Chapter

6 Localization

in order to get somewhere we need to know where we are

In our discussion of map-based navigation we assumed that the robot had a means of knowing its position. In this chapter we discuss some of the common techniques used to estimate the location of a robot in the world – a process known as localization.

Today GPS makes outdoor localization so easy that we often take this capability for granted. Unfortunately GPS is a far from perfect sensor since it relies on very weak radio signals received from distant orbiting satellites. This means that GPS cannot work where there is no *line of sight* radio reception, for instance indoors, underwater, underground, in urban canyons or in deep mining pits. GPS signals are also extremely weak and can be easily jammed and this is not acceptable for some applications.

GPS has only been in use since 1995 yet humankind has been navigating the planet and localizing for many thousands of years. In this chapter we will introduce the *classical* navigation principles such as dead reckoning and the use of landmarks on which modern robotic navigation is founded.

Dead reckoning is the estimation of location based on estimated speed, direction and time of travel with respect to a previous estimate. Figure 6.1 shows how a ship's position is updated on a chart. Given the average compass heading over the previous hour and a distance traveled the position at 3 P.M. can be found using elementary geometry from the position at 2 P.M. However the measurements on which the update is based are subject to both systematic and random error. Modern instruments are quite precise but 500 years ago clocks, compasses and speed measurement were primitive. The recursive nature of the process, each estimate is based on the previous

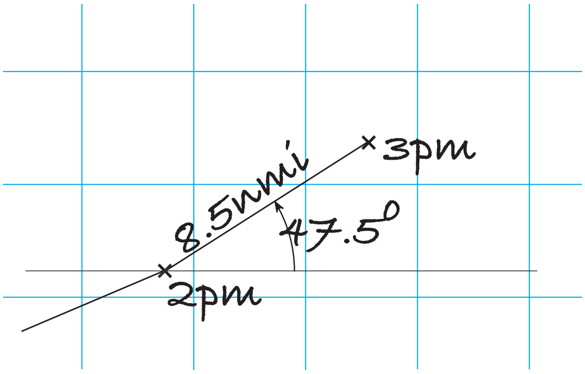
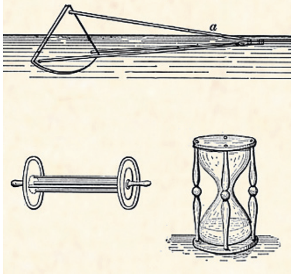


Fig. 6.1. Location estimation by dead reckoning. The ship's position at 3 P.M. is based on its position at 2 P.M., the estimated distance traveled since, and the average compass heading



Measuring speed at sea. A ship's log is an instrument that provides an estimate of the distance traveled. The oldest method of determining the speed of a ship at sea was the Dutchman's log – a floating object was thrown into the water at the ship's bow and the time for it to pass the stern was measured using an hourglass. Later came the chip log, a flat quarter-circle of wood with a lead weight on the circular side causing it to float upright and resist towing. It was tossed overboard and a line with knots at 50 foot intervals was payed out. A special hourglass, called a log glass, ran for 30 s, and each knot on the line over that interval corresponds to approximately 1 nmi h^{-1} or 1 knot. A nautical mile (nmi) is now defined as 1.852 km. (Image modified from Text-Book of Seamanship, Commodore S. B. Luce 1891)

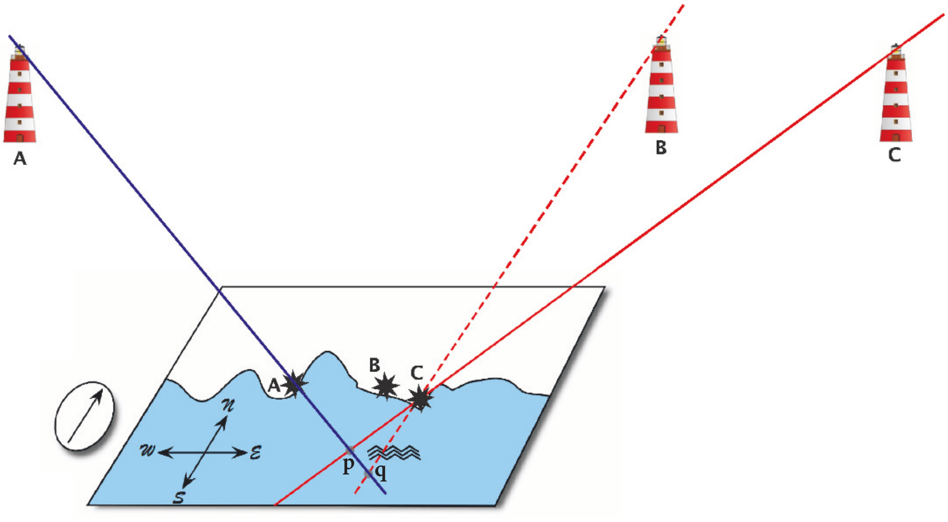


Fig. 6.2. Location estimation using a map. Lines of sight from two light-houses, ***A*** and ***C***, and their corresponding locations on the map provide an estimate ***p*** of our location. However if we mistake lighthouse ***B*** for ***C*** then we obtain an incorrect estimate ***q***

one, means that errors will accumulate over time and for sea voyages of many-years this approach was quite inadequate.

The Phoenicians were navigating at sea more than 4 000 years ago and they did not even have a compass – that was developed 2 000 years later in China. The Phoenicians navigated with crude dead reckoning but wherever possible they used *additional information* to correct their position estimate – sightings of islands and headlands, primitive maps and observations of the Sun and the Pole Star.

A landmark is a visible feature in the environment whose location is known with respect to some coordinate frame. Figure 6.2 shows schematically a map and a number of lighthouse landmarks. We first of all use a compass to align the north axis of our map with the direction of the north pole. The direction of a single landmark constrains our position to lie along a line on the map. Sighting a second landmark places our position on another constraint line, and our position must be at their intersection – a process known as resectioning.▸ For example lighthouse **A** constrains us to lie along the blue line. Lighthouse **C** constrains us to lie along the red line and the intersection is our true position **p**.

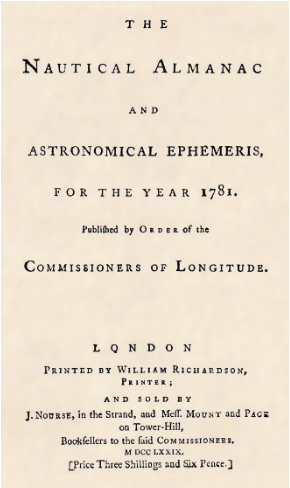
Resectioning is the estimation of position by measuring the bearing angles to known landmarks. Triangulation is the estimation of position by measuring the bearing angles to the unknown point from each of the landmarks.

However this process is critically reliant on correctly associating the observed landmark with the feature on the map. If we mistake one lighthouse for another, for example we see **B** but think it is **C** on the map, then the red dashed line leads to a

Celestial navigation. The position of celestial bodies in the sky is a predictable function of the time and the observer's latitude and longitude. This information can be tabulated and is known as ephemeris (meaning daily) and such data has been published annually in Britain since 1767 as the "*The Nautical Almanac*" by HM Nautical Almanac Office. The elevation of a celestial body with respect to the horizon can be measured using a sextant, a handheld optical instrument.

Time and longitude are coupled, the star field one hour later is the same as the star field 15° to the east. However the northern Pole Star, *Polaris* or the *North Star*, is very close to the celestial pole and its elevation angle is independent of longitude and time, allowing latitude to be determined very conveniently from a single sextant measuremernt.

Solving the longitude problem was the greatest scientific challenge to European governments in the eighteenth century since it was a significant impediment to global navigation and maritime supremacy. The British Longitude Act of 1714 created a prize of £20 000 which spurred the development of nautical chronometers, clocks that could maintain high accuracy onboard ships. More than fifty years later a suitable chronometer was developed by John Harrison, a copy of which was used by Captain James Cook on his second voyage of 1772–1775. After a three year journey the error in estimated longitude was just 13 km. With accurate knowledge of time, the elevation angle of stars could be used to estimate latitude and longitude. This technological advance enabled global exploration and trade. (Image courtesy archive.org)


THE

NAUTICAL ALMANAC

AND

ASTRONOMICAL EPHEMERIS,

FOR THE YEAR 1781.

Publiſhed by ORDER of the

COMMISSIONERS OF LONGITUDE.

LONDON

PRINTED BY WILLIAM RICHARDSON,
PRINTER;
AND SOLD BY
J. NOURSE, in the Strand, and Meſſ. MOUNT and PAGE
on Tower-Hill,
Bookſellers to the ſaid COMMISSIONERS.
M DCC LXXIX.
[Price Three Shillings and Six Pence.]

Radio-based localization. One of the earliest systems was LORAN, based on the British World War II GEE system. LORAN transmitters around the world emit synchronized radio pulses and a receiver measures the difference in arrival time between pulses from a pair of radio transmitters. Knowing the identity of two transmitters and the time difference (TD) constrains the receiver to lie along a hyperbolic curve shown on navigation charts as *TD lines*. Using a second pair of transmitters (which may include one of the first pair) gives another hyperbolic constraint curve, and the receiver must lie at the intersection of the two curves.

The Global Positioning System (GPS) was proposed in 1973 but did not become fully operational until 1995. It comprises around 30 active satellites orbiting the Earth in six planes at a distance of 20 200 km. A GPS receiver works by measuring the time of travel of radio signals from four or more satellites whose orbital position is encoded in the GPS signal. With four known points in space and four measured time delays it is possible to compute the (x, y, z) position of the receiver and the time. If the GPS signals are received after reflecting off some surface the distance traveled is longer and this will introduce an error in the position estimate. This effect is known as multi-pathing and is a common problem in large-scale industrial facilities.

Variations in the propagation speed of radio waves through the atmosphere is the main cause of error in the position estimate. However these errors vary slowly with time and are approximately constant over large areas. This allows the error to be measured at a reference station and transmitted as an augmentation to compatible nearby receivers which can offset the error – this is known as Differential GPS (DGPS). This information can be transmitted via the internet, via coastal radio networks to ships, or by satellite networks such as WAAS, EGNOS or OmniSTAR to aircraft or other users. RTK GPS achieves much higher precision in time measurement by using phase information from the carrier signal. The original GPS system deliberately added error, euphemistically termed selective availability, to reduce its utility to military opponents but this *feature* was disabled in May 2000. Other satellite navigation systems include the Russian GLONASS, the European Galileo, and the Chinese Beidou.

significant error in estimated position – we would believe we were at **q** instead of **p**. This belief would lead us to overestimate our distance from the coastline. If we decided to sail toward the coast we would run aground on rocks and be surprised since they were not where we expected them to be. This is unfortunately a very common error and countless ships have foundered because of this fundamental data association error. This is why lighthouses flash! In the eighteenth century technological advances enabled lighthouses to emit unique flashing patterns so that the identity of the particular lighthouse could be reliably determined and associated with a point on a navigation chart.

Of course for the earliest mariners there were no maps, or lighthouses or even compasses. They had to create maps as they navigated by incrementally adding new nonmanmade features to their maps just beyond the boundaries of what was already known. It is perhaps not surprising that so many early explorers came to grief◂ and that maps were tightly kept state secrets.

Magellan's 1519 expedition started with 237 men and 5 ships but most, including Magellan, were lost along the way. Only 18 men and 1 ship returned.

Robots operating today in environments without GPS face *exactly* the same problems as ancient navigators and, perhaps surprisingly, borrow heavily from navigational strategies that are centuries old. A robot's estimate of distance traveled will be imperfect and it may have no map, or perhaps an imperfect or incomplete map. Additional information from observed features in the world is critical to minimizing a robot's localization error but the possibility of data association error remains.

We can define the localization problem more formally where $\boldsymbol{x}$ is the true, but unknown, position of the robot and $\hat{\boldsymbol{x}}$ is our best estimate of that position. We also wish to know the *uncertainty* of the estimate which we can consider in statistical terms as the standard deviation associated with the position estimate $\hat{\boldsymbol{x}}$.

It is useful to describe the robot's estimated position in terms of a probability density function (PDF) over all possible positions of the robot.◂ Some example PDFs are shown in Fig. 6.3 where the magnitude of the function at any point is the relative likelihood of the vehicle being at that position. Commonly a Gaussian function is used which can be described succinctly in terms of its mean and standard deviation. The robot is most likely to be at the location of the peak (the mean) and increasingly less likely to be at positions further away from the peak. Figure 6.3a shows a peak with a small standard deviation which indicates that the vehicle's position is very well known. There is an almost zero probability that the vehicle is at the point indicated by the vertical black line. In contrast the peak in Fig. 6.3b has a large standard deviation which means that we are less certain about the location of the vehicle. There is a reasonable probability that the vehicle is at the point indicated by the vertical line.

For robot pose (x, y, θ) the PDF is a 4-dimensional surface.

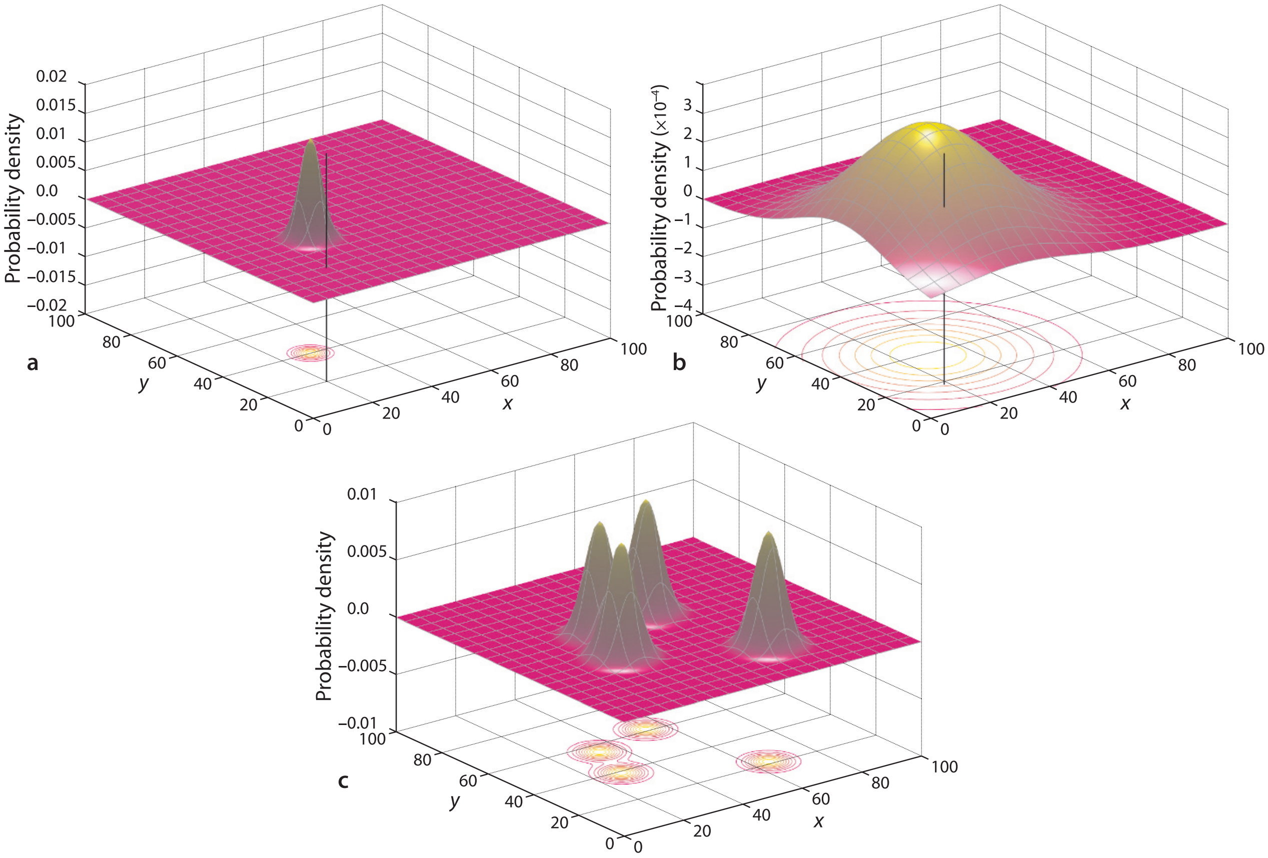


Fig. 6.3. Notions of vehicle position and uncertainty in the xy-plane, where the vertical axis is the relative likelihood of the vehicle being at that position, sometimes referred to as belief or bel(x). Contour lines are displayed on the lower plane. **a** The vehicle has low position uncertainty, $\sigma = 1$; **b** the vehicle has much higher position uncertainty, $\sigma = 20$; **c** the vehicle has multiple hypotheses for its position, each $\sigma = 1$

Using a PDF also allows for multiple hypotheses about the robot's position. For example the PDF of Fig. 6.3c describes a robot that is quite certain that it is at one of four places. This is more useful than it seems at face value. Consider an indoor robot that has observed a vending machine and there are four such machines marked on the map. In the absence of any other information the robot must be equally likely to be in the vicinity of *any* of the four vending machines. We will revisit this approach in Sect. 6.7.

Determining the PDF based on knowledge of how the vehicle moves and its observations of the world is a problem in estimation which we can define as:

> *the process of inferring the value of some quantity of interest, x, by processing data that is in some way dependent on x.*

For example a ship's navigator or a surveyor estimates location by measuring the bearing angles to known landmarks or celestial objects, and a GPS receiver estimates latitude and longitude by observing the time delay from moving satellites whose locations are known.

For our robot localization problem the true and estimated state are vector quantities so uncertainty will be represented as a covariance matrix, see Appendix G. The diagonal elements represent uncertainty of the corresponding states, and the off-diagonal elements represent correlations between states.

> The value of a PDF is *not* the probability of being at that location. Consider a small region of the *xy*-plane, the volume under that region of the PDF is the probability of being in that region.

6.1 Dead Reckoning

Dead reckoning is the estimation of a robot's pose based on its estimated speed, direction and time of travel with respect to a previous estimate.

An odometer is a sensor that measures distance traveled and sometimes also change in heading direction. For wheeled vehicles this can be determined by measuring the angular rotation of the wheels. The direction of travel can be measured using an electronic compass, or the change in heading can be measured using a gyroscope or differential odometry.◂ These sensors are imperfect due to systematic errors such an incorrect wheel radius or gyroscope bias, and random errors such as slip between wheels and the ground. Robots without wheels, such as aerial and underwater robots, can use visual odometry – a computer vision approach based on observations of the world moving past the robot which is discussed in Sect. 14.7.4.

Measuring the difference in angular velocity of a left- and right-hand side wheel.

6.1.1 Modeling the Vehicle

The first step in estimating the robot's pose is to write a function, $\boldsymbol{f}(\cdot)$, that describes how the vehicle's configuration changes from one time step to the next. A vehicle model such as Eq. 4.2 or 4.4 describes the evolution of the robot's configuration as a function of its control inputs, however for real robots we rarely have access to these control inputs. Most robotic platforms have proprietary motion control systems that accept motion commands from the user (speed and direction) and report odometry information.

Instead of using Eq. 4.2 or 4.4 directly we will write a discrete-time model for the evolution of configuration based on odometry where $\delta\langle k\rangle = (\delta_d, \delta_\theta)$ is the distance traveled and change in heading over the preceding interval, and k is the time step. The initial pose is represented in $\mathbf{SE}(2)$ as

$$\xi\langle k\rangle \sim \begin{pmatrix} \cos\theta\langle k\rangle & -\sin\theta\langle k\rangle & x\langle k\rangle \\ \sin\theta\langle k\rangle & \cos\theta\langle k\rangle & y\langle k\rangle \\ 0 & 0 & 1 \end{pmatrix}$$

We make a simplifying assumption that motion over the time interval is *small* so the order of applying the displacements is not significant. We choose to move forward in the vehicle x-direction by δ_d, and then rotate by δ_θ giving the new configuration

$$\begin{aligned} \xi\langle k{+}1\rangle &\sim \begin{pmatrix} \cos\theta\langle k\rangle & -\sin\theta\langle k\rangle & x\langle k\rangle \\ \sin\theta\langle k\rangle & \cos\theta\langle k\rangle & y\langle k\rangle \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \delta_d \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\delta_\theta & -\sin\delta_\theta & 0 \\ \sin\delta_\theta & \cos\delta_\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ &\sim \begin{pmatrix} \cos(\theta\langle k\rangle + \delta_\theta) & -\sin(\theta\langle k\rangle + \delta_\theta) & x\langle k\rangle + \delta_d\cos\theta\langle k\rangle \\ \sin(\theta\langle k\rangle + \delta_\theta) & \cos(\theta\langle k\rangle + \delta_\theta) & y\langle k\rangle + \delta_d\sin\theta\langle k\rangle \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

which we can represent concisely as a 3-vector $\boldsymbol{x} = (x, y, \theta)$

$$x\langle k{+}1\rangle = \begin{pmatrix} x\langle k\rangle + \delta_d\cos\theta\langle k\rangle \\ y\langle k\rangle + \delta_d\sin\theta\langle k\rangle \\ \theta\langle k\rangle + \delta_\theta \end{pmatrix} \qquad (6.1)$$

which gives the new configuration in terms of the previous configuration and the odometry.

In practice odometry is not perfect and we model the error by imagining a random number generator that corrupts the output of a perfect odometer. The measured output of the real odometer is the perfect, but unknown, odometry $(\delta_d, \delta_\theta)$ plus the output of the random number generator (v_d, v_θ). Such random errors are often referred to as noise, or more specifically as sensor noise. The random numbers are not known and cannot be measured, but we assume that we know the distribution from which they are drawn.

The robot's configuration at the next time step, including the odometry error, is

$$\boldsymbol{x}\langle k{+}1\rangle = \boldsymbol{f}\big(\boldsymbol{x}\langle k\rangle, \delta\langle k\rangle, \boldsymbol{v}\langle k\rangle\big) = \begin{pmatrix} x\langle k\rangle + \big(\delta_d + v_d\big)\cos\theta\langle k\rangle \\ y\langle k\rangle + \big(\delta_d + v_d\big)\sin\theta\langle k\rangle \\ \theta\langle k\rangle + \delta_\theta + v_\theta \end{pmatrix} \tag{6.2}$$

where k is the time step, $\delta\langle k\rangle = (\delta_d, \delta_\theta)^T \in \mathbb{R}^{2\times1}$ is the odometry measurement and $\boldsymbol{v}\langle k\rangle = (v_d, v_\theta)^T \in \mathbb{R}^{2\times1}$ is the random measurement noise over the preceding interval.

The odometry noise is *inside* the model of our process (vehicle motion) and is referred to as process noise.

In the absence of any information to the contrary we model the odometry noise as $\boldsymbol{v} = (v_d, v_\theta)^T \sim N(0, \boldsymbol{V})$, a zero-mean multivariate Gaussian process with variance

A normal distribution of angles on a circle is actually not possible since $\theta \in \mathbb{S}^1 \notin \mathbb{R}$, that is angles wrap around 2π. However if the covariance for angular states is small this problem is minimal. A normal-like distribution of angles on a circle is the von Mises distribution.

$$\boldsymbol{V} = \begin{pmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix}$$

This constant matrix, the covariance matrix, is diagonal which means that the errors in distance and heading are *independent*. Choosing a value for $\boldsymbol{V}$ is not always easy but we can conduct experiments or make some reasonable engineering assumptions. In the examples which follow, we choose $\sigma_d = 2$ cm and $\sigma_\theta = 0.5°$ per sample interval which leads to a covariance matrix of

In reality this is unlikely to be the case since odometry distance errors tend to be worse when change of heading is high.

```
>> V = diag([0.02, 0.5*pi/180].^2);
```

All objects of the Toolbox `Vehicle` superclass provide a method `f()` that implements the appropriate odometry update equation. For the case of a vehicle with bicycle kinematics that has the motion model of Eq. 4.2 and the odometric update Eq. 6.2, we create a `Bicycle` object

```
>> veh= Bicycle('covar', V)
veh =
Bicycle object
  L=1
  Superclass: Vehicle
    max speed=1, max steer input=0.5, dT=0.1, nhist=0
    V=(0.0004, 7.61544e-05)
    configuration: x=0, y=0, theta=0
```

which shows the default parameters such as the vehicle's length, speed, steering limit and the sample interval which defaults to 0.1 s. The object provides a method to simulate motion over one time step

```
>> odo = veh.step(1, 0.3)
odo =
    0.1108    0.0469
```

where we have specified a speed of 1 m s^{-1} and a steering angle of 0.3 rad. The function updates the robot's true configuration and returns a noise corrupted odometer reading. With a sample interval of 0.1 s the robot reports that is moving approximately 0.1 m each interval and changing its heading by approximately 0.03 rad. The robot's true (but 'unknown') configuration can be seen by

We simulate the odometry noise using MATLAB generated random numbers that have zero-mean and a covariance given by the diagonal of `V`. The random noise means that repeated calls to this function will return different values.

```
>> veh.x'
ans =
    0.1000         0    0.0309
```

Given the reported odometry we can estimate the configuration of the robot after one time step using Eq. 6.2 which is implemented by

```
>> veh.f([0 0 0], odo)
ans =
    0.1106    0.0052    0.0469
```

where the discrepancy with the exact value is due to the use of a noisy odometry measurement.

For the scenarios that we want to investigate we require the simulated robot to drive for a long time period within a defined spatial region. The `RandomPath` class is a *driver* that steers the robot to randomly selected waypoints within a specified region. We create an instance of the driver object and connect it to the robot

```
>> veh.add_driver( RandomPath(10) )
```

where the argument to the `RandomPath` constructor specifies a working region that spans ±10 m in the x- and y-directions. We can display an animation of the robot with its driver by

```
>> veh.run()
```

which repeatedly calls the `step` method and maintains a history of the true state of the vehicle over the course of the simulation within the `Bicycle` object.◂ The `RandomPath` and `Bicycle` classes have many parameters and methods which are described in the online documentation.

The number of history records is indicated by `nhist=` in the displayed value of the object. The `hist` property is an array of structures that hold the vehicle state at each time step.

6.1.2 Estimating Pose

The problem we face, just like the ship's navigator, is how to estimate our new pose given the previous pose and noisy odometry. We want the best estimate of where we are and how certain we are about that. The mathematical tool that we will use is the Kalman filter which is described more completely in Appendix H. This filter provides the optimal estimate of the system state, vehicle configuration in this case, assuming that the noise is zero-mean and Gaussian. The filter is a recursive algorithm that updates, at each time step, the optimal estimate of the unknown true configuration and the uncertainty associated with that estimate based on the previous estimate and noisy measurement data. The Kalman filter is formulated for linear systems but our model of the vehicle's motion Eq. 6.2 is nonlinear – the tool of choice is the extended Kalman filter (EKF).

For this problem the state vector is the vehicle's configuration

$$\boldsymbol{x} = \left(x_v, y_v, \theta_v\right)^T$$

and the prediction equations◂

The Kalman filter, Fig. 6.6, has two steps: prediction based on the model and update based on sensor data. In this dead-reckoning case we use only the prediction equation.

$$\hat{\boldsymbol{x}}^{+}\langle k{+}1\rangle = \boldsymbol{f}\left(\hat{\boldsymbol{x}}\langle k\rangle, \hat{\boldsymbol{u}}\langle k\rangle\right) \tag{6.3}$$

$$\hat{\boldsymbol{P}}^{+}\langle k{+}1\rangle = \boldsymbol{F}_x \hat{\boldsymbol{P}}\langle k\rangle \boldsymbol{F}_x^T + \boldsymbol{F}_v \hat{\boldsymbol{V}} \boldsymbol{F}_v^T \tag{6.4}$$

describe how the state and covariance evolve with time. The term $\hat{\boldsymbol{x}}^{+}\langle k{+}1\rangle$ indicates an estimate of $\boldsymbol{x}$ at time $k+1$ based on information up to, and including, time k. $\hat{\boldsymbol{u}}\langle k\rangle$ is the

Reverend Thomas Bayes (1702–1761) was a nonconformist Presbyterian minister. He studied logic and theology at the University of Edinburgh and lived and worked in Tunbridge-Wells in Kent. There, through his association with the 2nd Earl Stanhope he became interested in mathematics and was elected to the Royal Society in 1742. After his death his friend Richard Price edited and published his work in 1763 as *An Essay towards solving a Problem in the Doctrine of Chances* which contains a statement of a special case of Bayes' theorem. Bayes is buried in Bunhill Fields Cemetery in London.

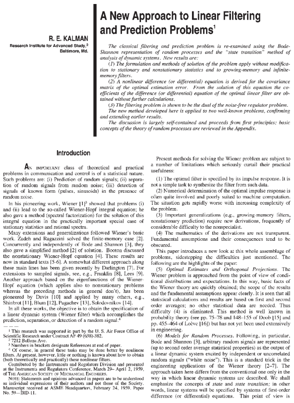
Bayes' theorem shows the relation between a conditional probability and its inverse: the probability of a hypothesis given observed evidence and the probability of that evidence given the hypothesis. Consider the hypothesis that the robot is at location X and it makes a sensor observation S of a known landmark. The *posterior* probability that the robot is at X given the observation S is

$$P(X|S) = \frac{P(S|X)P(X)}{P(S)}$$

where $P(X)$ is the *prior* probability that the robot is at X (not accounting for any sensory information), $P(S\,|\,X)$ is the likelihood of the sensor observation S given that the robot is at X, and $P(S)$ is the prior probability of the observation S. The Kalman filter, and the Monte-Carlo estimator we discuss later in this chapter, are essentially two different approaches to solving this inverse problem.

Rudolf Kálmán (1930–2016) was a mathematical system theorist born in Budapest. He obtained his bachelors and masters degrees in electrical engineering from MIT, and Ph.D. in 1957 from Columbia University. He worked as a Research Mathematician at the Research Institute for Advanced Study, in Baltimore, from 1958–1964 where he developed his ideas on estimation. These were met with some skepticism among his peers and he chose a mechanical (rather than electrical) engineering journal for his paper *A new approach to linear filtering and prediction problems* because "When you fear stepping on hallowed ground with entrenched interests, it is best to go sideways". He has received many awards including the IEEE Medal of Honor, the Kyoto Prize and the Charles Stark Draper Prize.

Stanley F. Schmidt (1926–2015) was a research scientist who worked at NASA Ames Research Center and was an early advocate of the Kalman filter. He developed the first implementation as well as the nonlinear version now known as the extended Kalman filter. This led to its incorporation in the Apollo navigation computer for trajectory estimation. (Extract from Kálmán's famous paper (1960) on the right reprinted with permission of ASME)



A New Approach to Linear Filtering and Prediction Problems[1]

R. E. KALMAN
Research Institute for Advanced Study,[2]
Baltimore, Md.

The classical filtering and prediction problem is re-examined using the Bode-Shannon representation of random processes and the "state transition" method of analysis of dynamic systems. New results are:

(1) The formulation and methods of solution of the problem apply without modification to stationary and nonstationary statistics and to growing-memory and infinite-memory filters.

(2) A nonlinear difference (or differential) equation is derived for the covariance matrix of the optimal estimation error. From the solution of this equation the coefficients of the difference (or differential) equation of the optimal linear filter are obtained without further calculations.

(3) The filtering problem is shown to be the dual of the noise-free regulator problem.

The new method developed here is applied to two well-known problems, confirming and extending earlier results.

The discussion is largely self-contained and proceeds from first principles; basic concepts of the theory of random processes are reviewed in the Appendix.

Introduction

An important class of theoretical and practical problems in communication and control is of a statistical nature. Such problems are: (i) Prediction of random signals; (ii) separation of random signals from random noise; (iii) detection of signals of known form (pulses, sinusoids) in the presence of random noise.

In his pioneering work, Wiener [1][3] showed that problems (i) and (ii) lead to the so-called Wiener-Hopf integral equation; he also gave a method (spectral factorization) for the solution of this integral equation in the practically important special case of stationary statistics and rational spectra.

Many extensions and generalizations followed Wiener's basic work. Zadeh and Ragazzini solved the finite-memory case [2]. Concurrently and independently of Bode and Shannon [3], they also gave a simplified method [2] of solution. Booton discussed the nonstationary Wiener-Hopf equation [4]. These results are now in standard texts [5-6]. A somewhat different approach along these main lines has been given recently by Darlington [7]. For extensions to sampled signals, see, e.g., Franklin [8], Lees [9]. Another approach based on the eigenfunctions of the Wiener-Hopf equation (which applies also to nonstationary problems whereas the preceding methods in general don't), has been pioneered by Davis [10] and applied by many others, e.g., Shinbrot [11], Blum [12], Pugachev [13], Solodovnikov [14].

In all these works, the objective is to obtain the specification of a linear dynamic system (Wiener filter) which accomplishes the prediction, separation, or detection of a random signal.[4]

Present methods for solving the Wiener problem are subject to a number of limitations which seriously curtail their practical usefulness:

(1) The optimal filter is specified by its impulse response. It is not a simple task to synthesize the filter from such data.

(2) Numerical determination of the optimal impulse response is often quite involved and poorly suited to machine computation. The situation gets rapidly worse with increasing complexity of the problem.

(3) Important generalizations (e.g., growing-memory filters, nonstationary prediction) require new derivations, frequently of considerable difficulty to the nonspecialist.

(4) The mathematics of the derivations are not transparent. Fundamental assumptions and their consequences tend to be obscured.

This paper introduces a new look at this whole assemblage of problems, sidestepping the difficulties just mentioned. The following are the highlights of the paper:

(5) *Optimal Estimates and Orthogonal Projections.* The Wiener problem is approached from the point of view of conditional distributions and expectations. In this way, basic facts of the Wiener theory are quickly obtained; the scope of the results and the fundamental assumptions appear clearly. It is seen that all statistical calculations and results are based on first and second order averages; no other statistical data are needed. Thus difficulty (4) is eliminated. This method is well known in probability theory (see pp. 75–78 and 148–155 of Doob [15] and pp. 455–464 of Loève [16]) but has not yet been used extensively in engineering.

(6) *Models for Random Processes.* Following, in particular, Bode and Shannon [3], arbitrary random signals are represented (up to second order average statistical properties) as the output of a linear dynamic system excited by independent or uncorrelated random signals ("white noise"). This is a standard trick in the engineering applications of the Wiener theory [2–7]. The approach taken here differs from the conventional one only in the way in which linear dynamic systems are described. We shall emphasize the concepts of *state* and *state transition;* in other words, linear systems will be specified by systems of first-order difference (or differential) equations. This point of view is

[1] This research was supported in part by the U. S. Air Force Office of Scientific Research under Contract AF 49 (638)-382.

[2] 7212 Bellona Ave.

[3] Numbers in brackets designate References at end of paper.

[4] Of course, in general these tasks may be done better by nonlinear filters. At present, however, little or nothing is known about how to obtain (both theoretically and practically) these nonlinear filters.

Contributed by the Instruments and Regulators Division and presented at the Instruments and Regulators Conference, March 29– April 2, 1959, of The American Society of Mechanical Engineers.

Note: Statements and opinions advanced in papers are to be understood as individual expressions of their authors and not those of the Society. Manuscript received at ASME Headquarters, February 24, 1959. Paper No. 59—IRD-11.

input to the process, which in this case is the measured odometry, so $\hat{\boldsymbol{u}}\langle k\rangle = \delta\langle k\rangle$. $\hat{\boldsymbol{P}} \in \mathbb{R}^{3\times 3}$ is a covariance matrix representing uncertainty in the estimated vehicle configuration. $\hat{\boldsymbol{V}}$ is our estimate of the covariance of the odometry noise which in reality we do not know.

F_x and F_v are Jacobian matrices – the vector version of a derivative. They are obtained by differentiating Eq. 6.2 and evaluating the result at $\boldsymbol{v} = 0$ giving▸

Since the noise value cannot actually be measured we use the mean value which is zero.

$$F_x = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{v}=0} = \begin{pmatrix} 1 & 0 & -\delta_d \sin\theta_v \\ 0 & 1 & \delta_d \cos\theta_v \\ 0 & 0 & 1 \end{pmatrix} \tag{6.5}$$

$$F_v = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{v}}\right|_{\boldsymbol{v}=0} = \begin{pmatrix} \cos\theta_v & 0 \\ \sin\theta_v & 0 \\ 0 & 1 \end{pmatrix} \tag{6.6}$$

which are functions of the current state and odometry.▸ Jacobians are reviewed in Appendix E. All objects of the `Vehicle` superclass provide methods `Fx` and `Fv` to compute these Jacobians, for example

The time step notation $\langle k\rangle$ is dropped to reduce clutter.

```
>> veh.Fx( [0,0,0], [0.5, 0.1] )
ans =
    1.0000         0   -0.0499
         0    1.0000    0.4975
         0         0    1.0000
```

where the first argument is the state at which the Jacobian is computed and the second is the odometry.

To simulate the vehicle and the EKF using the Toolbox we define the initial covariance to be quite small since, we assume, we have a good idea of where we are to begin with

```
>> P0 = diag([0.005, 0.005, 0.001].^2);
```

and we pass this to the constructor for an `EKF` object

```
>> ekf = EKF(veh, V, P0);
```

Running the filter for 1 000 time steps

```
>> ekf.run(1000);
```

drives the robot as before, along a random path. At each time step the filter updates the state estimate using various methods provided by the `Vehicle` superclass.

We can plot the true path taken by the vehicle, stored within the `Vehicle` superclass object, by

```
>> veh.plot_xy()
```

and the filter's estimate of the path stored within the `EKF` object,

```
>> hold on
>> ekf.plot_xy('r')
```

These are shown in Fig. 6.4 and we see some divergence between the true and estimated robot path.

The covariance at the 700th time step is

```
>> P700 = ekf.history(700).P
P700 =
    1.8929   -0.5575   -0.1851
   -0.5575    3.4184    0.3400
   -0.1851    0.3400    0.0533
```

The matrix is symmetric and the diagonal elements are the estimated variance associated with the states, that is σ_x^2, σ_y^2 and σ_θ^2 respectively. The standard deviation σ_x of the PDF associated with the vehicle's x-coordinate is

```
>> sqrt(P700(1,1))
ans =
   1.3758
```

There is a 95% chance that the robot's x-coordinate is within the $\pm 2\sigma$ bound or ± 2.75 m in this case. We can compute uncertainty for y and θ similarly.

The off-diagonal terms are correlation coefficients and indicate that the uncertainties between the corresponding variables are related. For example the value $P_{1,3} = P_{3,1} = -0.5575$ indicates that the uncertainties in x and θ are related – error in heading angle causes error in x-position and vice versa. Conversely new information about θ can be used to correct θ as well as x. The uncertainty in position is described by the top-left 2×2 covariance submatrix of $\hat{\boldsymbol{P}}$. This can be interpreted as an ellipse defining a confidence bound on position. We can overlay such ellipses on the plot by

```
>> ekf.plot_ellipse('g')
```

as shown in Fig. 6.4. These correspond to the default 95% confidence bound and are plotted by default every 20 time steps. The vehicle started at the origin and as it progresses we see that the ellipses become larger as the estimated uncertainty increases. The ellipses only show x- and y-position but uncertainty in θ also grows.

The total uncertainty,◂ position and heading, is given by $\sqrt{\det(\hat{\boldsymbol{P}})}$ and is plotted as a function of time

The elements of $\boldsymbol{P}$ have different units: m^2 and rad^2. The uncertainty is therefore a mixture of spatial and angular uncertainty with an implicit weighting. If the range of the position variables $x, y \gg \pi$ then positional uncertainty dominates.

```
>> ekf.plot_P();
```

as shown in Fig. 6.5 and we observe that it never decreases. This is because the second term in Eq. 6.4 is positive definite which means that $\boldsymbol{P}$, the position uncertainty, can never decrease.

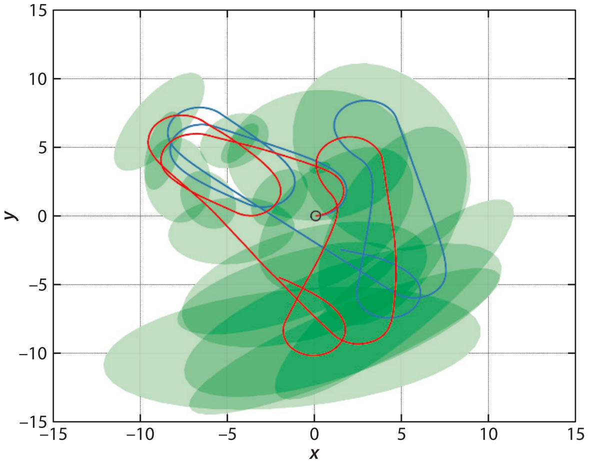


Fig. 6.4. Deadreckoning using the EKF. The true path of the robot, *blue*, and the path estimated from odometry in *red*. 95% confidence ellipses are indicated in *green*. The robot starts at the origin

Error ellipses. We consider the PDF of the robot's position (ignoring orientation) such as shown in Fig. 6.3 to be a 2-dimensional Gaussian probability density function

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)\det\left(\boldsymbol{P}_{xy}\right)^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\mu_x)^T \boldsymbol{P}_{xy}{}^{-1}(\boldsymbol{x}-\mu_x)\right\}$$

where $\boldsymbol{x} = (x, y)^T$ is the position of the robot, $\mu_x = (\hat{x}, \hat{y})^T$ is the estimated mean position and $\boldsymbol{P}_{xy} \in \mathbb{R}^{2\times2}$ is the position covariance matrix, the top left of the covariance matrix $\boldsymbol{P}$ computed by the Kalman filter. A horizontal cross-section is a contour of constant probability which is an ellipse defined by the points $\boldsymbol{x}$ such that

$$(\boldsymbol{x}-\mu_x)^T \boldsymbol{P}_{xy}{}^{-1}(\boldsymbol{x}-\mu_x) = s$$

Such error ellipses are often used to represent positional uncertainty as shown in Fig. 6.4. A large ellipse corresponds to a wider PDF peak and less certainty about position. To obtain a particular confidence contour (eg. 99%) we choose s as the inverse of the χ^2 cumulative distribution function for 2 degrees of freedom, in MATLAB that is `chi2inv(C, 2)` where $C \in [0, 1]$ is the confidence value. Such confidence values can be passed to several `EKF` methods when specifying error ellipses.

A handy scalar measure of total position uncertainty is the area of the ellipse $\pi r_1 r_2$ where the radii $r_i = \sqrt{\lambda_i}$ and λ_i are the eigenvalues of $\boldsymbol{P}_{xy}$. Since $\det(\boldsymbol{P}_{xy}) = \Pi\lambda_i$ the ellipse area – the scalar uncertainty – is proportional to $\sqrt{\det(\boldsymbol{P}_{xy})}$. See also Appendices C.1.4 and G.

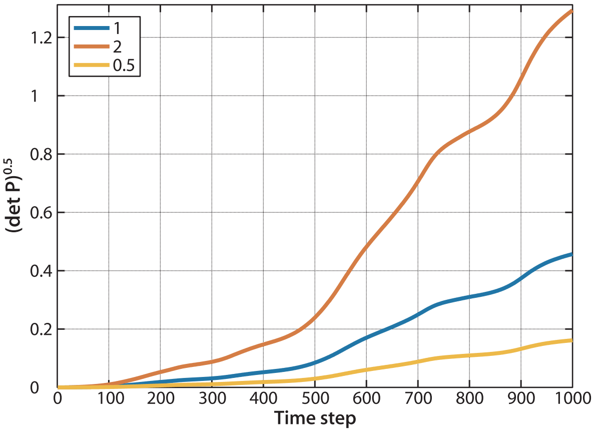


Fig. 6.5.
Overall uncertainty is given by $\sqrt{\det(\hat{\boldsymbol{P}})}$ which shows monotonically increasing uncertainty (*blue*). The effect of changing the magnitude of $\hat{\boldsymbol{V}}$ is to change the rate of uncertainty growth. Curves are shown for $\hat{\boldsymbol{V}} = \alpha\boldsymbol{V}$ where $\alpha = 1/2, 1, 2$

Note that we have used the odometry covariance matrix `V` twice. The first usage, in the `Vehicle` constructor, is the covariance $\boldsymbol{V}$ of the Gaussian noise source that is added to the true odometry to *simulate* odometry error in Eq. 6.2. In a real application this noise is generated by some physical process *hidden inside* the robot and we would not know its parameters. The second usage, in the `EKF` constructor, is $\hat{\boldsymbol{V}}$ which is our best *estimate* of the odometry covariance and is used in the filter's state covariance update equation Eq. 6.4.

The relative values of $\boldsymbol{V}$ and $\hat{\boldsymbol{V}}$ control the rate of uncertainty growth as shown in Fig. 6.5. If $\hat{\boldsymbol{V}} > \boldsymbol{V}$ then $\boldsymbol{P}$ will be larger than it should be and the filter is pessimistic – it overestimates uncertainty and is less certain than it should be. If $\hat{\boldsymbol{V}} < \boldsymbol{V}$ then $\boldsymbol{P}$ will be smaller than it should be and the filter will be *overconfident* of its estimate – the actual uncertainty is greater than the estimated uncertainty. In practice some experimentation is required to determine the appropriate value for the estimated covariance.

6.2 Localizing with a Map

We have seen how uncertainty in position grows without bound using dead-reckoning alone. The solution, as the Phoenicians worked out 4 000 years ago, is to bring in additional information from observations of known features in the world. In the examples that follow we will use a map that contains N fixed but randomly located landmarks whose positions are known.

The Toolbox supports a `LandmarkMap` object

```
>> map = LandmarkMap(20, 10)
```

that in this case contains $N = 20$ landmarks uniformly randomly spread over a region spanning ±10 m in the x- and y-directions and this can be displayed by

```
>> map.plot()
```

The robot is equipped with a sensor that provides observations of the landmarks *with respect to the robot* as described by

$$\boldsymbol{z} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{p}_i) \tag{6.7}$$

where $\boldsymbol{x} = (x_v, y_v, \theta_v)^T$ is the vehicle state, and $\boldsymbol{p}_i = (x_i, y_i)^T$ is the *known* location of the i^{th} landmark in the world frame.

To make this tangible we will consider a common type of sensor that measures the range and bearing angle to a landmark in the environment, for instance a radar or a scanning-laser rangefinder such as shown in Fig. 6.22a. The sensor is mounted onboard the robot so the observation of the i^{th} landmark is

$$\boldsymbol{z} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{p}_i) = \begin{pmatrix} \sqrt{(y_i - y_v)^2 + (x_i - x_v)^2} \\ \tan^{-1}(y_i - y_v)/(x_i - x_v) - \theta_v \end{pmatrix} + \begin{pmatrix} w_r \\ w_\beta \end{pmatrix} \tag{6.8}$$

where $\boldsymbol{z} = (r, \beta)^T$ and r is the range, β the bearing angle, and $\boldsymbol{w} = (w_r, w_\beta)^T$ is a zero-mean Gaussian random variable that models errors in the sensor

$$\begin{pmatrix} w_r \\ w_\beta \end{pmatrix} \sim N(0, \boldsymbol{W}), \quad \boldsymbol{W} = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\beta^2 \end{pmatrix}$$

The constant diagonal covariance matrix indicates that range and bearing errors are independent.◀

It also indicates that covariance is independent of range but in reality covariance may increase with range since the strength of the return signal, laser or radar, drops rapidly ($1/r^4$) with distance (r) to the target.

For this example we set the sensor uncertainty to be $\sigma_r = 0.1$ m and $\sigma_\beta = 1°$ giving a sensor covariance matrix

```
>> W = diag([0.1, 1*pi/180].^2);
```

We model this type of sensor with a `RangeBearingSensor` object◀

A subclass of `Sensor`.

```
>> sensor = RangeBearingSensor(veh, map, 'covar', W)
```

which is connected to the vehicle and the map, and the sensor covariance matrix `W` is specified along with the maximum range and the bearing angle limits. The `reading` method provides the range and bearing to a randomly selected visible◀ landmark along with its identity, for example

The landmark is chosen randomly from the set of visible landmarks, those that are within the field of view and the minimum and maximum range limits. If no landmark is visible `i` is assigned a value of 0.

```
>> [z,i] = sensor.reading()
z =
    9.0905
    1.0334
i =
    17
```

The identity is an integer $i \in [1, 20]$ since the map was created with 20 landmarks. We have avoided the data association problem by assuming that we know the identity of the sensed landmark. The position of landmark 17 can be looked up in the map

```
>> landmark(17)
   -4.4615
   -9.0766
```

Using Eq. 6.8 the robot can estimate the range and bearing angle to the landmark based on its own estimated position and the known position of the landmark from the map. Any difference between the observation $\boldsymbol{z}^{\#}$ and the estimated observation indicates an error in the robot's pose estimate $\hat{\boldsymbol{x}}$ – it isn't where it *thought* it was. However this *difference*

$$\boldsymbol{\nu} = \boldsymbol{z}^{\#}\langle k+1\rangle - \boldsymbol{h}\left(\hat{\boldsymbol{x}}^{+}\langle k+1\rangle, \boldsymbol{p}_i\right) \tag{6.9}$$

has real value and is key to the operation of the Kalman filter. It is called the innovation since it represents *new* information. The Kalman filter uses the innovation to correct the state estimate and update the uncertainty estimate in an optimal way.

The predicted state computed earlier using Eq. 6.3 and Eq. 6.4 is updated by

$$\hat{\boldsymbol{x}}\langle k+1\rangle = \hat{\boldsymbol{x}}^{+}\langle k+1\rangle + \boldsymbol{K}\boldsymbol{\nu} \tag{6.10}$$

$$\hat{\boldsymbol{P}}\langle k+1\rangle = \hat{\boldsymbol{P}}^{+}\langle k+1\rangle - \boldsymbol{K}\boldsymbol{H}_x\hat{\boldsymbol{P}}^{+}\langle k+1\rangle \tag{6.11}$$

which are the Kalman filter update equations. These take the *predicted* values for the next time step denoted with the $^+$ and compute the optimal estimate by applying landmark measurements from time step $k+1$. The innovation is added to the estimated state after multiplying by the Kalman gain matrix $\boldsymbol{K}$ which is defined as

$$\boldsymbol{K} = \boldsymbol{P}^{+}\langle k+1\rangle \boldsymbol{H}_x^T \boldsymbol{S}^{-1} \tag{6.12}$$

$$\boldsymbol{S} = \boldsymbol{H}_x \boldsymbol{P}^{+}\langle k+1\rangle \boldsymbol{H}_x^T + \boldsymbol{H}_w \hat{\boldsymbol{W}} \boldsymbol{H}_w^T \tag{6.13}$$

where $\hat{\boldsymbol{W}}$ is the estimated covariance of the sensor noise and $\boldsymbol{H}_x$ and $\boldsymbol{H}_w$ are Jacobians obtained by differentiating Eq. 6.8 yielding

$$\boldsymbol{H}_x = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{w=0} = \begin{pmatrix} -\dfrac{x_i - x_v}{r} & -\dfrac{y_i - y_v}{r} & 0 \\ \dfrac{y_i - y_v}{r^2} & -\dfrac{x_i - x_v}{r^2} & -1 \end{pmatrix} \tag{6.14}$$

which is a function of landmark position, vehicle pose and landmark range; and

$$\boldsymbol{H}_w = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{w}}\right|_{w=0} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{6.15}$$

The `RangeBearingSensor` object above includes methods `h` to implement Eq. 6.8 and `Hx` and `Hw` to compute these Jacobians respectively.

The Kalman gain matrix K in Eq. 6.10 *distributes* the innovation from the landmark observation, a 2-vector, to update every element of the state vector – the position and orientation of the vehicle. Note that the second term in Eq. 6.11 is *subtracted* from the estimated covariance and this provides a means for covariance to decrease which was

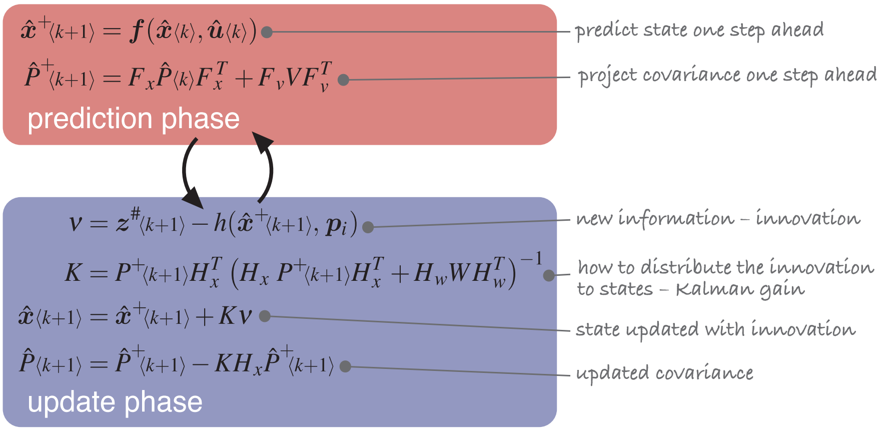

$\hat{\boldsymbol{x}}^{+}\langle k+1\rangle = \boldsymbol{f}(\hat{\boldsymbol{x}}\langle k\rangle, \hat{\boldsymbol{u}}\langle k\rangle)$
predict state one step ahead
$\hat{P}^{+}\langle k+1\rangle = F_x\hat{P}\langle k\rangle F_x^T + F_v V F_v^T$
project covariance one step ahead
prediction phase
$\nu = \boldsymbol{z}^{\#}\langle k+1\rangle - h(\hat{\boldsymbol{x}}^{+}\langle k+1\rangle, \boldsymbol{p}_i)$
new information – innovation
$K = P^{+}\langle k+1\rangle H_x^T \left(H_x P^{+}\langle k+1\rangle H_x^T + H_w W H_w^T\right)^{-1}$
how to distribute the innovation to states – Kalman gain
$\hat{\boldsymbol{x}}\langle k+1\rangle = \hat{\boldsymbol{x}}^{+}\langle k+1\rangle + K\nu$
state updated with innovation
$\hat{P}\langle k+1\rangle = \hat{P}^{+}\langle k+1\rangle - KH_x\hat{P}^{+}\langle k+1\rangle$
updated covariance
update phase

Fig. 6.6. Summary of extended Kalman filter algorithm showing the prediction and update phases

not possible for the dead-reckoning case of Eq. 6.4. The EKF comprises two phases: prediction and update, and these are summarized in Fig. 6.6.

We now have all the piece to build an estimator that uses odometry and observations of map features. The Toolbox implementation is

```
>> map = LandmarkMap(20);
>> veh = Bicycle('covar', V);
>> veh.add_driver( RandomPath(map.dim) );
>> sensor = RangeBearingSensor(veh, map, 'covar', W, 'angle',↵
 [-pi/2 pi/2], 'range', 4, 'animate');
>> ekf = EKF(veh, V, P0, sensor, W, map);
```

The `LandmarkMap` constructor has a default map dimension of ±10 m which is accessed by its `dim` property.

Running the simulation for 1 000 time steps

```
>> ekf.run(1000);
```

shows an animation of the robot moving and observations being made to the landmarks. We plot the saved results

```
>> map.plot()
>> veh.plot_xy();
>> ekf.plot_xy('r');
>> ekf.plot_ellipse('k')
```

which are shown in Fig. 6.7a. The error ellipses are now much smaller and many can hardly be seen.

Figure 6.7b shows a zoomed view of the robot's actual and estimated path – the robot is moving from top to bottom. We can see the error ellipses growing as the robot moves and then shrinking, just after a jag in the estimated path. This corresponds to the observation of a landmark. New information, beyond odometry, has been used to correct the state in the Kalman filter update phase.

Figure 6.8a shows that the overall uncertainty is no longer growing monotonically. When the robot sees a landmark it is able to dramatically reduce its estimated covariance. Figure 6.8b shows the error associated with each component of pose and the pink background is the estimated 95% confidence bound (derived from the covariance matrix) and we see that the error is mostly within this envelope. Below this is plotted the landmark observations and we see that the confidence bounds are tight (indicating low uncertainty) while landmarks are being observed but that they start to grow once observations stop. However as soon as an observation is made the uncertainty rapidly decreases.

This EKF framework allows data from many and varied sensors to update the state which is why the estimation problem is also referred to as sensor fusion. For example heading angle from a compass, yaw rate from a gyroscope, target bearing angle from a camera, position

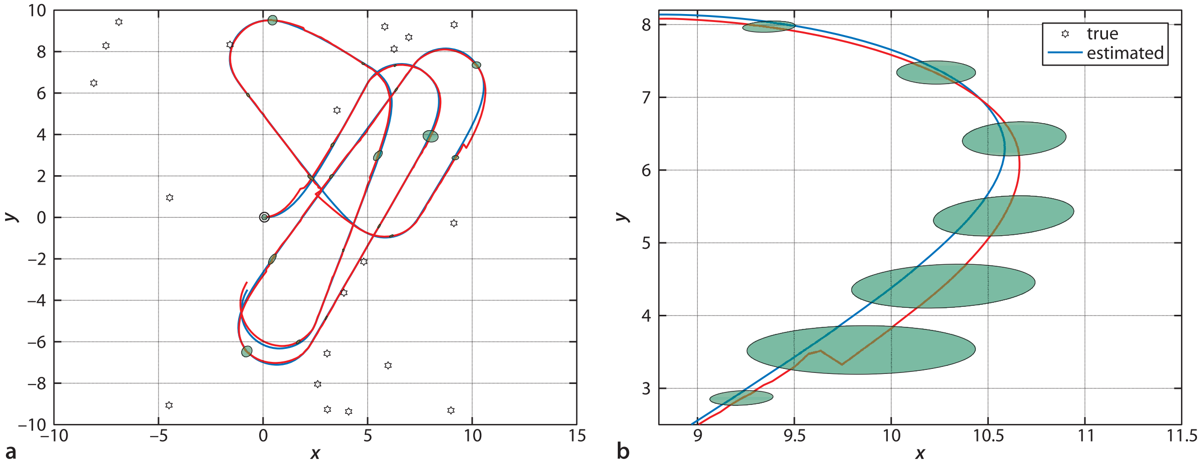


Fig. 6.7. **a** EKF localization showing the true path of the robot (*blue*) and the path estimated from odometry and landmarks (*red*). *Black stars* are landmarks. 95% confidence ellipses are indicated in *green*. The robot starts at the origin. **b** Closeup of the robot's true and estimated path

from GPS could all be used to update the state. For each sensor we need only to provide the observation function $\boldsymbol{h}(\cdot)$, the Jacobians $\boldsymbol{H}_x$ and $\boldsymbol{H}_w$ and some estimate of the sensor covariance $\boldsymbol{W}$. The function $\boldsymbol{h}(\cdot)$ can be nonlinear and even noninvertible – the EKF will do the rest.

> As discussed earlier for `V`, we also use `W` twice. The first usage, in the constructor for the `RangeBearingSensor` object, is the covariance $\boldsymbol{W}$ of the Gaussian noise that is added to the computed range and bearing to *simulate* sensor error as in Eq. 6.8. In a real application this noise is generated by some physical process *hidden inside* the sensor and we would not know its parameters. The second usage, $\hat{\boldsymbol{W}}$ is our best *estimate* of the sensor covariance which is used by the Kalman filter Eq. 6.12.

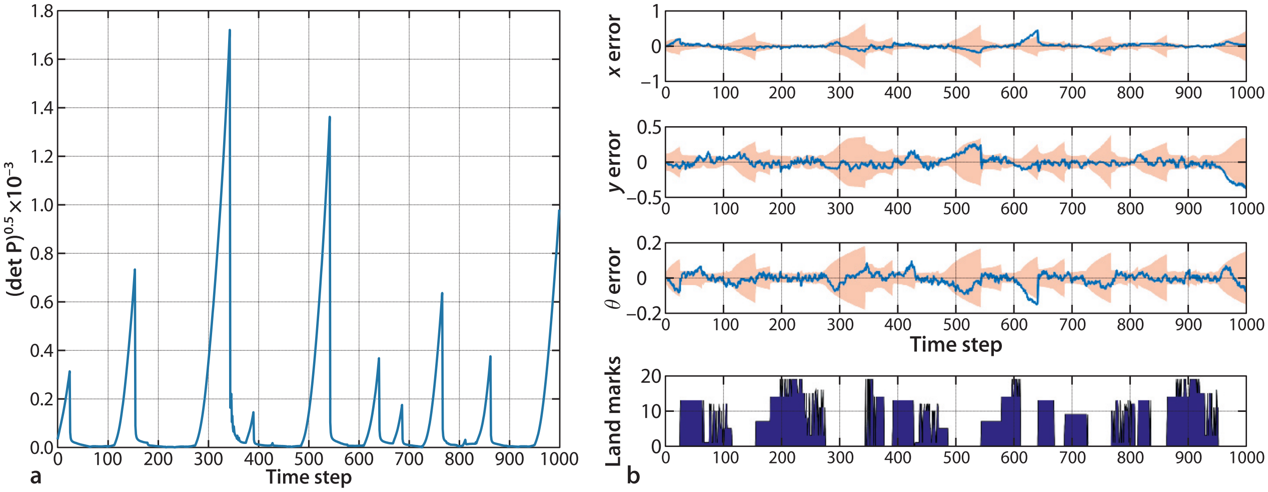


Fig. 6.8. a Covariance magnitude as a function of time. Overall uncertainty is given by $\sqrt{\det(\boldsymbol{P})}$ and shows that uncertainty does not continually increase with time. **b** Top: pose estimation error with 95% confidence bound shown in *pink*; bottom: observed landmarks the *bar* indicates which landmark is seen at each time step, 0 means no observation

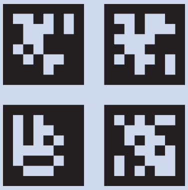
Data association. So far we have assumed that the observed landmark reveals its identity to us, but in reality this is rarely the case. Instead we compare our observation to the predicted position of all currently known landmarks and make a decision as to which landmark it is most likely to be, or whether it is a new landmark. This decision needs to take into account the uncertainty associated with the vehicle's pose, the sensor measurement and the landmarks in the map. This is the data association problem. Errors in this step are potentially catastrophic – incorrect innovation is coupled via the Kalman filter to the state of the vehicle and all the other landmarks which increases the chance of an incorrect data association on the next cycle. In practice, filters only use a landmark when there is a very high confidence in its estimated identity – a process that involves Mahalanobis distance and χ^2 confidence tests. If the situation is ambiguous it is best not to use the landmark – it can do more harm than good.

An alternative is to use a multi-hypothesis estimator, such as the particle filter that we will discuss in Sect. 6.7, which can model the possibility of observing landmark A or landmark B, and future observations will reinforce one hypothesis and weaken the others. The extended Kalman filter uses a Gaussian probability model, with just one peak, which limits it to holding only one hypothesis about the robot's pose. (Picture: the wreck of the Tararua, 1881)

Simple landmarks. For educational purposes it might be appropriate to use artificial landmarks that can be cheaply sensed by a camera. These need to be not only visually distinctive in the environment but also encode an identity. 2-dimensional bar codes such as QR codes or ARTags are well suited for this purpose. The Toolbox supports a variant called AprilTags, shown to the right, and

```
>> tags = apriltags(im);
```

returns a vector of `AprilTag` objects whose elements correspond to tags found in the image `im`. The centroid of the tag (`centre` property) can be used to determine relative bearing (see page 161), and the length of the edges (from the `corners` property) is a function of distance. The tag object also includes an homography (see Sect. 14.2.4) (`H` property) which encodes information about the orientation of the plane of the April tag. More details about April tags can be found at http://april.eecs.umich.edu.



A landmark might be some easily identifiable pattern such as this April tag (36h11) which can be detected in an image. Its position and size in the image encodes the bearing angle and range. The pattern itself encodes a number between 0 and 586 which could be used to uniquely identify the landmark in a map.

6.3 Creating a Map

So far we have taken the existence of the map for granted, an understandable mindset given that maps today are common and available for free via the internet. Nevertheless somebody, or something, has to create the maps we will use. Our next example considers the problem of a robot moving in an environment with landmarks and creating a map of their locations.

As before we have a range and bearing sensor mounted on the robot which measures, imperfectly, the position of landmarks with respect to the robot. There are a total of N landmarks in the environment and as for the previous example we assume that the sensor can determine the identity of each observed landmark. However for this case we assume that the robot knows its own location perfectly – it has ideal localization. This is unrealistic but this scenario is an important stepping stone to the next section.◀

A close and realistic approximation would be a high-end RTK GPS+INS system operating in an environment with no buildings or hills to obscure satellites.

Since the vehicle pose is known perfectly we do not need to estimate it, but we do need to estimate the coordinates of the landmarks. For this problem the state vector comprises the estimated coordinates of the M landmarks that have been observed so far

$$\hat{\boldsymbol{x}} = \left(x_1, y_1, x_2, y_2, \cdots x_M, y_M\right)^T \in \mathbb{R}^{2M\times 1}$$

The corresponding estimated covariance $\hat{\boldsymbol{P}}$ will be a $2M \times 2M$ matrix. The state vector has a variable length since we do not know in advance how many landmarks exist in the environment. Initially $M = 0$ and is incremented every time a previously unseen landmark is observed.

The prediction equation is straightforward in this case since the landmarks are assumed to be stationary

$$\hat{\boldsymbol{x}}^+\langle k{+}1\rangle = \hat{\boldsymbol{x}}\langle k\rangle \tag{6.16}$$

$$\hat{\boldsymbol{P}}^+\langle k{+}1\rangle = \hat{\boldsymbol{P}}\langle k\rangle \tag{6.17}$$

We introduce the function $\boldsymbol{g}(\cdot)$ which is the inverse of $\boldsymbol{h}(\cdot)$ and gives the coordinates of the observed landmark based on the known vehicle pose and the sensor observation

$$\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{z}) = \begin{pmatrix} x_v + r\cos(\theta_v + \beta) \\ y_v + r\sin(\theta_v + \beta) \end{pmatrix}$$

Since $\hat{\boldsymbol{x}}$ has a variable length we need to extend the state vector and the covariance matrix whenever we encounter a landmark we have not previously seen. The state vector is extended by the function $\boldsymbol{y}(\cdot)$

$$\boldsymbol{x}\langle k\rangle' = \boldsymbol{y}\big(\boldsymbol{x}\langle k\rangle, \boldsymbol{z}\langle k\rangle, \boldsymbol{x}_v\langle k\rangle\big) \tag{6.18}$$

$$= \begin{pmatrix} \boldsymbol{x}\langle k\rangle \\ \boldsymbol{g}\big(\boldsymbol{x}_v\langle k\rangle, \boldsymbol{z}\langle k\rangle\big) \end{pmatrix} \tag{6.19}$$

which appends the sensor-based estimate of the new landmark's coordinates to those already in the map. The order of feature coordinates within $\hat{\boldsymbol{x}}$ therefore depends on the order in which they are observed.

The covariance matrix also needs to be extended when a new landmark is observed and this is achieved by

$$\hat{\boldsymbol{P}}\langle k\rangle' = \boldsymbol{Y}_z \begin{pmatrix} \hat{\boldsymbol{P}}\langle k\rangle & 0 \\ 0 & \hat{\boldsymbol{W}} \end{pmatrix} \boldsymbol{Y}_z^T$$

where $\boldsymbol{Y}_z$ is the insertion Jacobian

$$\boldsymbol{Y}_z = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}} = \begin{pmatrix} \boldsymbol{I}_{n\times n} & & \boldsymbol{0}_{n\times 2} \\ \boldsymbol{G}_x & \boldsymbol{0}_{2\times n-3} & \boldsymbol{G}_z \end{pmatrix} \tag{6.20}$$

that relates the rate of change of the extended state vector to the new observation. n is the dimension of $\hat{\boldsymbol{P}}$ prior to it being extended and

$$\boldsymbol{G}_x = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{6.21}$$

$$\boldsymbol{G}_z = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}} = \begin{pmatrix} \cos(\theta_v + \beta) & -r\sin(\theta_v + \beta) \\ \sin(\theta_v + \beta) & r\cos(\theta_v + \beta) \end{pmatrix} \tag{6.22}$$

G_x is zero since $g(\cdot)$ is independent of the map in $\boldsymbol{x}$. An additional Jacobian for $\boldsymbol{h}(\cdot)$ is

$$\boldsymbol{H}_{p_i} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{p_i}} = \begin{pmatrix} \frac{x_i - x_v}{r} & \frac{y_i - y_v}{r} \\ -\frac{y_i - y_v}{r^2} & \frac{x_i - x_v}{r^2} \end{pmatrix} \tag{6.23}$$

which describes how the landmark observation changes with respect to landmark position for a particular robot pose, and is implemented by the method `Hp`.

For the mapping case the Jacobian $\boldsymbol{H}_x$ used in Eq. 6.11 describes how the landmark observation changes with respect to the full state vector. However the observation depends only on the position of that landmark so this Jacobian is mostly zeros

$$\boldsymbol{H}_x = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{w}=0} = \left(0 \cdots \boldsymbol{H}_{p_i} \cdots 0\right) \in \mathbb{R}^{2\times 2M} \tag{6.24}$$

where $\boldsymbol{H}_{p_i}$ is at the location in the vector corresponding to the state $\boldsymbol{p}_i$. This structure represents the fact that observing a particular landmark provides information to estimate the position of that landmark, but no others.

The Toolbox implementation is

```
>> map = LandmarkMap(20);
>> veh = Bicycle();  % error free vehicle
>> veh.add_driver( RandomPath(map.dim) );
>> W = diag([0.1, 1*pi/180].^2);
>> sensor = RangeBearingSensor(veh, map, 'covar', W);
>> ekf = EKF(veh, [], [], sensor, W, []);
```

the empty matrices passed to `EKF` indicate respectively that there is no estimated odometry covariance for the vehicle (the estimate is perfect), no initial vehicle state covariance, and the map is unknown. We run the simulation for 1 000 time steps

```
>> ekf.run(1000);
```

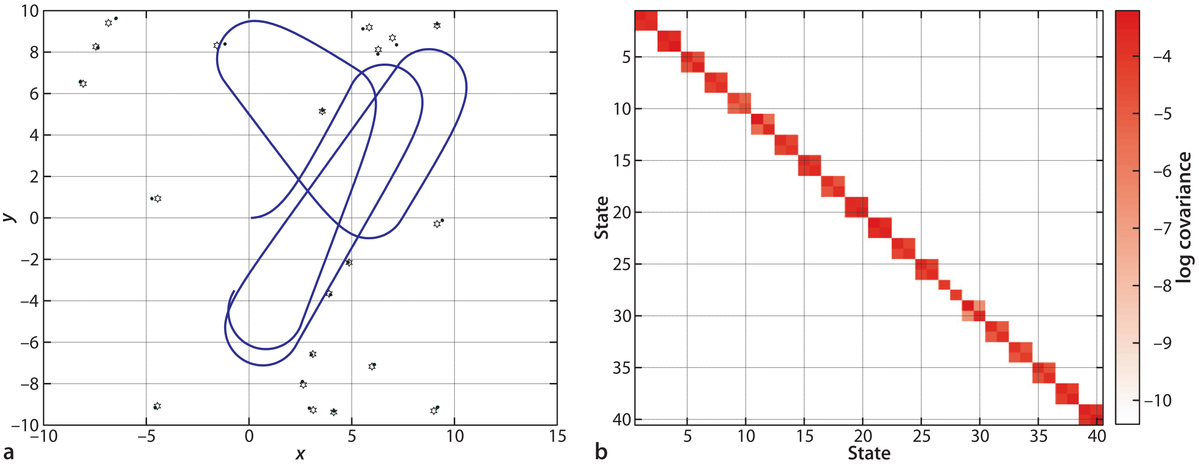


Fig. 6.9. EKF mapping results. **a** The estimated landmarks are indicated by *black dots* with 95% confidence ellipses (*green*), the true location (*black ✡-marker*) and the robot's path (*blue*). The landmark estimates have not fully converged on their true values and the estimated covariance ellipses can only be seen by zooming; **b** the nonzero elements of the final covariance matrix

and see an animation of the robot moving and the covariance ellipses associated with the map features evolving over time. The estimated landmark positions

```
>> map.plot();
>> ekf.plot_map('g');
>> veh.plot_xy('b');
```

are shown in Fig. 6.9a as 95% confidence ellipses along with the true landmark positions and the path taken by the robot. The covariance matrix has a block diagonal structure which is shown graphically in Fig. 6.9b. The off-diagonal elements are zero, which implies that the landmark estimates are uncorrelated or independent. This is to be expected since observing one landmark provides no new information about any other landmark.

Internally the EKF object maintains a table to relate the landmark's identity, returned by the RangeBearingSensor, to the position of that landmark's coordinates in the state vector. For example the landmark with identity 6

```
>> ekf.landmarks(:,6)
ans =
    19
    71
```

was seen a total of 71 times during the simulation and comprises elements 19 and 20 of $\hat{\boldsymbol{x}}$

```
>> ekf.x_est(19:20)'
ans =
   -6.4803    9.6233
```

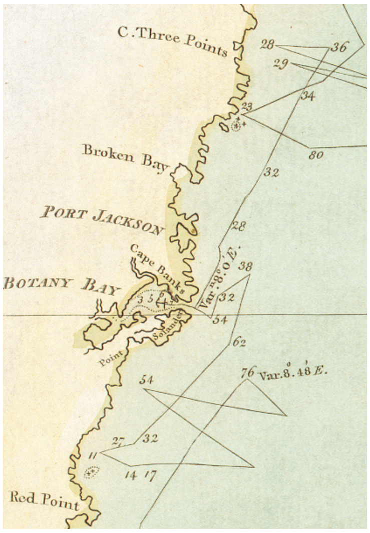
which is its estimated location. Its estimated covariance is a submatrix within $\hat{\boldsymbol{P}}$

```
>> ekf.P_est(19:20,19:20)
ans =
   1.0e-03 *
    0.2913    0.1814
    0.1814    0.3960
```

6.4 Localization and Mapping

Finally we tackle the problem of determining our position and creating a map at the same time. This is an old problem in marine navigation and cartography – incrementally extending maps while also using the map for navigation. Figure 6.10 shows what can be done without GPS from a moving ship with poor odometry and infrequent celestial position "fixes". In robotics this problem is known as simultaneous localization and mapping (SLAM) or concurrent mapping and localization (CML). This is often considered to be a "chicken and egg" problem – we need a map to localize and we need to localize to make the map. However based on what we have learned in the previous sections this problem is now quite straightforward to solve.

Fig. 6.10. Map of the New Holland coast (now eastern Australia) by Captain James Cook in 1770. The path of the ship and the map of the coast were determined at the same time. *Numbers* indicate depth in fathoms (1.83 m) (National Library of Australia, MAP NK 5557 A)



The state vector comprises the vehicle configuration *and* the coordinates of the M landmarks that have been observed so far

$$\hat{\boldsymbol{x}} = \left(x_v, y_v, \theta_v, x_1, y_1, x_2, y_2, \cdots x_M, y_M\right)^T \in \mathbb{R}^{2M+3\times1}$$

The estimated covariance is a $(2M+3) \times (2M+3)$ matrix and has the structure

$$\hat{\boldsymbol{P}} = \begin{pmatrix} \hat{\boldsymbol{P}}_{vv} & \hat{\boldsymbol{P}}_{vm} \\ \hat{\boldsymbol{P}}_{vm}^T & \hat{\boldsymbol{P}}_{mm} \end{pmatrix}$$

where $\hat{\boldsymbol{P}}_{vv}$ is the covariance of the vehicle pose, $\hat{\boldsymbol{P}}_{mm}$ the covariance of the map landmark positions, and $\hat{\boldsymbol{P}}_{vm}$ is the correlation between vehicle and landmark states.

The predicted vehicle state and covariance are given by Eq. 6.3 and Eq. 6.4 and the sensor-based update is given by Eq. 6.10 to 6.15. When a new feature is observed the state vector is updated using the insertion Jacobian $\boldsymbol{Y}_z$ given by Eq. 6.20 but in this case $\boldsymbol{G}_x$ is nonzero

$$G_x = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} = \begin{pmatrix} 1 & 0 & -r\sin(\theta_v+\beta) \\ 0 & 1 & r\cos(\theta_v+\beta) \end{pmatrix} \tag{6.25}$$

since the estimate of the new landmark depends on the state vector which now contains the vehicle's pose.

For the SLAM case the Jacobian $\boldsymbol{H}_x$ used in Eq. 6.11 describes how the landmark observation changes with respect to the state vector. The observation will depend on the position of the vehicle and on the position of the observed landmark and is

$$\boldsymbol{H}_x = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{w}=0} = \left(\boldsymbol{H}_{x_v} \cdots 0 \cdots \boldsymbol{H}_{p_i} \cdots 0\right) \in \mathbb{R}^{2\times(2M+3)} \tag{6.26}$$

where $\boldsymbol{H}_{p_i}$ is at the location corresponding to the landmark $\boldsymbol{p}_i$. This is similar to Eq. 6.24 but with an extra nonzero block $\boldsymbol{H}_{x_v}$ given by Eq. 6.14.

The Kalman gain matrix $\boldsymbol{K}$ *distributes* innovation from the landmark observation, a 2-vector, to update *every* element of the state vector – the pose of the vehicle *and* the position of *every* landmark in the map.

The Toolbox implementation is by now quite familiar

```
>> P0 = diag([.01, .01, 0.005].^2);
>> map = LandmarkMap(20);
>> veh = Bicycle('covar', V);
>> veh.add_driver( RandomPath(map.dim) );
>> sensor = RangeBearingSensor(veh, map, 'covar', W);
>> ekf = EKF(veh, V, P0, sensor, W, []);
```

and the empty matrix passed to `EKF` indicates that the map is unknown. `P0` is the initial 3 × 3 covariance for the vehicle state.

We run the simulation for 1 000 time steps

```
>> ekf.run(1000);
```

and as usual an animation is shown of the vehicle moving. We also see the covariance ellipses associated with the map features evolving over time. We can plot the results

```
>> map.plot();
>> ekf.plot_map('g');
>> ekf.plot_xy('r');
>> veh.plot_xy('b');
```

which are shown in Fig. 6.11.

Figure 6.12a shows that uncertainty is decreasing over time. The final covariance matrix is shown graphically in Fig. 6.12b and we see a complex structure. Unlike the mapping case af Fig. 6.9 $\hat{\boldsymbol{P}}_{mm}$ is not block diagonal, and the finite off-diagonal terms

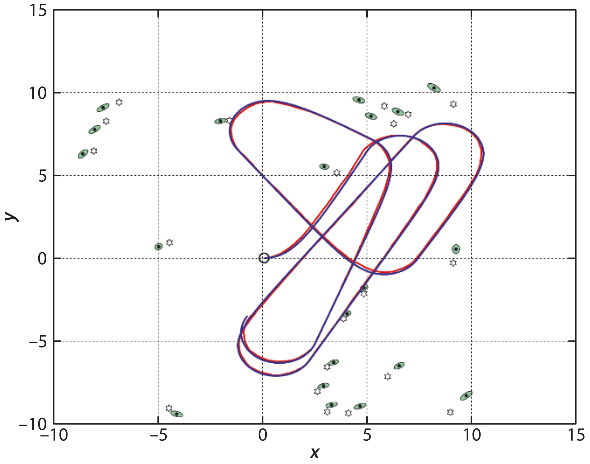


Fig. 6.11.
Simultaneous localization and mapping showing the true (*blue*) and estimated (*red*) robot path superimposed on the true map (*black ✡-marker*). The estimated map features are indicated by *black dots* with 95% confidence ellipses (*green*)

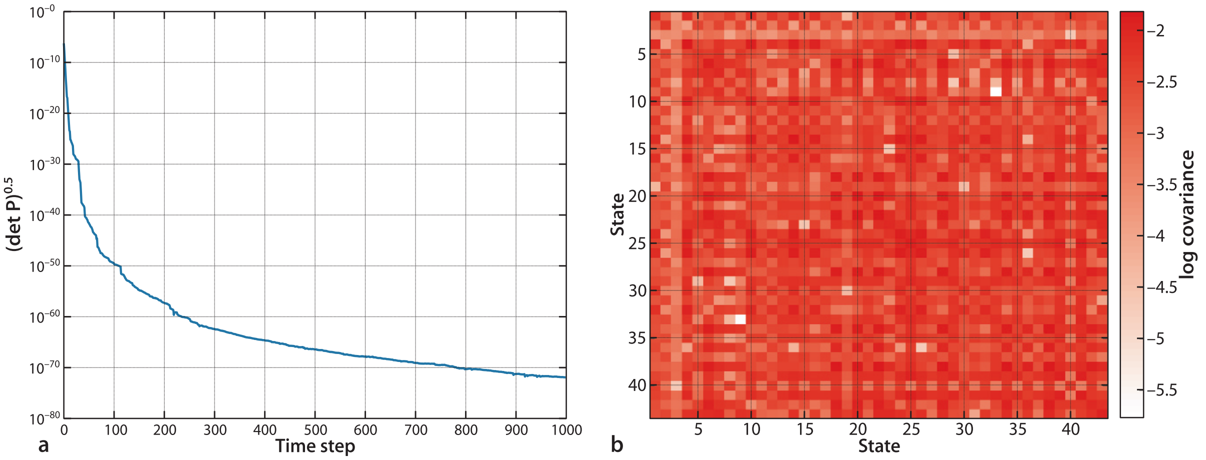


Fig. 6.12. Simultaneous localization and mapping. **a** Covariance versus time; **b** the final covariance matrix

represent correlation *between* the landmarks in the map. The landmark uncertainties never increase, the position prediction model is that they do not move, but they also never drop below the initial uncertainty of the vehicle which was set in $\boldsymbol{P}_0$. The block $\hat{\boldsymbol{P}}_{vm}$ is the correlation between errors in the vehicle pose and the landmark locations. A landmark's location estimate is a function of the vehicle's location and errors in the vehicle location appear as errors in the landmark location – and vice versa.

The correlations are used by the Kalman filter to connect the observation of any landmark to an improvement in the estimate of every other landmark in the map as well as the vehicle pose. Conceptually it is as if all the states were connected by springs and the movement of any one affects all the others.

The extended Kalman filter introduced here has a number of drawbacks. Firstly the size of the matrices involved increase with the number of landmarks and can lead to memory and computational bottlenecks as well as numerical problems. The underlying assumption of the Kalman filter is that all errors are Gaussian and this is far from true for sensors like laser rangefinders which we will discuss later in this chapter. We also need good estimates of covariance of the noise sources which in practice is challenging.

6.5 Rao-Blackwellized SLAM

We will briefly and informally introduce the underlying principle of Rao-Blackwellized SLAM of which FastSLAM is a popular and well known instance. The approach is motivated by the fact that the size of the covariance matrix for EKF SLAM is quadratic in the number of landmarks, and for large-scale environments becomes computationally intractable.

If we compare the covariance matrices shown in Fig. 6.9b and 6.12b we notice a stark difference. In both cases we were creating a map of unknown landmarks but Fig. 6.9b is mostly zero with a finite block diagonal structure whereas Fig. 6.12b has no zero values at all. The difference is that for Fig. 6.9b we assumed the robot trajectory was known exactly and that makes the landmark estimates *independent* – observing one landmark provides information about *only* that landmark. The landmarks are *uncorrelated*, hence all the zeros in the covariance matrix. If the robot trajectory is not known, the case for Fig. 6.12b, then the landmark estimates are correlated – error in one landmark position is related to errors in robot pose and other landmark positions. The Kalman filter uses the correlation information so that a measurement of any one landmark provides information to improve the estimate of all the other landmarks and the robot's pose.

In practice we don't know the true pose of the robot but imagine a multi-hypothesis estimator◀ where every hypothesis represents a robot trajectory that we *assume* is correct. This means that the covariance matrix will be block diagonal like Fig. 6.9b – rather than a filter with a 2N × 2N covariance matrix we can have N simple filters which are

Such as the particle filter that we will discuss in Sect. 6.7.

each *independently* estimating the position of a single landmark and have a 2 × 2 covariance matrix. Independent estimation leads to a considerable saving in both memory and computation. Importantly though, we are only able to do this because we *assumed* that the robot's estimated trajectory is correct.

Each hypothesis also holds an estimate of the robot's trajectory to date. Those hypotheses that best explain the landmark measurements are retained and propagated while those that don't are removed and recycled. If there are M hypotheses the overall computational burden falls from $O(N^2)$ for the EKF SLAM case to $O(M \log N)$ and in practice works well for M in the order of tens to hundreds but can work for a value as low as $M = 1$.

6.6 Pose Graph SLAM

An alternative approach to the SLAM problem is to separate it into two components: a front end and a back end, connected by a pose graph as shown in Fig. 6.13. The robot's path is considered to be a sequence of distinct poses and the task is to estimate those poses. Constraints between the unknown poses are based on measurements from a variety of sensors including odometry, laser scanners and cameras. The problem is formulated as a directed graph as shown in Fig. 6.14. A node corresponds to a robot pose or a landmark position. An edge between two nodes represents a spatial constraint between the nodes derived from some sensor data.

As the robot progresses it compounds an increasing number of uncertain relative poses so that the cumulative error in the pose of the nodes will increase – the problem with dead reckoning we discussed earlier. This is shown in exaggerated fashion in Fig. 6.14 where the robot is traveling around a square. By the time the robot reaches node 4 the error is significant. However when it makes a measurement of node 1 a constraint is added – the dashed edge – indicating that the nodes are closer than the estimated relative pose based on the chain of relative poses from odometry: ${}^{1}\xi_2^{\#} \oplus {}^{2}\xi_3^{\#} \oplus {}^{3}\xi_4^{\#}$. The back-end algorithm will then *pull* all the nodes closer to their correct pose.

The front end adds new nodes as the robot travels▸ as well as edges that define constraints between poses. For example, when moving from one place to another wheel odometry gives an estimate of distance and change in orientation which is a constraint. In addition the robot's exteroceptive sensors may observe the relative position of a landmark and this also adds a constraint. Every measurement adds a constraint – an edge in the graph. There is no limit to the number of edges entering or leaving a node.

Typically a new place is declared every meter or so of travel, or after a sharp turn.

The back end adjusts the poses of the nodes▸ so that the constraints are satisfied as well as possible, that is, that the sensor observations are best explained.

Also the positions of landmarks as we discuss later in this section.

Figure 6.15 shows the notation associated with two poses in the graph. Coordinate frames $\{i\}$ and $\{j\}$ are associated with robot poses i and j respectively and we seek to estimate ${}^{0}\xi_i$ and ${}^{0}\xi_j$ in the world coordinate frame. The robot makes a measurement of the relative pose ${}^{i}\xi_j^{\#}$ which will, in general, be different to the relative pose ${}^{i}\xi_j$ inferred from the poses ${}^{0}\xi_i$ and ${}^{0}\xi_j$. This difference, or innovation, is caused by error in the sensor measurement ${}^{i}\xi_j^{\#}$ and/or the node poses ${}^{0}\xi_i$ and ${}^{0}\xi_j$ and we use it to adjust the poses of the nodes. However there is insufficient information to determine where the error lies so naively adjusting ${}^{0}\xi_i$ and ${}^{0}\xi_j$ to better explain the measurement might increase

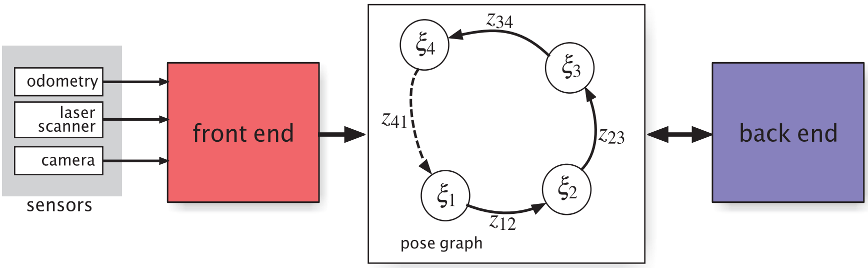


Fig. 6.13. Pose-graph SLAM system. The front end creates nodes as the robot travels, and creates edges based on sensor data. The back end adjusts the node positions to minimize total error

the error in another part of the graph – we need to minimize the error consistently over the whole graph.

The first step is to express the error associated with the graph edge in terms of the sensor measurement and our best estimates of the node poses with respect to the world frame◀

We have used our pose notation here but in the literature measurements are typically denoted by $\boldsymbol{z}$, error by $\boldsymbol{e}$ and pose or position by $\boldsymbol{x}$.

$$\xi_\varepsilon = \ominus {}^i\xi_j^{\#} \ominus {}^0\hat{\xi}_i \oplus {}^0\hat{\xi}_j \in \mathbf{SE}(2) \tag{6.27}$$

which is ideally zero.

We can formulate this as a minimization problem and attempt to find the poses of all the nodes $\boldsymbol{x} = \{\xi_1, \xi_2 \cdots \xi_N\}$ that minimizes the error across all the edges

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} \sum_k F_k(\boldsymbol{x}) \tag{6.28}$$

where $\boldsymbol{x}$ is the state of the pose graph and contains the pose of every node, and $F_k(\boldsymbol{x})$ is a nonnegative scalar cost associated with the edge k connecting node i to node j.

We convert the edge pose error in Eq. 6.27 to a vector representation $\xi_\varepsilon \sim (x, y, \theta)$ which is a function $\boldsymbol{f}_k(\boldsymbol{x}) \in \mathbb{R}^3$ of the state. The scalar cost can be obtained from a quadratic expression

$$F_k(\boldsymbol{x}) = \boldsymbol{f}_k^T(\boldsymbol{x})\boldsymbol{\Omega}_k\boldsymbol{f}_k(\boldsymbol{x}) \tag{6.29}$$

where $\boldsymbol{\Omega}_k$ is a positive-definite information matrix used as a weighting term.◀ Although Eq. 6.29 is written as a function of all poses $\boldsymbol{x}$, it in fact depends only on the pose of its two vertices ξ_i and ξ_j and the measurement ${}^i\xi_j^{\#}$. Solving Eq. 6.28 is a complex optimization problem which does not have a closed-form solution, but this kind of nonlinear least squares problem can be solved numerically if we have a good initial estimate of $\boldsymbol{x}$. Specifically this is a sparse nonlinear least squares problem which is discussed in Sect. F.2.4.

In practice this matrix is diagonal reflecting confidence in the x-, y- and θ-directions. The "bigger" (in a matrix sense) $\boldsymbol{\Omega}$ is, the more the edge *matters* in the optimization procedure. Different sensors have different accuracy and this must be taken into account. Information from a high-quality sensor should be given more weight than information from a low-quality sensor.

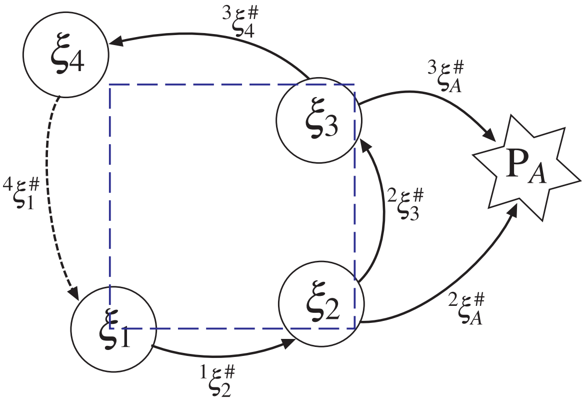


Fig. 6.14. Pose-graph SLAM example. Places are shown as *circular nodes* and have an associated pose. Landmarks are shown as *star-shaped nodes* and have an associated position. Edges represent a measurement of a relative pose or position with respect to the node at the tail of the arrow

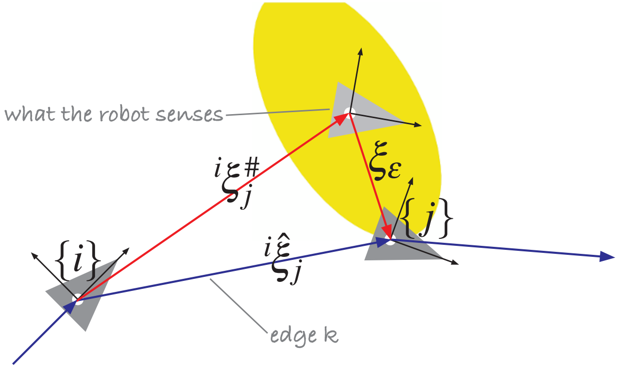


Fig. 6.15. Pose graph notation. The *light grey robot* is the estimated pose of $\{j\}$ based on the sensor measurement ${}^i\xi_j^{\#}$. The *yellow ellipse* indicates uncertainty associated with that measurement

The edge error $\boldsymbol{f}_k(\boldsymbol{x})$ can be linearized about the current state $\boldsymbol{x}_0$ of the pose graph

$$\boldsymbol{f}_k'(\boldsymbol{\Delta}) \approx \boldsymbol{f}_{0,k} + \boldsymbol{J}_k \boldsymbol{\Delta}$$

where $\boldsymbol{f}_{0,k} = \boldsymbol{f}_k(\boldsymbol{x}_0)$ and

$$\boldsymbol{J}_k = \frac{\partial \boldsymbol{f}_k(\boldsymbol{x})}{\partial \boldsymbol{x}} \in \mathbb{R}^{3\times 3N}$$

is a Jacobian matrix which depends only on the pose of its two vertices ξ_i and ξ_j so it is mostly zeros

$$\boldsymbol{J}_k = \left(0 \cdots \boldsymbol{A}_i \cdots \boldsymbol{B}_j \cdots 0\right), \quad \text{where } \boldsymbol{A}_i = \frac{\partial \boldsymbol{f}_k(\boldsymbol{x})}{\partial \xi_i} \in \mathbb{R}^{3\times 3}, \boldsymbol{B}_j = \frac{\partial \boldsymbol{f}_k(\boldsymbol{x})}{\partial \xi_j} \in \mathbb{R}^{3\times 3}$$

and more details are provided in Appendix E.

There are many ways to compute the Jacobians but here will demonstrate use of the MATLAB Symbolic Math Toolbox™

```
>> syms xi yi ti xj yj tj xm ym tm assume real
>> xi_e = inv( SE2(xm, ym, tm) ) * inv( SE2(xi, yi, ti) ) * SE2(xj, yj, tj);
>> fk = simplify(xi_e.xyt);
```

and the Jacobian which describes how the function $\boldsymbol{f}_k$ varies with respect to ξ_i is

```
>> jacobian( fk, [xi yi ti] );
>> Ai = simplify(ans)
Ai =
[ -cos(ti+tm), -sin(ti+tm), yj*cos(ti+tm)-yi*cos(ti+tm)+xi*sin(ti+tm)-xj*sin(ti+tm)]
[  sin(ti+tm), -cos(ti+tm), xi*cos(ti+tm)-xj*cos(ti+tm)+yi*sin(ti+tm)-yj*sin(ti+tm)]
[           0,           0,                                                      -1]
```

and we follow a similar procedure for $\boldsymbol{B}_j$.

It is quite straightforward to solve this type of pose-graph problem with the Toolbox. We load a simple pose graph, similar to Fig. 6.14, from a data file▸

The file format is one used by the popular posegraph optimization package g^{2}o which you can find at http://openslam.org.

```
>> pg = PoseGraph('pg1.g2o')
loaded g2o format file: 4 nodes, 4 edges in 0.00 sec
```

which returns a Toolbox `PoseGraph` object that describes the pose graph. We can visualize this by▸

The nodes have an orientation which is in the *z*-direction, rotate the graph to see this.

```
>> pg.plot()
```

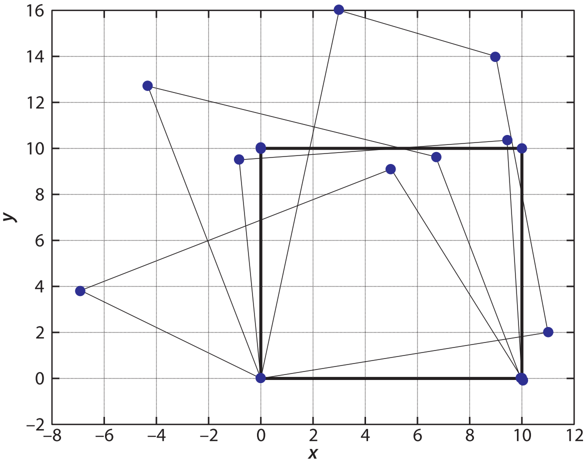


Fig. 6.16.
Pose graph optimization showing the result over consecutive iterations, the final configuration is the *square* shown in *bold*

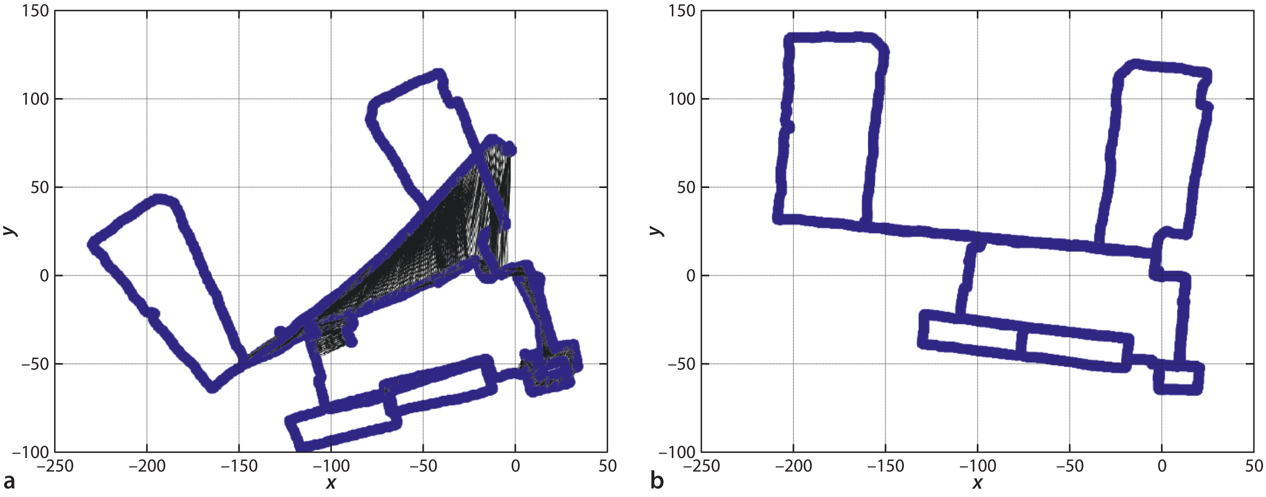


Fig. 6.17. Pose graph with 1 941 nodes and 3 995 edges from the MIT Killian Court dataset. **a** Initial configuration; **b** final configuration after optimization

The optimization reduces the error in the network while animating the changing pose of the nodes

```
>> pg.optimize('animate')
solving....done in 0.075 sec.  Total cost 316.88
solving....done in 0.0033 sec.  Total cost 47.2186
 .
 .
solving....done in 0.0023 sec.  Total cost 3.14139e-11
```

The displayed text indicates that the total cost is decreasing while the graphics show the nodes moving into a configuration that minimizes the overall error in the network. The pose graph configurations are overlaid and shown in Fig. 6.16.

Now let's look a much larger example based on real robot data

```
>> pg = PoseGraph('killian-small.toro');
loaded TORO/LAGO format file: 1941 nodes, 3995 edges in 0.68 sec
```

There are a lot of nodes and this takes a few seconds.

which we can plot◂

```
>> pg.plot()
```

and this is shown in Fig. 6.17a. Note the mass of edges in the center of the graph, and if you zoom in you can see these in detail. We optimize the pose graph by

```
>> pg.optimize()
solving....done in 0.91 sec.  Total cost 1.78135e+06
 .
 .
solving....done in 1.1 sec.  Total cost 5.44567
```

and the final configuration is shown in Fig. 6.17b. The original pose graph had severe pose errors from accumulated odometry error which meant that two trips along the corridor were initially very poorly aligned.

The pose graph can also include landmarks as shown in Fig. 6.18. Landmarks have a position $\boldsymbol{P}_j \in \mathbb{R}^2$ not a pose, and therefore differ from the nodes discussed so far. To accomodate this we redefine the state vector to be $\boldsymbol{x} = \{\xi_1, \xi_2 \cdots \xi_N \,|\, \boldsymbol{P}_1, \boldsymbol{P}_2 \cdots \boldsymbol{P}_M\}$ which includes N robot poses and M landmark positions. The robot at pose i observes landmark j at range and bearing $\boldsymbol{z}^\# = (r^\#, \beta^\#)$ which is converted to Cartesian form in frame $\{i\}$

$$^i\boldsymbol{P}_j^\# = \left(r^\# \cos\beta^\#, r^\# \sin\beta^\#\right) \in \mathbb{R}^2$$

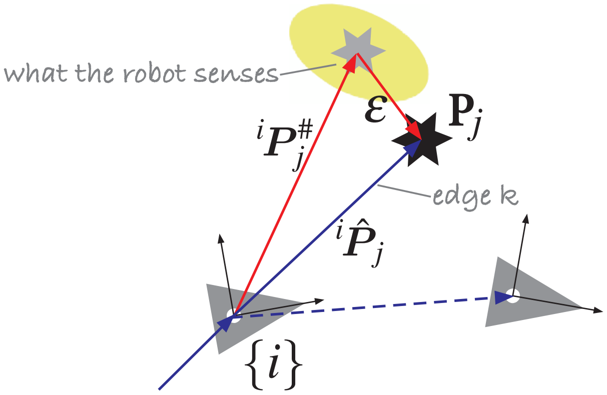


Fig. 6.18. Notation for a pose graph with a landmark indicated by the *star-shaped symbol*. The measured position of landmark j with respect to robot pose i is ${}^{i}\boldsymbol{P}_j^{\#}$. The *yellow ellipse* indicates uncertainty associated with that measurement

The estimated position of the landmark in frame $\{i\}$ is

$${}^{i}\hat{\boldsymbol{P}}_j = \left(\ominus {}^{0}\xi_i\right) \bullet \hat{\boldsymbol{P}}_j \in \mathbb{R}^2$$

and the error vector is

$$\boldsymbol{f}_k(\boldsymbol{x}) = \varepsilon = {}^{i}\hat{\boldsymbol{P}}_j - {}^{i}\boldsymbol{P}_j^{\#} \in \mathbb{R}^2$$

We follow a similar approach as earlier but the Jacobian matrix is now

$$\boldsymbol{J}_k = \frac{\partial \boldsymbol{f}_k(\boldsymbol{x})}{\partial \boldsymbol{x}} \in \mathbb{R}^{2\times(3N+2M)}$$

which again is mostly zero but the two nonzero blocks now have different widths

$$\boldsymbol{A}_i = \frac{\partial \boldsymbol{f}_k(\boldsymbol{x})}{\partial \xi_i} \in \mathbb{R}^{2\times 3},\ \boldsymbol{B}_j = \frac{\partial \boldsymbol{f}_k(\boldsymbol{x})}{\partial P_j} \in \mathbb{R}^{2\times 2}$$

and the solution can be achieved as before, see Sect. F.2.3 for more details.

Pose graph optimization results in a graph that has optimal relative poses and positions between the nodes but the absolute poses and positions are not necessarily correct. To remedy this we can fix or *anchor* one or more nodes (poses or landmarks) and not update them during the optimization, and this is discussed in Sect. F.2.4.

In practice the front and back ends can operate asynchronously. The graph is continually extended by the front end while the back end runs periodically to opti-

Monte Carlo methods are a class of computational algorithms that rely on repeated random sampling to compute their results. An early example of this idea is Buffon's needle problem posed in the eighteenth century by Georges-Louis Leclerc (1707–1788), Comte de Buffon: *Suppose we have a floor made of parallel strips of wood of equal width t, and a needle of length l is dropped onto the floor. What is the probability that the needle will lie across a line between the strips?* If n needles are dropped and h cross the lines, the probability can be shown to be $h / n = 2l / \pi t$ and in 1901 an Italian mathematician Mario Lazzarini performed the experiment, tossing a needle 3 408 times, and obtained the estimate $\pi \approx 355 / 113$ (3.14159292).

Monte Carlo methods are often used when simulating systems with a large number of coupled degrees of freedom with significant uncertainty in inputs. Monte Carlo methods tend to be used when it is infeasible or impossible to compute an exact result with a deterministic algorithm. Their reliance on repeated computation and random or pseudo-random numbers make them well suited to calculation by a computer. The method was developed at Los Alamos as part of the Manhattan project during WW II by the mathematicians John von Neumann, Stanislaw Ulam and Nicholas Metropolis. The name Monte Carlo alludes to games of chance and was the code name for the secret project.

mize the pose graph. Since the graph is only ever extended in a local region it is possible to optimize just a local subset of the pose graph and less frequently optimize the entire graph. If nodes are found to be equivalent after optimization they can be merged. The parallel tracking and mapping system (PTAM) is a vision-based SLAM system that has two parallel computational threads. One is the map builder which performs the front- and back-end tasks, adding landmarks to the pose graph based on estimated camera (vehicle) pose and performing graph optimization. The other thread is the localizer which matches observed landmarks to the estimated map to estimate the camera pose.

6.7 Sequential Monte-Carlo Localization

The estimation examples so far have assumed that the error in sensors such as odometry and landmark range and bearing have a Gaussian probability density function. In practice we might find that a sensor has a one sided distribution (like a Poisson distribution) or a multi-modal distribution with several peaks. The functions we used in the Kalman filter such as Eq. 6.2 and Eq. 6.7 are strongly nonlinear which means that sensor noise with a Gaussian distribution will not result in a Gaussian error distribution on the value of the function – this is discussed further in Appendix H. The probability density function associated with a robot's configuration may have multiple peaks to reflect several hypotheses that equally well explain the data from the sensors as shown for example in Fig. 6.3c.

The Monte-Carlo estimator that we discuss in this section makes no assumptions about the distribution of errors. It can also handle multiple hypotheses for the state of the system. The basic idea is disarmingly simple. We maintain many *different* values of the vehicle's configuration or state vector. When a new measurement is available we score how well each particular value of the state explains what the sensor just observed. We keep the best fitting states and randomly sample from the prediction distribution to form a new generation of states. Collectively these many possible states and their scores form a discrete approximation of the probability density function of the state we are trying to estimate. There is never any assumption about Gaussian distributions nor any need to linearize the system. While computationally expensive it is quite feasible to use this technique with today's standard computers. If we plot these state vectors as points in the state space we have a cloud of particles hence this type of estimator is often referred to as a particle filter.

We will apply Monte-Carlo estimation to the problem of localization using odometry and a map. Estimating only three states $\boldsymbol{x} = (x, y, \theta)$ is computationally tractable to solve with straightforward MATLAB code. The estimator is initialized by creating N particles $\boldsymbol{x}_i$, $i \in [1, N]$ distributed randomly over the configuration space of the vehicle. All particles have the same initial weight or likelihood $w_i = 1 / N$. The steps in the main iteration of the algorithm are:

1. Apply the state update to each particle

$$\boldsymbol{x}_i^+\langle k+1\rangle = \boldsymbol{f}\big(\boldsymbol{x}_i\langle k\rangle, \boldsymbol{u}\langle k\rangle + \boldsymbol{q}\langle k\rangle\big)$$

where $\hat{\boldsymbol{u}}\langle k\rangle$ is the input to the system or the measured odometry $\hat{\boldsymbol{u}}\langle k\rangle = \delta\langle k\rangle$. We also add a random vector $\boldsymbol{q}\langle k\rangle$ which represents uncertainty in the model or the odometry. Often $\boldsymbol{q}$ is drawn from a Gaussian random variable with covariance $\boldsymbol{Q}$ but any physically meaningful distribution can be used. The state update is often simplified to

$$\boldsymbol{x}_i^+\langle k+1\rangle = \boldsymbol{f}\big(\boldsymbol{x}_i\langle k\rangle, \boldsymbol{u}\langle k\rangle\big) + \boldsymbol{q}\langle k\rangle$$

where $\boldsymbol{q}\langle k\rangle$ represents uncertainty in the pose of the vehicle.

2. We make an observation $\boldsymbol{z}^{\#}$ of landmark j which has, according to the map, coordinate $\boldsymbol{p}_j$. For each particle we compute the innovation

$$\boldsymbol{\nu}_i = \boldsymbol{h}\left(\boldsymbol{x}_i^{+}, \boldsymbol{p}_j\right) - \boldsymbol{z}^{\#}$$

which is the error between the predicted and actual landmark observation. A likelihood function provides a scalar measure of how well the particular particle explains this observation. In this example we choose a likelihood function

$$w_i = \mathrm{e}^{-\nu_i^T L^{-1} \nu_i} + w_0$$

where w is referred to as the *importance* or *weight* of the particle, $\boldsymbol{L}$ is a covariance-like matrix, and $w_0 > 0$ ensures that there is a finite probability of a particle being retained despite sensor error. We use a quadratic exponential function only for convenience, the function does not need to be smooth or invertible but only to adequately describe the likelihood of an observation.▸

In this bootstrap type filter the weight is computed at each step, with no dependence on previous values.

3. Select the particles that best explain the observation, a process known as resampling▸ or importance sampling. A common scheme is to randomly select particles according to their weight. Given N particles $\boldsymbol{x}_i$ with corresponding weights w_i we first normalize the weights $w_i' = w_i / \Sigma_{i=1}^{N} w_i$ and construct a cumulative histogram $c_j = \Sigma_{i=1}^{j} w_i'$. We then draw a uniform random number $r \in [0, 1]$ and find

$$\arg\min_{i} r < c_i$$

where particle i is selected for the next generation. The process is repeated N times.

Particles with a large weight will correspond to a larger fraction of the vertical span of the cumulative histogram and therefore be more likely to be chosen. The result will have the same number of particles, some will have been copied▸ multiple times, others not at all. Resampling is a critical component of the particle filter without which the filter would quickly produce a degenerate set of particles where a few have high weights and the bulk have almost zero weight.

There are many resampling strategies for particle filters, both the resampling algorithm and the resampling frequency. Here we use the simplest strategy known variously as multinomial resampling, simple random resampling or select with replacement, at every time step. This is sometimes referred to as bootstrap particle filtering or condensation.

Step 1 of the next iteration will *spread out* these copies through the addition of $\boldsymbol{q}_{\langle k \rangle}$.

These steps are summarized in Fig. 6.19. The Toolbox implementation is broadly similar to the previous examples. We create a map

```
>> map = LandmarkMap(20);
```

and a robot with noisy odometry and an initial condition

```
>> W = diag([0.1, 1*pi/180].^2);
>> veh = Bicycle('covar', V);
>> veh.add_driver( RandomPath(10) );
```

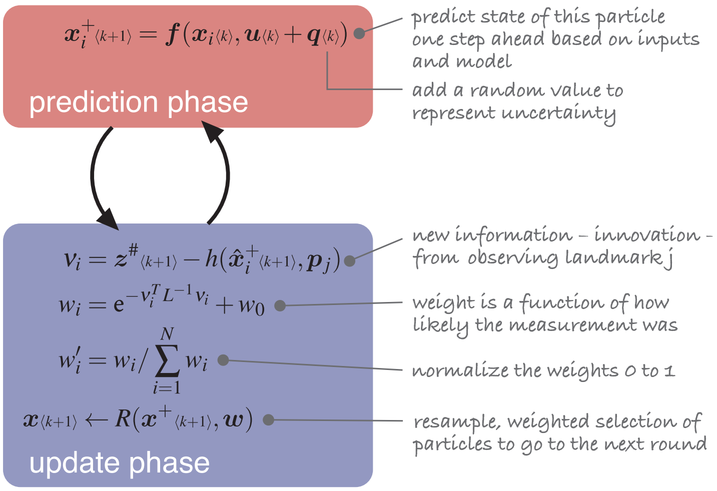

$\boldsymbol{x}_{i\langle k+1\rangle}^{+} = \boldsymbol{f}(\boldsymbol{x}_{i\langle k\rangle}, \boldsymbol{u}_{\langle k\rangle} + \boldsymbol{q}_{\langle k\rangle})$
prediction phase
predict state of this particle one step ahead based on inputs and model
add a random value to represent uncertainty
$\nu_i = \boldsymbol{z}^{\#}_{\langle k+1\rangle} - h(\hat{\boldsymbol{x}}_{i\langle k+1\rangle}^{+}, \boldsymbol{p}_j)$
new information – innovation – from observing landmark j
$w_i = \mathrm{e}^{-\nu_i^T L^{-1} \nu_i} + w_0$
weight is a function of how likely the measurement was
$w_i' = w_i / \sum_{i=1}^{N} w_i$
normalize the weights 0 to 1
$\boldsymbol{x}_{\langle k+1\rangle} \leftarrow R(\boldsymbol{x}^{+}_{\langle k+1\rangle}, \boldsymbol{w})$
resample, weighted selection of particles to go to the next round
update phase

Fig. 6.19. The particle filter estimator showing the prediction and update phases

and then a sensor with noisy readings

```
>> V = diag([0.005, 0.5*pi/180].^2);
>> sensor = RangeBearingSensor(veh, map, 'covar', W);
```

For the particle filter we need to define two covariance matrices. The first is the covariance of the random noise added to the particle states at each iteration to represent uncertainty in configuration. We choose the covariance values to be comparable with those of W

```
>> Q = diag([0.1, 0.1, 1*pi/180]).^2;
```

and the covariance of the likelihood function applied to innovation

```
>> L = diag([0.1 0.1]);
```

Finally we construct a `ParticleFilter` estimator

```
>> pf = ParticleFilter(veh, sensor, Q, L, 1000);
```

which is configured with 1 000 particles. The particles are initially uniformly distributed over the 3-dimensional configuration space.

We run the simulation for 1 000 time steps

```
>> pf.run(1000);
```

and watch the animation, two snapshots of which are shown in Fig. 6.20. We see the particles move about as their states are updated by odometry and random perturbation. The initially randomly distributed particles begin to aggregate around those regions of the configuration space that best *explain* the sensor observations that are made. In Darwinian fashion these particles become more highly weighted and survive the resampling step while the lower weight particles are extinguished.

The particles approximate the probability density function of the robot's configuration. The most likely configuration is the expected value or mean of all the particles. A measure of uncertainty of the estimate is the spread of the particle cloud or its standard deviation. The `ParticleFilter` object keeps the history of the mean and standard deviation of the particle state at each time step, taking into account the particle weighting. As usual we plot the results of the simulation

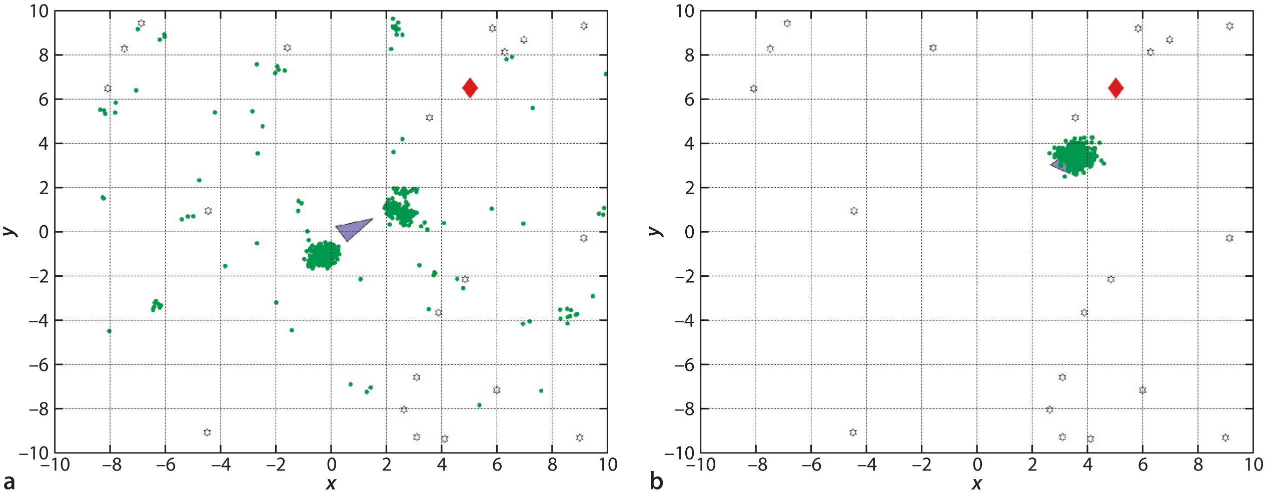
Here we take statistics over all particles. Other strategies are to estimate the kernel density at every particle – the sum of the weights of all neighbors within a fixed radius – and take the particle with the largest value.

```
>> map.plot();
>> veh.plot_xy('b');
```

and overlay the mean of the particle cloud

```
>> pf.plot_xy('r');
```

Fig. 6.20. Particle filter results showing the evolution of the particle cloud (*green dots*) over time. The vehicle is shown as a *blue triangle*. The *red diamond* is a waypoint, or temporary goal. When the simulation is running this is actually a 3D plot with orientation plotted in the *z*-direction, rotate the plot to see this dimension



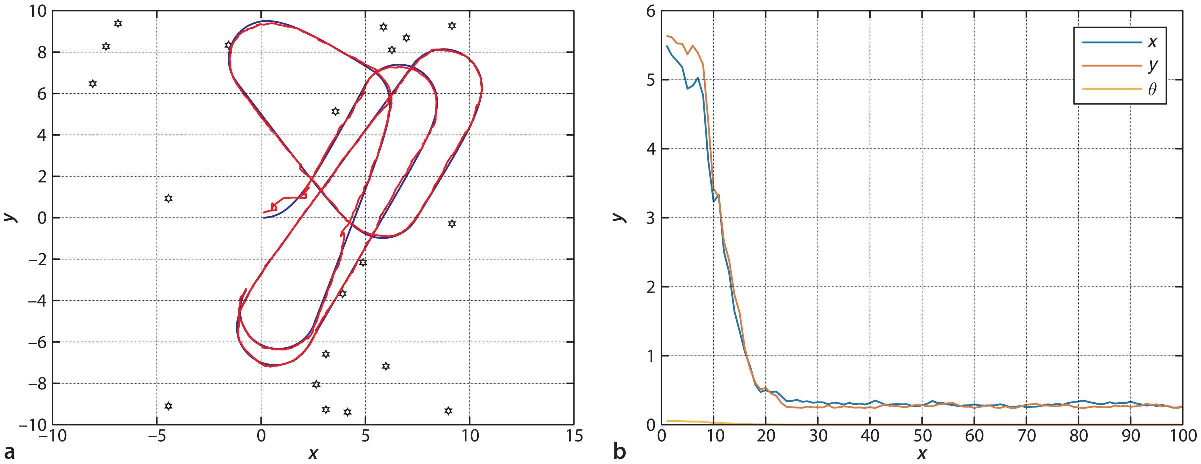


Fig. 6.21. Particle filter results. **a** True (*blue*) and estimated (*red*) robot path; **b** standard deviation of the particles versus time

which is shown in Fig. 6.21. The initial part of the estimated path has quite high standard deviation since the particles have not converged on the true configuration. We can plot the standard deviation against time

```
>> plot(pf.std(1:100,:))
```

and this is shown in Fig. 6.21b. We can see the sudden drop between timesteps 10–20 as the particles that are distant from the true solution are eliminated. As mentioned at the outset the particles are a sampled approximation to the PDF and we can display this as

```
>> pf.plot_pdf()
```

The problem we have just solved is known in robotics as the kidnapped robot problem where a robot is placed in the world with no idea of its initial location. To represent this large uncertainty we uniformly distribute the particles over the 3-dimensional configuration space and their sparsity can cause the particle filter to take a long time to converge unless a very large number of particles is used. It is debatable whether this is a realistic problem. Typically we have some approximate initial pose of the robot and the particles would be initialized to that part of the configuration space. For example, if we know the robot is in a corridor then the particles would be placed in those areas of the map that are corridors, or if we know the robot is pointing north then set all particles to have that orientation.

Setting the parameters of the particle filter requires a little experience and the best way to learn is to experiment. For the kidnapped robot problem we set $\boldsymbol{Q}$ and the number of particles high so that the particles explore the configuration space but once the filter has converged lower values could be used. There are many variations on the particle filter in the shape of the likelihood function and the resampling strategy.

6.8 Application: Scanning Laser Rangefinder

As we have seen, robot localization is informed by measurements of range and bearing to landmarks. Sensors that measure range can be based on many principles such as laser rangefinding (Fig. 6.22a, 6.22b), ultrasonic ranging (Fig. 6.22c), computer vision or radar.

A laser rangefinder emits short pulses of infra-red laser light and measures how long it takes for the reflected pulse to return. Operating range can be up to 50 m with an accuracy of the order of centimeters.

A *scanning* laser rangefinder, as shown in Fig. 6.22a, contains a rotating laser rangefinder and typically emits a pulse every quarter, half or one degree over an angular range of 180 or 270 degrees and returns a planar cross-section of the world in polar coordinate form $\{(r_i, \theta_i), i \in 1 \cdots N\}$. Some scanning laser rangefinders also measure

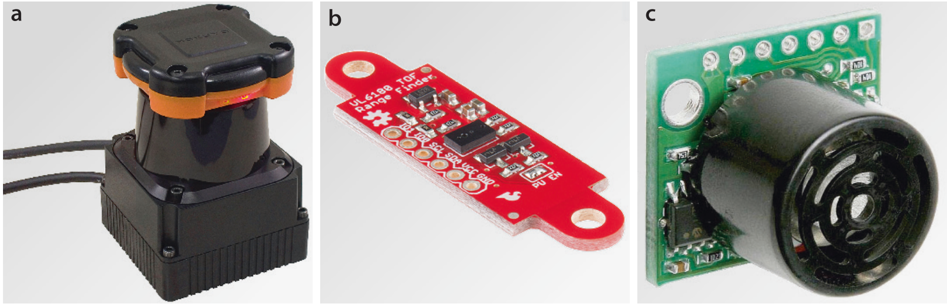


Fig. 6.22. Robot rangefinders. **a** A scanning laser rangefinder with a maximum range of 30 m, an angular range of 270 deg in 0.25 deg intervals at 40 scans per second (courtesy of Hokuyo Automatic Co. Ltd.); **b** a low-cost time-of-flight rangefinder with maximum range of 20 cm at 10 measurements per second (VL6180 courtesy of SparkFun Electronics); **c** a low-cost ultrasonic rangefinder with maximum range of 6.5 m at 20 measurements per second (LV-MaxSonar-EZ1 courtesy of SparkFun Electronics)

the return signal strength, remission, which is a function of the infra-red reflectivity of the surface. The rangefinder is typically configured to scan in a plane parallel to, and slightly above, the ground.

Laser rangefinders have advantages and disadvantages compared to cameras and computer vision which we discuss in Parts IV and V of this book. On the positive side laser scanners provide metric data, that is, the actual range to points in the world in units of meters, and they can work in the dark. However laser rangefinders work less well than cameras outdoors since the returning laser pulse is overwhelmed by infra-red light from the sun. Other disadvantages include providing only a linear cross section of the world, rather than an area as a camera does; inability to discern fine texture or color; having moving parts; as well as being bulky, power hungry and expensive compared to cameras.

Laser Odometry

A common application of scanning laser rangefinders is laser odometry, estimating the change in robot pose using laser scan data rather than wheel encoder data. We will illustrate this with laser scan data from a real robot

```
>> pg = PoseGraph('killian.g2o', 'laser');
loaded g2o format file: 3873 nodes, 4987 edges in 1.78 sec
  3873 laser scans: 180 beams, fov -90 to 90 deg, max range 50
```

and each scan is associated with a vertex of this already optimized pose graph. The range and bearing data for the scan at node 2 580 is

```
>> [r, theta] = pg.scan(2580);
>> about r theta
r [double] : 1x180 (1.4 kB)
theta [double] : 1x180 (1.4 kB)
```

represented by two vectors each of 180 elements. We can plot these in polar form

```
>> polar(theta, r)
```

or convert them to Cartesian coordinates and plot them

```
>> [x,y] = pol2cart(theta, r);
>> plot(x, y, '.')
```

The method `scanxy` is a simpler way to perform these operations. We load scans from two closely spaced nodes

```
>> p2580 = pg.scanxy(2580);
>> p2581 = pg.scanxy(2581);
>> about p2580
p2580 [double] : 2x180 (2.9 kB)
```

which creates two matrices whose columns are Cartesian point coordinates and these are overlaid in Fig. 6.23a.◄

Note that the points close to the laser, at coordinate (0,0) in this sensor reference frame are much more tightly clustered and this is a characteristic of laser scanners where the points are equally spaced in angle not over an area.

To determine the change in pose of the robot between the two scans we need to align these two sets of points and this can be achieved with iterated closest-point-matching

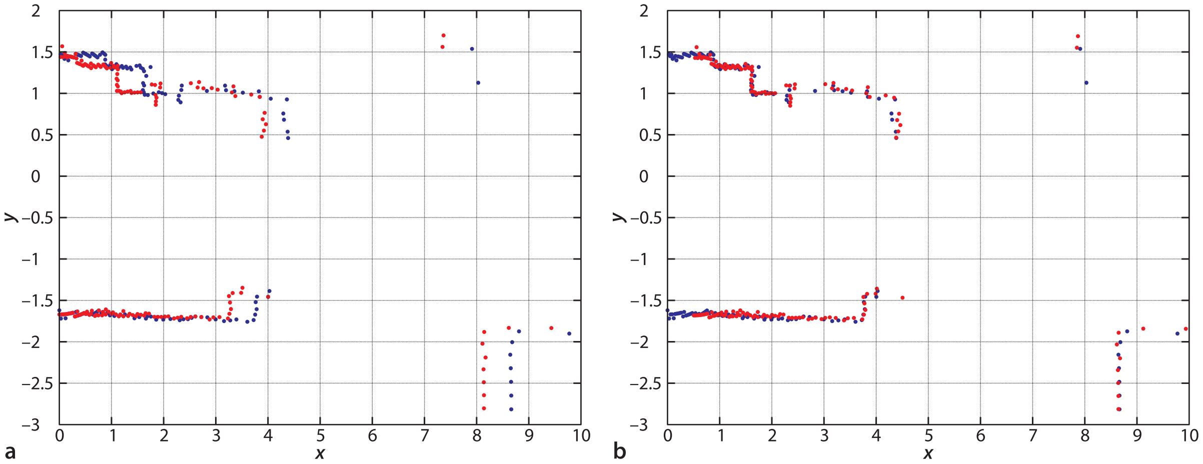


Fig. 6.23. Laser scan matching. **a** Laser scans from location 2 580 (*blue*) and 2 581 (*red*); **b** location 2 580 points (*blue*) and transformed points from location 2 581 (*red*)

or ICP. This is implemented by the Toolbox function `icp` and we pass in the second and first set of points, each organized as a $2 \times N$ matrix

```
>> T = icp( p2581, p2580, 'verbose' , 'T0', transl2(0.5, 0), 'distthresh', 3)
[1]: n=132/180, d=   0.466, t = (   0.499   -0.006), th = (  -0.0) deg
[2]: n=130/180, d=   0.429, t = (   0.500   -0.009), th = (   0.0) deg
   .
   .
[6]: n=130/180, d=   0.425, t = (   0.503   -0.011), th = (   0.0) deg

T =
    1.0000   -0.0002    0.5032
    0.0002    1.0000   -0.0113
         0         0    1.0000
```

The ICP algorithm is described more fully for the **SE**(3) case in Sect. 14.5.2.

and the algorithm converges after a few iterations with an estimate of $T \sim {}^{2580}\xi_{2581} \in \mathbf{SE}(2)$. This transform maps points from the second scan so that they are as close as possible to the points in the first scan. Figure 6.23b shows the first set of points transformed and overlaid on the second set and we see good alignment. The translational part of this transform is an estimate of the robot's motion between scans – around 0.50 m in the *x*-direction. The nodes of the graph also hold time stamp information and these two scans were captured

We demonstrate the principle using ICP but in practice more robust algorithms are used. Here we provide an initial estimate of the translation between frames, based on odometry, so as to avoid getting stuck in a local minimum. ICP works poorly in plain corridors where the points lie along lines – this example was deliberately chosen because it has wall segments in orthogonal directions.

```
>> pg.time(2581)-pg.time(2580)
ans =
    1.7600
```

seconds apart which indicates that the robot is moving quite slowly – a bit under 0.3 m s^{-1}.

At each iteration ICP assigns each point in the second set to the closest point in the first set and then computes a transform that minimizes the sum of distances between all corresponding points. Some points may not actually be corresponding but as long as enough are, the algorithm will converge. The `'verbose'` option causes data about each iteration to be displayed and `d` is the total distance between corresponding points which is decreasing but does not reach zero. This is due to many factors. The beams from the laser at the two different poses will not strike the walls at the same location so ICP's assumption about point correspondence is not actually valid.

To remove invalid correspondences we pass the `'distthresh'` option to `icp()`. This causes any correspondences that involve a distance more than three times the median distance between all corresponding points to be dropped. In the `icp()` output the notation `132/180` means that 132 out of 180 possible correspondences met this test, 48 were rejected.

In practice there are additional challenges. Some laser pulses will not return to the sensor if they fall on a surface with low reflectivity or on an oblique polished surface that specularly reflects the pulse away from the sensor – in these cases the sensor typically reports its maximum value. People moving through the environment change the shape of the world and temporarily cause a shorter range to be reported. In very large spaces all the walls may be beyond the maximum range of the sensor. Outdoors the beams can be reflected from rain drops, absorbed by fog or smoke and the return pulse can be overwhelmed by ambient sunlight. Finally the laser rangefinder, like all sensors, has measurement noise.

Laser-Based Map Building

If the robot pose is sufficiently well known, through some localization process, then we can transform all the laser scans to a global coordinate frame and build a map. Various map representations are possible but here we will outline how to build an occupancy grid as discussed in Chap. 5.

For a robot at a given pose, each beam in the scan is a ray and tells us several things. From the range measurement we can determine the coordinates of a cell that contains an obstacle but we can tell nothing about cells further along the ray. It is also implicit that all the cells between the sensor and the obtacle must be obstacle free. A maximum distance value, 50 m in this case, is the sensor's way of indicating that there was no returning laser pulse so we ignore all such measurements. We create the occupancy grid as a matrix and use the Bresenham algorithm to find all the cells along the ray based on the robot's pose and the laser range and bearing measurement, then a simple voting scheme to determine whether cells are free or occupied

```
>> pg.scanmap()
>> pg.plot_occgrid()
```

and the result is shown in Fig. 6.24. More sophisticated approaches treat the beam as a wedge of finite angular width and employ a probabilistic model of sensor return versus range. The principle can be extended to creating 3-dimensional point clouds from a scanning laser rangefinder on a moving vehicle as shown in Fig. 6.25.

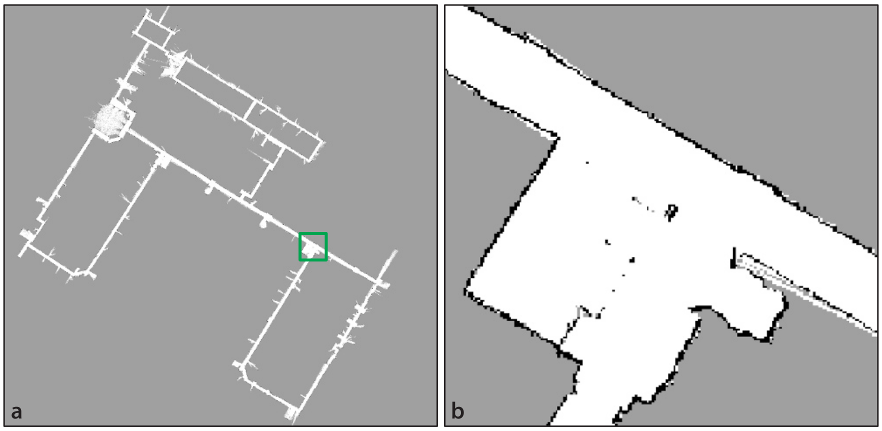


Fig. 6.24. **a** Laser scans rendered into an occupancy grid, the area enclosed in the *green square* is dispayed in **b**. *White cells* are free space, *black cells* are occupied and *grey cells* are unknown. Grid cell size is 10 cm

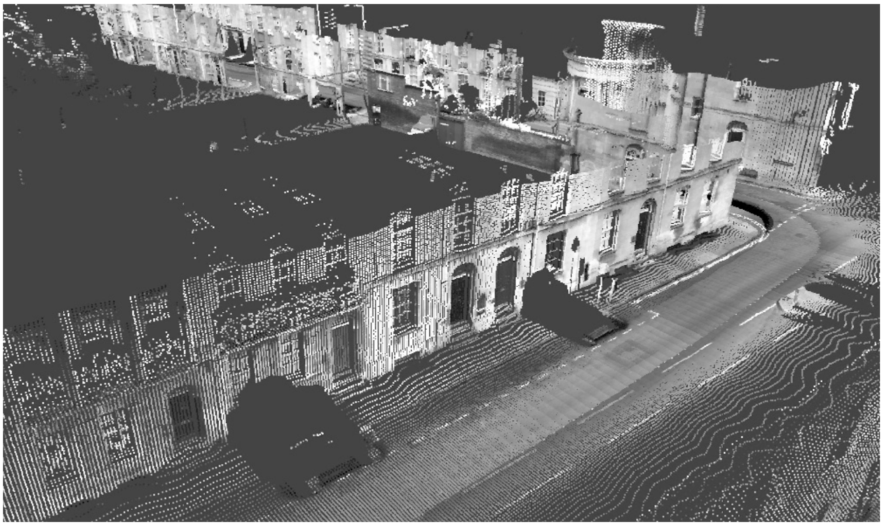


Fig. 6.25. 3D point cloud created by integrating multiple scans from a vehicle-mounted scanning laser rangefinder, where the scans are in a vertical plane normal to the vehicle's forward axis. This is sometimes called a "2.5D" representation since only the front surfaces of objects are described – note the range shadows on the walls behind cars. Note also that the density of laser points is not constant across the map, for example the point density on the road surface is much greater than it is high on the walls of buildings (image courtesy Alex Stewart; Stewart 2014)

Laser-Based Localization

We have mentioned landmarks a number of times in this chapter but avoided concrete examples of what they are. They could be distinctive visual features as discussed in Sect. 13.3 or artificial markers as discussed on page 164. If we consider a laser scan such as shown in Fig. 6.23a or 6.24b we see a fairly distinctive arrangement of points – a geometric signature – which we can use as a landmark. In many cases the signature will be ambiguous and of little value, for example a long corridor where all the points are collinear, but some signatures will be highly unique and can serve as a useful landmark. Naively we could match the current laser scan against all others and if the fit is good (the ICP error is low) we could add another constraint to the pose graph. However this strategy would be expensive with a large number of scans so typically only scans in the vicinity of the robot's estimated position are checked, and this once again raises the data association problem.

6.9 Wrapping Up

In this chapter we learned about two very different ways of estimating a robot's position: by dead reckoning, and by observing landmarks whose true position is known from a map. Dead reckoning is based on the integration of odometry information, the distance traveled and the change in heading angle. Over time errors accumulate leading to increased uncertainty about the pose of the robot.

We modeled the error in odometry by adding noise to the sensor outputs. The noise values are drawn from some distribution that describes the errors of that particular sensor. For our simulations we used zero-mean Gaussian noise with a specified covariance, but only because we had no other information about the specific sensor. The most realistic noise model available should be used. We then introduced the Kalman filter which provides an optimal estimate, in the least-squares sense, of the true configuration of the robot based on noisy measurements. The Kalman filter is however only optimal for the case of zero–mean Gaussian noise and a linear model. The model that describes how the robot's configuration evolves with time can be nonlinear in which case we approximate it with a linear model which included some partial derivatives expressed as Jacobian matrices – an approach known as extended Kalman filtering.

The Kalman filter also estimates uncertainty associated with the pose estimate and we see that the magnitude can never decrease and typically grows without bound. Only additional sources of information can reduce this growth and we looked at how observations of landmarks, with known locations, relative to the robot can be used. Once again we use the Kalman filter but in this case we use both the prediction and the update phases of the filter. We see that in this case the uncertainty can be decreased by a landmark observation, and that over the longer term the uncertainty does not grow. We then applied the Kalman filter to the problem of estimating the positions of the landmarks given that we knew the precise position of the vehicle. In this case, the state vector of the filter was the coordinates of the landmarks themselves.

Next we brought all this together and estimated the vehicle's position, the position of the landmarks and their uncertainties – simultaneous localization and mapping. The state vector in this case contained the configuration of the robot and the coordinates of the landmarks.

An important problem when using landmarks is data association, being able to determine which landmark has been known or observed by the sensor so that its position can be looked up in a map or in a table of known or estimated landmark positions. If the wrong landmark is looked up then an error will be introduced in the robot's position.

The Kalman filter scales poorly with an increasing number of landmarks and we introduced two alternative approaches: Rao-Blackwellized SLAM and pose-graph SLAM. The latter involves solving a large but sparse nonlinear least squares problem, turning the problem from one of (Kalman) filtering to one of optimization.

We finished our discussion of localization methods with Monte-Carlo estimation and introduced the particle filter. This technique is computationally intensive but makes no assumptions about the distribution of errors from the sensor or the linearity of the vehicle model, and supports multiple hypotheses. Particles filters can be considered as providing an approximate solution to the true system model, whereas a Kalman filter provides an exact solution to an approximate system model.

Finally we introduced laser rangefinders and showed how they can be applied to robot navigation, odometry and creating detailed floor plan maps.

Further Reading

Localization and SLAM. The tutorials by Bailey and Durrant-Whyte (2006) and Durrant-Whyte and Bailey (2006) are a good introduction to this topic, while the textbook *Probabilistic Robotics* (Thrun et al. 2005) is a readable and comprehensive coverage of all the material touched on in this chapter.

The book by Siegwart et al. (2011) also has a good treatment of robot localization. FastSLAM (Montemerlo et al. 2003; Montemerlo and Thrun 2007) is a state-of-the-art algorithm for Rao-Blackwellized SLAM.

Particle filters are described by Thrun et al. (2005), Stachniss and Burgard (2014) and the tutorial introduction by Rekleitis (2004). There are many variations such as fixed or adaptive number of particles and when and how to resample – and Li et al. (2015) provide a comprehensive review of resampling strategies. Determining the most likely pose was demonstrated by taking the weighted mean of the particles but many more approaches have been used. The kernel density approach takes the particle with the highest weight of neighboring particles within a fixed-size surrounding hypersphere.

Pose graph optimization, also known as GraphSLAM, has a long history starting with Lu and Milios (1997). There has been significant recent interest with many publications and open-source tools including g^{2}o (Kümmerle et al. 2011), $\sqrt{\text{SAM}}$ (Dellaert and Kaess 2006), iSAM (Kaess et al. 2007) and factor graphs. Agarwal et al. (2014) provides a gentle introduction to pose-graph SLAM and discusses the connection to land-based geodetic survey which is centuries old. Parallel Tracking and Mapping (PTAM) was described in Klein and Murray (2007), the code is available on github and there is also a blog.

There are many online resources related to SLAM. A collection of open-source SLAM implementations such as gmapping and iSam is available from OpenSLAM at http://www.openslam.org. An implementation of smoothing and mapping using factor graphs is available at https://bitbucket.org/gtborg/gtsam and has C++ and MATLAB bindings. MATLAB implementations include a 6DOF SLAM system at http://www.iri.upc.edu/people/jsola/JoanSola/eng/toolbox.html and the now dated CAS Robot Navigation Toolbox for planar SLAM at http://www.cas.kth.se/toolbox. Tim Bailey's website http://www-personal.acfr.usyd.edu.au/tbailey has MATLAB implementations of various SLAM and scan matching algorithms.

Many of the SLAM summer schools have websites that host excellent online resources such as lecture notes and practicals. Great teaching resources available online include Giorgio Grisetti's site http://www.dis.uniroma1.it/~grisetti and Paul Newman's *C4B Mobile Robots and Estimation Resources* ebook at https://www.free-ebooks.net/ebook/C4B-Mobile-Robotics.

Scan matching and map making. Many versions and variants of the ICP algorithm exist and it is discussed further in Chap. 14. Improved convergence and accuracy can be obtained using the normal distribution transform (NDT), originally proposed for 2D by Biber and Straßer (2003), extended to 3D by Magnusson et al. (2007) and implementations are available at pointclouds.org. A comparison of ICP and NDT for a field robotic application is described by Magnusson et al. (2009). A fast and popular approach to laser scan matching is that of Censi (2008).

When attempting to match a local geometric signature in a large point cloud (2D or 3D) to determine loop closure we often wish to limit our search to a local spatial region. An efficient way to achieve this is to organize the data using a kd-tree which is provided in MATLAB's Statistics and Machine Learning Toolbox™ and various contributions on File Exchange. FLANN (Muja and Lowe 2009) is a fast approximation which is available on github and has a MATLAB binding, and is also included in the VLFeat package.

For creating a map from robotic laser scan data in Sect. 6.8 we used a naive approach – a more sophisticated technique is the beam model or likelihood field as described in Thrun et al. (2005).

Kalman filtering. There are many published and online resources for Kalman filtering. Kálmán's original paper, Kálmán (1960), over 50 years old, is quite readable. The book by Zarchan and Musoff (2005) is a very clear and readable introduction to Kalman filtering. I have always found the classic book, recently republished, Jazwinski (2007) to be very readable. Bar-Shalom et al. (2001) provide comprehensive coverage of estimation theory and also the use of GPS. Groves (2013) also covers Kalman filtering. Welch and Bishop's online resources at http://www.cs.unc.edu/~welch/kalman have pointers to papers, courses, software and links to other relevant web sites.

A significant limitation of the EKF is its first-order linearization, particularly for processes with strong nonlinearity. Alternatives include the iterated EKF described by Jazwinski (2007) or the Unscented Kalman Filter (UKF) (Julier and Uhlmann 2004) which uses discrete sample points (sigma points) to approximate the PDF. Some of these topics are covered in the Handbook (Siciliano and Khatib 2016, §5 and §35). The information filter is an equivalent filter that maintains an inverse covariance matrix which has some useful properties, and is discussed in Thrun et al. (2005) as the sparse extended information filter.

Data association. SLAM techniques are critically dependent on accurate data association between observations and mapped landmarks, and a review of data association techniques is given by Neira and Tardós (2001). FastSLAM (Montemerlo and Thrun 2007) is capable of estimating data association as well as landmark position. The April tag which can be used as an artificial landmark is described in Olson (2011) and is supported by the Toolbox function `apriltags`. Mobile robots can uniquely identify places based on their visual appearance using tools such as OpenFABMAP (Glover et al. 2012).

Data association for Kalman filtering is covered in the Robotics Handbook (Siciliano and Khatib 2016). Data association in the tracking context is covered in considerable detail in, the now very old, book by Bar-Shalom and Fortmann (1988).

Sensors. The book by Kelly (2013) has a good coverage of sensors particularly laser range finders. For flying and underwater vehicles, odometry cannot be determined from wheel motion and an alternative, also suitable for wheeled vehicles, is visual odometry (VO). This is introduced in the tutorials by Fraundorfer and Scaramuzza (2012) and Scaramuzza and Fraundorfer (2011) and will be covered in Chap. 14. The Robotics Handbook (Siciliano and Khatib 2016) has good coverage of a wide range of robotic sensors. The principles of GPS and other radio-based localization systems are covered in some detail in the book by Groves (2013), and a number of links to GPS technical data are provided from this book's web site. The SLAM problem can be formulated in terms of bearing-only or range-only measurements. A camera is effectively a bearing-only sensor, giving the direction to a feature in the world. A VSLAM system is one that performs SLAM using bearing-only visual information, just a camera, and an introduction to the topic is given by Neira et al. (2008) and the associated special issue. Interestingly the robotic VSLAM problem is the same as the bundle adjustment problem known to the computer vision community and which will be discussed in Chap. 14.

The book by Borenstein et al. (1996) although dated has an excellent discussion of robotic sensors in general and odometry in particular. It is out of print but can be found

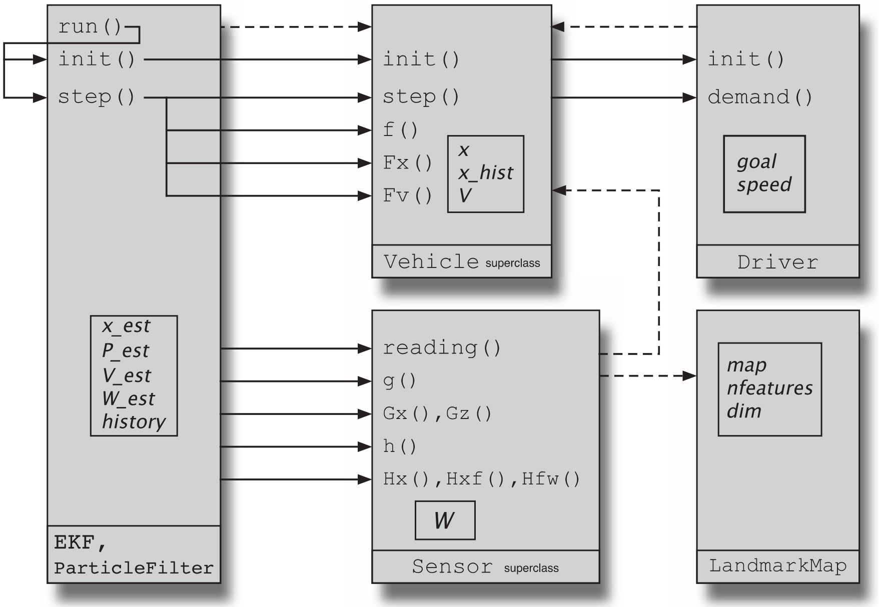


Fig. 6.26. Toolbox class relationship for localization and mapping. Each class is shown as a *rectangle*, method calls are shown as *arrows* from caller to callee, properties are *boxed*, and *dashed lines* represent object references

online. The book by Everett (1995) covers odometry, range and bearing sensors, as well as radio, ultrasonic and optical localization systems. Unfortunately the discussion of range and bearing sensors is now quite dated since this technology has evolved rapidly over the last decade.

General interest. Bray (2014) gives a very readable account of the history of techniques to determine our location on the planet. If you ever wondered how to navigate by the stars or use a sextant Blewitt (2011) is a slim book that provides an easy to understand introduction.

The book *Longitude* (Sobel 1996) is a very readable account of the longitude problem and John Harrison's quest to build a marine chronometer.

Toolbox and MATLAB Notes

This chapter has introduced a number of Toolbox classes to solve mapping and localization problems. The principle was to decompose the problem into clear functional subsystems and implement these as a set of cooperating classes, and this allows quite complex problems to be expressed in very few lines of code.

The relationships between the objects and their methods and properties are shown in Fig. 6.26. As always more documentation is available through the online help system or comments in the code. `Vehicle` is a superclass and concrete subclasses include `Unicycle` and `Bicycle`.

The MATLAB Computer Vision System Toolbox™ includes a fast version of ICP called `pcregrigid`. The Robotics System Toolbox™ contains a generic particle filter class `ParticleFilter` and a particle filter based localizer class `MonteCarloLocalization`.

Exercises

1. What is the value of the Longitude Prize in today's currency?
2. Implement a driver object (page 157) that drives the robot around inside a circle with specified center and radius.
3. Derive an equation for heading change in terms of the rotational rate of the left and right wheels for the car-like and differential-steer vehicle models.

4. Dead-reckoning (page 156)
   a) Experiment with different values of $\boldsymbol{P}_0$, $\boldsymbol{V}$ and $\hat{\boldsymbol{V}}$.
   b) Figure 6.4 compares the actual and estimated position. Plot the actual and estimated heading angle.
   c) Compare the variance associated with heading to the variance associated with position. How do these change with increasing levels of range and bearing angle variance in the sensor?
   d) Derive the Jacobians in Eq. 6.5 and 6.6 for the case of a differential steer robot.
5. Using a map (page 163)
   a) Vary the characteristics of the sensor (covariance, sample rate, range limits and bearing angle limits) and investigate the effect on performance
   b) Vary $\boldsymbol{W}$ and $\hat{\boldsymbol{W}}$ and investigate what happens to estimation error and final covariance.
   c) Modify the `RangeBearingSensor` to create a bearing-only sensor, that is, as a sensor that returns angle but not range. The implementation includes all the Jacobians. Investigate performance.
   d) Modify the sensor model to return occasional errors (specify the error rate) such as incorrect range or beacon identity. What happens?
   e) Modify the EKF to perform data association instead of using the landmark identity returned by the sensor.
   f) Figure 6.7 compares the actual and estimated position. Plot the actual and estimated heading angle.
   g) Compare the variance associated with heading to the variance associated with position. How do these change with increasing levels of range and bearing angle variance in the sensor?
6. Making a map (page 166)
   a) Vary the characteristics of the sensor (covariance, sample rate, range limits and bearing angle limits) and investigate the effect on performance.
   b) Use the bearing-only sensor from above and investigate performance relative to using a range and bearing sensor.
   c) Modify the EKF to perform data association instead of using identity returned by the sensor.
7. Simultaneous localization and mapping (page 168)
   a) Vary the characteristics of the sensor (covariance, sample rate, range limits and bearing angle limits) and investigate the effect on performance.
   b) Use the bearing-only sensor from above and investigate performance relative to using a range and bearing sensor.
   c) Modify the EKF to perform data association instead of using the landmark identity returned by the sensor.
   d) Figure 6.11 compares the actual and estimated position. Plot the actual and estimated heading angle.
   e) Compare the variance associated with heading to the variance associated with position. How do these change with increasing levels of range and bearing angle variance in the sensor?
8. Modify the pose-graph optimizer and test using the simple graph `pg1.g2o`
   a) anchor one node at a particular pose.
   b) add one or more landmarks. You will need to derive the relevant Jacobians first and add the landmark positions, constraints and information matrix to the data file.
9. Create a simulator for Buffon's needle problem, and estimate π for 10, 100, 1 000 and 10 000 needle throws. How does convergence change with needle length?
10. Particle filter (page 176)
    a) Run the filter numerous times. Does it always converge?
    b) Vary the parameters $\boldsymbol{Q}$, $\boldsymbol{L}$, w_0 and N and understand their effect on convergence speed and final standard deviation.

c) Investigate variations to the kidnapped robot problem. Place the initial particles around the initial pose. Place the particles uniformly over the *xy*-plane but set their orientation to its actual value.

d) Use a different type of likelihood function, perhaps inverse distance, and compare performance.

11. Experiment with April tags. Print some tags and extract them from images using the `apriltags` function. Check out Sect. 12.1 on how to acquire images using MATLAB.
12. Implement a laser odometer and test it over the entire path saved in `killian.g2o`. Compare your odometer with the relative pose changes in the file.
13. In order to measure distance using laser rangefinding what timing accuracy is required to achieve 1cm depth resolution?
14. Reformulate the localization, mapping and SLAM problems using a bearing-only landmark sensor.
15. Implement a localization or SLAM system using an external simulator such as V-REP or Gazebo. Obtain range measurements from the simulated robot, do laser odometry and landmark recognition, and send motion commands to the robot. You can communicate with these simulators from MATLAB using the ROS protocol if you have the Robotics System Toolbox. Alternatively you can communicate with V-REP using the Toolbox `VREP` class, see the documentation.