho = 0$ (uncorrelated).

A bivariate Gaussian PDF is often drawn as a confidence ellipse[6] for a given confidence interval. The confidence ellipse is an iso-contour of the Gaussian distribution.

[6] This is also called an error ellipse. For a trivariate Gaussian PDF this is an ellipsoid.

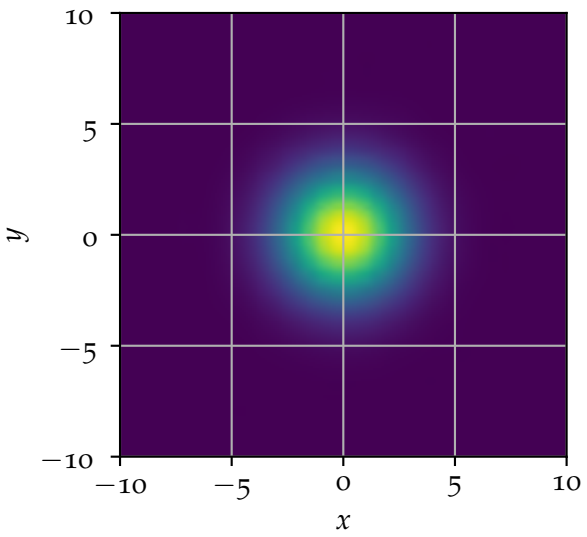


Figure 10.2: 2-D Gaussian PDF where $\mu_X = 0$, $\mu_Y = 0$, $\sigma_X = 2$, $\sigma_Y = 2$, $\rho = 0$ (uncorrelated).

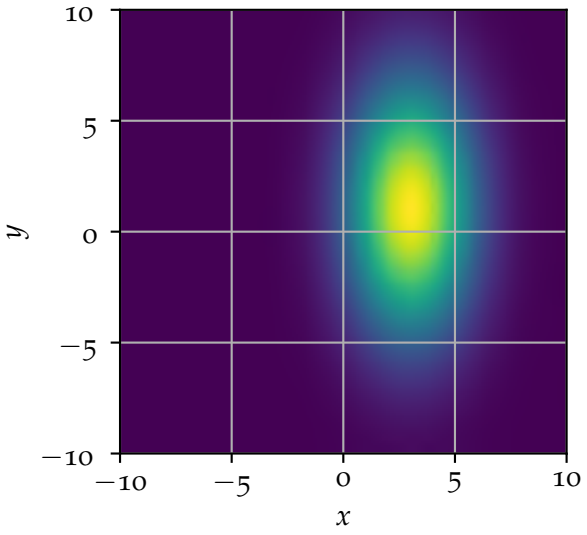


Figure 10.3: 2-D Gaussian PDF where $\mu_X = 3$, $\mu_Y = 1$, $\sigma_X = 2$, $\sigma_Y = 4$, $\rho = 0$ (uncorrelated).

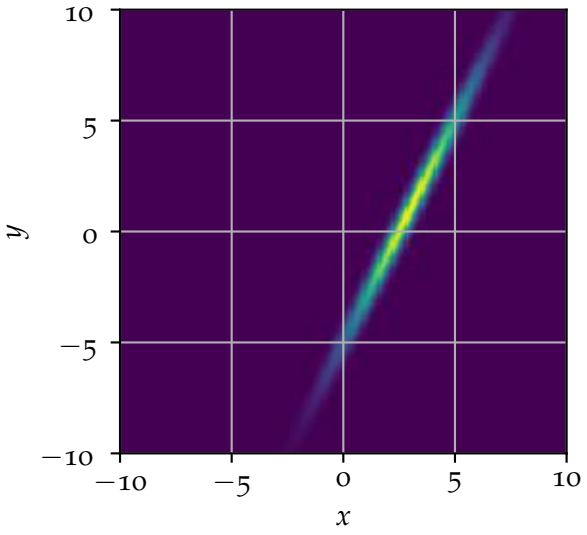


Figure 10.4: 2-D Gaussian function where $\mu_X = 3$, $\mu_Y = 1$, $\sigma_X = 2$, $\sigma_Y = 4$, $\rho = 0.9$ (highly correlated).

10.3 Multivariate Gaussian PDF

Equation (10.5) can describe Gaussian PDFs of arbitrary dimensionality. All that is required is a mean vector and a covariance matrix.

For the trivariate Gaussian distribution, $\mathbf{X} = (X_1, X_2, X_3)^{\mathrm{T}}$, $\mu_{\mathbf{X}} = (\mu_{X_1}, \mu_{X_2}, \mu_{X_3})^{\mathrm{T}}$, and the covariance matrix is

$$\mathbf{\Sigma_X} = \begin{bmatrix} \sigma^2_{X_1} & \rho_{12}\sigma_{X_1}\sigma_{X_2} & \rho_{13}\sigma_{X_1}\sigma_{X_3} \\ \rho_{12}\sigma_{X_1}\sigma_{X_2} & \sigma^2_{X_2} & \rho_{23}\sigma_{X_2}\sigma_{X_3} \\ \rho_{13}\sigma_{X_1}\sigma_{X_3} & \rho_{23}\sigma_{X_2}\sigma_{X_3} & \sigma^2_{X_3} \end{bmatrix}. \tag{10.6}$$

The distribution can be described by 9 parameters; three means, three variances, and three correlations.

10.4 Covariance matrices

With a multivariate distribution, the variances of each variate need to be described as well as the correlations between them. For a random vector, $\mathbf{V}$, its covariance matrix is defined by

$$\mathbf{\Sigma_V} = \mathrm{E}\left[(\mathbf{V} - \mu_{\mathbf{V}})(\mathbf{V} - \mu_{\mathbf{V}})^{\mathrm{T}}\right]. \tag{10.7}$$

Most sensor noise is zero mean, and in this case the covariance matrix is the same as the correlation matrix and is defined by

$$\mathrm{E}\left[\mathbf{V}\mathbf{V}^{\mathrm{T}}\right]. \tag{10.8}$$

For example, consider a two element sensor noise random vector

$$\mathbf{V} = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}. \tag{10.9}$$

Assuming the noise is zero mean, the covariance matrix is

$$\mathbf{\Sigma_V} = E\left\{ \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix} \right\}, \tag{10.10}$$

$$= \begin{bmatrix} \mathrm{E}[V_1V_1] & \mathrm{E}[V_1V_2] \\ \mathrm{E}[V_1V_2] & \mathrm{E}[V_2V_2] \end{bmatrix}. \tag{10.11}$$

Since $\mathrm{E}[V_1] = 0$ then $\mathrm{E}[V_1V_1] = \sigma^2_{V_1}$, etc., and so

$$\mathbf{\Sigma_V} = \begin{bmatrix} \sigma^2_{V_1} & \rho_{V_1,V_2}\sigma_{V_1}\sigma_{V_2} \\ \rho_{V_1,V_2}\sigma_{V_1}\sigma_{V_2} & \sigma^2_{V_2} \end{bmatrix}. \tag{10.12}$$

Note that the covariance matrix encodes the variances of each random variable and the correlation, ρ_{V_1,V_2}, between them. If the correlation is zero, then the covariance matrix is diagonal,

$$\mathbf{\Sigma_V} = \begin{bmatrix} \sigma_{V_1}^2 & 0 \\ 0 & \sigma_{V_2}^2 \end{bmatrix}. \tag{10.13}$$

10.5 Multivariate histograms

Histogram filters are not feasible for large state vectors since if M state variables are required, then a M-dimensional histogram is required. If N bins are required for each variable then a total of N^M bins are required for the histogram. For example, if there are six state variables and 50 bins are required for each variable, then $50^6 \approx 15 \times 10^9$ bins are required in total.

10.6 Multivariate particles

Similarly, particle filters are not feasible for large state vectors. If N particles are required for each state variable, then N^M particles are required for M state variables. Due to exponential growth in the required number of particles with the number of state variables, particle filters are not used for a large number of state variables. For example, with $M = 6$ state variables and $N = 100$ particles per state variable, then 1×10^{12} particles are required. Reducing N to 20, then 64×10^6 particles are required. This is more achievable but results in a poorer modelling of the PDF.

10.7 Linear mapping of a random vector

Consider the linear mapping between a random vector[7] $\mathbf{X}$ to another random vector $\mathbf{Y}$

$$\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{b}, \tag{10.14}$$

where $\mathbf{A}$ is a transfer matrix and $\mathbf{b}$ is an offset vector. The mean of $\mathbf{Y}$ is related to the mean of $\mathbf{X}$ by

$$\mu_\mathbf{Y} = \mathbf{A}\mu_\mathbf{X} + \mathbf{b}. \tag{10.15}$$

The covariance[8] of $\mathbf{Y}$ is related to the covariance of $\mathbf{X}$ by

[7] To avoid confusion between vectors and matrices, lower case is often used to denote a random variable vector.

[8] A covariance matrix has variances on the diagonal and correlations off the diagonal.

$$\boldsymbol{\Sigma}_{\mathbf{Y}} = \mathbf{A}\,\boldsymbol{\Sigma}_{\mathbf{X}}\,\mathbf{A}^{\mathrm{T}}. \tag{10.16}$$

Even if the errors of $\mathbf{X}$ are uncorrelated so that $\boldsymbol{\Sigma}_{\mathbf{X}}$ is diagonal, then in general the covariance $\boldsymbol{\Sigma}_{\mathbf{Y}}$ is full.

For example, consider the case where

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \tag{10.17}$$

and

$$\boldsymbol{\Sigma}_{\mathbf{X}} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}. \tag{10.18}$$

The covariance of $\mathbf{Y}$ is,

$$\boldsymbol{\Sigma}_{Y} = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}, \tag{10.19}$$

$$= \begin{bmatrix} 10 & 10 \\ 10 & 20 \end{bmatrix}. \tag{10.20}$$

Since the off-diagonal elements are non-zero, there is correlation.

10.8 Summary

1. A multivariate Gaussian PDF is parameterised in terms of a mean vector and a covariance matrix.
2. A covariance matrix specifies the variance of each variable and the correlations between them.
3. A bivariate Gaussian PDF is often described by a confidence ellipse.

10.9 Exercises

1. How many parameters are required to describe a bivariate Gaussian distribution?
2. How many parameters are required to describe a trivariate Gaussian distribution?
3. How many parameters are required to describe an N-variate Gaussian distribution?
4. When is the covariance matrix of a random vector equal to the correlation matrix?

5. If the covariance matrix of a random vector has a diagonal form, what does this imply?
6. From a sensor fusion perspective, is it better to have sensors with correlated or un-correlated errors?

11
Multivariate Kalman filters

Kalman, histogram, and particle filters can be applied to estimate multiple state variables simultaneously. For a linear system, the Kalman filter is the most efficient since it only needs to track a mean vector and a covariance matrix. This is sufficient to describe a multidimensional Gaussian belef.

11.1 Deterministic linear model

A multidimensional linear system can be modelled[1] as

$$\begin{aligned} \mathbf{x}_n &= \mathbf{A}\mathbf{x}_{n-1} + \mathbf{B}\mathbf{u}_{n-1}, \quad \text{(motion model)} \\ \mathbf{z}_n &= \mathbf{C}\mathbf{x}_n + \mathbf{D}\mathbf{u}_{n-1}. \quad \text{(sensor model)} \end{aligned} \tag{11.1}$$

Here $\mathbf{x}_n$ denotes the system state at time-step n, $\mathbf{u}$ is the control input, and $\mathbf{z}_n$ is the measured output. The matrices $\mathbf{A}$ and $\mathbf{B}$ describe the motion model and the matrices $\mathbf{C}$ and $\mathbf{D}$ describe the sensor model.

[1] A time variant system can be modelled by replacing $\mathbf{A}$ with $\mathbf{A}_n$, etc. Some authors use $\mathbf{F}$, $\mathbf{H}$, etc., to distinguish the discrete-time and continuous time cases.

11.2 Stochastic linear model with additive noise

In practice, we need to consider both process noise and measurement noise, so the models (11.1) become

$$\begin{aligned} \mathbf{X}_n &= \mathbf{A}\mathbf{X}_{n-1} + \mathbf{B}\mathbf{u}_{n-1} + \mathbf{W}_n, \\ \mathbf{Z}_n &= \mathbf{C}\mathbf{X}_n + \mathbf{D}\mathbf{u}_{n-1} + \mathbf{V}_n. \end{aligned} \tag{11.2}$$

Here $\mathbf{X}_n$ is the random state vector, $\mathbf{u}_{n-1}$ is the known control vector, $\mathbf{Z}_n$ is the random measurement vector, $\mathbf{W}_n$ is a zero-mean random vector[2] that models the process noise, and $\mathbf{V}_n$ is a zero-mean random vector that models the measurement noise.

[2] Multivariate random variable.

11.2.1 Example: 1-D moving robot

The 1-D moving robot problem has a motion model:

$$X_n = X_{n-1} + v\Delta t + W_n, \tag{11.3}$$

and a sensor model:

$$Z_{1,n} = X_n + V_{1,n}, \tag{11.4}$$
$$Z_{2,n} = X_n + V_{2,n}. \tag{11.5}$$

These equations expressed in discrete-time state-space form[3] are:

[3] See (11.2).

$$\begin{aligned} \begin{bmatrix} X_n \end{bmatrix} &= \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} X_{n-1} \end{bmatrix} + \begin{bmatrix} \Delta t \end{bmatrix} \begin{bmatrix} v \end{bmatrix} + \begin{bmatrix} W_n \end{bmatrix}, \\ \begin{bmatrix} Z_{1,n} \\ Z_{2,n} \end{bmatrix} &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} X_n \end{bmatrix} + \begin{bmatrix} V_{1,n} \\ V_{2,n} \end{bmatrix} \end{aligned} \tag{11.6}$$

Thus for this problem,

$$\mathbf{X}_n = \begin{bmatrix} X_n \end{bmatrix}, \tag{11.7}$$
$$\mathbf{u}_{n-1} = \begin{bmatrix} v \end{bmatrix}, \tag{11.8}$$
$$\mathbf{Z}_n = \begin{bmatrix} Z_{1,n} \\ Z_{2,n} \end{bmatrix}, \tag{11.9}$$
$$\mathbf{A} = \begin{bmatrix} 1 \end{bmatrix}, \tag{11.10}$$
$$\mathbf{B} = \begin{bmatrix} \Delta t \end{bmatrix}, \tag{11.11}$$
$$\mathbf{C} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \tag{11.12}$$
$$\mathbf{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{11.13}$$
$$\mathbf{W}_n = \begin{bmatrix} W_n \end{bmatrix}, \tag{11.14}$$
$$\mathbf{V}_n = \begin{bmatrix} V_{1,n} \\ V_{2,n} \end{bmatrix}. \tag{11.15}$$

11.3 Kalman filter

The Kalman filter attempts to estimate the mean of the state using a time average. The state equation is used as the motion model; the output equation is used for the sensor model. Again there are two steps, predict and update. In the following $\mathbf{D} = 0$.

11.3.1 Predict step

The predicted (prior) state estimate is

$$\hat{\mathbf{x}}_n^- = \mathbf{A}\hat{\mathbf{x}}_{n-1}^- + \mathbf{B}\mathbf{u}_{n-1}. \tag{11.16}$$

This has a covariance,

$$\mathbf{P}_n^- = \text{Covar}\left[\hat{\mathbf{X}}_n^-\right] = \mathbf{A}\mathbf{P}_{n-1}^+\mathbf{A}^\mathrm{T} + \boldsymbol{\Sigma}_\mathbf{W}, \tag{11.17}$$

where, $\boldsymbol{\Sigma}_\mathbf{W} = \text{Covar}\left[\mathbf{W}\right] = \mathrm{E}\left[\mathbf{W}\mathbf{W}^\mathrm{T}\right]$ is the process noise covariance matrix[4].

11.3.2 Update step

The updated (posterior) state estimate is[5]

$$\hat{\mathbf{x}}_n^+ = \hat{\mathbf{x}}_n^- + \mathbf{K}_n\left(\mathbf{Z}_n - \mathbf{C}\hat{\mathbf{x}}_n^-\right), \tag{11.18}$$

where $\mathbf{K}_n$ is the Kalman gain matrix,

$$\mathbf{K}_n = \mathbf{P}_n^-\mathbf{C}^\mathrm{T}\left(\mathbf{C}\mathbf{P}_n^-\mathbf{C}^\mathrm{T} + \boldsymbol{\Sigma}_\mathbf{V}\right)^{-1}. \tag{11.19}$$

Here, $\boldsymbol{\Sigma}_\mathbf{V} = \text{Covar}\left[\mathbf{V}\right] = \mathrm{E}\left[\mathbf{V}\mathbf{V}^\mathrm{T}\right]$ is the measurement noise covariance matrix[6].

The covariance of the posterior state estimator is

$$\mathbf{P}_n^+ = \text{Covar}\left[\hat{\mathbf{X}}_n^+\right] = \left(\mathbf{I} - \mathbf{K}_n\mathbf{C}\right)\mathbf{P}_n^-. \tag{11.20}$$

[4] In general, this can vary with time.

[5] The first term is the predicted state estimate. The second term is called the correction since it corrects the predicted state using the measurement. It is the product of the Kalman gain and the innovation, the difference between the actual and predicted measurements.

[6] In general, this can vary with time.

11.4 Kalman filter implementation

Kalman filters can be efficiently applied to systems with many state variables. Since the state belief is Gaussian, the Kalman filter only needs to track the current mean estimate and its variance.

11.4.1 Predict step

Given the previous posterior estimate, $\hat{\mathbf{x}}_{n-1}^+$, the predicted (prior) estimate is given by

$$\hat{\mathbf{x}}_n^- = \mathbf{A}\hat{\mathbf{x}}_{n-1}^+ + \mathbf{B}\mathbf{u}_{n-1}. \tag{11.21}$$

The variance of this estimate from (11.17) is

$$\mathbf{P}_n^- = \mathbf{A}\mathbf{P}_{n-1}^+\mathbf{A}^\mathrm{T} + \boldsymbol{\Sigma}_\mathbf{W}. \tag{11.22}$$

11.4.2 *Update step*

Given a measurement vector, $\mathbf{z}_n$, the mean of the posterior is estimated using

$$\hat{\mathbf{x}}_n^+ = \hat{\mathbf{x}}_n^- + \mathbf{K}_n \left(\mathbf{z}_n - \mathbf{C}\hat{\mathbf{x}}_n^- \right), \tag{11.23}$$

where $\mathbf{K}_n$ is the Kalman gain matrix given by

$$\mathbf{K}_n = \mathbf{P}_n^- \mathbf{C}^\mathrm{T} \left(\mathbf{C}\mathbf{P}_n^- \mathbf{C}^\mathrm{T} + \mathbf{\Sigma}_\mathbf{V} \right)^{-1}. \tag{11.24}$$

The covariance of the posterior state estimator is

$$\mathbf{P}_n^+ = \mathrm{Covar}\left[\hat{\mathbf{X}}_n^+ \right] = (\mathbf{I} - \mathbf{K}_n \mathbf{C})\, \mathbf{P}_n^-. \tag{11.25}$$

For the next iteration, $\hat{\mathbf{x}}_n^+$ becomes $\hat{\mathbf{x}}_{n-1}^+$ and $\mathbf{P}_n^+$ becomes $\mathbf{P}_{n-1}^+$.

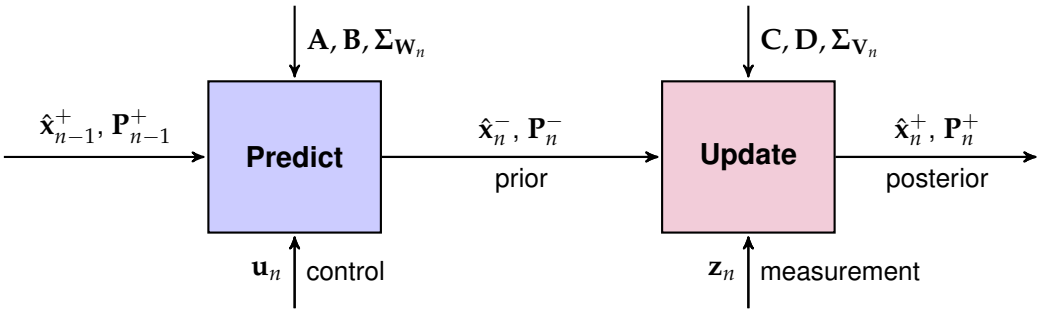


Figure 11.1: Kalman filter predict and update steps.

11.4.3 *Comments*

1. The belief of the state of the Kalman filter is represented at each time-step by[7]

   $\hat{\mathbf{x}}_n^+$ — the posterior state estimate.

   $\mathbf{P}_n^+$ — the posterior covariance matrix.

   Together, these are sufficient to describe a multivariate Gaussian PDF.

2. The predict and update steps alternate but if no measurements are available, the update step is skipped.

3. The Kalman filter requires a matrix inversion and this may require too much computation for a microcontroller. A sub-optimal variant called the Steady State Kalman filter precomputes the Kalman gain to avoid this matrix inversion at each time-step, see Section 22.11.4.

[7] See Figure 11.1.

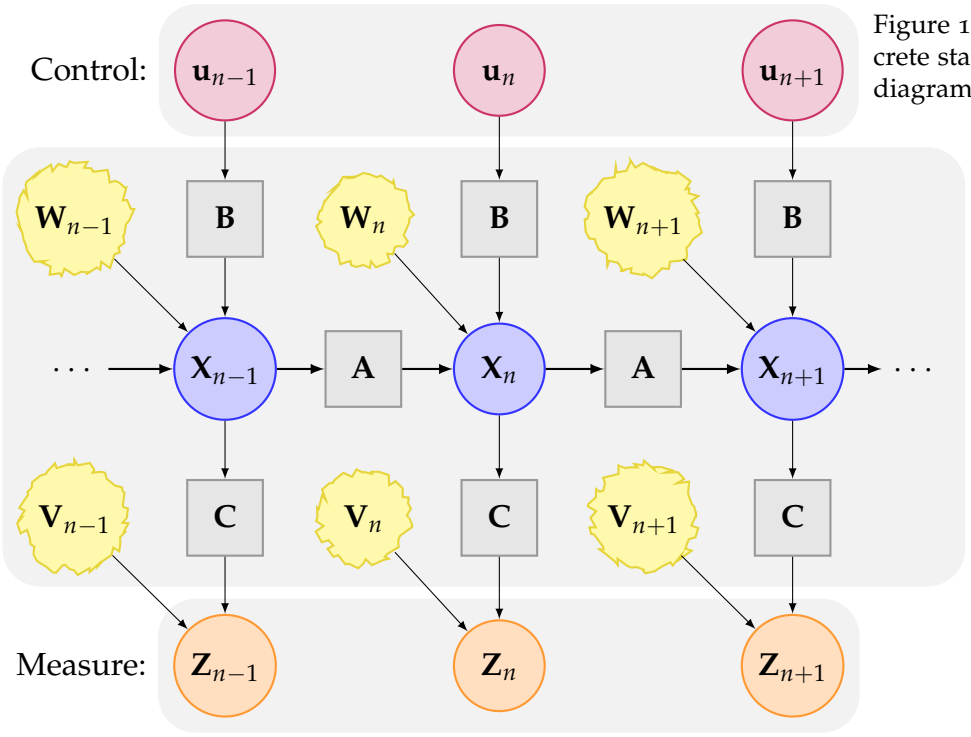


Figure 11.2: Probabilistic discrete state-space system block diagram.

The sizes of the vectors and matrices in a Kalman filter are:

$\mathbf{x}$ $N_x \times 1$

$\mathbf{z}$ $N_z \times 1$

$\mathbf{u}$ $N_u \times 1$

$\mathbf{A}$ $N_x \times N_x$

$\mathbf{B}$ $N_x \times N_u$

$\mathbf{C}$ $N_z \times N_x$

$\mathbf{D}$ $N_z \times N_u$

$\mathbf{K}$ $N_x \times N_z$

$\mathbf{P}$ $N_x \times N_x$

$\mathbf{V}$ $N_z \times N_z$

$\mathbf{W}$ $N_x \times N_x$

11.5 Information filters

There are two common ways to parameterise multivariate Gaussian distributions:

moments parameterisation mean vector and covariance matrix:

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \det(\mathbf{\Sigma}_{\mathbf{X}})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_{\mathbf{X}})^{\mathrm{T}} \mathbf{\Sigma}_{\mathbf{X}}^{-1}(\mathbf{x}-\mu_{\mathbf{X}})\right), \tag{11.26}$$

where $\mathbf{\Sigma}_{\mathbf{X}}$ is the covariance matrix and $\mu_{\mathbf{X}} = \mathrm{E}[\mathbf{X}]$ is the mean of $\mathbf{X}$.

canonical parameterisation information vector and information matrix:

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{\exp\left(-\frac{1}{2}\boldsymbol{\nu}^{\mathrm{T}} \mathbf{\Lambda}_{\mathbf{X}}^{-1} \boldsymbol{\nu}\right)}{\sqrt{(2\pi)^N \det\left(\mathbf{\Lambda}_{\mathbf{X}}^{-1}\right)}} \exp\left(-\frac{1}{2}\mathbf{x}^{\mathrm{T}} \mathbf{\Lambda}_{\mathbf{X}} \mathbf{x} + \mathbf{x}^{\mathrm{T}} \boldsymbol{\nu}\right), \tag{11.27}$$

where $\mathbf{\Lambda}_{\mathbf{X}} = \mathbf{\Sigma}_{\mathbf{X}}^{-1}$ is the (Fisher) information matrix and $\boldsymbol{\nu} = \mathbf{\Lambda}_{\mathbf{x}} \mu_{\mathbf{X}}$ is the information vector (scaled mean).

The Kalman filter is derived when applying a Bayes filter with the moments parameterisation. The information filter is derived when applying a Bayes filter with the canonical parameterisation of a Gaussian. They can be considered duals of each other.

11.6 The Kalman filter family

Here are some variations of the Kalman filter (see Figure 11.3):

Kalman filter (KF) Uses Gaussian PDF assumption and linear system model.

Extended Kalman filter (EKF) Uses Gaussian PDF assumption and linear approximations of non-linear system and sensor models (using first term of Taylor series).

Unscented Kalman filter (UKF) Another Kalman filter for non-linear systems but with a different linear approximation (a stochastic linearisation).

Information filter (IF) This is like a Kalman filter but uses a canonical parameterisation of a Gaussian[8]. This can simplify the update equations[9].

[8] Instead of a mean vector and covariance matrix an information vector and information matrix is used. The (Fisher) information matrix is the inverse of the covariance matrix and the information vector is the product of the inverse covariance matrix and mean vector.

[9] Global uncertainty is simply a zero information matrix.

Extended information filter (EIF) This is similar to an extended Kalman filter but using an information filter.

Multi-hypothesis extended Kalman filter (MHEKF) This represents posteriors with a mixture (or sums) of Gaussians to represent multiple modes.

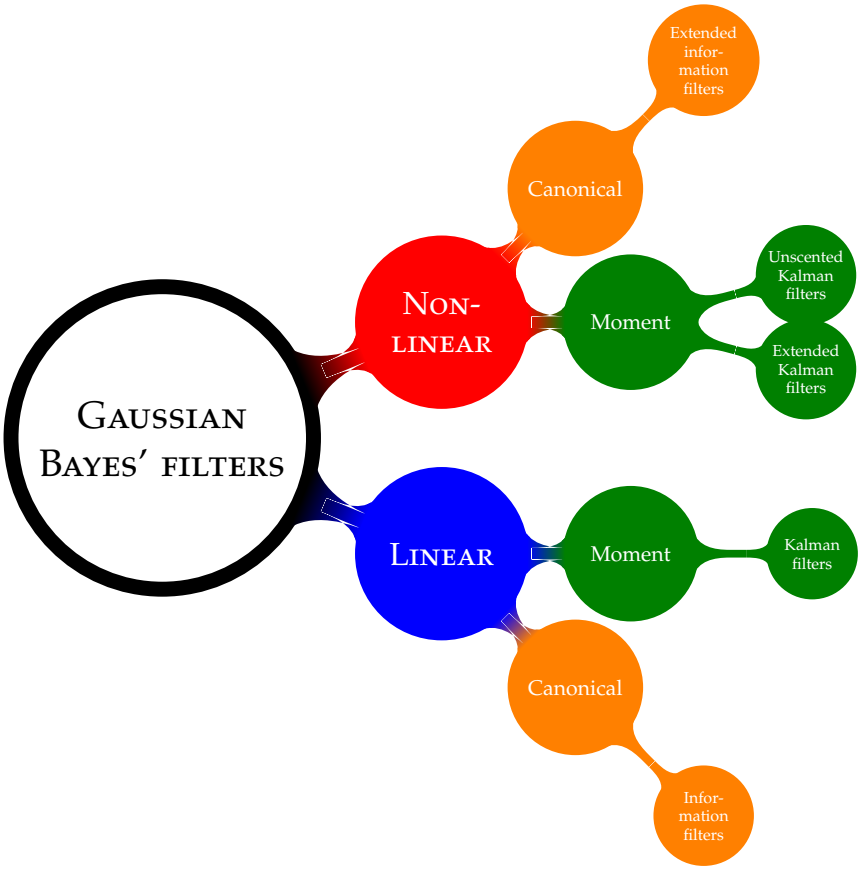


Figure 11.3: Categories of Gaussian Bayes' filters.

11.7 Summary

1. The Kalman filter is an on-the-fly (online) averaging filter.
2. The Kalman gain weights the contribution of each new measurement.
3. The Kalman filter stores only the estimated state and the estimated state covariance matrix.
4. The Kalman filter is only applicable for linear systems.
5. The Kalman filter requires the noise to be additive and Gaussian.

11.8 Exercises

1. Can a Kalman filter be applied to a non-linear system?

2. A Kalman filter is applied to estimate the location of a robot using two state variables. How many floating point values does the Kalman filter need to store at each iteration?

3. A Kalman filter is applied to estimate the pose of a robot using ten state variables. How many floating point values does the Kalman filter need to store at each iteration?

4. Under what conditions does the Kalman gain converge to a constant vector?

5. What is a steady state Kalman filter?

6. A simple robot has the state transition equation,

$$\begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} x_{n-1} \\ y_{n-1} \\ \theta_{n-1} \end{bmatrix} + \begin{bmatrix} v\Delta t \cos\theta_{n-1}, \\ v\Delta t \sin\theta_{n-1}, \\ 0 \end{bmatrix}, \tag{11.28}$$

   where x and y denote the Robot's location and θ denotes its heading. If the measurement vector is a linear function of the robot's state, what sort of Kalman filter would you recommend to estimate the robot's state?

7. If for the previous example the measurement vector is a non-linear function of the robot's state, what sort of Kalman filter would you recommend to estimate the robot's state?

8. If there are 1000 state variables, how many parameters does an extended Kalman filter need to track?

9. If an unmanned aerial vehicle (drone) has a constant linear and angular accelerations, is the motion model linear or non-linear?

10. What sort of Kalman filter would you recommend for pose estimation of a drone? State any assumptions.

11. A Kalman filter can be implemented using the follow-

ing equations:

$$\begin{aligned}
\hat{\mathbf{x}}_n^- &= \mathbf{A}\hat{\mathbf{x}}_{n-1} + \mathbf{B}\mathbf{u}_{n-1} \\
\mathbf{P}_n^- &= \mathbf{A}\mathbf{P}_{n-1}^+\mathbf{A}^\mathsf{T} + \mathbf{\Sigma}_\mathbf{W} \\
\mathbf{K}_n &= \mathbf{P}_n^-\mathbf{C}^\mathsf{T}\left(\mathbf{C}\mathbf{P}_n^-\mathbf{C}^\mathsf{T} + \mathbf{\Sigma}_\mathbf{V}\right)^{-1} \\
\hat{\mathbf{x}}_n^+ &= \hat{\mathbf{x}}_n^- + \mathbf{K}_n\left(\mathbf{z}_n - \mathbf{C}\hat{\mathbf{x}}_n^-\right) \\
\mathbf{P}_n^+ &= \left(\mathbf{I} - \mathbf{K}_n\mathbf{C}\right)\mathbf{P}_n^-.
\end{aligned}$$

Consider these equations and complete the table for a system with a three element state vector, a two element measurement vector, and a one element control vector.

Symbol	Description	Dimension
$\mathbf{z}_n$	*Measurement vector at time-step n*	2×1
$\mathbf{u}_{n-1}$		
$\hat{\mathbf{x}}_n^+$		
$\hat{\mathbf{x}}_n^-$		
$\mathbf{P}_n^+$		
$\mathbf{\Sigma}_\mathbf{W}$		
$\mathbf{A}$		
$\mathbf{C}$		
$\mathbf{K}_n$		
$\mathbf{\Sigma}_\mathbf{V}$		
$\mathbf{I}$		

12
Multivariate particle filters

Particle filters can be employed to estimate the belief of multiple state variables, for example, the location of a robot. They are useful for non-linear problems since particles can represent arbitrary PDFs. However, they can only estimate a few state variables due to exponential growth in the total number of particles required.

12.1 Number of particles

If you have N state variables and desire M particles per state, the total number of particles is

$$N^M. \tag{12.1}$$

For example, consider estimating the pose[1] of a drone. This comprises three positions and three angles, so $N = 6$. If $M = 10$ particles are required per state, then the required number of particles is

[1] Position and orientation.

$$10^{3+3} = 1 \times 10^6. \tag{12.2}$$

If the linear and angular speeds are also required, then $N = 12$ and the number of particles is

$$10^{3+3+3+3} = 1 \times 10^{12}. \tag{12.3}$$

This is an unfeasible number of particles to process.

12.2 Robot particle filter example

Consider a robot exploring a 2-D world. Let's assume it moves at constant speed and so the state vector representing its pose is $\mathbf{x}_n = (x_n, y_n, \theta_n)^{\mathrm{T}}$, where x_n, y_n is its position and θ_n is its heading at time-step n. Let's also assume that the robot has no idea of its position but it

has a sensor that can measure the range and bearing to a beacon. It knows where the beacons are but they all look the same.

Let's assume that the robot's world is bounded by $x_{\min} \leq x \leq x_{\max}$ and $y_{\min} \leq y \leq y_{\max}$ and its heading angle is bounded by $-\pi \leq \theta \leq \pi$. If the robot could be anywhere with any orientation, the initial belief is a three-dimensional uniform distribution.

A particle filter starts by sampling from the initial belief. The m^{th} particle is initialised by $\mathbf{x}_0^{(m)} = \left(x_0^{(m)}, y_0^{(m)}, \theta_0^{(m)}\right)^{\mathrm{T}}$, where

$$x_0^{(m)} \sim \mathcal{U}\left(x_{\min}, x_{\max}\right), \tag{12.4}$$
$$y_0^{(m)} \sim \mathcal{U}\left(y_{\min}, y_{\min}\right), \tag{12.5}$$
$$\theta_0^{(m)} \sim \mathcal{U}\left(-\pi, \pi\right). \tag{12.6}$$

Here $\sim$ denotes taking a sample from a specified PDF. Each particle also has a weight initialised to $a_0^{(m)} = 1$.

As the robot moves around making measurements we desire to determine its pose[2]. For each measurement, we need to update the weights of the particles based on the measurements. For each particle it determines the closest beacon and then compares the distance and bearing to the beacon with the distance and bearing measured by the robot. The particles with the smallest discrepancy will get higher weights.

[2] This is a localisation problem.

The range and bearing[3] measured by the robot is

[3] Note, the measurements are a non-linear function of $\mathbf{x}_n$.

$$r_n = \sqrt{(x_b(\mathbf{x}_n) - x_n)^2 + (y_b(\mathbf{x}_n) - y_n)^2} + v_{r,n}, \tag{12.7}$$
$$\phi_n = \text{angdiff}\left(\theta_n, \tan^{-1}\frac{y_b(\mathbf{x}_n) - y_n}{x_b(\mathbf{x}_n) - x_n} - v_{\phi,n}\right), \tag{12.8}$$

and the range and bearing measured by the m^{th} particle are

$$r_n^{(m)} = \sqrt{(x_b(\mathbf{x}_n^{(m)}) - x_n)^2 + (y_b(\mathbf{x}_n^{(m)}) - y_n)^2}, \tag{12.9}$$
$$\phi_n^{(m)} = \text{angdiff}\left(\theta_n^{(m)}, \tan^{-1}\frac{y_b(\mathbf{x}_n^{(m)}) - y_n}{x_b(\mathbf{x}_n^{(m)}) - x_n}\right). \tag{12.10}$$

In these equations $(x_b(\mathbf{x}_n), y_b(\mathbf{x}_n))$ are the coordinates of the beacon closest to the robot with state $\mathbf{x}_n$, $v_{r,n}$ represents range measurement noise, $v_{\phi,n}$ represents bearing measurement, and $\text{angdiff}(\theta_1, \theta_2)$ is a function[4] that calculates the difference between two angles so that the

[4] $\text{angdiff}(\theta_1, \theta_2) = \min\left(2\pi - (\theta_1 - \theta_2), \theta_1 - \theta_2\right)$.

result is between $-\pi$ and π. Note, that there is no measurement noise for the particles; they know where they and the beacons are.

The particle weights are chosen using the joint probability density of the range and bearing errors. If the range and bearing errors are independent, then the joint probability density is simply the product of the range and bearing probability densities. This simplifies the weight calculation to

$$a_n^{(m)} = a_{n-1}^{(m)} \, f_R\left(r_n - r_n^{(m)}\right) f_\Phi\left(\text{angdiff}\left(\phi_n, \phi_n^{(m)}\right)\right), \tag{12.11}$$

where $f_R(r)$ is the range error PDF and $f_\Phi(\phi)$ is the bearing error PDF[5]. Note, it is not necessary to get the amplitude scaling correct since the weights will be normalised.

[5] In practice, this will get poorer with r.

12.3 *Lost in the country particle filter example*

Imagine that you are lost in the pitch black countryside. All you can see are the lights of farmhouses[6] in the distance. You have a phone but no GPS or map. So how can you find out where you are?

[6] Did I say that all the farmers had loaded shotguns and were suspicious of strangers?

One approach is to phone all your friends and relations and tell them to spread themselves randomly around the countryside at random angles. You then phone them up and tell them the approximate distance and bearing to the nearest farmhouse you can see. They then perform the same exercise but since they have a map and GPS they can accurately determine the correct range and distance to the nearest farmhouse they can see. Using the information you gave them and their own measurement, they can assign a weight to the consistency of the measurements. If the weight is small then they are in the wrong location or facing in the wrong direction. If the weight is high, then they could have the same pose as you (or the same pose with respect to another farmhouse). You then tell the people whose measurements have a low weight to hurry to a new place in the countryside, preferably closer to the people with high weights. You then walk along for a while and tell everyone to do the same. After a while, you stop, estimate the range and bearing to the nearest farmhouse, phone your friends and get them to re-estimate their weights, etc. The process then repeats until hopefully

you find all your friends and relatives huddled around you.

This algorithm is analogous to a particle filter where each of your friends and relatives is a particle. Since they have no prior knowledge of where you are, they need to spread themselves out uniformly over the countryside—this models the initial belief that you could be anywhere. As you wander along they wander themselves at a similar speed. Then using the measurements you give them, they collectively reestimate the posterior density for their belief of your location[7].

[7] And orientation.

12.4 Exercises

1. If you need approximately 100 particles per state variable, how many particles are required if the state space has ten dimensions?
2. What is the point of the angdiff function in (12.8)?

13
Wheeled robot motion models

The goal of the motion model is to predict the pose of a robot given the previous pose. For a rigid mobile robot, the pose can be described by 3-D Cartesian coordinates and three Euler angles (pitch, roll, and yaw). If the robot is constrained to a plane, the pose is reduced to three variables: a 2-D location and a heading angle:

$$\mathbf{x}_n = \begin{bmatrix} x_n & y_n & \theta_n \end{bmatrix}^{\mathrm{T}}. \tag{13.1}$$

13.1 Deterministic motion models

There are two common mobile robot motion models:

Velocity motion model This only considers the control outputs, assumed to be velocity commands given to the robot motors.

Odometry motion model This requires odometry sensors[1]. It is more accurate than the velocity motion model but since it requires measurements it cannot be used for path planning.

[1] Usually by having encoders to measure the revolution of the robot's wheels. Alternatively, IMUs are used with GPS (when outside) or light sensors, such as: monocular cameras, stereo cameras, laser scanners, time-of-flight cameras, and structured light cameras.

13.2 Velocity motion model

The velocity motion model[2] assumes there are two control outputs:

Linear speed [3] v

Rotational speed [4] ω

The control vector[5] at time-step n is thus

$$\mathbf{u}_n = \begin{bmatrix} v_n \\ \omega_n \end{bmatrix}. \tag{13.2}$$

[2] For a differential drive robot.

[3] A positive value implies a forward motion.

[4] A positive value implies an anti-clockwise rotation.

[5] In the following the time dependence of the speed controls are dropped to simplify the notation.

In the following we assume that there is no process error, for example, there is no wheel slippage. There are three cases to consider.

Pure translation, $\omega = 0$. Here the robot moves along a straight path and the motion model is

$$\begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} x_{n-1} \\ y_{n-1} \\ \theta_{n-1} \end{bmatrix} + \begin{bmatrix} v\Delta t \cos\theta_{n-1} \\ v\Delta t \sin\theta_{n-1} \\ 0 \end{bmatrix}. \tag{13.3}$$

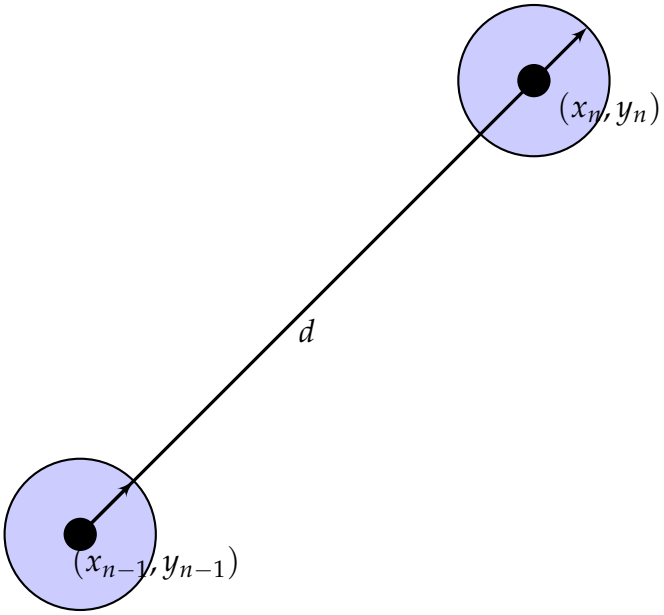


Figure 13.1: Velocity model for a zero angular speed. In this example, $\theta_n = \pi/4$. The distance travelled is $d = v\Delta t$.

Pure rotation, $v = 0$. Here the robot will rotate on the spot, and the motion model is

$$\begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} x_{n-1} \\ y_{n-1} \\ \theta_{n-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \omega_{n-1}\Delta t \end{bmatrix}. \tag{13.4}$$

Translation with rotation Here the robot moves on a circle of radius $r = |v/\omega|$ so the next pose[6] is given by

$$\begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} x_{n-1} \\ y_{n-1} \\ \theta_{n-1} \end{bmatrix} + \begin{bmatrix} -\frac{v}{\omega}\sin\theta_{n-1} + \frac{v}{\omega}\sin(\theta_{n-1} + \omega\Delta t) \\ \frac{v}{\omega}\cos\theta_{n-1} - \frac{v}{\omega}\cos(\theta_{n-1} + \omega\Delta t) \\ \omega\Delta t \end{bmatrix}. \tag{13.5}$$

The distance travelled along the arc between poses is $s_n = |v|\,\Delta t$.

[6] In practice, the updated heading needs to be converted into the region $-\pi \leq \theta < \pi$.

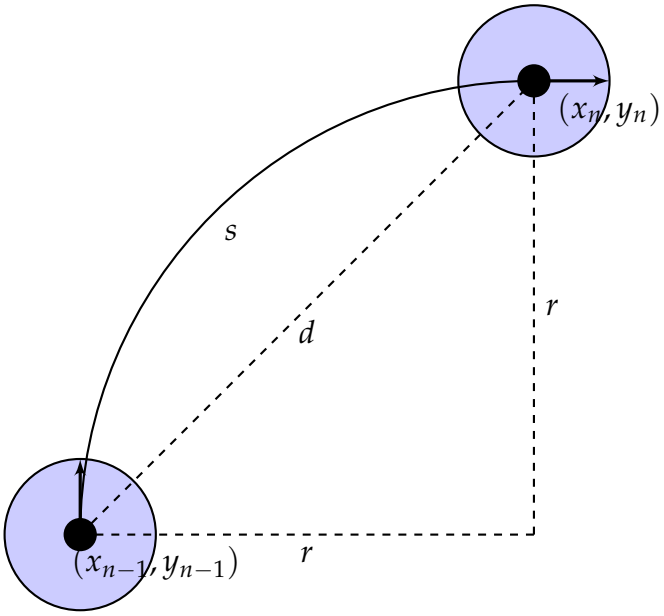


Figure 13.2: Velocity model for non-zero linear and angular speeds. In this example, $\theta_n = 0$. The distance travelled along the arc of radius, $r = v/\omega$, is $s = v\Delta t$. The Cartesian distance travelled is $d = 2r \tan \frac{\omega \Delta t}{2}$.

The motion model (13.5) can be summarised as

$$\mathbf{x}_n = g(\mathbf{x}_{n-1}, \mathbf{u}_{n-1}). \tag{13.6}$$

However, it cannot be expressed[7] as

$$\mathbf{x}_n = \mathbf{A}_n \mathbf{x}_{n-1} + \mathbf{B}_n \mathbf{u}_{n-1}. \tag{13.7}$$

[7] Due to the reciprocal of ω and trig. functions.

This means we cannot use a standard Kalman filter. We either need to linearise the motion model and apply the EKF or UKF or use a histogram or particle filter.

13.3 Odometry motion model

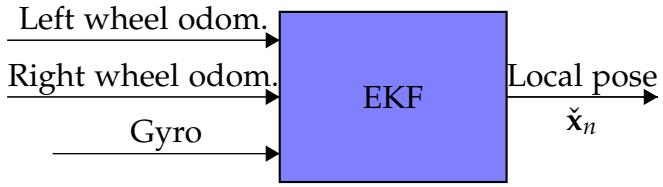


Figure 13.3: Block diagram for estimation of local pose using odometry and gyroscope measurements.

The odometry motion model is the state-of-the-art motion model. It is more accurate than the velocity motion model since it uses measurements.

Most wheeled robots have odometry sensors on the wheels to estimate the distance each wheel travels. This is fused[8] with IMU measurements to estimate the local pose of the robot,

$$\check{\mathbf{x}}_n = \left(\check{x}_n, \check{y}_n, \check{\theta}_n\right)^{\mathrm{T}}. \tag{13.8}$$

[8] The Turtlebot uses an EKF to fuse odometry with a gyro to estimate the local pose.

Here the check accent denotes a local pose (relative to the starting pose of the robot). Note, the local pose is subject to drift[9] since the errors accumulate.

Since odometry is a form of measurement it should be incorporated in the update step of a Bayesian filter[10]. In practice, a sleight-of-hand is employed to approximate a pair of local poses estimated from odometry as a control signal for the pose prediction.

[9] However, the drift between two consecutive local poses is small.

[10] Unfortunately, to achieve this requires augmenting the robot's state with velocity and thus the estimation problem becomes harder.

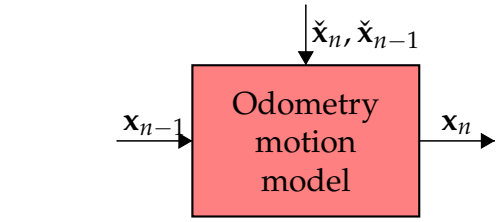


Figure 13.4: Block diagram for the odometry motion model. The check accent indicates local poses estimated from the odometry measurements.

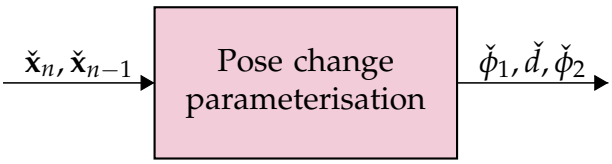


Figure 13.5: Parameterisation of a local pose change into $\check{\phi}_1$, $\check{d}$, and $\check{\phi}_2$.

The odometry motion model uses a change in the local pose to estimate the change in the global pose. In other words, given the current and previous local poses, $\check{\mathbf{x}}_n$ and $\check{\mathbf{x}}_{n-1}$, the odometry motion model infers the probability density of the transition of $\mathbf{x}_{n-1}$ to $\mathbf{x}_n$.

The odometry motion model parameterises a local pose change[11] as a rotation, ϕ_1, a translation, d, and another rotation, ϕ_2, (see Figure 13.6):

[11] The pose change parameterisation is denoted by the operator $\mathcal{D}$.

$$\left(\check{\phi}_1, \check{d}, \check{\phi}_2\right)^{\mathrm{T}} = \mathcal{D}\left\{\check{\mathbf{x}}_n, \check{\mathbf{x}}_{n-1}\right\}, \tag{13.9}$$

where

$$\begin{bmatrix} \check{x}_n \\ \check{y}_n \\ \check{\theta}_n \end{bmatrix} = \begin{bmatrix} \check{x}_{n-1} \\ \check{y}_{n-1} \\ \check{\theta}_{n-1} \end{bmatrix} + \begin{bmatrix} d\cos\left(\check{\theta}_{n-1} + \check{\phi}_1\right) \\ d\sin\left(\check{\theta}_{n-1} + \check{\phi}_1\right) \\ \check{\theta}_{n-1} + \check{\phi}_1 + \check{\phi}_2 \end{bmatrix}. \tag{13.10}$$

Assuming the error in the estimated local pose change is small, the global pose change can be parameterised the same as the local pose change, i.e.,

$$(d, \phi_1, \phi_2)^{\mathrm{T}} = \left(\check{d}, \check{\phi}_1, \check{\phi}_2\right)^{\mathrm{T}}, \tag{13.11}$$

and so given a previous global pose $(x_n, y_n, \theta_n)^{\mathrm{T}}$, the

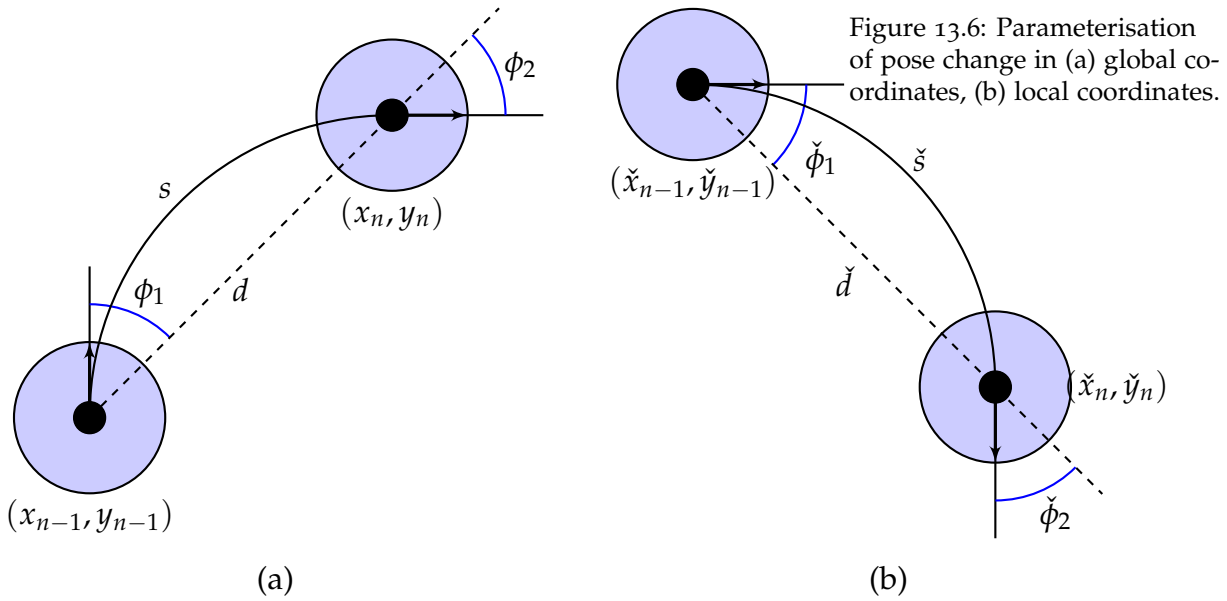


Figure 13.6: Parameterisation of pose change in (a) global coordinates, (b) local coordinates.

predicted global pose is found from

$$\begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} x_{n-1} \\ y_{n-1} \\ \theta_{n-1} \end{bmatrix} + \begin{bmatrix} d\cos(\theta_{n-1} + \phi_1) \\ d\sin(\theta_{n-1} + \phi_1) \\ \phi_1 + \phi_2 \end{bmatrix}. \qquad (13.12)$$

13.4 Summary

1. The velocity motion model is better for path planning.
2. The odometry motion model fudges the odometry measurements as a control output to avoid having to augment the robot pose with velocities.
3. The odometry measurement uses differences between local pose estimates to determine the conditional probability for the pose transition.
4. These motion models have ignored dynamics; thus a fast moving robot or heavily loaded robot will behave differently.

13.5 Exercises

1. What is the advantage of the odometry motion model over the velocity model?
2. Is the velocity motion model linear?
3. Is the odometry motion model linear?
4. How do the motion models handle dynamics?
5. What is the principle of the odometry motion model?
6. Why cannot the odometry model be used by a path planner?

14
Probabilistic wheeled robot motion models

Traditionally, kinematics has been considered deterministic but there is always uncertainty due to control noise or unmodelled exogenous effects. These uncertainties can be considered using *probabilistic kinematics*[1].

Probabilistic motion models are described by a conditional density $f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1};\mathbf{u}_{n-1})$. They have two uses:

1. To determine the belief of the next pose[2] given the belief of the current pose and the control output.
2. To sample from when propagating the particles in a particle filter.

[1] This collapses to deterministic kinematics when the modelled standard deviations are zero.

[2] Say for a histogram filter.

14.1 Probabilistic velocity motion model

One approach is to assume additive process noise:

$$\mathbf{X}_n = g(\mathbf{X}_{n-1}, \mathbf{u}_{n-1}) + \mathbf{W}_n(\mathbf{X}_{n-1}, \mathbf{u}_{n-1}). \tag{14.1}$$

However, the additive process noise is likely to be some complicated function of the speeds and previous pose. For example, when driving in the x-direction, the slip in the y-direction is likely to be small.

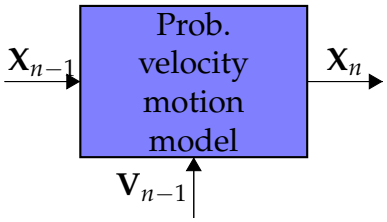


Figure 14.1: Block diagram for probabilistic velocity motion model where the control outputs are treated as random variables.

A better approach is to consider the speeds as random variables[3]:

$$\mathbf{X}_n = g(\mathbf{X}_{n-1}, \mathbf{V}(\mathbf{u}_{n-1})), \tag{14.2}$$

[3] Note, in the following $\mathbf{V}$ denotes the speed random vector and not sensor noise.

where

$$\mathbf{V}(\mathbf{u}) = \begin{bmatrix} V(\mathbf{u}) \\ \Omega(\mathbf{u}) \end{bmatrix}. \tag{14.3}$$

The random speed vector is parameterised by the control vector,

$$\mathbf{u} = \begin{bmatrix} v_d \\ \omega_d \end{bmatrix}, \tag{14.4}$$

where v_d and ω_d are the desired linear and angular speeds.

Assuming that V and Ω speeds are independent, their joint PDF simplifies to

$$f_{V,\Omega}(v, \omega; \mathbf{u}) = f_V(v; \mathbf{u}) f_\Omega(\omega; \mathbf{u}). \tag{14.5}$$

Here $f_V(v; \mathbf{u})$ and $f_\Omega(\omega; \mathbf{u})$ describe suitable PDFs[4] with mean values corresponding to the desired speeds. The tricky thing is how to choose the standard deviations of the distributions. A common approach is to assume that the error increases with the desired speeds, i.e.,

$$\begin{bmatrix} \sigma_V \\ \sigma_\Omega \end{bmatrix} = \begin{bmatrix} \alpha_1 & \alpha_2 \\ \alpha_3 & \alpha_4 \end{bmatrix} \begin{bmatrix} v_d \\ \omega_d \end{bmatrix}. \tag{14.6}$$

Here α_i are robot-specific error parameters[5].

[4] Say a Gaussian or a triangle distribution.

[5] These are smaller for more accurate robots.

14.1.1 *Sampling from the velocity motion model*

When a particle filter updates the poses of the particles it needs to sample from the conditional PDF describing the velocity motion model, $f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1}; \mathbf{u}_{n-1})$. This is achieved by sampling[6] v_{n-1} and ω_{n-1} from the joint error density, $f_{V_{n-1},\Omega_{n-1}}(v_{n-1}, \omega_{n-1}; \mathbf{u}_{n-1})$ (see (14.5)), and then mapping these speeds through the motion model (13.6) to find the new pose,

$$\mathbf{x}_n = g(\mathbf{x}_{n-1}, \mathbf{v}_{n-1}), \tag{14.7}$$

where $\mathbf{v}_{n-1} = (v_{n-1}, \omega_{n-1})^{\mathrm{T}}$.

[6] If the distributions are uniform or Gaussian, then the math functions `rand`/`uniform` and `randn` can be used. Otherwise it is necessary to use inverse transform sampling with the inverse of the desired cumulative distribution.

14.2 *Probabilistic odometry motion model*

The stochastic odometry motion model fudges the change between two local pose measurements as a control. The conditional motion PDF is thus

$$f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1}; \check{\mathbf{x}}_n, \check{\mathbf{x}}_{n-1}). \tag{14.8}$$

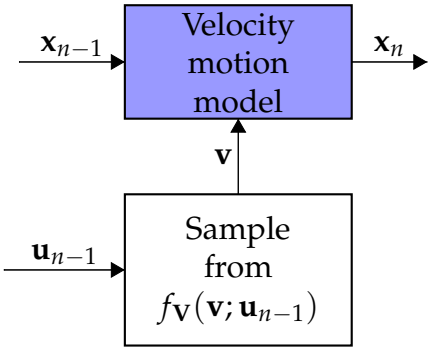


Figure 14.2: Block diagram showing how samples from the velocity motion-model PDF are chosen as errors when determining global pose transition.

This is approximated by a joint PDF that depends on the difference between the local and global pose change parameterisations,

$$f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1};\mathbf{u}_{n-1}) \approx f_{\mathbf{D}}(\mathbf{d}_{n-1};\check{\mathbf{d}}_{n-1}), \tag{14.9}$$

where $\mathbf{D} = (D, \Phi_1, \Phi_2)^{\mathrm{T}}$ is the global pose change parameterisation random vector, $\mathbf{d} = (d, \phi_1, \phi_2)^{\mathrm{T}}$ is the global pose change parameterisation vector, and $\check{\mathbf{d}} = \left(\check{d}, \check{\phi}_1, \check{\phi}_2\right)^{\mathrm{T}}$ is the local pose change parameterisation vector.

The joint PDF given by (14.9) can be simplified by assuming that D, Φ_1, and Φ_2 are independent:

$$f_{\mathbf{D}}(\mathbf{d};\check{\mathbf{d}}) \approx f_{\Phi_1}(\phi_1;\check{\mathbf{d}}) f_D(d;\check{\mathbf{d}}) f_{\Phi_2}(\phi_2;\check{\mathbf{d}}). \tag{14.10}$$

The PDFs for Φ_1, Φ_2, and D are often guessed to be Gaussian or triangular. Ideally, they should be determined from histograms of measured data. The standard deviations are commonly determined using

$$\sigma_{\Phi_1} = \alpha_1 |\phi_1| + \alpha_2 d, \tag{14.11}$$
$$\sigma_D = \alpha_3 (|\phi_1| + |\phi_2|) + \alpha_4 d, \tag{14.12}$$
$$\sigma_{\Phi_2} = \alpha_1 |\phi_2| + \alpha_2 d. \tag{14.13}$$

Here α_i are constants to be determined for each robot.

14.2.1 *Sampling from the odometry motion model*

When a particle filter updates the poses of the particles it needs to sample from the odometry motion model. This is done in two steps:

1. Sample[7] ϕ_1, d, and ϕ_2 from the joint PDF $f_{\mathbf{D}}(\mathbf{d};\check{\mathbf{d}})$.
2. Estimate the new pose from the previous pose:

$$x_n = x_{n-1} + d\cos(\theta_{n-1} + \phi_1), \tag{14.14}$$
$$y_n = y_{n-1} + d\sin(\theta_{n-1} + \phi_1), \tag{14.15}$$
$$\theta_n = (\theta_{n-1} + \phi_1 + \phi_2). \tag{14.16}$$

[7] Again it is easier to assume that the joint PDF is the product of three 1-D PDFs.

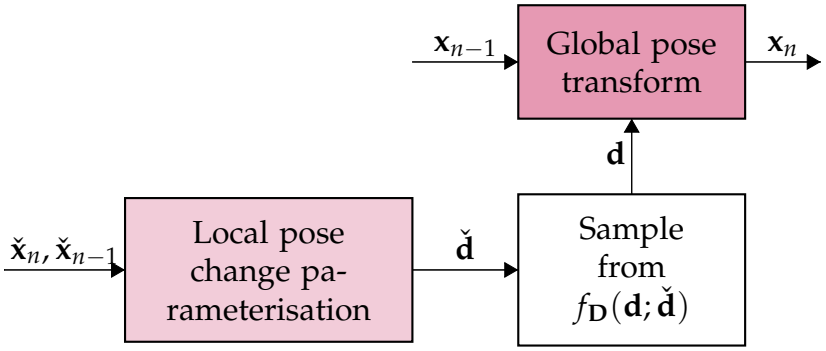


Figure 14.3: Block diagram showing how samples from the motion-model PDF are chosen as errors when determining global pose transition.

14.3 *Map-based motion models*

An occupancy map, $\mathbf{M}$, can be used to constrain a robot's position[8]. A map based motion model has a conditional state transition probability, $f_{\mathbf{X}_n|\mathbf{X}_{n-1},\mathbf{M}}(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{m}; \mathbf{u}_{n-1})$. This can be approximated by

$$f_{\mathbf{X}_n|\mathbf{X}_{n-1},\mathbf{M}}(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{m}; \mathbf{u}_{n-1}) = \eta \frac{f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1}; \mathbf{u}_{n-1})\, f_{\mathbf{X}_n|\mathbf{M}}(\mathbf{x}_n|\mathbf{m})}{f_{\mathbf{X}_n}(\mathbf{x}_n)}, \tag{14.17}$$

where η is a normaliser. The second factor expresses the consistency of the pose with the map[9].

[8] It is unlikely that a robot can move into an occupied cell.

[9] For example, it is zero for robot poses in an occupied grid cell.

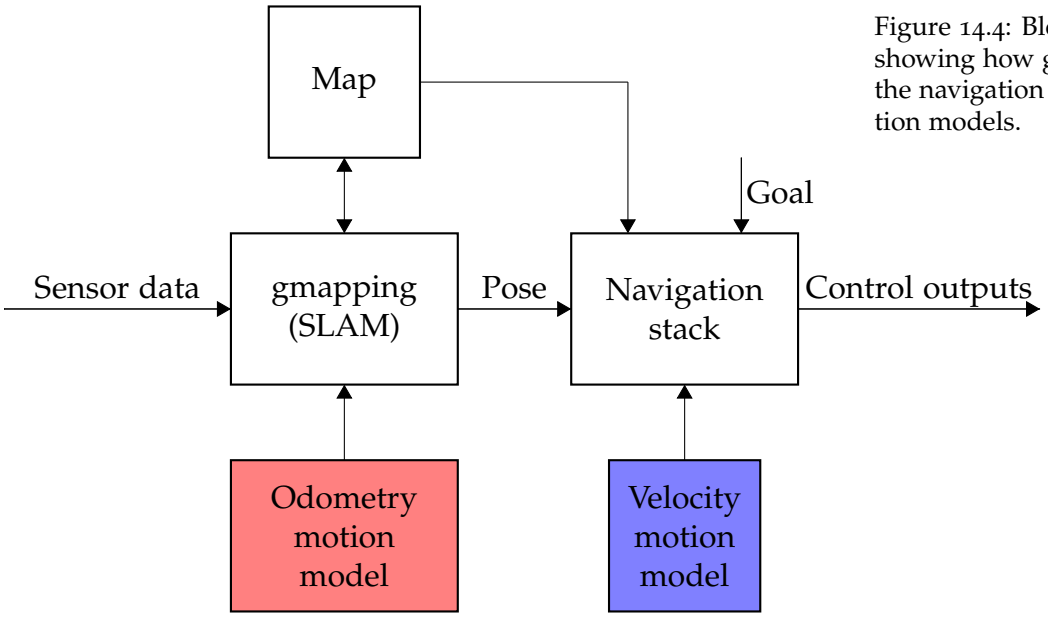


Figure 14.4: Block diagram showing how gmapping and the navigation stack use motion models.

14.4 Summary

1. The velocity model treats the control outputs as random variables.
2. The odometry measurement uses differences between local pose estimates to determine the conditional probability for the pose transition.
3. A map can assist a motion model.
4. Bayesian filters (except particle filters) require an inverse motion model.

14.5 Exercises

1. What are the two ways a probabilistic motion model is used?
2. If the joint PDF for the speeds is Gaussian, how would you choose $\mathbf{x}_n$ given $\mathbf{u}_{n-1} = (v_d, \omega_d)^{\mathrm{T}}$ and $\mathbf{x}_{n-1}$ for a particle filter?
3. When is the inverse motion model needed?
4. Comment on the validity of

$$f_{V,\Omega}(v, \omega; \mathbf{u}_{n-1}) = f_V(v; \mathbf{u}_{n-1}) f_\Omega(\omega; \mathbf{u}_{n-1}).$$

15
Mapping

Robots use maps to model the world but there are a number of problems:

Size The world is large and thus maps require a lot of storage and computing.

Noise Sensors and actuators are noisy.

Perceptual ambiguity Many places look the same.

Cycles A robot can traverse many paths throughout its world.

15.1 Maps

A map is a list of objects and their locations. There are two classes of map representation:

Feature based where each entry corresponds to a feature. For example, a feature might be a beacon or a road.

Location based where each entry corresponds to a location. For example, a topographic map or an occupancy map. In general, these are volumetric maps and thus can be wasteful of memory. Location based maps, such as an occupancy grid map indexed by robot position, make it easy to find paths.

There are many different map implementations, using grids (multidimensional arrays), graphs, etc. There is no universally 'best' implementation.

15.2 Occupancy maps

Occupancy maps are useful to determine if a location is occupied or free; this is useful for robot navigation since a robot cannot move into an occupied location. Most occupancy maps use grids to discretise the world.

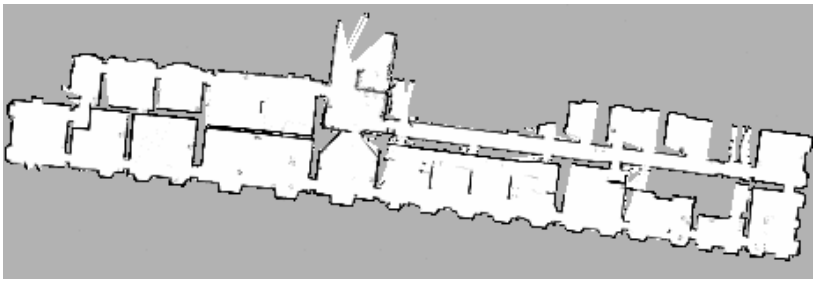


Figure 15.1: Occupancy grid belief example. White is free, black is occupied, grey is unsure. ROS occupancy values are given by $(255 - v)/255$ where v is the pixel value (in the range 0 to 255).

15.2.1 Occupancy probability

Occupancy maps represent a map, $\mathbf{M}$, of the environment as a collection of cells. Each cell is a binary random variable with two states: occupied and free, representing the presence of an obstacle at that location.

For example, consider a single cell, M. The probability mass function (PMF) for the cell, denoted $f_M(m)$, can be parameterised[1] by the occupancy probability, $p = P(m = \text{occupied})$, since

$$P(m = \text{free}) = 1 - P(m = \text{occupied}). \tag{15.1}$$

[1] The PMF is a Bernoulli distribution: $f_M(m; p) = p$ if $m = \text{occupied}$ and $f_M(m; p) = 1 - p$ if $m = \text{free}$.

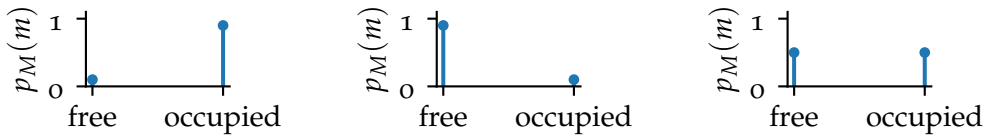


Figure 15.2: PMFs for three occupancy beliefs: likely, unlikely, no idea.

15.2.2 Occupancy grid joint PMF

Most occupancy grids are 2-D floor plans and thus the map $\mathbf{M}$ can be interpreted as a 2-D array[2] of $N_i \times N_j$ grid cells,

$$\mathbf{M} = \{M_{ij}\}, \tag{15.2}$$

where M_{ij} is a discrete random variable for the grid cell with indices i and j.

[2] Matrix.

Unfortunately, it is infeasible to calculate the joint PMF for all possible maps due to combinatorial explosion. To see this, consider a map of two cells, $\mathbf{M} = (M_0, M_1)^{\mathrm{T}}$. There are four possible maps:

The joint PMF[3] of the map is

$$f_{\mathbf{M}}(\mathbf{m}) = f_{M_0,M_1}(m_0, m_1). \tag{15.3}$$

This requires a $2^2 = 4$ element matrix to store the different combinations.

[3] The joint PMF for an occupancy grid is a multivariate Bernoulli distribution with $2^{N_i \times N_j} - 1$ parameters.

15.2.3 *Occupancy grid independence*

Let's now consider a small 2-D grid of 100×100 map cells. This has $2^{100 \times 100}$ possible map combinations. Since this is an extremely large number it is impossible to represent the belief for this map. Instead it is necessary to assume that each grid cell is independent[4], so that the joint PMF of the map is a product of the PMFs for each cell:

$$f_{\mathbf{M}}(\mathbf{m}) = \prod_{i,j} f_{M_{ij}}(m_{ij}). \tag{15.4}$$

This only requires $N_i \times N_j$ parameters which are the occupancy probabilities for the cells.

[4] This prevents modelling dependencies between neighbouring cells.

15.2.4 *Occupancy grid algorithm*

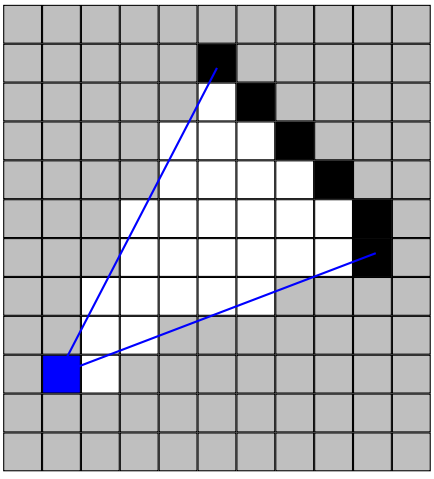


Figure 15.3: Occupancy grid scanning. The robot is at the blue square with position (x, y) and orientation θ. The blue lines denote the limits of the perceptual field of the sensor. The black cells at range r are considered occupied; the white cells within the scanning region are considered free; gray cells are unknown.

The belief of a map at the current time-step depends on all the sensor measurements from known locations[5].

[5] The control information, $\mathbf{u}_{0:n-1}$, is not required since the path is specified by $\mathbf{x}_{0:n}$.

With the assumption that each cell is independent:

$$f_{\mathbf{M}_n|\mathbf{Z}_{0:n},\mathbf{X}_{0:n}}\left(\mathbf{m}_n|\mathbf{z}_{0:n},\mathbf{x}_{0:n}\right) \approx \prod_{ij} f_{M_{ijn}|\mathbf{Z}_{0:n},\mathbf{X}_{0:n}}\left(m_{ijn}|\mathbf{z}_{0:n},\mathbf{x}_{0:n}\right). \tag{15.5}$$

A discrete Bayes filter algorithm[6] to calculate the occupancy grid probabilities is shown in Algorithm 1.

[6] Note, there is no predict step since the map is assumed to be static.

for *each measurement* $\mathbf{z}_n$ *at pose* $\mathbf{x}_n$**:**
  for *each map cell* i, j *in perceptual field of* $\mathbf{z}_n$**:**
    $$P(M_{ij}) = P(M_{ij}) \times \frac{l_{\mathbf{X}_n|M_{ij},\mathbf{Z}_n}(\mathbf{x}_n|m_{ij},\mathbf{z}_n)}{f_{\mathbf{Z}_n,\mathbf{X}_n}(\mathbf{z}_n,\mathbf{x}_n)}$$

Algorithm 1: Occupancy grid algorithm for the posterior belief. $P(M_{ij}) = f_{M_{ij}}(m_{ij})$ is the probability for cell M_{ij} being occupied. It is initialised with the prior that m_{ij} is occupied, usually 0.5. $l_{\mathbf{X}_n|M_{ij},\mathbf{Z}_n}(\mathbf{x}_n|m_{ij},\mathbf{z}_n)$ is the likelihood of $\mathbf{x}_n$ given measurement $\mathbf{z}_n$ and m_{ij}.

The measurements assign likelihoods using:

$$l_{\mathbf{X}_n|M_{ij},\mathbf{Z}_n}(\mathbf{x}_n|m_{ij},\mathbf{z}_n) = \begin{cases} P_{\text{occupied}} & \text{if grid cell at measured range} \\ P_{\text{free}} & \text{if grid cell closer than measured range} \\ 0.5 & \text{if grid cell beyond measured range} \end{cases} \tag{15.6}$$

The values of P_{occupied} and P_{free} depends on the amount of evidence a measurement carries. If the measurements were noise free, then $P_{\text{occupied}} = 1$ and $P_{\text{free}} = 0$.

for *each measurement* $\mathbf{z}_n$ *at pose* $\mathbf{x}_n$**:**
  for *each map cell* i, j *in perceptual field of* $\mathbf{z}_n$**:**
    $$\text{lo}(M_{ij}) = \text{lo}(M_{ij}) + \log \frac{l_{\mathbf{X}_n|M_{ij},\mathbf{Z}_n}(\mathbf{x}_n|m_{ij},\mathbf{z}_n)}{f_{\mathbf{Z}_n|\overline{M}_{ij},\mathbf{X}_n}(\mathbf{z}_n|m_{ij},\mathbf{x}_n)}$$

Algorithm 2: Occupancy grid algorithm using log odds for the posterior belief, c.f., Algorithm 1. $\text{lo}(M_{ij})$ is the log odds for cell m_{ij} being occupied (usually initialised with a prior of 0), $l_{\mathbf{X}_n|M_{ij},\mathbf{Z}_n}(\mathbf{x}_n|m_{ij},\mathbf{z}_n)$ is the likelihood of measurement $\mathbf{z}_n$ given m_{ij} is occupied, and $f_{\mathbf{Z}_n|\overline{M}_{ij},\mathbf{X}_n}(\mathbf{z}_n|m_{ij},\mathbf{x}_n)$ is the likelihood of measurement $\mathbf{z}_n$ given m_{ij} is free.

In practice, log odds are used to reduce numerical instabilities when $P \approx 0$ or $P \approx 1$,

$$l = \log \frac{P}{1-P}. \tag{15.7}$$

The probability is easily found from the log odds:

$$P = 1 - \frac{1}{1 + \exp(\text{lo})}. \tag{15.8}$$

P	lo
0.99	4.9
0.9	2.2
0.5	0
0.1	−2.2
0.01	−4.9

Table 15.1: Mapping between probabilities and log odds.

15.2.5 *Loop closure*

Grid maps are difficult to correct when there is a steady drift or jumps. For example, consider the case where a robot navigates in a loop back to its known starting position. If the odometry is inaccurate, then the map of its path will not form a closed curve, see Figure 15.4.

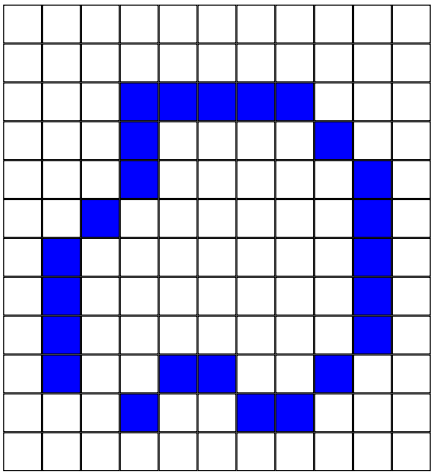


Figure 15.4: The loop closure problem with a grid map. A robot explores its environment and then returns to its starting point; however, due to errors in localisation the robot's path (denoted by blue cells) is not closed.

15.3 *Topological maps*

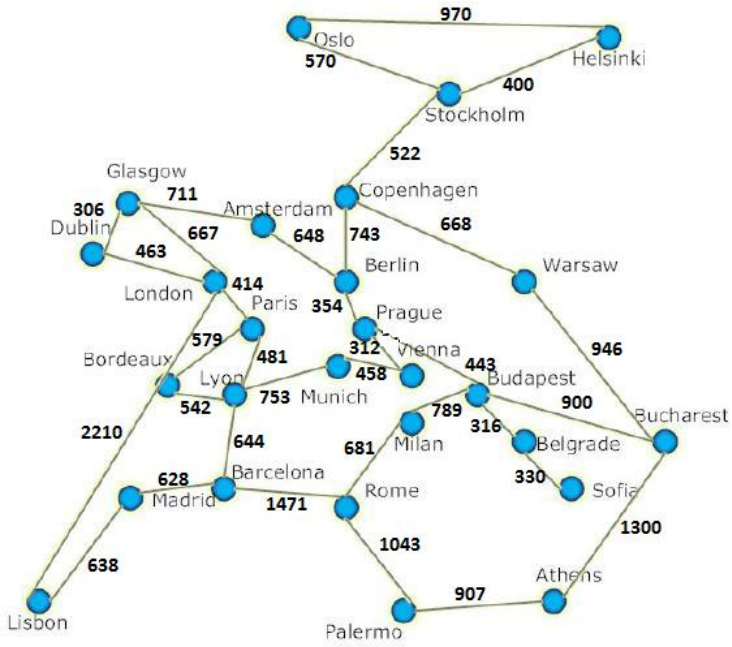


Figure 15.5: Example topological map showing some of the cities in Europe; the edges indicate the distances between the cities.

Topological maps (or graph based maps) store the map of the environment using a graph data structure. In the graph, each vertex represents robot and landmark poses, while the edges represent constraints on the relative poses of the two nodes. Since topological maps do not rely on metric measurements, they avoid some of the difficulties scaling to large environments. Additional advantages include:

- Efficient and compact representation of the map.
- Logical organisation for tasks such as path planning. The map is stored as a graph and this makes it possible to use existing graph algorithms, such as finding the shortest path between non-adjacent vertices.
- Loop closure on a local graph-based map is automatic and straight forward, as it just involves adding an edge between the start and end vertices. When a global map is required, a geometric spanner (t-spanner) can be used to close distant gaps introduced by the loop closure.
- If two places appear similar, then loop closure is attempted resulting in a data association failure.

15.4 Hybrid maps

Hybrid maps have been developed to take advantage of the complementary strengths that grid and topological maps provide. For example, the map may be partitioned as independent local maps and these are stored in a graph structure. This is called sub-mapping. To produce a global map, the Dijkstra projection is used to find the pose of each submap relative to a single reference frame. However, this only works if there are no loops, otherwise a non-linear least squares optimisation is required.

15.5 Sensor fusion

Using multiple sensors to generate a map is difficult since different sensors respond to the environment differently. For example, a laser scanner can 'see' through glass whereas an ultrasonic sensor cannot.

The popular solution to generating a map with different sensor modalities is to create separate maps for each sensor and then fuse the maps together. For example, consider an occupancy grid. Denoting M_{ij}^k as the ij^{th} map cell estimated with the k^{th} sensor, the PMF of the pessimistic belief for the combined map cell is

$$f_{M_{ij}}(m_{ij}) = \max_k f_{M_{ij}^k}(m_{ij}). \tag{15.9}$$

If any sensor says a cell is occupied then so will the combined map cell.

15.6 Dynamic environments

One approach to deal with dynamic environments is to create both short-term and long-term maps. The short term map contains recent sensor data, and as it does not contain significant odometry error, it is used for obstacle avoidance and localisation. Once a short-term map has matured, it is then added to the long-term map and used for navigation and path-planning.

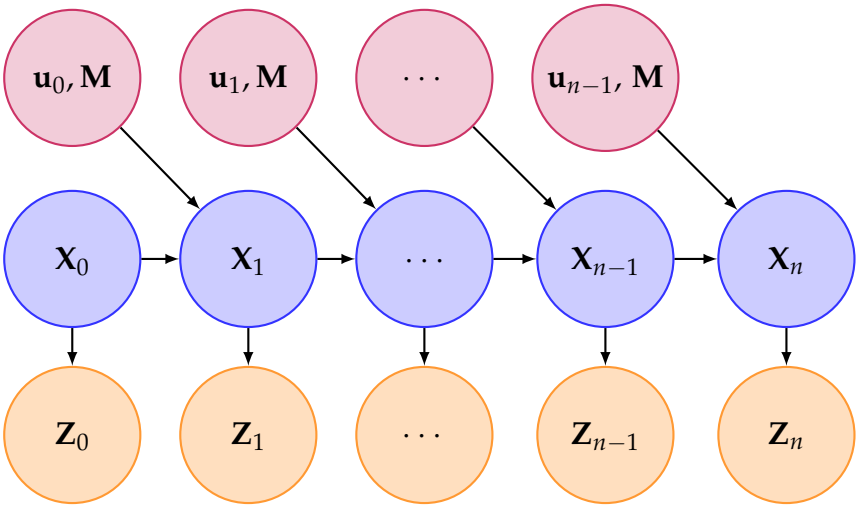


Figure 15.6: Diagram of the mapping problem. The goal is to determine the belief of the map, $\mathbf{M}$, given the known poses, $\mathbf{X}_n = \mathbf{x}_n$, of the robot and the measurements, $\mathbf{Z}_n = \mathbf{z}_n$.

15.7 Summary

1. Occupancy grids are simple to implement and view.
2. Occupancy grids provide a dense representation of the environment.
3. Occupancy grids provide an explicit representation of occupied and free space, useful for path planning.
4. Occupancy grids can be created with a prior map, say a floorplan.
5. A rectangular grid is not an efficient representation for non-rectangular environments.
6. There is a trade-off between grid resolution and computational complexity.
7. Graph-based maps are efficient and compact.
8. Loop-closure is difficult for grid-based maps.
9. Loop-closure is automatic for graph-based maps.

15.8 Exercises

1. What is the difference between a feature-based and a location-based map?
2. What is the difference between a grid map and a topological map?
3. If an occupancy grid cell has a value of 0.8, what would you concur?

4. If an occupancy grid cell has a value of 0.5, what would you concur?
5. How is an occupancy grid obtained?
6. If a robot has not viewed a region in the occupancy grid, what cell values do you expect there?
7. What are log odds?
8. How should maps from different sensors be combined?
9. What are short-term maps primarily used for?
10. Why is loop closure difficult with a grid map?
11. Why are the map cells considered independent.
12. A grid map is comprised of 200×100 cells. If each cell can be either occupied or free, how many different maps are there?

16
Localisation

Localisation is a position estimation problem. The task is to establish a correspondence between a map (described in a global coordinate system) with the robot's local coordinate system.

The pose of a robot cannot be sensed directly[1]; instead it must be inferred. Due to ambiguities, it requires multiple measurements[2].

The goal of localisation is to determine the belief of the robot pose given a map and sensor measurements,

$$\mathbf{X}_n | \mathbf{X}_0, \mathbf{Z}_{0:n}, \mathbf{M}_n; \mathbf{u}_{0:n-1}, \tag{16.1}$$

where $\mathbf{M}_n$ denotes the current map[3]. Prior information, such as the initial pose, $\mathbf{X}_0$, is incorporated using a Bayes filter.

Localisation assumes that the map is known but, in practice, this needs to be estimated as well (using a simultaneous mapping and localisation (SLAM) algorithm).

[1] There is no noise-free pose sensor.

[2] For example, pretend you are a robot in the corridor of a large hospital. How would you determine which wing your were in or on which floor you were on?

[3] This is a random quantity to account for errors in the map.

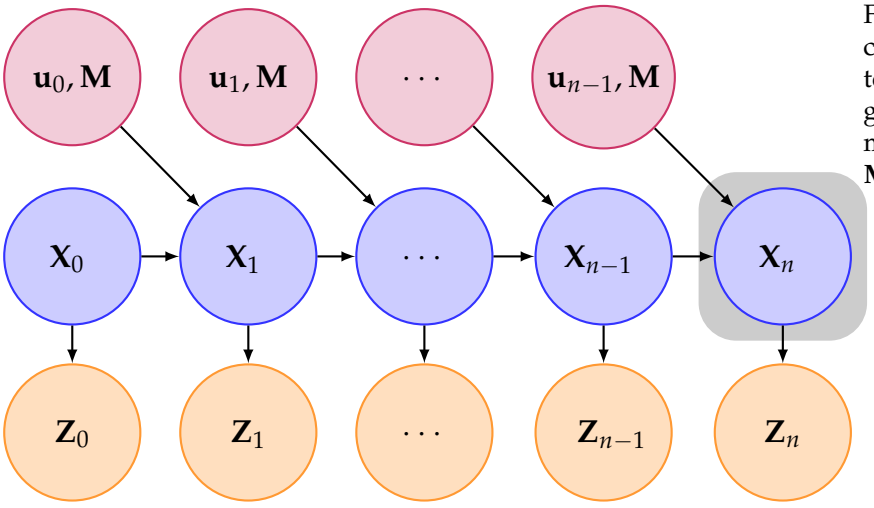


Figure 16.1: Diagram of the localisation problem. The goal is to estimate the pose belief, $\mathbf{X}_n$, given the control outputs, $\mathbf{u}_n$, measurements, $\mathbf{Z}_n$, and a map, $\mathbf{M}$ (here assumed unchanging).

16.1 Localisation problems

There are a number of localisation problems, including:

position tracking — the initial pose is known

global localisation — the initial pose is unknown

kidnapped robot problem — variant of global localisation where a robot may be repositioned

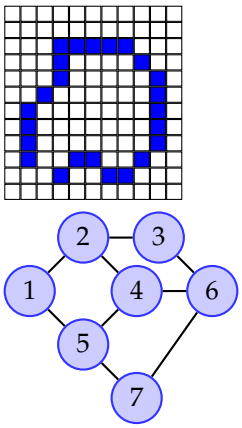


Figure 16.2: Location-based maps can be represented by grids or graphs.

16.2 Localisation algorithms

There are many approaches to the localisation problem. The algorithms can be categorised by:

- The map representation: grids or graphs.
- The use of features or dense measurements.
- The measurement noise model.
- The representation of the posterior pose distribution: Gaussians, histograms, or particles.

Most localisation algorithms employ the *Markov assumption* (complete state assumption) so that only the previous state of the robot is required and not the entire history. For robot localisation, the Markov assumption is a good approximation but fails if a measurement bias is introduced, say if the robot crashes or a wheel gets worn.

The advantage of having a multimodal pose posterior is that multiple hypotheses are supported. Thus a robot can recover from ambiguous situations, say when it crashes into something and the odometry is lost.

16.2.1 EKF localisation

EKF localisation algorithms use a continuous representation of the robot pose (x, y, θ) and linearised motion and sensor models. They are precise and efficient but need to start with an initially known position.

The disadvantage of EKF localisation is that its pose beliefs are unimodal (single hypothesis). The multiple hypothesis tracking (MHT) algorithm overcomes this using a mixture of Gaussians to support multiple hypotheses[4].

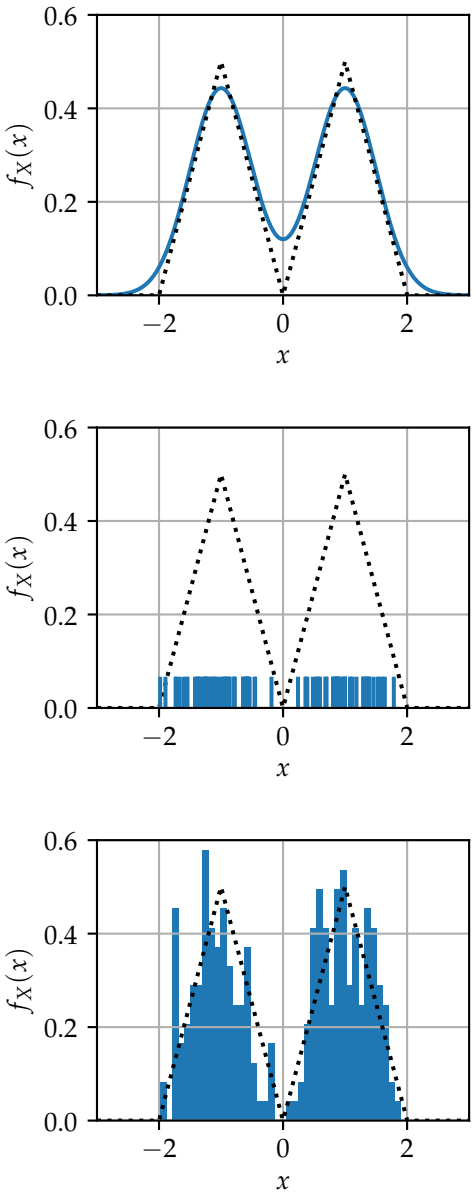


Figure 16.3: Posteriors can be represented by mixtures of Gaussians, particles, and histograms.

16.2.2 Monte-Carlo localisation (MCL)

Monte-Carlo localisation (MCL) represents the pose posterior using particles. MCL is popular since it is easy to implement and it can approximate nearly any distribution. AMCL (Adapative Monte-Carlo Localisation) chooses the number of particles dynamically[5].

By adding random particles, the kidnapped robot problem can be solved. However, many additional particles are required and this slows the particle filter.

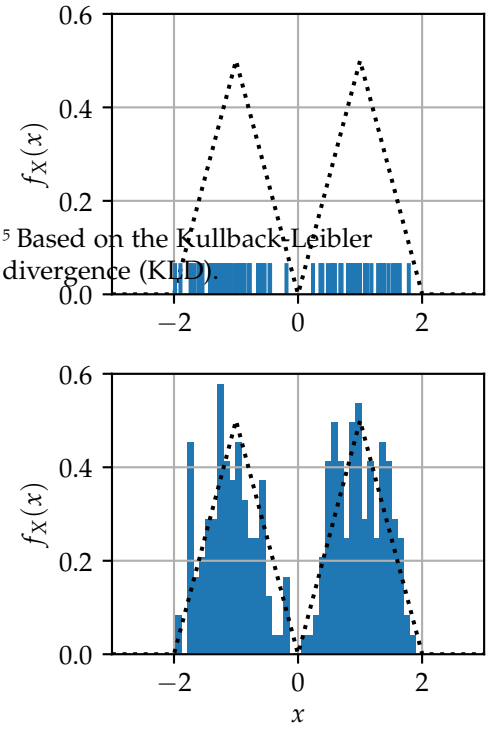


Figure 16.5: Posterior represented by a histogram.

16.2.3 Grid localisation

Grid localisation uses a histogram filter so multiple hypotheses can be tracked. Usually a fixed partitioning of pose space is employed, typically, 15 cm × 15 cm for location and 5° for heading. A finer representation yields better results but is slower.

16.3 SLAM

Simultaneous localisation and mapping (SLAM) is difficult to solve in practice, since an unbiased map is required for localisation, while an accurate pose estimation is needed to build a map. Moreover, errors in the map will propagate into the localisation estimate and vice-versa.

16.3.1 Online SLAM

This estimates the belief of the current robot pose along with the map given all the past measurements and controls:

$$\mathbf{X}_n, \mathbf{M}_n | \mathbf{X}_0, \mathbf{Z}_{0:n}; \mathbf{u}_{0:n}. \tag{16.2}$$

This is called online SLAM since it only estimates variables that persist at time-step n. Note, many algorithms assume the complete state (Markov) approximation and so discard[6] past measurements and controls. In practice, the map is assumed to be unchanging, but the belief of the map will change as more measurements are obtained.

[5] Based on the Kullback-Leibler divergence (KLD).

[6] This is important to avoid filling up memory of a computer.

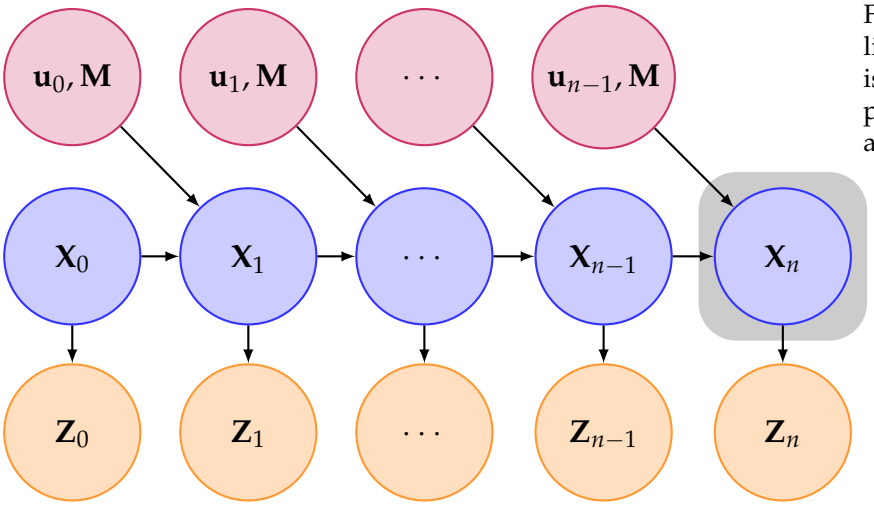


Figure 16.6: Diagram of the online SLAM problem. The goal is to estimate the current robot pose and the map. The map is assumed to be unchanging.

16.3.2 *Full SLAM*

This estimates the entire pose history, $\mathbf{X}_{0:n}$, along with the map,

$$\mathbf{X}_{0:n}, \mathbf{M}_n | \mathbf{X}_0, \mathbf{Z}_{0:n}; \mathbf{u}_{0:n}. \tag{16.3}$$

Full SLAM is very slow since for each new control, $\mathbf{u}_n$, and measurement, $\mathbf{Z}_n = \mathbf{z}_n$, all the previous pose estimates, $\mathbf{X}_{0:n-1}$, need to be re-estimated.

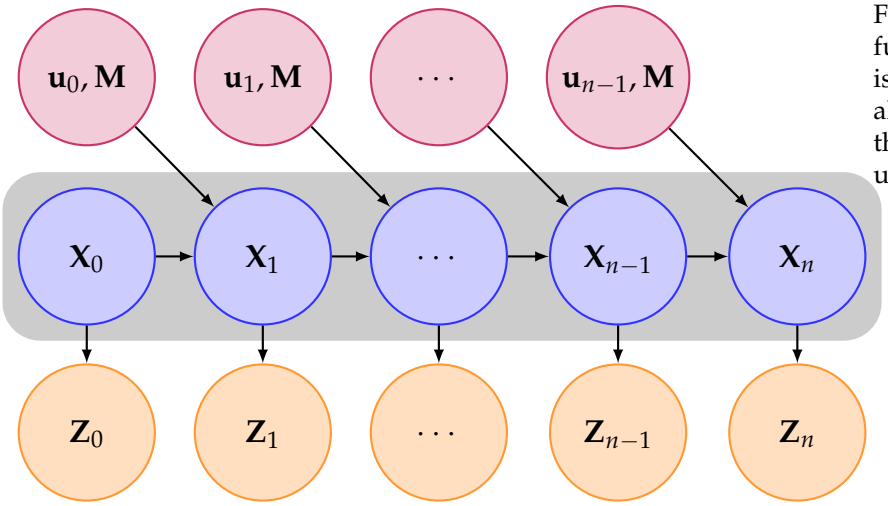


Figure 16.7: Diagram of the full SLAM problem. The goal is to estimate **all** robot poses along with the map. Again, the map is assumed to be unchanging.

16.4 *SLAM algorithms*

There are many SLAM algorithms, due to the choice of map representation, choice of posterior representation for the pose and map, sensor noise model, etc.

The chief difficulty of applying a Bayes filter to the SLAM problem is the dimensionality of the state space;

not only does it need to maintain the belief of the robot pose, it also needs to maintain the belief of its map.

Using a particle filter for both the pose and map is infeasible[7]. To reduce the dimensionality of the problem, two popular SLAM algorithms, FastSLAM and Gmapping, use Rao-Blackwellized particle filters. These exploit the idea that knowledge of the robot's true path allows landmark positions to be conditionally independent. Thus the posterior distribution can be factorised. A particle filter can then be used to track the robot's pose and an EKF can be employed to track each map feature.

[7] The required number of particles grows exponentially with the dimension of the state space.

16.4.1 Loop closure

Loop closure[8] is the problem of recognising a previously visited location and updating the beliefs of the location and map accordingly. This is easier to implement for graph-based maps compared to grid-based maps.

[8] Have I been there before?

16.4.2 Frontier exploration

This is the problem of deciding to move next in order to build the map as efficiently as possible. Multiple robots can use "Multi agent SLAM" algorithms to coordinate the exploration.

16.5 Summary

1. Localisation involves estimating the robot's pose, given a motion model, a sensor model, control outputs, sensor measurements, and a map.
2. There are many localisation algorithms, based on different Bayes filters to represent the posterior pose distribution, e.g., EKF, histogram filter, particle filter.
3. Features greatly reduce the quantity of measurement data to process. However, a lot of information is lost using features[9].
4. As computers have become faster, there has been less reliance on features for localisation. Instead, dense measurements are used for state-of-the-art localisation algorithms.
5. Position tracking requires a known initial pose.

[9] You can probably recognise where you are by looking around. However, if you only looked at the locations of doors, your localisation will be more ambiguous.

6. Localisation estimates the latest pose from the measurements given a map and initial pose: $\mathbf{X}_n|\mathbf{X}_0, \mathbf{Z}_{0:n}, \mathbf{M}_n; \mathbf{u}_{0:n}$.
7. Online SLAM estimates the latest pose and the map: $\mathbf{X}_n, \mathbf{M}_n|\mathbf{X}_0, \mathbf{Z}_{0:n}; \mathbf{u}_{0:n}$.
8. Full SLAM estimates all poses and the map: $\mathbf{X}_{0:n}, \mathbf{M}_n|\mathbf{X}_0, \mathbf{Z}_{0:n}; \mathbf{u}_{0:n}$.
9. Mapping estimates the map from the measurements given the pose: $\mathbf{M}_n|\mathbf{Z}_{0:n}, \mathbf{X}_n; \mathbf{u}_{0:n}$.
10. Many SLAM algorithms use a particle filter to estimate the pose and EKFs to estimate the map features.

16.6 Exercises

1. What sort of Bayesian filters are commonly used for localisation?
2. What is the difference between position tracking and global localisation?
3. What is the kidnapped robot problem?
4. What is the MCL algorithm?
5. What is the AMCL algorithm?
6. What is a disadvantage of using features compared to dense sensor measurements?
7. What is an advantage of using features compared to dense sensor measurements?
8. Why is SLAM difficult?
9. Why do SLAM algorithms often use a particle filter for the robot's pose?
10. Why do SLAM algorithms not use a particle filter for the map?
11. What is the difference between the online and full SLAM algorithms?
12. What Bayes filtering is used by the gmapping algorithm?
13. Explain what the gmapping algorithm does and how it uses Bayesian estimation.

17
Navigation

There are many algorithms for robot navigation. There are two types:

Local navigation applies to navigation over short distances, often less than the maximum range of the obstacle detecting sensor(s). The primary application is obstacle avoidance. Current sensor data is used and thus it handles dynamic environments.

Global navigation is also known as *path planning*. It is employed for travelling between two locations and requires the use of a map.

Some navigation algorithms are probabilistic; others are deterministic.

Most robots use a set of navigation algorithms for motion planning. These execute at different rates, e.g., global path planners (e.g., A*, 0.1 Hz), mid-level path deformation (e.g., elastic band, 5 Hz), and collision / obstacle avoidance algorithms (20 Hz).

17.1 Local navigation algorithms

Local navigation is usually reactive. There are three common approaches:

Potential-field based where each obstacle has an obstacle 'force field' for repelling the robot, and the goal has a attraction field[1].

Dynamics based where the algorithm consider the robot's dynamics in calculating a solution. (e.g., Velocity Obstacles, Dynamic Window Approach, and Trajectory Rollout).

Sampling based where various collision free states are sampled and then combined, (e.g., Reachability graph, Probabilistic roadmaps).

[1] A similar approach is Vector-Field Histograms (VFH) and Virtual Force Field. VFH also considers dynamics.

17.2 *Dynamic window approach (DWA)*

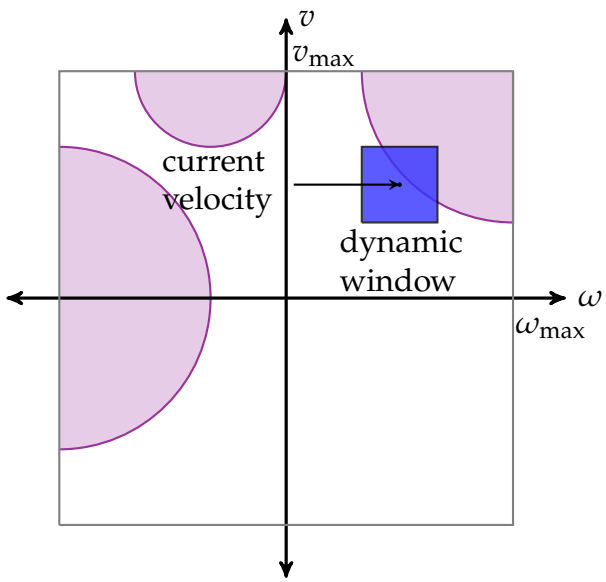


Figure 17.1: Dynamic window approach for a non-holonomic robot with differential drive. The blue box denotes the dynamic window surrounding the current velocity vector (these are the achievable velocities given acceleration constraints). The outer box denotes the limits of the robot's velocity. The violet regions indicate inadmissible velocities that will result in obstacle crashes.

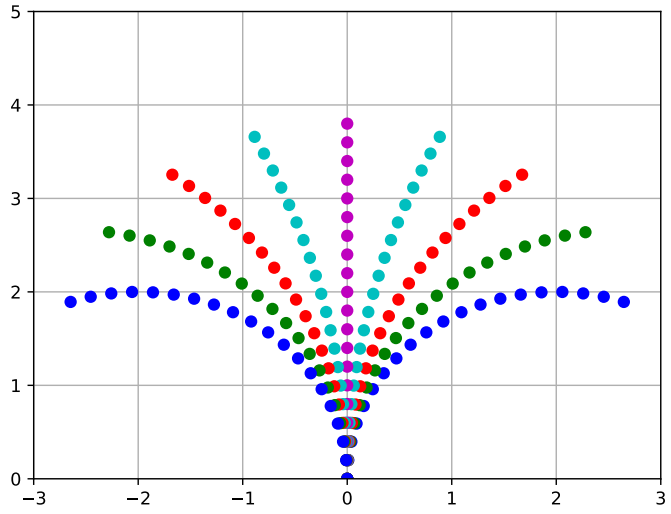


Figure 17.2: Path trajectories for $v = 2\,\text{m/s}$ and ω =-1, -0.75, -0.5, -0.25, 0.25, 0.5, 0.71, and 1 rad/s.

The dynamic window approach (DWA) operates directly in the velocity space of a robot. It has two steps:

1. Generate a valid search space by finding the (translational and rotational) velocities that lead to a safe trajectory in a short time interval without collision.
2. Maximize an objective function so that the chosen velocities moves the robot to the goal with maximum clearance from any obstacle.

DWA reduces the search space for possible velocities with the following criteria:

1. Dynamic window: the velocity pairs must be possible to achieve within a short time interval, given the acceleration limits of the robot[2].
2. Admissible velocities: only safe velocities are considered, i.e., velocities which allow the robot to stop before it reaches the closest obstacle on a trajectory.

Once the search space for the velocities has been generated, the velocity is selected that maximizes the objective function, a weighted sum of three components:

1. Heading to the target.
2. Clearance to the closest obstacle.
3. Forward velocity of the robot.

The Dynamic Window Approach (DWA) algorithm can be summarised as (see Listing 17.1):

1. Discretely sample in the robot's control space (v, ω)[3].
2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed.
4. Discard illegal trajectories (those that collide with obstacles).
5. Pick the highest-scoring trajectory and send the associated velocity to the robot.

[2] If the time-step is Δt, the current linear speed is v, and the maximum linear acceleration is a, then the dynamic window for the linear speed is set by the range $(v - a\Delta t, v + a\Delta t)$. Similarly, if the current angular speed is ω, and the maximum angular acceleration is α, then the dynamic window for the angular speed is set by the range $(\omega - \alpha\Delta t, \omega + \alpha\Delta t)$.

[3] For a holonomic robot on a plane, the control space is (v_x, v_y, ω) often expressed as $(\Delta x, \Delta y, \Delta\theta)$ assuming a fixed time interval Δt.

17.3 *Trajectory rollout*

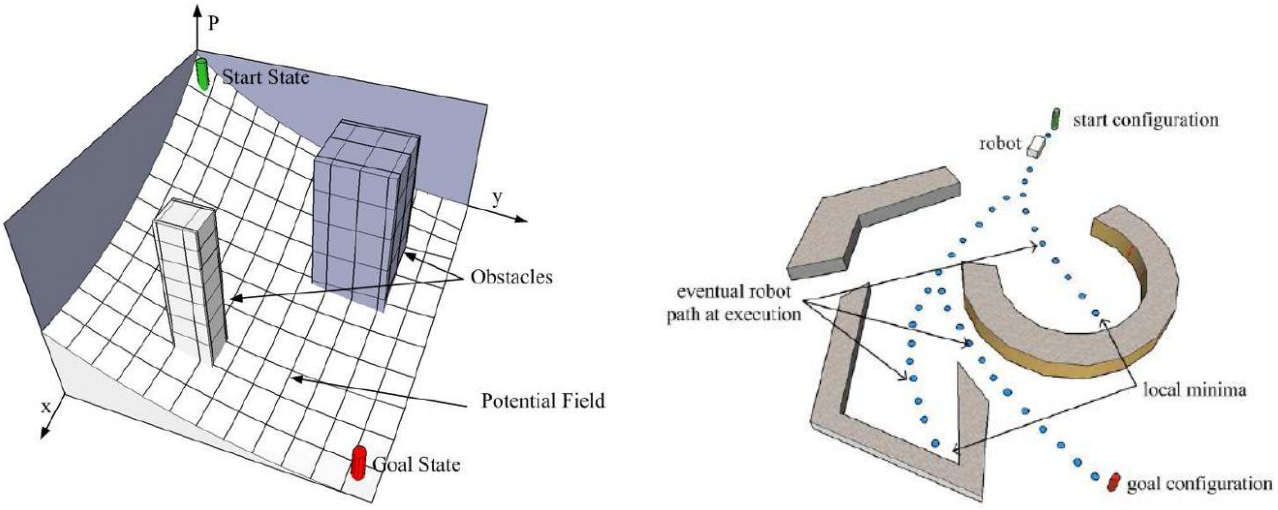
Trajectory rollout is similar to DWA but samples the robot's control space differently. Trajectory Rollout samples from the set of achievable velocities over the entire forward simulation period given the acceleration limits of the robot, while DWA samples from the set of achievable velocities for just one simulation step given the acceleration limits of the robot. This means that DWA is a more efficient algorithm because it samples a smaller space, but it may be outperformed by Trajectory Rollout for robots with low acceleration limits.

17.4 Potential fields

This models the robot as a charged particle in a potential field. Goals provide an attractive force; obstacles provide repulsive forces. The robot tries to move from a high potential at the starting point to the low potential at the goal (see Figure 17.3(a)).

Potential fields are simple to implement but:

- A robot can get stuck in a local minima due to U-shaped obstacles, see Figure 17.3(b).
- A robot struggles to pass through narrow gaps (the large contribution to the resultant force by the obstacles can make the robot reverse).
- A robot can oscillate in the presence of obstacles and in narrow passages.



(a) (b)

Figure 17.3: (from S. Petti, PhD thesis, 2007).

17.4.1 Vector field histogram (VFH)

This is one of the most popular algorithms. It achieves real-time obstacle avoidance by maintaining a 2-D histogram around the robot using range-bearing sensors such as a laser scanner. This provides a statistical representation of obstacles and is useful for inaccurate sensor data and fusion of multiple sensor readings.

There are three steps:

1. For each new sensor reading, only the 2-D histogram cell at the measured range and bearing is updated[4].

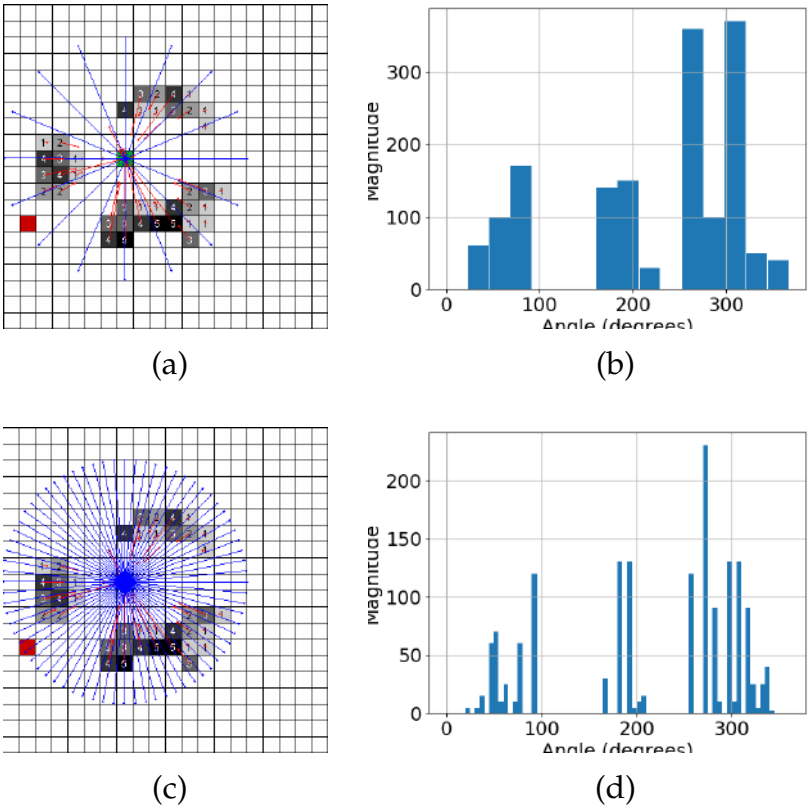
[4] Unlike with the creation of an occupancy grid where the data is projected to find free cells.

2. The 2-D histogram is transformed into a 1-D polar histogram, where the magnitude of each sector represents a measure of obstacle density in that direction.

3. The polar histogram is then smoothed and thresholded; the resulting steering angle is found by finding an appropriate 'valley' of the polar histogram with which the robot can be aligned.

One of the benefits of VFH is that it is relatively robust to bad sensor measurements because readings are averaged out in the data reduction processes. Another key advantage with VFH is its low computational requirement. However, the VFH method requires a manual tuning of the threshold value (which decides whether a sector in the polar histogram is a valley).

There are a number of extensions to the VFH algorithm, including VFH+ and VFH*. VFH+ incorporates the robot dynamics and uses a four-stage data reduction process to smooth the trajectory. VFH* uses the A* algorithm to avoid dead-end paths.



(a) (b) (c) (d)

Figure 17.4: Conversion of Cartesian histogram to polar histogram for VFH: (a), (b) 22.5° sampling; (c), (d) 5° sampling. The green square denotes the robot; the red square the goal.

17.5 Summary

1. Local navigation applies to navigation over short distances (in the range of the sensors).
2. The primary application of local navigation is for obstacle avoidance.
3. Local navigation runs at a faster rate, typ. 20 Hz, than global navigation, typ. 0.1 Hz.
4. The DWA algorithm is popular since it operates in the control-space of a robot and uses dynamics.

17.6 Exercises

1. What is the prime purpose of local navigation?
2. Why is the DWA algorithm popular?
3. Describe how the DWA algorithm operates?
4. Speculate on the speed difference of the DWA algorithm for holonomic and non-holonomic robots.
5. How does the DWA algorithm deal with robot dynamics?
6. Comment on the rates that local and global navigation need to be performed.

```
def DWA(robot, goal_pose, laserscan):
    """Implement DWA for differential drive robot trying to achieve
    goal_pose given laserscan data"""

    # Calculate desired speed; fast if far away, slow if close
    desired_v = robot.calculate_v(goal_pose)

    # Find allowable speeds in dynamic window around current speeds
    # based on achievable accelerations
    allowable_v = robot.allowable_v()
    allowable_w = robot.allowable_w()

    optimal_cost = None

    # Iterate over all possible speed pairs in dynamic window
    for v in allowable_v:
        for w in allowable_w:

            # Find closest object on trajectory
            obstacle_distance = robot.find_distance(v, w, laserscan)

            # Disallow speed pair if cannot stop in time
            braking_distance = robot.braking_distance(v)
            if obstacle_distance < braking_distance:
                continue

            heading_error = robot.heading_error(v, w, goal_pose)

            clearance = (obstacle_distance - braking_distance) /
                        (robot.dmax - braking_distance)

            cost = cost_function(heading_error, clearance,
                                 abs(desired_v - v))
            if optimal_cost is None or cost < optimal_cost:
                optimal_v = v
                optimal_w = w
                optimal_cost = cost

    return optimal_v, optimal_w
```

Listing 17.1: Python pseudo-code for DWA algorithm.

18
Path planning

Path planning is also known as global navigation. Due to its combinatorial complexity there are many algorithms with different heuristics.

Figure 18.1: Tee hee.

18.1 *Configuration space*

Before path planning it is common is to grow the obstacles in the map[1] of the environment, a process called *inflation*. This uses the radius of a circle that encompasses the robot[2] (see Figure 18.2). The new map representation is called the *configuration space* (C-space); it represents the legal positions of the robot in the environment. This simplifies path planning since it reduces the robot to a single point.

[1] Usually an occupancy grid.

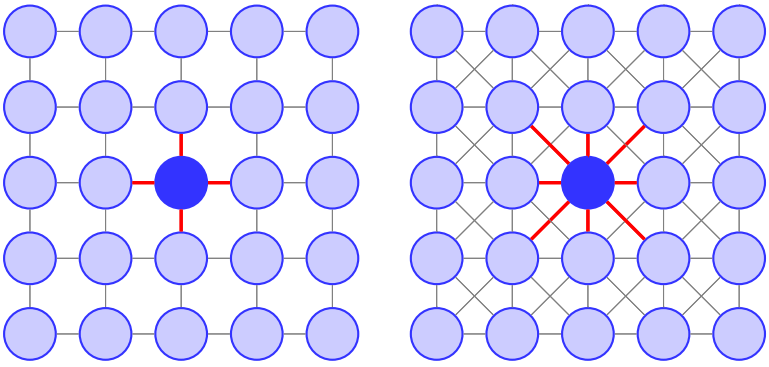
[2] Assuming it can turn in-place.

Figure 18.2: Red cells represent obstacles in the costmap, the blue cells represent obstacles inflated by the inscribed radius of the robot, and the red polygon represents the footprint of the robot. For the robot to avoid collision, the footprint of the robot should never intersect a red cell and the centre point of the robot should never cross a blue cell. From `http://wiki.ros.org/costmap_2d`.

18.2 Grids and graphs

Global navigation algorithms require a map. Usually this is stored as a grid and treated as a graph with eight-way connectivity.

18.2.1 Neighbourhoods



(a) (b)

Figure 18.3: A grid treated as a graph: (a) von Neumann neighbourhood, (b) Moore neighbourhood.

There are two ways to connect the cells in a grid:

von Neumann neighbourhood consists of the four neighbours on the faces of a square.

Moore neighbourhood consists of the eight[3] neighbours on the faces and corners of a square.

[3] The number of nearest neighbours for n-D space is $3^n - 1$ I think!

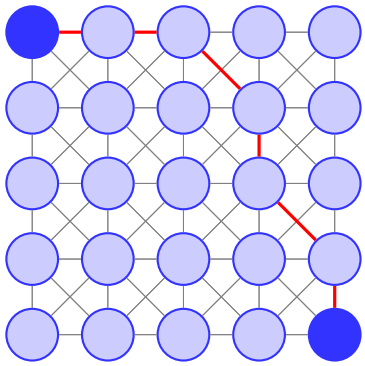


Figure 18.4: One of many possible paths (shown in red) between a pair of nodes using Moore neighbourhoods.

18.3 Global navigation algorithms

There are many path planning algorithms[4]. Most use graphs.

[4] There's even a book on them!

18.3.1 Dijkstra path planning

Dijkstra's algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms.

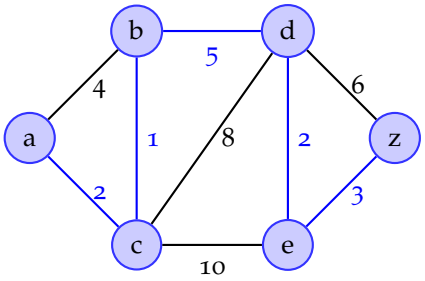


Figure 18.5: Dijkstra's shortest-path example.

18.3.2 A* path planning

A* is another algorithm for pathfinding and graph traversal. It is a generalization of Dijkstra's algorithm that reduces the size of the subgraph that must be explored, if additional information provides a lower bound on the "distance" to the target.

A* uses heuristics to only expand points it believes to be the most likely to provide the shortest route. Dijkstra's is more likely to produce an optimal path to the goal.

Like all informed search algorithms, A* first searches the routes that appear to be most likely to lead towards the goal. What sets A* apart from a greedy best-first search is that it also takes the distance already travelled into account.

18.3.3 Wavefront path planning

Wavefront path planning is a specialisation of Dijkstra's algorithm. It considers the configuration space to be like a heat conducting material. Obstacles have zero conductivity, while open space has an infinite conductivity.

The wavefront algorithm expanding wavefronts that emanate from the goal, and thus is a breadth-first search. The cost assigned to a node is essentially the distance from the goal. The search stops when a wavefront has hit the current position, or there are no more nodes to explore (i.e., in this case there is no path). When a wavefront hits the robot, the path to the goal is then found by exploring the node with a lower cost until the goal is reached.

More difficult terrains can be modelled by reducing their conductivity.

18.3.4 Sampling methods

To simplify the search, sampling methods are employed. The algorithms either assume that the measurements are certain (RRT, PRM) or uncertain (POMDPs).

Rapidly exploring random tree (RRT) an algorithm designed to efficiently search non-convex, high-dimensional spaces by randomly building a space-filling tree[5]. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem. They easily handle problems with obstacles and differential constraints (nonholonomic and kinodynamic).

[5] http://en.wikipedia.org/wiki/Rapidly_exploring_random_tree

Probabilistic roadmaps (PRM) have two phases: a construction and a query phase[6]. In the construction phase a roadmap (graph) is built, approximating the motions that can be made in the environment (using a local planner), using random sampling from the configuration space of the robot. In the query phase, the start and goal configurations are added and a path is found using Dijkstra's algorithm.

[6] http://en.wikipedia.org/wiki/Probabilistic_roadmap

Partially observable Markov decision processes (POMDPs) assume that the system dynamics are determined by a Markov decision process (MDP) but that the underlying state cannot be directly observed. This is a general framework for many path planning algorithms that operate with uncertainty.

Like Bayes filters, POMDPs are difficult to implement efficiently in practice due to the combinatorial nature of the problem.

18.4 Realtime path deformation

If something alters the planned path, say someone crosses the path, you need to recover the path again. Algorithms such as D* lite allow efficient re-computation. However, the elastic band method allows the existing path to be adjusted to handle deformations. It enables realtime modifications to a precomputed path that considers additional obstacles not considered during the global planning.

18.5 Topological graph creation

A topological map can be created from a grid-based one. This has a number of advantages:

- Faster path planning since the graph is smaller[7].
- The graph can be simplified using pruning.
- Alternative paths can be generated more quickly for dynamic environments.

[7] Large free areas, such as corridors, can be represented by a few nodes.

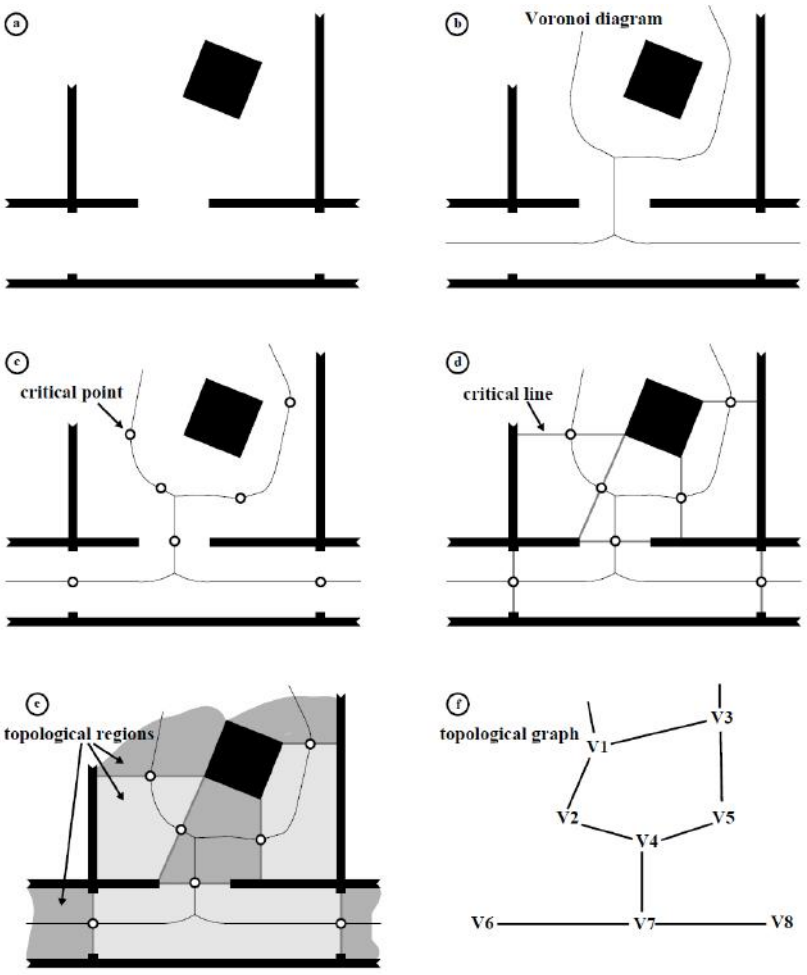


Figure 18.6: Constructing a topological map from an occupancy grid map (from S. Thrun. "Learning maps for indoor mobile robot navigation").

An algorithm to construct a topological map from a grid map is:

1. Threshold each cell in the occupancy grid map.
2. Construct a Voronoi diagram[8].
3. Locate the critical points: these are points that lie on the Voronoi diagram and are local minima with regards to clearances between obstacles (and that with the neighbouring points).

[8] In mathematics, a Voronoi diagram is a way of dividing space into a number of regions. A set of points (called seeds, sites, or generators) is specified beforehand and for each seed there will be a corresponding region consisting of all points closer to that seed than to any other. The regions are called Voronoi cells. It is dual to the Delaunay triangulation.

4. Construct critical lines: these are simply lines extended from each critical point to their nearest obstacle in the thresholded map.
5. Construct topological graph by assigning a node for each region separated by critical lines and obstacles, while the Voronoi diagram specifies how the nodes are connected.

18.6 ROS navigation

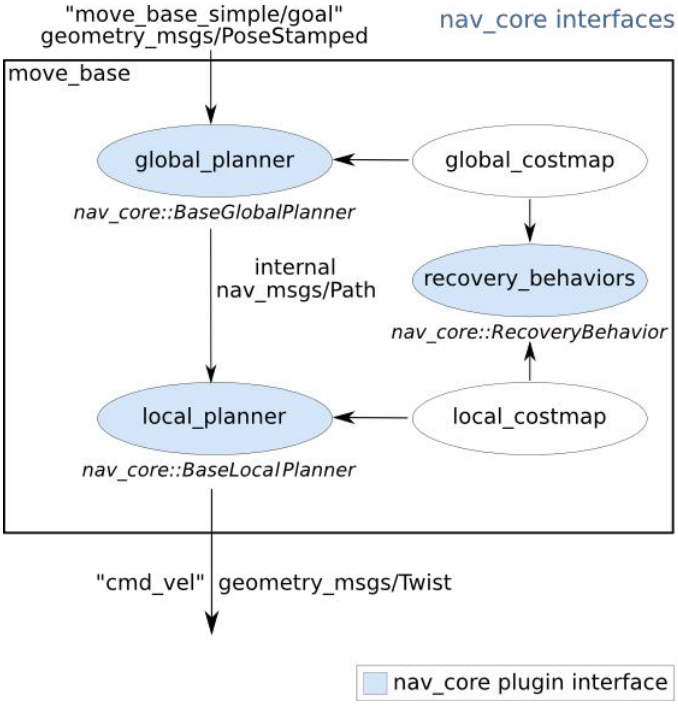


Figure 18.7: Interfaces for the ROS nav_core, from http://wiki.ros.org/nav_core.

ROS provides a navigation node (nav_core) with plugins for a global path planner and a local planner. In addition, it supports recovery behaviours when it gets confused by the sensor data conflicting with the map.

18.6.1 ROS global planners

There are two global planners used with ROS nav_core:

navfn uses Dijkstra's algorithm on a grid.

carrot_planner takes a goal point and attempts to move the robot as close to it as possible, even when that goal point is in an obstacle[9].

[9] It tries to place carrots in front of the robot for it to follow.

18.6.2 ROS local planners

ROS has two local planners for navigation on a plane:

DWA local planner implements the Dynamic Window Approach[10] (DWA). This supports holonomic or pseudo-holonomic robots since the control-space is (v_x, v_y, ω).

Base local planner implements Trajectory Rollout and the DWA[11] for a control-space (v, ω).

[10] http://wiki.ros.org/dwa_local_planner

[11] http://wiki.ros.org/base_local_planner

18.6.3 Recovery behaviours

Recovery behaviours used with ROS nav_core:

clear_costmap_recovery reverts the costmaps used by move_base to the static map outside of a user-specified range.

rotate_recovery performs a 360 degree rotation of the robot to attempt to clear out space.

18.6.4 Cost maps

The costmap_2d package[12] provides a configurable structure that maintains information about where the robot should navigate in the form of an occupancy grid.

The costmap package automatically subscribes to ROS sensor topics and creates an occupancy grid. It inflates the occupancy grid to create a costmap. There are two costmaps:

[12] http://wiki.ros.org/costmap_2d

local costmap This only uses the sensor data and is used by local navigation.

global costmap This uses the sensor data and the map and is used by by the path planner.

18.6.5 Complete system

1. Localise robot, say using AMCL, given map.
2. Use global planner, say Dijkstra's algorithm, to produce a path.
3. Use local planner, say DWA, to generate velocity commands to follow path and avoid obstacles.

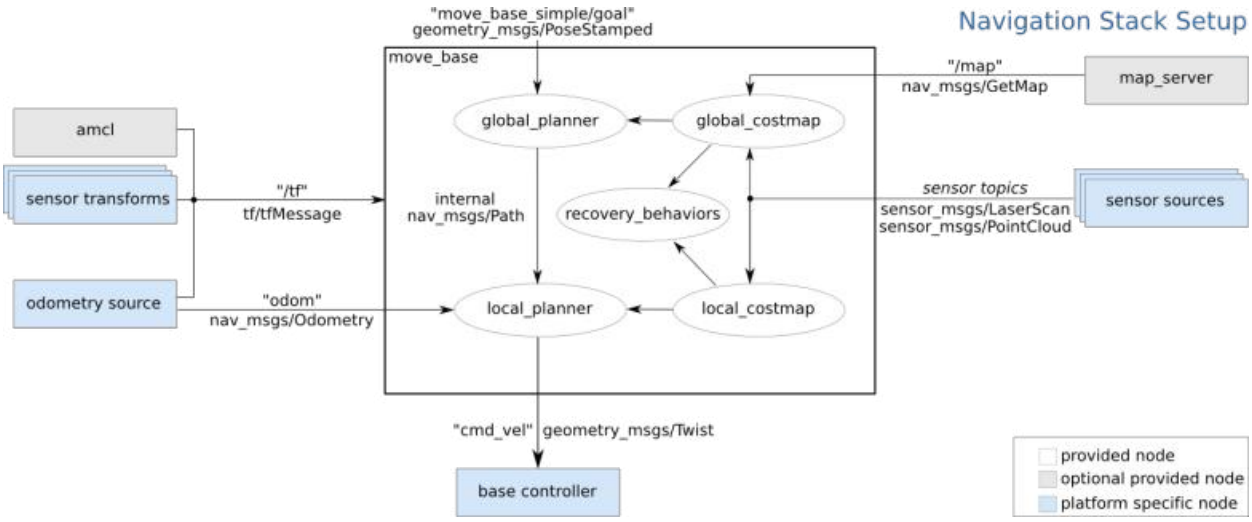


Figure 18.8: ROS move_base_node.

18.7 Exercises

1. How is an occupancy grid treated as a graph?
2. What is the advantage of a topological map for path planning?
3. How can topological maps be created from grid maps?
4. Is the A* algorithm always faster than the Dijkstra algorithm?
5. Why is the A* algorithm faster than the Dijkstra algorithm?
6. Does the A* algorithm always find an optimal solution?
7. Why is it not worthwhile searching for an optimal solution?
8. What is configuration space and what is it used for?
9. What is a recovery behaviour?
10. ROS navigation uses a local costmap and a global costmap. What are the differences?
11. How does a costmap differ from an occupancy grid?

19
ROS introduction

ROS is an open-source, meta-operating system[1] for a robot[2]. It operates on a heterogeneous network of computers using TCP/IP over ethernet, WiFi, etc. It provides hardware abstraction, low-level device control, message-passing, etc. It also provides tools and libraries for running and developing code across multiple computers, including package management.

[1] It currently runs on top of a Unix operating system.
[2] ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.

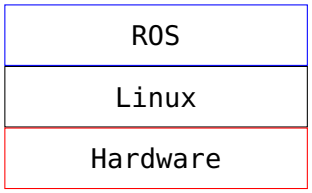


Figure 19.1: ROS is meta operating system, usually in conjunction with Linux.

19.1 ROS computation graph

The ROS Computation Graph[3] is a peer-to-peer network of processes (*nodes*) that communicate using *messages*.

[3] A picture of the graph is generated by the program `rxgraph`. This displays the nodes currently running and the topics that connect them.

19.1.1 Nodes

A node performs an aspect of computation, for example,

- laser range-finder control
- motor control
- localisation
- path planning

There are two special nodes:

Master provides name registration and lookup[4].

Parameter Server allows data to be stored by key in a central repository[5].

[4] Similar to a Domain Name Server (DNS). It tracks publishers and subscribers of topics as well as services.
[5] Currently the Parameter Server is part of the Master.

19.1.2 *Messages*

Nodes communicate using message passing. Messages are data structures with typed fields. These can be primitive types include integer, floating point, Boolean, etc., arrays of primitive types, and arbitrarily nested structures and arrays (like C structs). There are two ways of passing messages:

Topics provide an asynchronous publish/subscribe model. A node transmits a message by publishing it with a topic identifying the message content (for example, laser-scan data). Other nodes subscribed to the topic will receive the message.

Services provide a synchronous request/reply model similar to a remote procedure call. A node sends a request message and then waits for a reply message (for example, requesting a map).

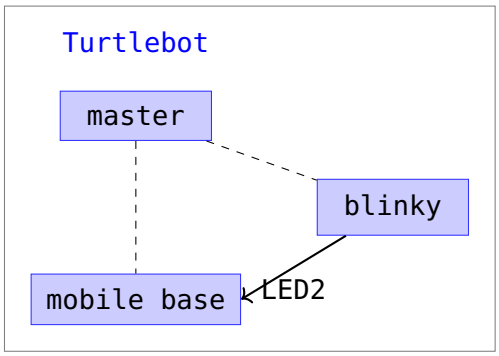


Figure 19.2: Node communication example using the LED2 topic. The blinky node publishes messages on the LED2 topic subscribed to by the base node.

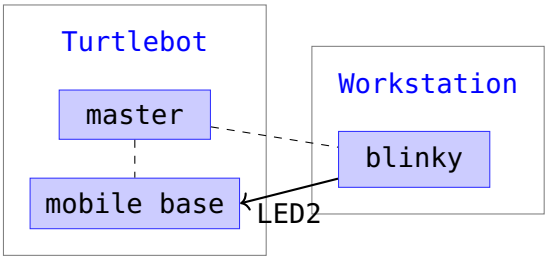


Figure 19.3: Node communication example using the LED2 topic with the nodes running on different computers.

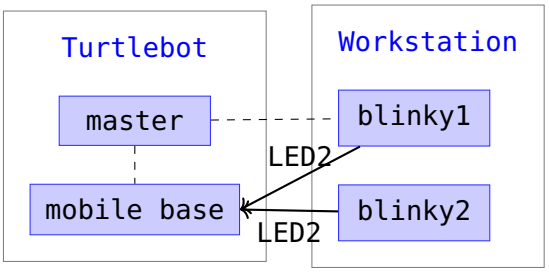


Figure 19.4: Node communication example using the LED2 topic with two publishers.

19.1.3 Message bags

Messages can be saved in *bag*[6] files using the rosbag node. Message bags are useful for developing and testing algorithms since the data can be recorded once and replayed[7].

[6] For example, to record a bag called `wander.bag` with messages from the topics `imu` and `scan`, use `rosbag record -O wander.bag /imu /scan`.
[7] `rosbag play wander.bag`.

19.2 ROS commands

There are many ROS commands, the common ones being:

rospack displays information about packages.

rosnode displays information about nodes.

rostopic displays information about topics.

rossrv displays information about services.

rosmsg displays information about message types.

rosrun runs an executable in a package.

roslaunch launches multiple nodes.

rosbag records messages to a bag or replays messages from a bag.

The nodes running on a system can be determined with the `rosnode` command, for example:

```
$ rosnode list
/app_manager
/bumper2pointcloud
/cmd_vel_mux
/diagnostic_aggregator
/gateway
/mobile_base
/mobile_base_nodelet_manager
/robot_state_publisher
/rosout
/zeroconf/zeroconf
```

All the message topics can be determined with the `rostopic` command, for example:

```
$ rostopic list
/cmd_vel_mux/active
/cmd_vel_mux/input/navi
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/teleop
```

```
/cmd_vel_mux/parameter_descriptions
/cmd_vel_mux/parameter_updates
/diagnostics
/diagnostics_agg
/diagnostics_toplevel_state
/gateway/force_update
/gateway/gateway_info
/joint_states
/mobile_base/commands/controller_info
/mobile_base/commands/digital_output
/mobile_base/commands/external_power
/mobile_base/commands/led1
/mobile_base/commands/led2
/mobile_base/commands/motor_power
/mobile_base/commands/reset_odometry
/mobile_base/commands/sound
/mobile_base/commands/velocity
/mobile_base/controller_info
/mobile_base/debug/raw_control_command
/mobile_base/debug/raw_data_command
/mobile_base/debug/raw_data_stream
/mobile_base/events/bumper
/mobile_base/events/button
/mobile_base/events/cliff
/mobile_base/events/digital_input
/mobile_base/events/power_system
/mobile_base/events/robot_state
/mobile_base/events/wheel_drop
/mobile_base/sensors/bumper_pointcloud
/mobile_base/sensors/core
/mobile_base/sensors/dock_ir
/mobile_base/sensors/imu_data
/mobile_base/sensors/imu_data_raw
/mobile_base/version_info
/mobile_base_nodelet_manager/bond
/odom
/rosout
/rosout_agg
/tf
/turtlebot/app_list
/zeroconf/lost_connections
/zeroconf/new_connections
```

The format of a message can be printed using `rosmsg`, for example, a Pose message is a structure comprised of a position and an orientation[8]

```
$ rosmsg show Pose
```

[8] Quaternions are used to avoid gimbal lock that can occur when representing orientations with Euler angles. They simply specify a single rotation around an arbitrary vector.

```
[geometry_msgs/Pose]:
geometry_msgs/Point position
  float64 x
  float64 y
  float64 z
geometry_msgs/Quaternion orientation
  float64 x
  float64 y
  float64 z
  float64 w
```

19.3 ROS packages

ROS comprises a large number of packages[9]; each package might contain ROS nodes, a dataset, configuration files, etc. Example packages[10] include:

amcl a localisation algorithm given a map.

gmapping a SLAM algorithm for producing a map.

map-server a server for map data.

tf an infrastructure for coordinate transforms.

base_local_planner a local navigation algorithm.

global_planner a global path planning algorithm.

[9] Early versions of ROS (using rosbuild) used stacks that are similar to metapackages

[10] Use `rospack list-names` to find installed ROS packages.

19.3.1 The tf package

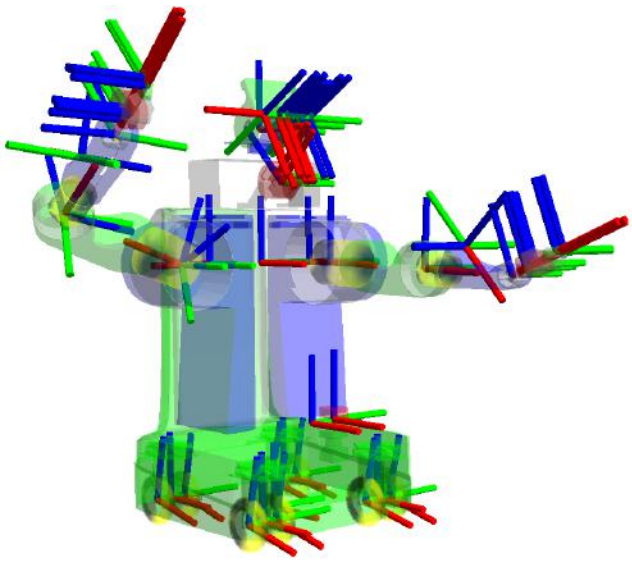


Figure 19.5: Visualising the frames of a robot. Red (x), green (y), and blue (z).

A robot typically has many 3D coordinate frames such as a world frame, base frame, IMU frame, gripper

frame, etc. tf keeps track of all the frame relationships[11] over time, and can answer questions like:

[11] Using a transform tree.

- Where was the base frame relative to the world frame, 5 seconds ago?
- What is the pose of the object in the gripper relative to the base?
- What is the current pose of the base frame in the map frame?

19.4 Launchers

A launcher is a file[12] that is used for starting multiple ROS nodes and setting parameters on the Parameter Server. It is used by the `roslaunch` program.

[12] An XML file with a `.launch` suffix.

Names of services and topics can be remapped. For example, say we wanted to use a laser scanner instead of the Kinect. Rather than changing our applications, we can remap the message names.

```
<launch>
  <!-- turtlebot_teleop_key already has its own built in velocity smoother
      -->
  <node pkg="turtlebot_teleop" type="turtlebot_teleop_key" name="
     turtlebot_teleop_keyboard"  output="screen">
    <param name="scale_linear" value="0.5" type="double"/>
    <param name="scale_angular" value="1.5" type="double"/>
    <remap from="turtlebot_teleop_keyboard/cmd_vel" to="cmd_vel_mux/input/
       teleop"/>
  </node>
</launch>
```

Notes:

1. The launcher defines two parameters with default values, `scale_linear` and `scale_angular`.
2. The launcher remaps the topic name `cmd_vel` to `cmd_vel_mux`.

19.5 Message passing implementation

Messages are passed peer to peer and not via the master node[13]. When a node publishes or subscribes to a topic, it communicates with the master. The master thus keeps track of all the topics and whoever is subscribing or publishing.

When a topic has both a publisher and a subscriber, the master informs publishers and subscribers of each others existence. Each publisher can then send messages to the subscribers. Note, no messages are sent on a topic unless there is at least one publisher and one subscriber.

[13] http://wiki.ros.org/Master

19.6 Exercises

1. How does a node determine which node publishes a topic?
2. Is the master node required for message transfers?
3. How are parameters specified?
4. What does a heterogenous operating system mean?
5. What does a meta operating system mean?
6. How does a topic differ from a service?
7. What is a ROS node?
8. What is a launcher?
9. What is the point of message name remapping?
10. Give examples of four different ROS commands and what they do.
11. What is a bag file and state an advantage of using one.
12. What does the tf package do?
13. How many floating point numbers are required for a Pose message?
14. Why are quaternions used for Pose messages?
15. In broad terms how does rosbag work?
16. What is the purpose of the ROS master node?
17. Explain how ROS nodes can communicate using topics.

18. Give an example of what a ROS service may be used for.
19. Give an example of what a ROS topic may be used for.

20 Probability supplement

Probability theory concerns models of how *data will likely behave*, for example, what is the expected robot position? Statistics concerns analysis of how *data did behave*. For example, what was the average robot position?

20.1 Random variables

Random variables can be discrete or continuous. In the following continuous random variables are assumed[1]. With discrete random variables the probability density function (PDF) is replaced with a probability mass function (PMF).

[1] For discrete random variables it is necessary to replace the integrals with discrete summations.

20.1.1 Notation

A random variable is usually denoted by a capital letter[2], for example, X. A realization, observation, or observed value, of X is denoted by the corresponding lowercase letter, for example, x.

[2] Most books on robotics blur the distinction between a random variable and its value. You have to guess!

20.1.2 Random process

A random process can result in different outcomes, even though it is repeated in the same manner every time. The result of a discrete-time random process is a sequence of random variables,

$$\mathbf{X} = \{X_0, X_1, X_2, \cdots, X_{N-1}\}. \tag{20.1}$$

Here X_n is a random variable at time-step n.

If we observe a random process we get a sequence of observations (the data)[3]

$$\mathbf{x} = \{x_0, x_1, x_2, \cdots, x_{N-1}\}. \tag{20.2}$$

Here each x_n is an observed value for the corresponding random variable, X_n.

[3] The discrete-time sequence $\mathbf{x}$ is often denoted $x[n]$.

20.1.3 Probability

The probability that a continuous random variable X is less than some value x is

$$\mathrm{P}(X \leq x) = F_X(x), \tag{20.3}$$

where $F_X(x)$ is the cumulative probability distribution of X. In general, the probability that X lies in the semi-closed interval $(a, b]$ is

$$\mathrm{P}(a < X \leq b) = \int_a^b f_X(x)\mathrm{d}x, \tag{20.4}$$
$$= F_X(b) - F_X(a), \tag{20.5}$$

where $f_X(x)$ is the probability density function[4] of X, related to the cumulative distribution by

$$F_X(x) = \int_{-\infty}^{x} f_X(t)\,\mathrm{d}t. \tag{20.6}$$

[4] The notation $p_X(x)$ is also common and so is $p(x)$, although the latter is ambiguous. The notation is further blurred when using a lowercase symbol for a random variable.

20.1.4 Expectation

The mean of a continuous random variable is given by

$$\mu_X = \mathrm{E}[X] = \int_{-\infty}^{\infty} x f_X(x)\mathrm{d}x. \tag{20.7}$$

The expected value of a function $g(X)$ is given by

$$\mathrm{E}[g(X)] = \int_{-\infty}^{\infty} g(x) f_X(x)\,\mathrm{d}x. \tag{20.8}$$

20.1.5 Variance

The variance of a continuous random variable is given by

$$\sigma_X^2 = \mathrm{Var}[X] = \int_{-\infty}^{\infty} (x - \mu_X)^2 f_X(x)\,\mathrm{d}x, \tag{20.9}$$
$$= \int_{-\infty}^{\infty} x^2 f_X(x)\,\mathrm{d}x - \mu_X^2. \tag{20.10}$$

20.1.6 Scaling of a random variable

Let a random variable Y be formed from the scaling of a random variables X,

$$Y = aX. \tag{20.11}$$

The resultant mean is scaled by the same factor a,

$$\mu_Y = a\mu_X, \tag{20.12}$$

and the resultant variance is scaled by a^2,

$$\sigma_Y^2 = a^2\sigma_X^2. \tag{20.13}$$

The PDF of Y is related to the PDF of X by

$$f_Y(y) = \frac{1}{|a|} f_X\left(\frac{y}{a}\right). \tag{20.14}$$

20.1.7 *Linear mapping of a random variable*

Consider the linear mapping between a random variable X to another random variable Y, where

$$Y = aX + b. \tag{20.15}$$

Here a is a scale factor and b is an offset. The mean of Y is related to the mean of X by

$$\begin{aligned} \mu_Y &= \mathrm{E}[Y], && (20.16)\\ &= \mathrm{E}[aX] + b, && (20.17)\\ &= a\,\mathrm{E}[X] + b, && (20.18)\\ &= a\mu_X + b. && (20.19) \end{aligned}$$

The variance of Y is related[5] to the variance of X by

$$\begin{aligned} \sigma_Y^2 &= \mathrm{Var}[Y], && (20.20)\\ &= \mathrm{E}\left[(Y-\mu_Y)^2\right], && (20.21)\\ &= \mathrm{E}\left[(aX - b - (a\mu_X + b))^2\right], && (20.22)\\ &= a^2\,\mathrm{E}\left[(X-\mu_X)^2\right], && (20.23)\\ &= a^2\sigma_X^2. && (20.24) \end{aligned}$$

[5] This relation is called the propagation of uncertainty.

If $f_X(x)$ is Gaussian[6] then it can be characterised by the mean and variance and so with a linear mapping, $f_Y(y)$ will also be Gaussian. However, in general, there is no simple mapping of the PDF, $f_Y(y)$, given the PDF, $f_X(x)$.

[6] With other distributions, the mapping of higher order statistics are required.

20.1.8 *Non-linear mapping of a random variable*

Consider a non-linear mapping of a Gaussian random variable X,

$$Y = g(X). \tag{20.25}$$

The cumulative distribution functions are related by

$$F_Y(y) = F_X\left(g^{-1}(y)\right), \tag{20.26}$$

and thus the probability density function for Y is

$$f_Y(y) = \frac{\mathrm{d}F_Y(y)}{\mathrm{d}y} = \frac{\mathrm{d}F_X\left(g^{-1}(y)\right)}{\mathrm{d}y}. \tag{20.27}$$

An approximate approach to characterise the distribution of Y is to linearise the mapping, $g(x)$, of a value x of X around the mean, μ_X, of X,

$$Y \approx \frac{\mathrm{d}}{\mathrm{d}X} g(X)\bigg|_{X=\mu_X} (X - \mu_X) + g(\mu_X). \tag{20.28}$$

20.1.9 *Linear mapping of random vector*

Consider the linear mapping between a random vector[7] $\mathbf{X}$ to another random vector $\mathbf{Y}$

$$\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{b}, \tag{20.29}$$

where $\mathbf{A}$ is a transfer matrix and $\mathbf{b}$ is an offset vector. The mean of $\mathbf{Y}$ is related to the mean of $\mathbf{X}$ by

$$\mu_{\mathbf{Y}} = \mathbf{A}\mu_{\mathbf{X}} + \mathbf{b}. \tag{20.30}$$

The covariance[8] of $\mathbf{Y}$ is related to the covariance of $\mathbf{X}$ by

$$\mathbf{\Sigma}_{\mathbf{Y}} = \mathbf{A}\,\mathbf{\Sigma}_{\mathbf{X}}\,\mathbf{A}^{\mathrm{T}}. \tag{20.31}$$

Even if the errors of $\mathbf{X}$ are uncorrelated so that $\mathbf{\Sigma}_{\mathbf{X}}$ is diagonal, then in general the covariance $\mathbf{\Sigma}_{\mathbf{Y}}$ is full.

[7] To avoid confusion between vectors and matrices, lower case is often used to denote a random variable vector.

[8] A covariance matrix has variances on the diagonal and correlations off the diagonal.

20.1.10 *Non-linear mapping of random vector*

When the system is non-linear and multidimensional,

$$\mathbf{Y} = g(\mathbf{X}), \tag{20.32}$$

then a linear approximation can be employed around the mean, $\boldsymbol{\mu}_{\mathbf{X}}$,

$$\mathbf{Y} \approx \mathbf{J}\,(\mathbf{X} - \boldsymbol{\mu}_{\mathbf{X}}) + g(\boldsymbol{\mu}_{\mathbf{X}}), \tag{20.33}$$

where $\mathbf{J}$ is a Jacobian matrix (evaluated at $\mathbf{X} = \boldsymbol{\mu}_{\mathbf{X}}$),

$$\mathbf{J} = \begin{bmatrix} \frac{\partial Y_1}{\partial X_1} & \cdots & \frac{\partial Y_1}{\partial X_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial Y_m}{\partial X_1} & \cdots & \frac{\partial Y_m}{\partial X_n} \end{bmatrix}. \tag{20.34}$$

With this approximation, the covariance of $\mathbf{Y}$ is related to the covariance of $\mathbf{X}$ by

$$\mathbf{\Sigma}_{\mathbf{Y}} = \mathbf{J}\,\mathbf{\Sigma}_{\mathbf{X}}\,\mathbf{J}^{\mathrm{T}}. \tag{20.35}$$

20.2 Joint probability density

The joint probability density function of two random variables X and Y is $f_{X,Y}(x,y)$. If the variables are independent, then

$$f_{X,Y}(x,y) = f_X(x)f_Y(y). \tag{20.36}$$

If the variables are completely dependent (i.e., the same)

$$f_{X,Y}(x,y) = f_X(x)\delta(y-x). \tag{20.37}$$

20.2.1 Marginal probability density

The marginal probability densities for X and Y are obtained from the joint probability density using integration over the other variable(s):

$$f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x,y)\mathrm{d}y, \tag{20.38}$$

$$f_Y(y) = \int_{-\infty}^{\infty} f_{X,Y}(x,y)\mathrm{d}x. \tag{20.39}$$

20.2.2 Conditional probability density

The conditional probability density of a random variable Y given another random variable X is

$$f_{Y|X}(y|x) = \frac{f_{X,Y}(x,y)}{f_X(x)}. \tag{20.40}$$

An alternative notation[9] is

$$f_Y(y|X=x) = \frac{f_{X,Y}(x,y)}{f_X(x)}. \tag{20.41}$$

[9] See en.wikipedia.org/wiki/Conditional_probability_distribution.

When X and Y are independent, then

$$f_{Y|X}(y|x) = f_Y(y). \tag{20.42}$$

Thus knowledge of a value of X has no effect on the probability density for Y.

The marginal probability densities and joint probability density are related to the conditional probability densities using

$$f_{X,Y}(x,y) = f_{X|Y}(x|y)f_Y(y) = f_{Y|X}(y|x)f_X(x). \tag{20.43}$$

Rearranging (20.43) gives Bayes' theorem

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)f_X(x)}{f_Y(y)}. \tag{20.44}$$

Often this is written as

$$f_{X|Y}(x|y) = \eta f_{Y|X}(y|x) f_X(x), \tag{20.45}$$

where η is a normalising factor calculated using the law of total probability

$$\eta = \frac{1}{f_X(x)} = \frac{1}{\int_{-\infty}^{\infty} f_{Y|X}(y|x) f_X(x) \mathrm{d}x}. \tag{20.46}$$

When X and Y are independent, then

$$f_{X,Y}(x,y) = f_X(x) f_Y(y) \tag{20.47}$$

and thus

$$f_{X|Y}(x|y) = \frac{f_X(x) f_Y(y)}{f_Y(y)}, \tag{20.48}$$

$$= f_X(x). \tag{20.49}$$

If X and Y are completely dependent, i.e., $Y = X$, then

$$f_{X,Y} = f_X(x)\delta(y - x) \tag{20.50}$$

and thus

$$f_{X|Y}(x|y) = \frac{f_X(x)\delta(y-x)}{f_Y(y)}, \tag{20.51}$$

$$= \delta(y - x). \tag{20.52}$$

20.3 Transformation of a joint distribution

Consider the mapping of two random variables to two new random variables

$$U = g_1(X, Y), \tag{20.53}$$

$$V = g_2(X, Y). \tag{20.54}$$

The joint PDF of the new random variables is given by

$$f_{U,V}(u,v) = f_{X,Y}(x,y) \left| J_g(x,y) \right|^{-1}, \tag{20.55}$$

where J_g is the Jacobian of the functions $g_1(x,y)$ and $g_2(x,y)$,

$$J_g = \det \begin{pmatrix} \frac{\partial g_1}{\partial x} & \frac{\partial g_1}{\partial y} \\ \frac{\partial g_2}{\partial x} & \frac{\partial g_2}{\partial y} \end{pmatrix} = \frac{\partial g_1}{\partial x}\frac{\partial g_2}{\partial y} - \frac{\partial g_1}{\partial y}\frac{\partial g_2}{\partial x} \tag{20.56}$$

X and Y are found from an inverse mapping,

$$X = g_1^{-1}(U, V), \tag{20.57}$$
$$Y = g_2^{-1}(U, V), \tag{20.58}$$

and thus (20.55) becomes

$$f_{U,V}(u,v) = f_{X,Y}(g_1^{-1}(u,v), g_2^{-1}(u,v)) \left| J_g(g_1^{-1}(u,v), g_2^{-1}(u,v)) \right|^{-1}, \tag{20.59}$$

A special case is where we combine two random variables into a single random variable,

$$U = g(X, Y). \tag{20.60}$$

Here we need to generate a dummy random variable V, say $V = Y$, so $g_1(X,Y) = g(X,Y)$ and $g_2(X,Y) = Y$. In this case, $\frac{\partial}{\partial x} g_2 = 0$, $\frac{\partial}{\partial y} g_2 = 1$, and the resulting Jacobian is $\frac{\partial}{\partial x} g$. The new joint PDF is

$$f_{U,V}(u,v) = f_{X,Y}(x,y) \left| \frac{\partial g}{\partial x} \right|^{-1}. \tag{20.61}$$

The desired marginal PDF can be found by integrating over the dummy variable $v = y$ to give

$$f_U(u) = \int_{-\infty}^{\infty} f_{X,Y}(x,y) \left| \frac{\partial g}{\partial x} \right|^{-1} \mathrm{d}y. \tag{20.62}$$

x is found from $g^{-1}(u,y)$ and so

$$f_U(u) = \int_{-\infty}^{\infty} f_{X,Y}(g^{-1}(u,y), y) \left| \frac{\partial g}{\partial x} \right|^{-1} \mathrm{d}y. \tag{20.63}$$

An alternative choice of dummy variable $V = X$ leads to

$$f_U(u) = \int_{-\infty}^{\infty} f_{X,Y}(x, g^{-1}(x,u)) \left| \frac{\partial g}{\partial y} \right|^{-1} \mathrm{d}x. \tag{20.64}$$

Note, the g^{-1} function is different to the one in (20.63) since it provides a value for y and not x.

20.3.1 *Derivation of PDF for the sum of two random variables*

Let's consider the case where we add two random variables together,

$$U = X + Y. \tag{20.65}$$

Let's choose a dummy variable $V = Y$ and so

$$u = g_1(x, y) = x + y, \tag{20.66}$$
$$v = g_2(x, y) = y, \tag{20.67}$$

and thus the Jacobian is

$$J_g = 1 \times 1 - 1 \times 0 = 1. \tag{20.68}$$

Hence from (20.63),

$$f_U(u) = \int_{-\infty}^{\infty} f_{X,Y}(u - y, y)\mathrm{d}y. \tag{20.69}$$

If X and Y are independent, then

$$f_{X,Y}(x, y) = f_X(x) f_Y(y), \tag{20.70}$$

and thus (20.69) turns into a convolution integral,

$$f_U(u) = \int_{-\infty}^{\infty} f_X(u - y) f_Y(y)\mathrm{d}y. \tag{20.71}$$

Note, convolution is a blurring process and the PDF for U will be wider than the PDFs of X and Y. This should be expected since the random variables are added together and thus the resulting variance is broader.

If X and Y are completely dependent, i.e., $Y = X$, then

$$f_{X,Y}(x, y) = f_X(x)\delta(y - x) \tag{20.72}$$

and thus

$$f_U(u) = \int_{-\infty}^{\infty} f_{X,Y}(u - y, y)\mathrm{d}y, \tag{20.73}$$
$$= \int_{-\infty}^{\infty} f_X(u - y)\delta(u - 2y)\mathrm{d}y, \tag{20.74}$$
$$= \frac{1}{2}\int_{-\infty}^{\infty} f_X\left(u - \frac{\lambda}{2}\right)\delta(u - \lambda)\mathrm{d}\lambda, \tag{20.75}$$
$$= \frac{1}{2} f_X\left(\frac{u}{2}\right). \tag{20.76}$$

This is equivalent to $U = 2X$. In general, if $U = aX$, then

$$f_U(u) = \frac{1}{|a|} f_X\left(\frac{u}{a}\right). \tag{20.77}$$

Equation (20.69) can be written using conditional probabilities as

$$f_U(u) = \int_{-\infty}^{\infty} f_{X|Y}(x|u - x) f_Y(u - x)\mathrm{d}x. \tag{20.78}$$

20.3.2 PDF of the product of two random variables

Let's consider the case where we multiply two random variables together,

$$U = XY. \tag{20.79}$$

Let's choose a dummy variable $V = Y$ and so

$$u = g_1(x, y) = xy, \tag{20.80}$$
$$v = g_2(x, y) = y, \tag{20.81}$$

and thus the Jacobian is

$$J_g = y \times 1 - x \times 0 = y. \tag{20.82}$$

Hence from (20.63),

$$f_U(u) = \int_{-\infty}^{\infty} f_{X,Y}\left(\frac{u}{y}, y\right) \frac{1}{|y|} \mathrm{d}y. \tag{20.83}$$

If X and Y are independent, then

$$f_U(u) = \int_{-\infty}^{\infty} f_X\left(\frac{u}{y}\right) f_Y(y) \frac{1}{|y|} \mathrm{d}y. \tag{20.84}$$

20.3.3 Transformation of a random variable

A transformation of a random variable, X, given by

$$Y = g(X), \tag{20.85}$$

has a PDF

$$f_Y(y) = f_X\left(g^{-1}(y)\right) \left| \frac{\mathrm{d}g^{-1}(y)}{\mathrm{d}y} \right|. \tag{20.86}$$

For example, with $Y = aX$ then $g(X) = aX$, $g^{-1}(Y) = Y/a$ and so

$$f_Y(y) = \frac{1}{|a|} f_X\left(\frac{y}{a}\right). \tag{20.87}$$

20.3.4 PDF for the sum of two random variables

Consider the sum of two random variables,

$$U = X + Y. \tag{20.88}$$

If X and Y are independent[10] then the PDF of U is

$$f_U(u) = \int_{-\infty}^{\infty} f_X(u - y) f_Y(y) \mathrm{d}y. \tag{20.89}$$

This is a convolution integral. Since, convolution is a blurring process the PDF for U will be wider than the PDFs of X and Y. This should be expected since the random variables are added together and thus the resulting variance is larger.

[10] See Section 20.3.1 for a derivation.

20.3.5 PDF for the sum of two Gaussian random variables

Consider two independent Gaussian random variables,

$$X \sim \mathcal{N}\left(\mu_X, \sigma_X^2\right), \tag{20.90}$$

$$Y \sim \mathcal{N}\left(\mu_Y, \sigma_Y^2\right). \tag{20.91}$$

The random variable formed from the sum of X and Y,

$$U = X + Y, \tag{20.92}$$

also has a Gaussian distribution,

$$U \sim \mathcal{N}\left(\mu_U, \sigma_U^2\right), \tag{20.93}$$

where

$$\mu_U = \mu_X + \mu_Y, \tag{20.94}$$

$$\sigma_U^2 = \sigma_X^2 + \sigma_Y^2. \tag{20.95}$$

Note, since the variances add, the variance of the result is larger as expected.

20.4 Gaussian PDF

The noise distribution for most sensors is Gaussian[11]. This distribution for a random variable X can be parameterised by two parameters, the mean, μ_X, and variance, σ_X^2, (see Figure 20.1)

[11] If it is not, averaging often produces a Gaussian distribution due to the central limit theorem.

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_X}{\sigma_X}\right)^2\right). \tag{20.96}$$

20.4.1 Multivariate Gaussian PDF

The N-dimensional Gaussian probability density for a real random vector $\mathbf{X} = (X_1, X_2, \cdots X_N)^{\mathrm{T}}$, is[12]

[12] Note, in the 1-D case, this collapses to $f_X(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left(-0.5(x-\mu_X)^2/\sigma_X^2\right)$.

$$f_{\mathbf{X}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})\right), \tag{20.97}$$

where $\boldsymbol{\mu} = \mathrm{E}[\mathbf{X}]$ is the mean of $\mathbf{X}$ and

$$\boldsymbol{\Sigma} = \mathrm{E}\left[(\mathbf{X}-\boldsymbol{\mu})(\mathbf{X}-\boldsymbol{\mu})^{\mathrm{T}}\right] \tag{20.98}$$

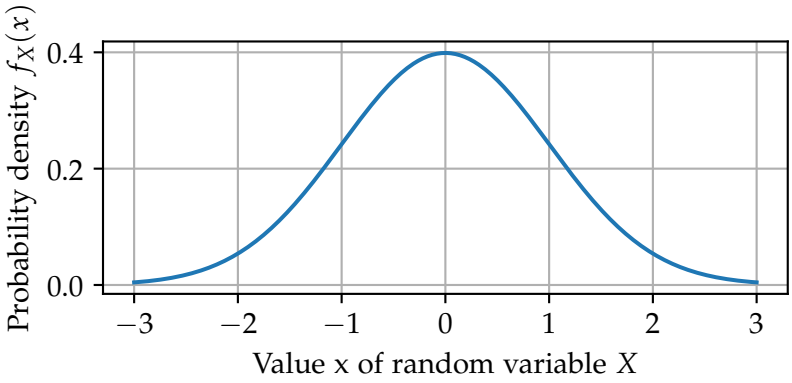


Figure 20.1: Gaussian probability density, zero mean and unit variance: $\mu_X = 0$, $\sigma_X^2 = 1$.

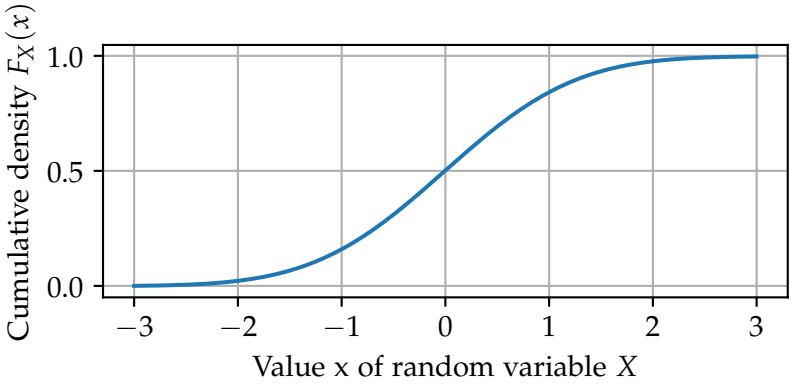


Figure 20.2: Gaussian cumulative probability distribution, zero mean and unit variance: $\mu_X = 0$, $\sigma_X^2 = 1$.

is the *covariance matrix* of $\mathbf{X}$. Note, the PDF is parameterised by a vector of mean values and a covariance matrix. No higher order statistics are required.

For the bivariate Gaussian distribution, $\mathbf{X} = (X, Y)^{\mathrm{T}}$, $\boldsymbol{\mu} = (\mu_X, \mu_Y)^{\mathrm{T}}$, and the covariance matrix is

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_X^2 & \rho\sigma_X\sigma_Y \\ \rho\sigma_X\sigma_Y & \sigma_Y^2 \end{bmatrix}, \tag{20.99}$$

where ρ is the correlation coefficient between X and Y. When $\rho = 0$ then X and Y are uncorrelated[13]. When $\rho = 1$ then X and Y are perfectly correlated[14].

[13] $f_{X,Y}(x, y) = f_X(x) f_Y(y)$.

[14] $f_{X,Y}(x, y) = f_X(x)\delta(y - x)$.

For the trivariate Gaussian distribution, $\mathbf{X} = (X_1, X_2, X_3)^{\mathrm{T}}$, $\boldsymbol{\mu} = (\mu_{X_1}, \mu_{X_2}, \mu_{X_3})^{\mathrm{T}}$, and the covariance matrix is

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho_{12}\sigma_{X_1}\sigma_{X_2} & \rho_{13}\sigma_{X_1}\sigma_{X_3} \\ \rho_{12}\sigma_{X_1}\sigma_{X_3} & \sigma_{X_2}^2 & \rho_{23}\sigma_{X_2}\sigma_{X_3} \\ \rho_{13}\sigma_{X_2}\sigma_{X_3} & \rho_{23}\sigma_{X_2}\sigma_{X_3} & \sigma_{X_3}^2 \end{bmatrix}. \tag{20.100}$$

The distribution can be described by 9 parameters; three means, three variances, and three correlations.

20.5 Quirks

Random variables obey many rules of algebra but there are some quirks. Consider the sum of two independent random variables,

$$Z = X + V. \tag{20.101}$$

The PDF of the result is given by a convolution of the PDFS,

$$f_Z(z) = \int_{-\infty}^{\infty} f_X(z - v) f_V(v) \mathrm{d}v. \tag{20.102}$$

Equation (20.101) looks innocuous but it cannot be arranged to give,

$$X = Z - V. \tag{20.103}$$

This would require a deconvolution of the PDF $f_Z(z)$ with $f_V(v)$ to find $f_X(x)$.

21
Estimation supplement

Measurements from sensors are corrupted by noise. However, if we have a model of the sensor and the noise we can apply estimation techniques to get a better answer.

Estimation techniques (estimators) include averaging and filtering. The answer they give is called an estimate. Ideally an estimator should not have a bias and its variance should be as small as possible.

The best possible estimator is called a minimum variance unbiased estimator (MVUE). This has a variance given by the Cramér-Rao lower bound (CRLB)[1].

Bayesian estimators use prior information that biases the result. Thus they can achieve a variance lower than the CRLB.

[1] No unbiased estimators can achieve a lower variance than the CRLB.

21.1 Observation model

Estimators require an observation model. A typical observation model with additive noise is

$$Z_n = f(\boldsymbol{\beta}) + V_n \quad \text{for } n = 0, 1, \cdots N-1, \tag{21.1}$$

where Z_n is the n-th random variable of a random process describing the measurements, V_n is the n-th random variable of a random process describing additive noise, and $f(\boldsymbol{\beta})$ is a function of some parameters, $\boldsymbol{\beta}$, that we wish to estimate[2]. For example, consider a DC signal corrupted by additive noise. The observation model is

$$Z_n = a + V_n, \tag{21.2}$$

where a is a constant but unknown parameter that we wish to estimate. This observation model is linear in terms of the parameter, a.

A second example is determining the parameters of an AC signal corrupted by additive noise,

$$Z_n = a\cos(2\pi f_0 n\Delta t + \phi) + V_n. \tag{21.3}$$

[2] The noise process also has parameters (mean, variance, etc) that are assumed to be known.

In this example, there are three unknown parameters[3]: the amplitude a, the frequency f_0, and the phase ϕ. The observation model is linear in the amplitude parameter a but is non-linear in the frequency and phase parameters.

A third example is estimating the gravitational acceleration and the initial speed of a ball thrown into the air from height measurements. The height measurements can be modelled as

$$Z_n = v_i t_n + a t_n^2 + V_n, \tag{21.4}$$

where $t_n = n\Delta t$. Here the measurements are non-linear in t_n but linear in the parameters v_i and a.

A fourth example is the estimation of parameters of an unknown noise process,

$$Z_n = V_n(\mu, \sigma). \tag{21.5}$$

Here μ is the unknown mean of the noise process and σ is the unknown variance.

[3] The sampling period, Δt, is assumed to be known.

21.2 Estimators

An estimator is a function of the observed random process and is itself a random variable. It is usually denoted with a hat, $\hat{X}$. A particular realization of this random variable is called an *estimate*, $\hat{x}$. This is the value determined from some specific measurements[4].

[4] A realisation of the observed random process.

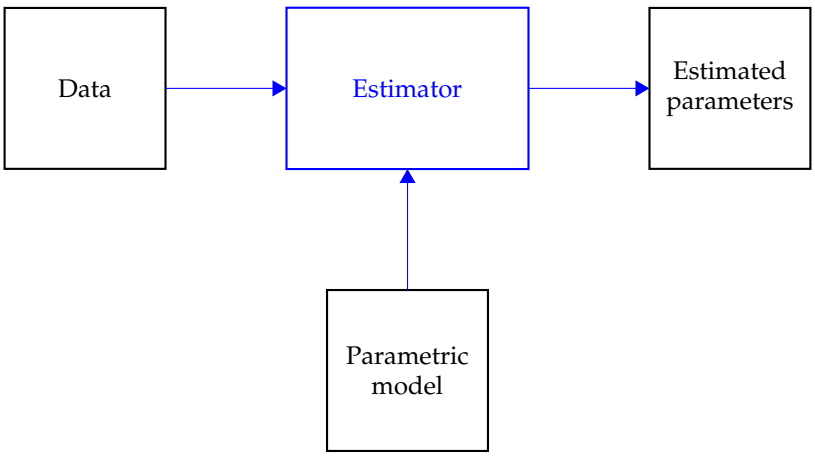


Figure 21.1: The estimation process.

21.2.1 Minimum variance unbiased estimator (MVUE)

A desirable estimator has no bias and the smallest possible variance. This is called a minimum variance unbi-

ased estimator (MVUE). The smallest possible achievable variance is given by the Cramér-Rao Lower Bound (CRLB) or (CRB)[5]. The CRLB depends on the signal to noise ratio (SNR); the higher the SNR, the lower the variance of the estimator.

The minimum variance unbiased estimator (MVUE) may not exist or may be hard to determine for certain estimation problems. In practice, a maximum likelihood estimator (MLE) is used since it asymptotically reaches the CRLB with a large number of measurements.

[5] Much effort in estimators has been expended by the military. A better estimator allows an invader to be more reliably detected at greater distances, say using sonar or radar.

21.2.2 *Best linear unbiased estimator (BLUE)*

The best linear unbiased estimator (BLUE) is a weighted average where the weights are chosen to avoid a bias and to minimise the variance of the result. It is a MVUE for linear problems.

21.2.3 *Estimator notation*

$\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ commonly denote a vector of parameters[6] of a statistical model. The estimator[7] of β is denoted $\hat{\mathbf{B}}$. This is short-hand for $\hat{\mathbf{B}}(\mathbf{Z})$ where $\mathbf{Z}$ is a sequence of random variables $(Z_{0:N-1})$ corresponding to the observed data. Thus $\hat{\mathbf{B}}(\mathbf{Z})$ is a random variable that is a function of the observed data. $\hat{\beta}(\mathbf{z})$ denotes the estimate for a particular sequence of observed data, $\mathbf{Z} = \mathbf{z}$.

[6] Mean and variance for a Gaussian, rate parameter for Poisson distribution, etc.

[7] An estimator is a statistic since it is a function of the data.

21.3 *Maximum likelihood estimator (MLE)*

When it is difficult to find a MVUE, the maximum likelihood estimator (MLE) is popular. Unfortunately, this has a bias[8], but the bias gets smaller for larger numbers of samples, N. In the limit as $N \longrightarrow \infty$, there is no bias and MLE is equivalent to MVUE.

MLE requires the joint PDF for the random process to be known. Using this with the observation model, a likelihood function is formed as a function of the measurements and unknown parameters. A search is then performed to find the parameters that maximises the likelihood function given the measurements. Mathematically the maximum likelihood estimate can be formulated as an optimisation:

[8] There are ML estimators that estimate and subtract the bias to first-order.

$$\hat{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta}} l_{Z_{0:N-1}}(\boldsymbol{\beta}; z_{0:N-1}), \tag{21.6}$$

where $l_{Z_{0:N-1}}(\boldsymbol{\beta}; z_{0:N-1})$ is the likelihood function for the set of N measurements, $z_{0:N-1} = z_0, z_1, \cdots z_{N-1}$, and parameter vector $\boldsymbol{\beta}$.

The likelihood function is greatly simplified if each random variable in the random process is independent and has the same distribution[9]. In this case, the joint PDF is the product of the PDFs of the random variables in the random process. For example, this applies for additive white Gaussian noise (AWGN).

When the noise process is Gaussian distributed, MLE is equivalent to generalised least squares (GLS). If the noise process is also IID, MLE is equivalent to ordinary least squares (OLS). Finally, if the observation model is linear in the parameters, a search over the parameter space is not required and a direct solution is possible. In this case, the observation model can be written in vector form[10] as

$$\mathbf{Z} = \mathbf{A}\boldsymbol{\beta} + \mathbf{N}. \tag{21.7}$$

The least squares estimator[11] is

$$\hat{\mathbf{B}} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}\mathbf{Z}, \tag{21.8}$$

and for a specific vector of measurements, $\mathbf{z}$, the least squares estimate is

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}\mathbf{z}. \tag{21.9}$$

Note, that the estimated parameters are given by a linear weighted sum of the measurements.

[9] Independent and identically distributed (IID).

[10] The columns of $\mathbf{A}$ are the basis vectors.

[11] The matrix $\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}$ is called the Moore-Penrose pseudoinverse of $\mathbf{A}$. For complex data, the matrix transpose is replaced with the Hermitian transpose, $\mathbf{A}^{\mathrm{H}}$.

21.4 *Bayesian estimation*

ML estimators are completely data driven. However, there are times when we know something about the possible parameters. This additional prior information can be utilised by Bayesian estimators.

Bayesian estimators treat the unknown parameters in an observation model as random variables. Thus each parameter has a PDF. The PDF before a measurement is called the *prior PDF*; this represents our prior knowledge of the possible solutions. For example, a robot may be physically constrained and so its position can only have a limited range of values. If the constrained robot positions are equally likely, the prior PDF has a uniform distribution.

After measurements have been made, a Bayesian estimator combines the likelihood function with the prior PDF to achieve the *posterior PDF*. Finally, the 'best' value of the parameters is chosen as the estimate. This is found either by looking for the parameters that maximise the posterior PDF, the maximum a posteriori (MAP) estimate, or using the mean value of the posterior PDF, the minimum mean square estimate (MMSE)[12].

[12] If the posterior PDF is Gaussian, the MAP and MMSE estimates are equivalent.

21.5 Mean estimation

Consider a sequence of N independent and identically distributed (IID) random variables from a random process,

$$\mathbf{Z} = \{Z_0, Z_1, Z_2, \cdots, Z_{N-1}\}. \tag{21.10}$$

Since Z_n are IID, they have the same mean μ and variance σ^2. Let's say we wish to estimate the mean. This can be achieved with a well-known estimator[13], the time average[14]

$$\hat{\mu}(\mathbf{Z}) = \frac{1}{N}\sum_{n=0}^{N-1} Z_n. \tag{21.11}$$

[13] This is a point estimator.

[14] Sample average.

The estimator for the mean produces a random variable $\hat{\mu}(\mathbf{Z})$. This is a random variable since if someone else took N samples, they would get a different value, $\hat{\mu}(\mathbf{z})$, where

$$\hat{\mu}(\mathbf{z}) = \frac{1}{N}\sum_{n=0}^{N-1} z_n. \tag{21.12}$$

Here $\mathbf{z}$ denotes the sequence of observed values for a realisation of $\mathbf{Z}$,

$$\mathbf{z} = \{z_0, z_1, z_2, \cdots, z_{N-1}\}. \tag{21.13}$$

Equation (21.11) is an unbiased estimator since

$$\mathrm{E}\left[\overline{\mathbf{Z}}\right] = \mu, \tag{21.14}$$

The variance of the estimator is

$$\mathrm{Var}\left[\overline{\mathbf{Z}}\right] = \frac{\sigma^2}{N}. \tag{21.15}$$

21.6 Maximum likelihood amplitude estimation

Consider an observation model:

$$Z[n] = As[n] + V[n], \tag{21.16}$$

where $s[n]$ is a known signal, $V[n]$ is an additive zero-mean white Gaussian random process, and A is a constant but unknown amplitude.

The likelihood function for N observations is

$$l\left(A; z_{0:N-1}\right) = \prod_{n=0}^{N-1} f_V(Z[n] - As[n]), \tag{21.17}$$

and the maximum likelihood estimate can be found from a search,

$$\hat{A} = \arg\max_A l\left(A; z_{0:N-1}\right). \tag{21.18}$$

Since the noise is independent zero-mean Gaussian,

$$l\left(A; z_{0:N-1}\right) = \prod_{n=0}^{N-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{z[n] - As[n]}{\sigma}\right)^2\right), \tag{21.19}$$

and using the log-likelihood, the product is converted to a summation

$$\log l\left(A; z_{0:N-1}\right) = -\log\sqrt{2\pi\sigma^2} - \sum_{n=0}^{N-1}\left(\frac{1}{2}\left(\frac{z[n] - As[n]}{\sigma}\right)^2\right). \tag{21.20}$$

Since the log function is monotonic, the maximum likelihood estimate is equivalent to

$$\hat{A} = \arg\min_A \sum_{n=0}^{N-1}\left(\frac{1}{2}\left(\frac{z[n] - As[n]}{\sigma}\right)^2\right). \tag{21.21}$$

Since the noise is identically distributed, the ML estimate is equivalent to the least-squares estimate,

$$\hat{A} = \arg\min_A \sum_{n=0}^{N-1} \left(z[n] - As[n]\right)^2. \tag{21.22}$$

Expanding the square and dropping terms independent of A,

$$\hat{A} = \arg\min_A \sum_{n=0}^{N-1} -2As[n]z[n] + A^2 s^2[n]. \tag{21.23}$$

Differentiating the objective function with respect to A and equating to zero, the estimate of A can be directly obtained as

$$\hat{A} = \frac{\sum_{n=0}^{N-1} z[n]s[n]}{\sum_{n=0}^{N-1} s^2[n]}. \tag{21.24}$$

The ML estimator is known to have a bias since it can be shown that

$$\mu_{\hat{A}} = \mathrm{E}\left[\hat{A}\right] \approx A + \frac{\sigma^2}{NA}. \tag{21.25}$$

21.7 Summary of estimation techniques

Bayes-filter A on-line probabilistic state estimator using Bayes theorem with probabilistic motion and sensor models and an initial PDF for the state. With the complete state (Markov) assumption they can be implemented recursively.

Best linear unbiased estimate (BLUE) The best linear combination that minimises the variance without a bias. This is equivalent to MVUE for linear problems with additive Gaussian noise.

Cramér Rao lower bound (CRLB) This is the limit on the minimum variance of an unbiased estimator.

Curve-fitting a.k.a. model fitting. This is a form of regression analysis. It does not assume a prior. There are two variants: (1) interpolation, where a exact fit is required to the data points; (2) smoothing, where a smooth function is approximately fitted to the data.

Extended Kalman filter (EKF) A Kalman filter the approximates non-linear models with linear models.

Estimate A specific value of an estimator.

Estimator A random variable or an algorithm. For example, maximum likelihood estimator. A specific value of an estimator is an estimate. An estimator is analogous to a function and an estimate is analogous to the function value at a point.

Filtering Using past and present data, cf., smoothing.

Generalised least squares (GLS) This handles heteroscedasity where the errors have different variances and are

correlated. The estimated parameter vector is found using:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} (\mathbf{z} - \mathbf{A}\boldsymbol{\beta})^{\mathrm{T}} \boldsymbol{\Omega}^{-1} (\mathbf{z} - \mathbf{A}\boldsymbol{\beta}), \tag{21.26}$$

$$= \left(\mathbf{A}^{\mathrm{T}}\boldsymbol{\Omega}^{-1}\mathbf{A}\right)^{-1} \mathbf{A}^{\mathrm{T}}\boldsymbol{\Omega}^{-1}\mathbf{z}. \tag{21.27}$$

Note, a direct solution is obtained without a search over the parameter space.

When there are no correlations (the off diagonal terms of the covariance matrix) are zero, this simplifies to weighted least squares. If all the error variances are the same, $\boldsymbol{\Omega} = \sigma^2\mathbf{I}$, and (21.27) simplifies to ordinary least squares.

Histogram filter A Bayes filter that represents the posterior distribution with a histogram. It is suited to non-linear problems of low-dimensionality.

Kalman filter On-line recursive Bayes-filter, a.k.a. linear quadratic estimation. The noise must be additive Gaussian noise and motion and sensor models must be linear. This is an efficient algorithm since it only needs to track the mean vector of the estimate state and its covariance. Asymptotically for stationary noise the Kalman filter becomes a Wiener filter.

Information filter Equivalent to a Kalman filter but using information matrices rather than covariance matrices to parameterise multi-dimensional Gaussian PDFs.

Iteratively reweighteed least squares (IRLS) This is a form of robust estimation that handles outliers.

Least squares There are many different variants, the most common are linear least squares and non-linear least squares.

Linear least squares There are many different variants, generalised least squares, weighted least squares, ordinary least squares, total least squares.

Linear quadratic estimation (LQE) a.k.a. Kalman filter.

Linear regression A linear model between the dependent variable (measurement) and the independent variables (explanatory variables). The model needs to be linear in the parameters but can be non-linear, for example,

$$y(t) = v_i t + 0.5at^2, \tag{21.28}$$

is non-linear in t but linear in the parameters v_i and a.

Maximum likelihood (ML) The estimate found from the mode of the likelihood function. This only uses a sensor model; compare with maximum a posteriori. The maximum likelihood can be biased but asymptotically becomes the minimum variance unbiased estimate. With additive Gaussian noise, the ML estimate is equivalent to the least squares estimate.

Maximum a posteriori (MAP) The estimate found from the mode of the posterior PDF. Compare with maximum likelihood.

Mean likelihood The estimate found from the mean of the likelihood function. This only uses a sensor model. It is less common than maximum likelihood. It is equivalent to the maximum likelihood for Gaussian shaped likelihood functions.

Measurement model a.k.a. observation model, sensor model. A model relating the hidden system state to the measurements.

Minimum mean squared error (MMSE) The estimate found from the mean of the posterior PDF.

Motion model a.k.a. system model, transition model. A model for the state transitions.

Multiple hypothesis Kalman filter (MHKF) Uses a mixture of Gaussians to represent multi-modal PDFs.

Multiple linear regression This is a linear model with more than one explanatory (independent) variable. For example, the model $h_n = h_0 + v_0 t_n + \frac{1}{2} a t_n^2$ has three parameters: h_0, v_0, and a. While the model is non-linear in t_n, it is linear in the parameters.

Multivariate regression The is a form of linear regression with more than one dependent variable. For example, a robot with two different range sensors.

Non-linear least squares Require search of unknown parameters to find minimum in an objective function. With multiple minima (non-convex), the search is slow for large parameter spaces. The search is faster if the gradient is known (see steepest-descent).

Off-line a.k.a. noncausal, post-processing. Using past, present data, and 'future' data. The entire time-series is available.

On-line a.k.a. on-the-fly, real-time, causal. Using past and present data.

Ordinary least squares This is a form of linear least squares where the unknown parameters can be found directly from a matrix inversion:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left(\mathbf{z} - \mathbf{A}\boldsymbol{\beta}\right)^{\mathrm{T}} \left(\mathbf{z} - \mathbf{A}\boldsymbol{\beta}\right), \tag{21.29}$$

$$= \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1} \mathbf{A}^{\mathrm{T}}\mathbf{z}. \tag{21.30}$$

Particle filter A Bayes filter that represents the posterior distribution with particles. It is suited to non-linear problems of low-dimensionality.

Predictor A filter that predicts future values from past and present values.

Polynomial regression A special case of multiple linear regression where the fitted curve is a polynomial.

Recursive Bayes-filter A Bayes filter implemented recursively on the basis of the complete state (Markov) assumption. Only the previous posterior PDF needs to be stored.

Regression analysis A set of statistical processes for estimating the relationships among variables. There are many types including linear regression and polynomial regression. Linear regression has one dependent variable and one independent variable. Multiple regression has multiple independent variables. Polynomial regression fits a polynomial to non-linear relationships.

Regressors independent variables used in a regression (a.k.a. explanatory variables). With least squares estimation, these are constructed into a design matrix, $\mathbf{A}$. Often a constant is included with the regressors.

Robust regression This can handle outliers. A popular technique is Iteratively reweighted least squares.

Sensor model a.k.a. measurement model, observation model. A model relating the hidden system state to the measurements.

Simple linear regression This has a measurement model $z_n = \alpha + \beta x_n + \epsilon_n$.

Smoothing Using past, present, and 'future' data, cf., filtering. For example, a forward-backwards Kalman filter is a smoother since it interpolates.

Steady state Kalman filter This is a Kalman filter with the Kalman gains pre-computed.

System identification The estimation of the parameters of a dynamic system.

System model a.k.a. motion model, transition model. A model for the state transitions.

Total least squares (TLS) This handles the case where both the dependent and independent variables have uncertainty. There also variants such as weighted total least squares (WTLS).

The parameter vector is estimated by solving the system of equations,

$$\begin{bmatrix} \mathbf{A} & -\mathbf{z} \end{bmatrix} \begin{bmatrix} \beta \\ 1 \end{bmatrix} = 0. \tag{21.31}$$

The solution is the singular vector in the right singular matrix $\mathbf{V}$ (determined from a singular value decomposition of $[\mathbf{A} \ -\mathbf{z}]$ corresponding to the minimum singular value, normalised so the last value is unity.

Transition model a.k.a. motion model, system model. A model for the state transitions.

Unscented Kalman filter (UKF) A Kalman filter for non-linear models with linear models that fits a Gaussian to the posterior PDF.

Weighted least squares (WLS) This is a special case of generalised least squares when there are no correlations but the variances of the errors differ.

Wiener filter on-line algorithm assuming stationary noise. It is used for prediction and extraction of a stochastic signal from additive stationary noise. For a Gaussian process, no non-linear filter can do better. It can be considered a special case of a Kalman filter for stationary noise and a time-invariant model.

The Wiener filter is also used for deconvolution.

22
Derivations

22.1 BLU estimation

The best linear unbiased (BLU) estimator, a.k.a. minimum variance linear unbiased (MVLU) estimator, is a minimum variance unbiased (MVU) estimator for the case of a linear system with additive zero-mean noise[1].

[1] In general, the MVU does not exist or cannot be found. In this case the maximum likelihood (ML) estimator is used. This is a computationally complex turn-the-crank method that is biased for small datasets.

Consider a system where we wish to estimate the state, X, from two sensor measurements. Assuming the noise is additive and zero-mean, the random variable describing the measurements can be expressed as

$$Z_1 = X + V_1, \tag{22.1}$$
$$Z_2 = X + V_2, \tag{22.2}$$

where V_1 and V_2 are zero-mean noise random variables. In vector form,

$$\mathbf{Z} = \mathbf{A}X + \mathbf{V}, \tag{22.3}$$

where $\mathbf{A} = [1, 1]^{\mathrm{T}}$.

The goal is to estimate X from the two measurements using a weighted average. This can be expressed as

$$\hat{X} = \mathbf{w}^{\mathrm{T}}\mathbf{Z}, \tag{22.4}$$

where the hat over X denotes an estimator. The mean of the estimator is,

$$\mathrm{E}\left[\hat{X}\right] = \mathrm{E}\left[\mathbf{w}^{\mathrm{T}}\mathbf{A}X\right] + \mathrm{E}\left[\mathbf{w}^{\mathrm{T}}\mathbf{V}\right], \tag{22.5}$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{A}\,\mathrm{E}\left[X\right] + \mathbf{w}^{\mathrm{T}}\,\mathrm{E}\left[\mathbf{V}\right], \tag{22.6}$$
$$= \mathbf{w}^{\mathrm{T}}\mathbf{A}\,\mathrm{E}\left[X\right], \tag{22.7}$$

since the noise was assumed to be zero-mean. For an unbiased estimator we require that $\mathrm{E}\left[\hat{X}\right] = \mathrm{E}\left[X\right]$, and thus

$$\mathbf{w}^{\mathrm{T}}\mathbf{A} = 1. \tag{22.8}$$

With $\mathbf{A} = [1,1]^{\mathrm{T}}$ this requires that the weights sum to unity.

The variance of the estimator is

$$\mathrm{Var}\left[\hat{X}\right] = \mathrm{E}\left[\hat{X}^2\right] - \mathrm{E}\left[\hat{X}\right]^2. \tag{22.9}$$

For a minimum variance estimator we need to minimise the mean-squared estimate since an unbiased estimate will not alter the mean of the estimate. The mean-squared estimator is

$$\begin{aligned}
\mathrm{E}\left[\hat{X}^2\right] &= \mathrm{E}\left[\left(\mathbf{w}^{\mathrm{T}}\mathbf{Z}\right)\left(\mathbf{w}^{\mathrm{T}}\mathbf{Z}\right)^{\mathrm{T}}\right], && (22.10)\\
&= \mathrm{E}\left[\mathbf{w}^{\mathrm{T}}\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\mathbf{w}\right], && (22.11)\\
&= \mathbf{w}^{\mathrm{T}}\,\mathrm{E}\left[\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\right]\mathbf{w}, && (22.12)\\
&= \mathbf{w}^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathbf{Z}}\mathbf{w}, && (22.13)
\end{aligned}$$

where $\boldsymbol{\Sigma}_{\mathbf{Z}}$ is the covariance matrix of the sensor signals. This is given by

$$\begin{aligned}
\boldsymbol{\Sigma}_{\mathbf{Z}} &= \mathrm{E}\left[\left(\mathbf{Z} - \mathrm{E}\left[\mathbf{Z}\right]\right)\left(\mathbf{Z} - \mathrm{E}\left[\mathbf{Z}\right]\right)^{\mathrm{T}}\right], && (22.14)\\
&= \mathrm{E}\left[\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\right] - \mathrm{E}\left[\mathbf{Z}\right]\mathrm{E}\left[\mathbf{Z}\right]^{\mathrm{T}}, && (22.15)
\end{aligned}$$

where

$$\begin{aligned}
\mathrm{E}\left[\mathbf{Z}\mathbf{Z}^{\mathrm{T}}\right] &= \mathrm{E}\left[\left(\mathbf{A}X + \mathbf{V}\right)\left(\mathbf{A}X + \mathbf{V}\right)^{\mathrm{T}}\right], && (22.16)\\
&= \mathrm{E}\left[\mathbf{A}X^2\mathbf{A}^{\mathrm{T}} + \mathbf{A}X\mathbf{V}^{\mathrm{T}} + \mathbf{V}X\mathbf{A}^{\mathrm{T}} + \mathbf{V}\mathbf{V}^{\mathrm{T}}\right], && (22.17)\\
&= \mathbf{A}\,\mathrm{E}\left[X^2\right]\mathbf{A}^{\mathrm{T}} + \mathbf{A}\,\mathrm{E}\left[X\mathbf{V}^{\mathrm{T}}\right] + \mathrm{E}\left[\mathbf{V}X\right]\mathbf{A}^{\mathrm{T}} + \mathrm{E}\left[\mathbf{V}\mathbf{V}^{\mathrm{T}}\right], && (22.18)
\end{aligned}$$

and

$$\mathrm{E}\left[\mathbf{Z}\right]\mathrm{E}\left[\mathbf{Z}\right]^{\mathrm{T}} = \mathbf{A}\,\mathrm{E}\left[X\right]\mathrm{E}\left[X^{\mathrm{T}}\right]\mathbf{A}^{\mathrm{T}}. \tag{22.19}$$

Assuming that the sensor noise is uncorrelated with X, then the cross-terms are zero, so

$$\boldsymbol{\Sigma}_{\mathbf{Z}} = \mathbf{A}\,\mathrm{E}\left[X^2\right]\mathbf{A}^{\mathrm{T}} + \mathrm{E}\left[\mathbf{V}\mathbf{V}^{\mathrm{T}}\right]. \tag{22.20}$$

Thus

$$\mathrm{Var}\left[X\right] = \mathbf{w}^{\mathrm{T}}\mathbf{A}\,\mathrm{E}\left[X^2\right]\mathbf{A}^{\mathrm{T}}\mathbf{w} + \mathbf{w}^{\mathrm{T}}\boldsymbol{\Sigma}_{\mathbf{V}}w - \mathrm{E}\left[X\right]^2. \tag{22.21}$$

Assuming the sensor noise is uncorrelated, its covariance matrix has zero elements off the diagonal,

$$\boldsymbol{\Sigma}_{\mathbf{V}} = \mathrm{E}\left[\mathbf{V}\mathbf{V}^{\mathrm{T}}\right] = \begin{bmatrix} \mathrm{Var}\left[V_1\right] & 0 \\ 0 & \mathrm{Var}\left[V_2\right] \end{bmatrix}. \tag{22.22}$$

22.1.1 BLU estimator without vector algebra

Let's consider the case of forming an estimate from a weighted sum of two measurements,

$$\hat{X} = w_1 Z_1 + w_2 Z_2, \tag{22.23}$$

where

$$Z_1 = X + V_1, \tag{22.24}$$
$$Z_2 = X + V_2, \tag{22.25}$$

and V_1 and V_2 are zero-mean noise random variables.

The mean of the estimator is

$$\begin{aligned} \mathrm{E}\left[\hat{X}\right] &= \mathrm{E}\left[w_1 Z_1 + w_2 Z_2\right], && (22.26)\\ &= \mathrm{E}\left[w_1 X + w_1 V_1 + w_2 X + V_2\right], && (22.27)\\ &= w_1 \mathrm{E}\left[X\right] + w_1 \mathrm{E}\left[V_1\right] + w_2 \mathrm{E}\left[X\right] + w_2 \mathrm{E}\left[V_2\right], && (22.28)\\ &= (w_1 + w_2)\,\mathrm{E}\left[X\right]. && (22.29) \end{aligned}$$

For an unbiased estimator, this requires that $w_1 + w_2 = 1$.

The variance of the estimator is

$$\mathrm{Var}\left[\hat{X}\right] = \mathrm{E}\left[\hat{X}^2\right] - \mathrm{E}\left[\hat{X}\right]^2, \tag{22.30}$$

where $\mathrm{E}\left[\hat{X}^2\right]$ is the mean-square value of the estimator. This is given by

$$\begin{aligned} \mathrm{E}\left[\hat{X}^2\right] &= \mathrm{E}\left[\left(w_1 X + w_1 V_1 + w_2 X + w_2 V_2\right)^2\right], && (22.31)\\ &= w_1^2 \mathrm{E}\left[X^2\right] + w_2^2 \mathrm{E}\left[X^2\right] + 2w_1 w_2 \mathrm{E}\left[X V_1\right]\\ &\quad + 2w_1 w_2 \mathrm{E}\left[X V_2\right] + 2w_1 w_2 \mathrm{E}\left[V_1 V_2\right]\\ &\quad + w_1^2 \mathrm{E}\left[V_1^2\right] + w_2^2 \mathrm{E}\left[V_2^2\right]. && (22.32) \end{aligned}$$

If the sensor noise is uncorrelated with each other then $\mathrm{E}\left[V_1 V_2\right] = 0$. If the sensor noise is also uncorrelated with X then $\mathrm{E}\left[V_1 X\right] = \mathrm{E}\left[V_2 X\right] = 0$. This simplifies (22.32) to

$$\mathrm{E}\left[\hat{X}^2\right] = w_1^2 \mathrm{E}\left[X^2\right] + w_2^2 \mathrm{E}\left[X^2\right] + w_1^2 \mathrm{E}\left[V_1^2\right] + w_2^2 \mathrm{E}\left[V_2^2\right]. \tag{22.33}$$

Since the noise is zero mean, the mean-square noise values are equal to their variances, and thus

$$\mathrm{E}\left[\hat{X}^2\right] = w_1^2 \mathrm{E}\left[X^2\right] + w_2^2 \mathrm{E}\left[X^2\right] + w_1^2 \mathrm{Var}\left[V_1\right] + w_2^2 \mathrm{Var}\left[V_2\right]. \tag{22.34}$$

Furthermore, since $w_2 = 1 - w_1$,

$$\mathrm{E}\left[\hat{X}^2\right] = \mathrm{E}\left[X^2\right] w_1^2 \operatorname{Var}[V_1] + (1 - w_1)^2 \operatorname{Var}[V_2]\,. \quad (22.35)$$

To minimise this with respect to w_1 we differentiate it with respect to w_1 and solve for w_1 where the derivative is zero. This gives

$$w_1 = \frac{\operatorname{Var}[V_2]}{\operatorname{Var}[V_1] + \operatorname{Var}[V_2]}. \quad (22.36)$$

This can be expressed as

$$w_1 = \frac{\frac{1}{\operatorname{Var}[V_1]}}{\frac{1}{\operatorname{Var}[V_1]} + \frac{1}{\operatorname{Var}[V_2]}}, \quad (22.37)$$

and so

$$w_2 = \frac{\frac{1}{\operatorname{Var}[V_2]}}{\frac{1}{\operatorname{Var}[V_1]} + \frac{1}{\operatorname{Var}[V_2]}}, \quad (22.38)$$

Thus, the optimal choice of weights are proportional to the inverse of the variance. This yields the BLU estimator

$$\hat{X}_{\mathrm{BLU}} = \frac{\frac{1}{\operatorname{Var}[V_1]} Z_1 + \frac{1}{\operatorname{Var}[V_2]} Z_2}{\frac{1}{\operatorname{Var}[V_1]} + \frac{1}{\operatorname{Var}[V_2]}}, \quad (22.39)$$

$$= \frac{\operatorname{Var}[V_2]\, Z_1 + \operatorname{Var}[V_1]\, Z_2}{\operatorname{Var}[V_1] + \operatorname{Var}[V_2]}. \quad (22.40)$$

Using these optimal weights, the resulting variance is

$$\operatorname{Var}\left[\hat{X}_{\mathrm{BLU}}\right] = \frac{1}{\frac{1}{\operatorname{Var}[V_1]} + \frac{1}{\operatorname{Var}[V_2]}}, \quad (22.41)$$

$$= \frac{\operatorname{Var}[V_1] \operatorname{Var}[V_2]}{\operatorname{Var}[V_1] + \operatorname{Var}[V_2]}. \quad (22.42)$$

22.2 *PDF of a weighted sum*

Consider the weighted sum of two random variables,

$$Z = w_1 X + (1 - w_1) Y. \quad (22.43)$$

If X and Y are independent, the PDF of Z is given by a convolution

$$f_Z(z) = \int_{-\infty}^{\infty} f_{X'}(x) f_{Y'}(z - x)\mathrm{d}x, \quad (22.44)$$

where

$$f_{X'}(x') = \frac{1}{w_1} f_X\left(\frac{x'}{w_1}\right), \tag{22.45}$$

$$f_{Y'}(y') = \frac{1}{1-w_1} f_Y\left(\frac{y'}{1-w_1}\right). \tag{22.46}$$

Thus[2],

$$f_Z(z) = \int_{-\infty}^{\infty} \frac{1}{w_1} f_X\left(\frac{x}{w_1}\right) \frac{1}{1-w_1} f_Y\left(\frac{z-x}{1-w_1}\right) \mathrm{d}x. \tag{22.47}$$

[2] Note, when $w_1 = 0$ or $w_1 = 1$, care needs to be taken with the integral since Dirac deltas are involved.

For example, consider two uniform random variables, where

$$f_X(x) = \frac{1}{W_x} \operatorname{rect}\left(\frac{x-\mu_X}{W_x}\right), \tag{22.48}$$

$$f_Y(y) = \frac{1}{W_y} \operatorname{rect}\left(\frac{y-\mu_Y}{W_y}\right). \tag{22.49}$$

The convolution of two rectangle functions gives a trapezoid function with a centre value,

$$\mu_Z = w_1\mu_X + (1-w_1)\mu_Y, \tag{22.50}$$

a base width

$$W_z = w_1 W_x + (1-w_1) W_y, \tag{22.51}$$

and a top width,

$$T_z = \left|w_1 W_x - (1-w_1) W_y\right|. \tag{22.52}$$

22.3 *Differentiation of discrete time signal with noise*

Consider a sampled measured signal that comprises the ideal signal, s_n, with additive noise. We can represent the measured signal at each sample as a random variable, X_n,

$$X_n = s_n + W_n, \tag{22.53}$$

where W_n is a random-variable for the n-th sample denoting the noise.

Let's now consider a first order discrete differentiator,

$$D_n = \frac{X_n - X_{n-1}}{\Delta t}, \tag{22.54}$$

where Δt is the sampling interval. Using (22.53), (22.54) can be written as

$$D_n = \frac{s_n - s_{n-1}}{\Delta t} + \frac{W_n - W_{n-1}}{\Delta t}. \tag{22.55}$$

If the noise is zero-mean, then the expected value for the differentiated signal is

$$\begin{aligned} \mathrm{E}[D_n] &= \frac{s_n - s_{n-1}}{\Delta t} + \frac{\mathrm{E}[W_n - W_{n-1}]}{\Delta t}, && (22.56)\\ &= \frac{s_n - s_{n-1}}{\Delta t} + \frac{\mathrm{E}[W_n] - \mathrm{E}[W_{n-1}]}{\Delta t}, && (22.57)\\ &= \frac{s_n - s_{n-1}}{\Delta t}. && (22.58) \end{aligned}$$

The variance is

$$\begin{aligned} \mathrm{Var}[D_n] &= \mathrm{E}\left[(D_n - \mathrm{E}[D_n])^2\right], && (22.59)\\ &= \mathrm{E}\left[D_n^2\right] - \mathrm{E}[D_n]^2, && (22.60)\\ &= \mathrm{E}\left[\left(\frac{\mathrm{E}[W_n] - \mathrm{E}[W_{n-1}]}{\Delta t}\right)^2\right], && (22.61)\\ &= \frac{1}{(\Delta t)^2}\left(\mathrm{E}[W_n]^2 + \mathrm{E}[W_{n-1}]^2 - 2\,\mathrm{E}[W_n W_{n-1}]\right). && (22.62) \end{aligned}$$

The first term is the mean square value of the noise at sample n. Since the noise is assumed to be zero mean, this is equal to the variance. If the noise is stationary (i.e., its mean and variance does not change), then the variance of each noise sample is the same. The last term is the covariance between W_n and W_{n-1}. If the noise is uncorrelated (i.e., has a white power spectrum) then this term is zero. Denoting the noise variance by $\mathrm{Var}[W]$, the variance of the differentiated signal is

$$\mathrm{Var}[D_n] = \frac{2\,\mathrm{Var}[W]}{(\Delta t)^2}. \tag{22.63}$$

Note, the smaller the sampling interval, Δt, the greater the noise amplification.

22.4 *Integration of discrete time signal with noise*

The difference equation for a first-order integrator is

$$I_n = I_{n-1} + X_n \Delta t. \tag{22.64}$$

Assuming that $I_0 = 0$, this can be expressed as

$$I_n = \sum_{i=0}^{n-1} I_i + X_n \Delta t, \tag{22.65}$$

$$= \Delta t \sum_{i=0}^{n} X_i, \tag{22.66}$$

$$= \Delta t \sum_{i=0}^{n} s_i + \Delta t \sum_{i=0}^{n} W_i. \tag{22.67}$$

Since the noise is assumed to be additive and zero mean, it is straightforward to show that,

$$\mathrm{E}\left[I_n\right] = \Delta t \sum_{i=0}^{n} s_i. \tag{22.68}$$

This is the noise-free integrated signal.

Again assuming zero-mean, stationary, uncorrelated noise, the variance of the integrated signal can be shown to be

$$\mathrm{Var}\left[I_n\right] = n\left(\Delta t\right)^2 \mathrm{Var}\left[W\right]. \tag{22.69}$$

Note, the integrated noise is not stationary[3] since the variance varies with n.

[3] This is an example of a Gaussian random walk process.

Let's consider a discrete-time sinewave of the form,

$$s_n = A \sin\left(2\pi f n \Delta t\right). \tag{22.70}$$

The first order difference is

$$\frac{s_n - s_{n-1}}{\Delta t} = \frac{A}{\Delta t}\left(\sin\left(2\pi f n \Delta t\right) - \sin\left(2\pi f (n-1)\Delta t\right)\right). \tag{22.71}$$

Using a trigonometric identity, this can be expressed as

$$\frac{s_n - s_{n-1}}{\Delta t} = \frac{2A}{\Delta t}\left(\sin\left(2\pi f \frac{\Delta t}{2}\right)\cos\left(2\pi f\left(n - \frac{1}{2}\right)\Delta t\right)\right). \tag{22.72}$$

If the frequency, f, of the sinusoid is small such that $\sin\theta \approx \theta$, then

$$\frac{s_n - s_{n-1}}{\Delta t} \approx 2\pi f A \cos\left(2\pi f\left(n - \frac{1}{2}\right)\Delta t\right). \tag{22.73}$$

This can be written as

$$\frac{s_n - s_{n-1}}{\Delta t} \approx 2\pi f A \cos\left(2\pi f n \Delta t - \pi f \Delta t\right). \tag{22.74}$$

It can be seen to be equivalent to the derivative of a sinewave but with a phase delay of $\pi f \Delta t$. If the frequency and sampling interval are small, this phase shift

is negligible. For example, if the signal is oversampled by a factor of 10, then $f\Delta t = 0.1$ and the phase delay is 18 degrees.

Differentiating a sinusoid scales the amplitude by $2\pi f$. At the same time the standard deviation of the noise scales by $\sqrt{2}/\Delta t$. Thus the more the signal is oversampled, the greater the level of noise compared to the signal when it is differentiated.

22.5 Propagation of belief

The PDF of the predicted state belief, X_n, can be found using

$$f_{X_n}(x_n; u_{n-1}) = \int_{-\infty}^{\infty} f_{X_n|X_{n-1}}(x_n|x_{n-1}; u_{n-1}) f_{X_{n-1}}(x_{n-1}) \mathrm{d}x_{n-1}. \tag{22.75}$$

Since the process noise is additive,

$$f_{X_n|X_{n-1}}(x_n|x_{n-1}; u_{n-1}) = f_{W_n}(x_n - Ax_{n-1} - Bu_{n-1}), \tag{22.76}$$

and so the PDF of X_n can be found from a convolution of the PDF of X_{n-1} with the PDF of W_n,

$$f_{X_n}(x_n) = \int_{-\infty}^{\infty} f_{X_{n-1}}(x_{n-1}) f_{W_n}(x_n - Ax_{n-1} - Bu_{n-1}) \mathrm{d}x_{n-1}. \tag{22.77}$$

Note, if there is no uncertainty in the motion model

$$f_{W_n}(w_n) = \delta(w_n), \tag{22.78}$$

and due to the sifting property[4] of a Dirac delta, (22.77) collapses to

$$f_{X_n}(x_n) = \frac{1}{|A|} f_{X_{n-1}}\left(\frac{x_n - Bu_{n-1}}{A}\right). \tag{22.79}$$

In this case, the PDF is just shifted and scaled.

[4] $\int_{-\infty}^{\infty} \delta(x-u) f(u) \mathrm{d}u = f(x)$.

22.6 Linear mapping of a random variable

Consider the linear mapping between a random variable X to another random variable Y, where

$$Y = aX + b. \tag{22.80}$$

Here a is a scale factor and b is an offset. The mean of Y is related to the mean of X by

$$\begin{aligned} \mu_Y &= \mathrm{E}[Y], && (22.81)\\ &= \mathrm{E}[aX] + b, && (22.82)\\ &= a\,\mathrm{E}[X] + b, && (22.83)\\ &= a\mu_X + b. && (22.84) \end{aligned}$$

The variance of Y is related[5] to the variance of X by

$$\begin{aligned} \sigma_Y^2 &= \mathrm{Var}[Y], && (22.85)\\ &= \mathrm{E}\left[(Y-\mu_Y)^2\right], && (22.86)\\ &= \mathrm{E}\left[(aX+b-(a\mu_X+b))^2\right], && (22.87)\\ &= a^2\,\mathrm{E}\left[(X-\mu_X)^2\right], && (22.88)\\ &= a^2\,\mathrm{Var}[X]. && (22.89) \end{aligned}$$

[5] This relation is called the propagation of uncertainty.

If $f_X(x)$ is Gaussian[6] then it can be characterised by the mean and variance and so with a linear mapping, $f_Y(y)$ will also be Gaussian. However, in general, there is no simple mapping of the PDF, $f_Y(y)$, given the PDF, $f_X(x)$.

[6] With other distributions, the mapping of higher order statistics are required.

22.7 General mapping of a random variable

Consider a non-linear mapping of a Gaussian random variable X,

$$Y = g(X). \tag{22.90}$$

An approximate approach to characterise the distribution of Y is to linearise the mapping, $g(x)$, of a value x of X around the mean, μ_X, of X,

$$Y \approx \frac{\mathrm{d}}{\mathrm{d}X} g(X)\Big|_{X=\mu_X} (X-\mu_X) + g(\mu_X). \tag{22.91}$$

22.8 Derivation of Kalman filter as special case of a recursive Bayes filter

In the following derivation, a single state variable, x_n, with a single control action, u_n, and single measurement, z_n is assumed.

The initial Gaussian belief distribution is

$$f_{X_0^+}(x_0) = \frac{1}{\sqrt{2\pi\,\mathrm{Var}\left[X_0^+\right]}} \exp\left(-0.5\frac{(x_0-\hat{x}_0^+)^2}{\mathrm{Var}\left[X_0^+\right]}\right), \tag{22.92}$$

where $\hat{x}_0^+$ is the initial guess with variance $\text{Var}\left[X_0^+\right]$.

Using the state transition conditional probability density, $f_{X_1|X_0}(x_1|x_0;u_1)$, the predicted belief distribution is

$$f_{X_1^-}(x_1) = \int_{-\infty}^{\infty} f_{X_1|X_0}(x_1|x_0;u_1) f_{X_0^+}(x_0)\mathrm{d}x_0. \tag{22.93}$$

With additive Gaussian process noise, the state transition can be described by a function g,

$$f_{X_1|X_0}(x_1|x_0;u_1) = \frac{1}{\sqrt{2\pi\,\text{Var}\left[W\right]}} \exp\left(-0.5\frac{\left(x_1 - g(x_0,u_1)\right)^2}{\text{Var}\left[W\right]}\right). \tag{22.94}$$

Assuming a linear system,

$$g(x_0,u_1) = Ax_0 + Bu_1, \tag{22.95}$$

then

$$f_{X_1|X_0}(x_1|x_0;u_1) = \frac{1}{\sqrt{2\pi\,\text{Var}\left[W\right]}} \exp\left(-0.5\frac{\left(x_1 - Ax_0 - Bu_1\right)^2}{\text{Var}\left[W\right]}\right). \tag{22.96}$$

Thus (22.94) can be expressed as

$$f_{X_1^-}(x_1) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\,\text{Var}\left[W\right]}} \exp\left(-0.5\frac{\left(x_1 - Ax_0 - Bu_1\right)^2}{\text{Var}\left[W\right]}\right) \tag{22.97}$$

$$\frac{1}{\sqrt{2\pi\,\text{Var}\left[X_0^+\right]}} \exp\left(-0.5\frac{\left(x_0 - \hat{x}_0^+\right)^2}{\text{Var}\left[X_0^+\right]}\right)\mathrm{d}x_0.$$

Solving the integral gives

$$f_{X_1^-}(x_1) = \frac{\sqrt{2\pi\,\text{Var}\left[W\right]\text{Var}\left[X_1^+\right]}}{\text{Var}\left[X_1^-\right]} \exp\left(-0.5\frac{\left(x_1 - \hat{x}_1^-\right)^2}{\text{Var}\left[X_1^-\right]}\right), \tag{22.98}$$

where

$$\hat{x}_1^- = A\hat{x}_0^+ + Bu_1, \tag{22.99}$$

$$\text{Var}\left[X_1^-\right] = A^2\,\text{Var}\left[X_0^+\right] + \text{Var}\left[W\right]. \tag{22.100}$$

Using Bayes' theorem, the posteriori belief distribution is

$$f_{X_1^+}(x_1) = \eta\, L_{X_1|Z_1}(x_1|z_1) f_{X_1^-}(x_1). \tag{22.101}$$

Assuming additive Gaussian sensor noise, the measurement likelihood is

$$L_{X_1|Z_1}(x_1|z_1) = \frac{1}{\sqrt{2\pi \operatorname{Var}[V]}} \exp\left(-0.5\frac{(z_1 - h(x_1))^2}{\operatorname{Var}[V]}\right). \tag{22.102}$$

Assuming a linear system,

$$h(x_1) = Cx_1, \tag{22.103}$$

then the likelihood becomes

$$L_{X_1|Z_1}(x_1|z_1) = \frac{1}{\sqrt{2\pi \operatorname{Var}[V]}} \exp\left(-0.5\frac{(z_1 - Cx_1)^2}{\operatorname{Var}[V]}\right). \tag{22.104}$$

Using this, the posterior belief distribution is given by

$$f_{X_1^+}(x_1) = \eta \frac{1}{\sqrt{2\pi \operatorname{Var}[V]}} \exp\left(-0.5\frac{(z_1 - Cx_1)^2}{\operatorname{Var}[V]}\right) \frac{\sqrt{2\pi \operatorname{Var}[W] \operatorname{Var}\left[X_0^+\right]}}{\operatorname{Var}\left[X_1^-\right]} \exp\left(-0.5\frac{(x_1 - \hat{x}_1^-)^2}{\operatorname{Var}\left[X_1^-\right]}\right), \tag{22.105}$$

or

$$f_{X_1^+}(x_1) = \eta' \exp\left(-0.5\frac{(x_1 - \hat{x}_1^+)^2}{\operatorname{Var}\left[X_1^+\right]}\right), \tag{22.106}$$

where

$$\hat{x}_1^+ = \hat{x}_1^- + \frac{C \operatorname{Var}\left[X_1^-\right](z_1 - C\hat{x}_1^-)}{C^2 \operatorname{Var}\left[X_1^-\right] + \operatorname{Var}[V]}, \tag{22.107}$$

$$\operatorname{Var}[X_1] = \frac{\operatorname{Var}[V] \operatorname{Var}\left[X_1^-\right]}{C^2 \operatorname{Var}\left[X_1^-\right] + \operatorname{Var}[V]}, \tag{22.108}$$

$$\eta' = \frac{1}{\sqrt{2\pi \operatorname{Var}\left[X_1^+\right]}}. \tag{22.109}$$

This result is the same as that given by (22.141).

22.9 *Particle filter maths*

Particle filters use a set of M particles to represent the belief posterior density, $f_{X_n^+}(x_n)$, at each time-step n:

$$\mathcal{X}_n = \left\{x_n^{(0)}, x_n^{(1)}, \cdots, x_n^{(M-1)}\right\}. \tag{22.110}$$

Initially, the particles are chosen using *inverse transform sampling* to approximate the initial belief distribution.

Then for each iteration, the predict and update steps of a Bayes filter are applied:

$$\text{predict: } f_{X_n^-}(x_n) = \int f_{X_n|X_{n-1}}(x_n|x_{n-1};u_{n-1}) f_{X_{n-1}^+}(x_{n-1})\mathrm{d}x_{n-1}, \tag{22.111}$$

$$\text{update: } f_{X_n^+}(x_n) = \eta L_{X_n|Z_n}(x_n|z_n) f_{X_n^-}(x_n). \tag{22.112}$$

The prediction step is achieved by selecting a new set of particles to represent the prior PDF $f_{X_n^-}(x_n)$,

$$\mathcal{X} = \left\{x^{(0)}, x^{(1)}, \cdots, x^{(M-1)}\right\}, \tag{22.113}$$

where each new particle is sampled from the state transition probability density[7], conditioned on the previous control, u_{n-1}, and each previous particle, $x_{n-1}^{(m)} \in \mathcal{X}_{n-1}$,

$$x^{(m)} \sim p\left(x_n^{(m)}|x_{n-1}^{(m)}, u_{n-1}\right). \tag{22.114}$$

[7] This is usually called the motion model.

The update step creates a set of weights, one per particle,

$$\mathcal{A}_n = \left\{a_n^{(0)}, a_n^{(1)}, \cdots, a_n^{(M-1)}\right\}, \tag{22.115}$$

where the weights are selected using the measurement likelihood[8],

$$a_n^{(m)} = L_{X_n|Z_n}\left(x_n^{(m)}|z_n\right). \tag{22.116}$$

[8] The sensor model.

22.9.1 *Resampling*

While the set of weights, $\mathcal{A}_n$, and set of particles, $\mathcal{X}_n$ represent the new belief[9], $f_{X_n^+}(x_n)$, some of the weights may be close to zero and so a resampling step is performed to create a new set of particles, $\mathcal{X}_n$, sampled from $f_{X_n^+}(x_n)$.

The resampling step draws particles from $\mathcal{X}$ (with replacement) with a probability given by $\mathcal{A}_n$. This can result in many duplicate particles in $\mathcal{X}_n$ but they will be dispersed after the motion model is applied. Note, that the new set of particles approximate the posterior density without the weights, for example, for a 1-D case

$$f_X(x) \approx \sum_{m=0}^{M-1} \frac{w^{(m)}}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x-x^{(m)}}{\sigma}\right)^2\right) \tag{22.118}$$

$$\approx \sum_{m=0}^{M-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{x-x'^{(m)}}{\sigma}\right)^2\right) \tag{22.119}$$

[9] For example, the estimated mean state can be calculated using a weighted average of the particles:

$$\overline{x}_n = \frac{\sum_{m=0}^{M-1} a_n^{(m)} x_n^{(m)}}{\sum_{m=0}^{M-1} a_n^{(m)}}. \tag{22.117}$$

22.9.2 Practical considerations

In practice, the dimensionality of $f_{X_n|X_{n-1}}\left(x_n|x_{n-1}^{(m)};u_{n-1}\right)$ is too high for it to be used as a lookup table. So again the state transition is often considered to be deterministic with additive process noise. The resultant state transition conditional probability density is

$$f_{X_n|X_{n-1}}(x_n|x_{n-1};u_{n-1}) = f_W\left(x_n - g(x_{n-1},u_{n-1})\right). \tag{22.120}$$

where $f_W(w_n)$ is the process noise probability density. Similarly, the measurement is considered to be corrupted by additive noise so the measurement likelihood is

$$L_{X_n|Z_n}(x_n|z_n) = f_V\left(z_n - h(x_n,u_{n-1})\right), \tag{22.121}$$

where $f_V(v_n)$ is the measurement noise probability density.

With these approximations, the particle filter algorithm for each time-step is:

1. Create a set of particles, $\mathcal{X}$, to represent the predicted belief where the particles are sampled according to

$$x^{(m)} \sim f_W\left(x_n - g(x_{n-1}^{(m)},u_{n-1})\right). \tag{22.122}$$

   For additive noise, this is equivalent to

$$x^{(m)} = g(x_{n-1}^{(m)},u_{n-1}) + w_n, \tag{22.123}$$

   where w_n is the process noise sampled from $f_W(w)$.

2. Create a set of particle weights, $\mathcal{A}_n$, where

$$a_n^{(m)} = f_V\left(z_n - h(x^{(m)},u_{n-1})\right). \tag{22.124}$$

3. Resample the particles to create $\mathcal{X}_n$, where the m^{th} particle is drawn (with replacement) from $\mathcal{X}$ with a probability proportional to $a_n^{(m)}$.

22.10 General mapping of a random vector

Consider a non-linear and multidimensional system,

$$\mathbf{Y} = g(\mathbf{X}), \tag{22.125}$$

then a linear approximation can be employed around the mean, $\boldsymbol{\mu}_{\mathbf{X}}$,

$$\mathbf{Y} \approx \mathbf{J}\,(\mathbf{X} - \boldsymbol{\mu}_{\mathbf{X}}) + g(\boldsymbol{\mu}_{\mathbf{X}}), \tag{22.126}$$

where $\mathbf{J}$ is a Jacobian matrix (evaluated at $\mathbf{X} = \boldsymbol{\mu}_{\mathbf{X}}$),

$$\mathbf{J} = \begin{bmatrix} \frac{\partial Y_1}{\partial X_1} & \cdots & \frac{\partial Y_1}{\partial X_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial Y_m}{\partial X_1} & \cdots & \frac{\partial Y_m}{\partial X_n} \end{bmatrix}. \tag{22.127}$$

With this approximation, the covariance of $\mathbf{Y}$ is related to the covariance of $\mathbf{X}$ by

$$\boldsymbol{\Sigma}_{\mathbf{Y}} = \mathbf{J}\,\boldsymbol{\Sigma}_{\mathbf{X}}\,\mathbf{J}^{\mathrm{T}}. \tag{22.128}$$

22.11 Kalman filters

22.11.1 Bayes filters to Kalman filters

A Bayes filter represents a model of a discrete-time continuous system by the *state transition probability* density, $f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1};\mathbf{u}_{n-1})$, and the *measurement likelihood*, $L_{\mathbf{X}_n|\mathbf{Z}_n}(\mathbf{x}_n|\mathbf{z}_n)$. Its belief of the system state is represented by an arbitrary probability density, $f_{\mathbf{X}_n|\mathbf{Z}_{0:n}}(\mathbf{x}_n|\mathbf{z}_{0:n};\mathbf{u}_{0:n})$.

A simpler, but less general, probabilistic system model assumes zero-mean additive Gaussian[10] process and measurement noise:

[10] This assumption is not required for particle filters.

$$\mathbf{X}_n = g(\mathbf{X}_{n-1}, \mathbf{u}_{n-1}) + \mathbf{W}_n, \tag{22.129}$$
$$\mathbf{Z}_n = h(\mathbf{X}_n, \mathbf{u}_{n-1}) + \mathbf{V}_n, \tag{22.130}$$

so the state transition probability density is

$$f_{\mathbf{x}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1};\mathbf{u}_{n-1}) = \eta \exp\left(-\frac{1}{2}\,(\mathbf{x}_n - \mu_{\mathbf{X}_n})^{\mathrm{T}}\,\boldsymbol{\Sigma}_{\mathbf{W}}^{-1}\,(\mathbf{x}_n - \mu_{\mathbf{X}_n})\right), \tag{22.131}$$

where

$$\mu_{\mathbf{X}_n} = g(\mathbf{X}_{n-1}, \mathbf{u}_{n-1}), \tag{22.132}$$

and the measurement likelihood is

$$L_{\mathbf{X}_n|\mathbf{Z}_n}(\mathbf{x}_n|\mathbf{z}_n) = \eta \exp\left(-\frac{1}{2}\,(\mathbf{z}_n - \mu_{\mathbf{Z}_n})^{\mathrm{T}}\,\boldsymbol{\Sigma}_{\mathbf{V}}^{-1}\,(\mathbf{z}_n - \mu_{\mathbf{Z}_n})\right), \tag{22.133}$$

where

$$\mu_{\mathbf{Z}_n} = h(\mathbf{X}_n, \mathbf{u}_{n-1}). \tag{22.134}$$

In both (22.131) and (22.133), η is a normalising constant.

Both the extended and unscented Kalman filters use this approximation and, in addition, represent the belief by a Gaussian density[11]. However, in general, a Gaussian belief density is a poor approximation for a non-linear system.

The Kalman filter requires linear models[12], so

$$\mathbf{X}_n = \mathbf{A}\mathbf{X}_{n-1} + \mathbf{B}\mathbf{u}_{n-1} + \mathbf{W}_n, \tag{22.135}$$
$$\mathbf{Z}_n = \mathbf{C}\mathbf{X}_n + \mathbf{D}\mathbf{u}_{n-1} + \mathbf{V}_n. \tag{22.136}$$

[11] This only requires a mean vector and covariance matrix to parameterise.

[12] With a linear model, a Gaussian belief always maps to another Gaussian belief at the next time-step.

22.11.2 *Single state Kalman filter*

The Kalman filter equations for a single state variable, x, single control action, u, and single measurement, z, are:

$$\hat{x}_n^- = A\hat{x}_{n-1}^- + Bu_n \tag{22.137}$$
$$\operatorname{Var}\left[\hat{X}_n^-\right] = A^2 \operatorname{Var}\left[\hat{X}_{n-1}^-\right] + \operatorname{Var}[W] \tag{22.138}$$
$$K_n = \frac{C \operatorname{Var}\left[\hat{X}_n^-\right]}{C^2 \operatorname{Var}\left[\hat{X}_n^-\right] + \operatorname{Var}[V]} \tag{22.139}$$
$$\hat{x}_{n-1}^+ = \hat{x}_n^- + K_n\left(z_n - C\hat{x}_n^-\right) \tag{22.140}$$
$$\operatorname{Var}\left[\hat{X}_{n-1}^+\right] = (1 - K_n C)\operatorname{Var}\left[\hat{X}_n^-\right] = \frac{\operatorname{Var}[V]\operatorname{Var}\left[\hat{X}_n^-\right]}{C^2 \operatorname{Var}\left[\hat{x}_n^-\right] + \operatorname{Var}[V]} \tag{22.141}$$

22.11.3 *Kalman filter alternative implementations*

There are many descriptions of the Kalman filter. Some model the state and output equations as:

$$\mathbf{X}_{n+1} = \mathbf{A}\mathbf{X}_n + \mathbf{B}\mathbf{u}_n + \mathbf{W}_n,$$
$$\mathbf{Z}_n = \mathbf{C}\mathbf{X}_n + \mathbf{D}\mathbf{u}_n + \mathbf{V}_n. \tag{22.142}$$

This leads to the state posterior estimate

$$\hat{\mathbf{x}}_{n+1}^+ = \left(\mathbf{A}\hat{\mathbf{x}}_n^+ + \mathbf{B}\mathbf{u}_n\right) + \mathbf{K}_n\left(\mathbf{z}_n - \mathbf{C}\hat{\mathbf{x}}_n^+\right), \tag{22.143}$$

where the Kalman gain matrix is

$$\mathbf{K}_n = \mathbf{A}P_n^+\mathbf{C}^{\mathrm{T}}\left(\mathbf{C}P_n^+\mathbf{C}^{\mathrm{T}} + \mathbf{\Sigma_V}\right)^{-1}, \tag{22.144}$$

and the state posterior estimate covariance matrix is

$$P_{n+1}^+ = \mathbf{A}P_n^+\mathbf{A}^{\mathrm{T}} + \mathbf{\Sigma_W} - \mathbf{A}P_n^+\mathbf{C}^{\mathrm{T}}\mathbf{\Sigma_V}^{-1}\mathbf{C}P_n^+\mathbf{A}^T. \tag{22.145}$$

22.11.4 Steady-state Kalman filter

The Kalman filter requires a matrix inversion. To avoid this overhead, one solution is to use a pre-computed Kalman gain matrix ($\mathbf{K}_n = \mathbf{K}$). This assumes that the noise statistics are unchanging, the system is time-invariant[13], and the system is stable. With a constant Kalman gain matrix, the steady-state Kalman filter equations simplify to:

[13] With a time-invariant system, the state-space matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ do not change.

$$\hat{\mathbf{x}}_n^- = \mathbf{A}\hat{\mathbf{x}}_n^- + \mathbf{B}\mathbf{u}_{n-1}, \tag{22.146}$$

$$\hat{\mathbf{x}}_n^+ = \hat{\mathbf{x}}_n^- + \mathbf{K}\left(\mathbf{z}_n - \mathbf{C}\hat{\mathbf{x}}_n^-\right). \tag{22.147}$$

Given the steady-state covariance, $\mathbf{P}$, the Kalman gain matrix is found from

$$\mathbf{K} = \mathbf{P}\mathbf{C}^\mathrm{T}\left(\mathbf{C}\mathbf{P}\mathbf{C}^\mathrm{T} + \mathbf{\Sigma}_{\mathbf{V}_n}\right)^{-1}. \tag{22.148}$$

Unfortunately, finding the steady-state covariance, $\mathbf{P}$, is non trivial. It requires solving a matrix Ricatti equation:

$$\mathbf{P} = \mathbf{A}\left(\mathbf{P} - \mathbf{P}\mathbf{C}^\mathrm{T}\left(\mathbf{C}\mathbf{P}\mathbf{C}^\mathrm{T} + \mathbf{\Sigma}_{\mathbf{V}_n}\right)^{-1}\mathbf{C}\mathbf{P}\right)\mathbf{A}^\mathrm{T} + \mathbf{B}\mathbf{\Sigma}_{\mathbf{W}_n}\mathbf{B}^\mathrm{T}. \tag{22.149}$$

22.12 Differential drive motion models

This section gives more detailed mathematics for the velocity model derivation, linearised velocity model, inverse velocity model, and odometry model parameterisation for differential drive robots. The current pose is denoted as $\mathbf{x}_n = \mathbf{x} = (x, y, \theta)^\mathrm{T}$ and the previous pose is denoted with a prime, i.e., $\mathbf{x}_{n-1} = \mathbf{x}' = (x', y', \theta')^\mathrm{T}$.

22.12.1 Velocity model derivation

Assuming a constant translation speed v and rotational speed ω, the robot moves on a circle of radius, r,

$$r = \left|\frac{v}{\omega}\right|. \tag{22.150}$$

Note, when the rotational speed, ω, is zero then the circle has an infinite radius corresponding to a straight line. The centre of the circle is given by

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} x - r\sin\theta \\ y + r\cos\theta \end{bmatrix}. \tag{22.151}$$

Thus

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} x' - r\sin\theta' \\ y' + r\cos\theta' \end{bmatrix} = \begin{bmatrix} x - r\sin\theta \\ y + r\cos\theta \end{bmatrix}, \tag{22.152}$$

and so

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' - r\sin\theta + r\sin(\theta' + \omega\Delta t) \\ y' + r\cos\theta - r\cos(\theta' + \omega\Delta t) \end{bmatrix}. \tag{22.153}$$

22.12.2 *Linearised velocity motion model*

Using the double angle formulae, (13.5) can be expanded as

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} + \begin{bmatrix} -\frac{v}{\omega}\sin\theta' + \frac{v}{\omega}(\sin\theta'\cos\omega\Delta t + \cos\theta'\sin\omega\Delta t) \\ \frac{v}{\omega}\cos\theta' - \frac{v}{\omega}(\cos\theta'\cos\omega\Delta t - \sin\theta'\sin\omega\Delta t) \\ \omega\Delta t \end{bmatrix}. \tag{22.154}$$

If $\omega\Delta t \ll 1$ then[14]

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \approx \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} + \begin{bmatrix} -\frac{v}{\omega}\sin\theta' + \frac{v}{\omega}(\sin\theta' + \omega\Delta t\cos\theta') \\ \frac{v}{\omega}\cos\theta' - \frac{v}{\omega}(\cos\theta' - \omega\Delta t\sin\theta') \\ \omega\Delta t \end{bmatrix}, \tag{22.155}$$

and so

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \approx \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} + \begin{bmatrix} v\Delta t\cos\theta' \\ v\Delta t\sin\theta' \\ \omega\Delta t \end{bmatrix}. \tag{22.156}$$

This is now a linear model[15] given θ',

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \approx \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} + \begin{bmatrix} \Delta t\cos\theta' & 0 \\ \Delta t\sin\theta' & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{22.157}$$

Another thing to note is this motion model assumes the speeds can instantaneously jump between time-steps. It also assumes that the robot velocity changes without delay. In practice, the effect of the velocity jump is reduced by using a smaller time-step Δt.

22.12.3 *Inverse velocity model*

Estimates of the effective speeds given the difference between two poses can be found naïvely[16] from

[14] Using $\sin\alpha \approx \alpha$ and $\cos\alpha \approx 1$ when $\alpha \ll 1$.

[15] It assumes the robot has followed a straight path instead of a curved path.

[16] There are more accurate but more complicated inverse models. This version fails when $\hat{\omega} = 0$. Also note, some pose changes cannot be represented by a parameterisation in terms of v and ω.

$$\hat{w} = \frac{\theta - \theta'}{\Delta t}, \tag{22.158}$$

$$\hat{v} = \frac{\hat{w}d}{2\tan\left(\frac{\hat{\omega}\Delta t}{2}\right)}, \tag{22.159}$$

where

$$d = \sqrt{(y-y')^2 + (x-x')^2}. \tag{22.160}$$

22.12.4 Odometry model pose change parameterisation

The odometry model pose change parameterisation denoted by $\mathcal{D}\{\mathbf{x}, \mathbf{x}'\}$ is given by[17]

$$\phi_1 = \tan^{-1}\frac{y-y'}{x-x'} - \theta', \tag{22.161}$$

$$d = \sqrt{(y-y')^2 + (x-x')^2}, \tag{22.162}$$

$$\phi_2 = \theta - \theta' - \phi_1. \tag{22.163}$$

[17] Note, when computing the inverse tan use the arctan2 function since the arctan function only gives an answer in the first quadrant.

Note ϕ_1 depends on the previous heading angle θ'.

Inversely, given a pose change parameterisation $(d, \phi_1, \phi_2)^{\mathrm{T}}$ and a previous pose $(x', y', \theta')^{\mathrm{T}}$, the predicted pose is found from

$$x = x' + d\cos(\theta' + \phi_1), \tag{22.164}$$

$$y = y' + d\sin(\theta' + \phi_1), \tag{22.165}$$

$$\theta = \theta' + \phi_1 + \phi_2. \tag{22.166}$$

22.13 Inverse motion models

Inverse motion models are required for histogram filters.

22.13.1 Inverse velocity motion model

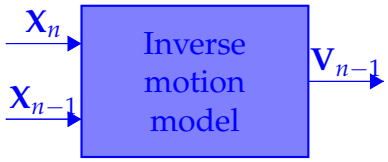


Figure 22.1: Block diagram for how an inverse motion model is used.

In the predict step, Bayes filters[18] require the condi-

[18] This is implicit for Kalman filters (they just require the variance). Particle filters avoid the integration by sampling the motion model.

tional PDF, $f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1};\mathbf{u}_{n-1})$, since

$$f_{\hat{\mathbf{X}}^-n}(\mathbf{x}_n) = \int f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1};\mathbf{u}_{n-1}) f_{\hat{\mathbf{x}}^+n-1}(\mathbf{x}_{n-1}) \mathrm{d}\mathbf{x}_{n-1}. \tag{22.167}$$

To evaluate $f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1};\mathbf{u}_{n-1})$ we need an inverse motion model, where the estimated speed vector for the pose to change from $\mathbf{x}_{n-1}$ to $\mathbf{x}_n$ is found using

$$\mathbf{v}_{n-1} = g^{-1}(\mathbf{x}_n, \mathbf{x}_{n-1}), \tag{22.168}$$

where g^{-1} is the inverse of (13.6) and $\mathbf{v} = (v, \omega)^{\mathrm{T}}$. Then we can find the transition probability from the joint PDF for the speeds:

$$f_{\mathbf{X}_n|\mathbf{X}_{n-1}}(\mathbf{x}_n|\mathbf{x}_{n-1};\mathbf{u}_{n-1}) = f_{V,\Omega}(v, \omega; \mathbf{v}_{n-1}). \tag{22.169}$$

22.13.2 *Inverse odometry motion model*

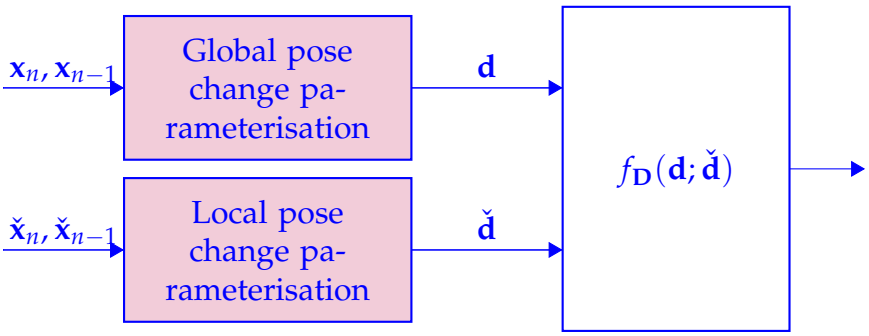


Figure 22.2: Block diagram showing how the PDFs are calculated for the inverse odometry motion model.

The inverse odometry motion model is required to determine the belief of a pose, $\mathbf{x}_n$, given a previous pose, $\mathbf{x}_{n-1}$, and the change of local pose measured from odometry. This is obtained from (14.10).

22.14 *Mapping algorithm derivation*

Using Bayes' theorem, the probability that a map cell is occupied given a measurement is

$$P(M|Z) = \frac{P(Z|M)P(M)}{P(Z)}, \tag{22.170}$$

and the probability that a map cell is free given a measurement is

$$P(\overline{M}|Z) = \frac{P(Z|\overline{M})P(\overline{M})}{P(Z)}. \tag{22.171}$$

Using odds,

$$O(M) = \frac{P(M)}{P(\overline{M})} \tag{22.172}$$

and so Bayes' theorem for odds is

$$\begin{aligned} O(M|Z) &= \frac{P(M|Z)}{P(\overline{M}|Z)} = \frac{P(Z|M)P(M)}{P(Z|\overline{M})P(\overline{M})}, & (22.173)\\ &= \lambda(Z|M)O(M), & (22.174)\end{aligned}$$

where

$$\lambda(Z|M) = \frac{P(Z|M)}{P(Z|\overline{M})}. \tag{22.175}$$

Using log odds,

$$\log O(M|Z) = \log \lambda(Z|M) + \log O(M). \tag{22.176}$$

There are three cases to consider, where R is the measured range and D is the distance to the map grid cell of interest:

1. If the measured range is less than the map cell range then we cannot infer anything,

$$\log \lambda(R < D|M) = \log 1 = 0. \tag{22.177}$$

2. If the measured range is the same as map cell range, then it is likely that the map cell is occupied. If we assume that $P(R = D|M) = 0.06$ and $P(R = D|\overline{M}) = 0.005$, then

$$\log \lambda(R < D|M) = \log \frac{0.06}{0.005} = \log 12 = 2.5. \tag{22.178}$$

3. If the measured range is further than the map cell range, then it is likely that the map cell is free. If we assume that $P(R > D|M) = 0.2$ and $P(R > D|\overline{M}) = 0.9$, then

$$\log \lambda(R < D|M) = \log \frac{0.2}{0.9} = \log 0.22 = -1.5. \tag{22.179}$$

22.15 *Bayes filter for SLAM*

The Bayes filter for the SLAM problem[19] is:

$$\begin{aligned} p(\mathbf{x}_n, \mathbf{m}|\mathbf{z}_{0:n}, \mathbf{u}_{0:n}) &= \eta p(\mathbf{z}_n|\mathbf{x}_n, \mathbf{m}) \\ &\times \int p(\mathbf{x}_n|\mathbf{u}_n, \mathbf{x}_{n-1}) p(\mathbf{x}_{n-1}, \mathbf{m}|\mathbf{z}_{0:n-1}, \mathbf{u}_{0:n-1}) \mathrm{d}\mathbf{x}_{n-1}. \end{aligned} \tag{22.180}$$

[19] This formulation combines the predict and update steps and assumes dense sensor measurements and not features. For features, replace $\mathbf{z}_{0:n}$ with $f(\mathbf{z}_{0:n})$ and add a correspondence vector history, $\mathbf{c}_{0:n}$.

The complete state assumption simplifies the Bayes filter to

$$p(\mathbf{x}_n, \mathbf{m}|\mathbf{z}_{0:n}, \mathbf{u}_{0:n}) = \eta p(\mathbf{z}_n|\mathbf{x}_n, \mathbf{m}) \int p(\mathbf{x}_n|\mathbf{u}_n, \mathbf{x}_{n-1}, \mathbf{m}) \mathrm{d}\mathbf{x}_{n-1}. \tag{22.181}$$

However, even with this approximation, the posterior is intractable since it involves a multidimensional probability density over a continuous space.

The PDF of the online SLAM estimate can be obtained from the PDF of the full SLAM estimate by integrating over past poses,

$$p(\mathbf{x}_n, \mathbf{m}|\mathbf{z}_{0:n}, \mathbf{u}_{0:n}) = \int \int \cdots \int p(\mathbf{x}_{0:n}, \mathbf{m}|\mathbf{z}_{0:n}, \mathbf{u}_{0:n}) \mathrm{d}\mathbf{x}_0 \mathrm{d}\mathbf{x}_1 \cdots \mathrm{d}\mathbf{x}_{n-1}. \tag{22.182}$$