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 human-kind 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 travelled 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

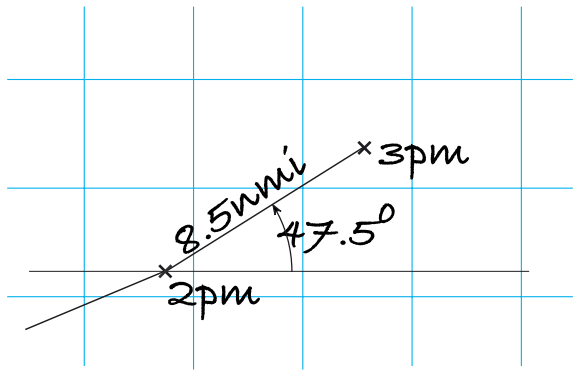


Fig. 6.1. Location estimation by dead reckoning. The ship's position at 3 P.M. is based on its position at 2 P.M., the estimated distance travelled since, and the average compass heading

Measuring speed at sea. A ship's log is an instrument that provides an estimate of the distance travelled. 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.

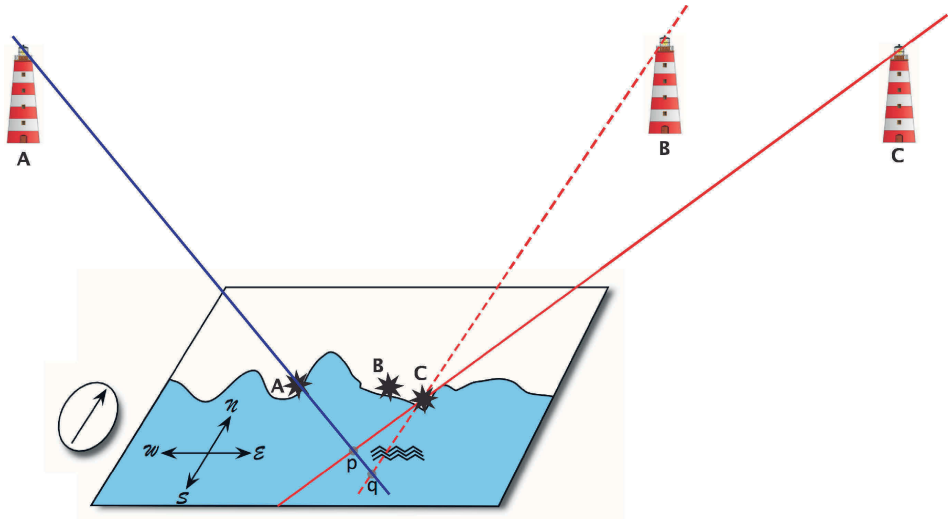


Fig. 6.2.
Location estimation using a map. Lines of sight from two lighthouses, **A** and **C**, and their corresponding locations on the map provide an estimate **p** of our location. However if we mistake lighthouse **C** for **B** then we obtain an incorrect estimate **q**

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 one, means that errors will accumulate over time and for sea voyages of many-months 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 whereever 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.

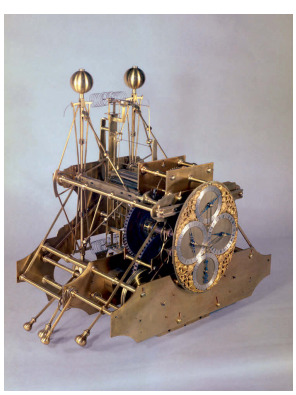
A landmark is a visible feature in the environment whose location is known with respect to some coordinate frame. Figure 6.2 shows schematically a map and a number of lighthouse landmarks. We first of all use a compass to align the north axis of our map with the direction of the north pole. The direction 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 **B** 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.

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 lattitude to be determined very conveniently from a single sextant measurememt.

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. This 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.



Harrison's H1 chronometer (1735), © National Maritime Museum, Greenwich, London

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 currently 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 travelled is longer and this will introduce an error in the position estimate. This effect is known as multi-pathing and is common 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 to compatible nearby receivers which can offset the error – this is known as Differential GPS (DGPS). Many countries have coastal radio networks that broadcast this correction information, and for aircraft it is broadcast by another satellite network called the Wide Area Augmentation System (WAAS). 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.

It is just as bad is to see **C** but think it is **B** on the map.

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 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.

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.

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.

A wheeled robot can estimate distance travelled by measuring wheel rotation, but an aerial or underwater robot cannot do this. Wheel rotation is imperfect due to variation and uncertainty in wheel radius, slippage and the effects of turning. Computer vision can be used to create a visual odometry system based on observations of the world moving past the robot.

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 travelled will be imperfect◂ and it may have no map, or perhaps an imperfect or incomplete map. Additional information from observation of features 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 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 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

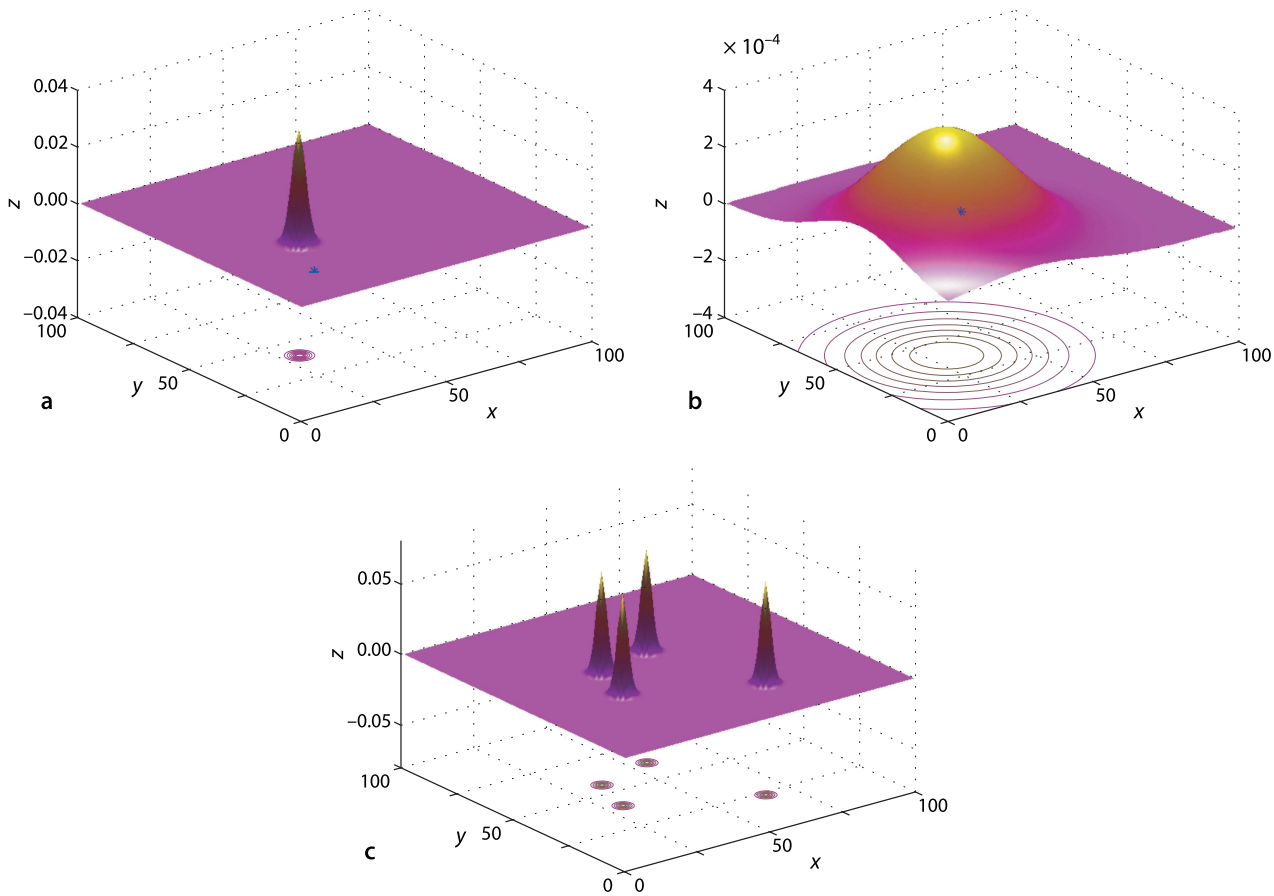


Fig. 6.3. Notions of vehicle position and uncertainty in the *xy*-plane, where the vertical axis is the relative likelihood of the vehicle being at that position. Contour lines are displayed on the lower plane. **a** The vehicle has low position uncertainty, $\sigma = 1$; **b** the vehicle has much higher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

probability that the vehicle is at the point indicated by the $*$-marker. 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 $*$-marker. 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. 6.5.

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 usefully 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 location is 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 F. The diagonal elements represent uncertainty of the corresponding states, and the off-diagonal elements represent correlations between states.

6.1 Dead Reckoning

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

6.1.1 Modeling the Vehicle

The first step in estimating the robot's position 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 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.

An odometer is a sensor that measures distance travelled, typically 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, or the effect of turning. We consider odometry to comprise both distance and heading information.

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

When turning, the outside wheel travels faster than the inside wheel, and this can be accounted for by measuring the speed of both wheels.

Instead of using Eq. 4.2 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 travelled and change in heading over the preceding interval, and k is the time step. The initial pose is represented in $SE(2)$ 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 assume 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

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

or as a 3-vector

$$\xi\langle k+1\rangle \sim \begin{pmatrix} x\langle k\rangle + \delta_d\langle k\rangle \cos\big(\theta\langle k\rangle + \delta_\theta\big) \\ y\langle k\rangle + \delta_d\langle k\rangle \sin\big(\theta\langle k\rangle + \delta_\theta\big) \\ \theta\langle k\rangle + \delta_\theta \end{pmatrix} \tag{6.1}$$

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

However this assumes that odometry is perfect, which is not realistic. To model the error in odometry we add continuous random variables v_d and v_θ to δ_d and δ_θ respectively. The robot's configuration at the next time step, including the odometry error, is

$$\xi_{\langle k+1\rangle} = \begin{pmatrix} x\langle k\rangle + (\delta_d\langle k\rangle + v_d)\cos(\theta\langle k\rangle + \delta_\theta + v_\theta) \\ y\langle k\rangle + (\delta_d\langle k\rangle + v_d)\sin(\theta\langle k\rangle + \delta_\theta + v_\theta) \\ \theta\langle k\rangle + \delta_\theta + v_\theta \end{pmatrix} \tag{6.2}$$

which is the required function $\boldsymbol{f}(\cdot)$

$$\boldsymbol{x}\langle k+1\rangle = \boldsymbol{f}(\boldsymbol{x}\langle k\rangle, \delta\langle k\rangle, \boldsymbol{v}\langle k\rangle) \tag{6.3}$$

where k is the time step, $\delta\langle k\rangle$ is the odometry measurement and $\boldsymbol{v}\langle k\rangle$ the random measurement noise over the preceding interval.

In this case the odometry noise is *inside* the process model and is referred to as process noise.

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

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

This 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);
```

The Toolbox `Vehicle` class simulates the bicycle model of Eq. 4.2 and the odometric configuration update Eq. 6.2. To use it we create a `Vehicle` object

```
>> veh = Vehicle(V)
Vehicle object
  L=1, maxspeed=5, alphalim=0.5, T=0.100000, V=(0.0004,0.00121847), nhist=0
  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 one time step

```
>> odo = veh.step(1, 0.3)
odo =
    0.1002    0.0322
>> odo = veh.step(1, 0.3)
odo =
    0.0991    0.0311
```

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 hidden) configuration can be seen by displaying the object

```
>> veh
veh =
Vehicle object
  L=1, maxspeed=5, alphalim=0.5, T=0.100000, V=(0.0004,0.00121847), nhist=2
  x=0.199955, y=0.00299955, theta=0.06
```

We want to run the simulation over a long time period but we also want to keep the vehicle 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 `step` and maintains a history of the true state of the vehicle over the course of the simulation within the `Vehicle` object. The `RandomPath` and `Vehicle` 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 best estimate our new pose given the previous pose and noisy odometry. 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, position in this case, assuming that the noise is zero-mean and Gaussian. In this application the state of the Kalman filter is the estimated configuration of the robot. 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. That is, it provides the best estimate of where we are and how certain we are about that.

The Kalman filter is formulated for linear systems but our model of the vehicle's motion Eq. 6.3 is non-linear. We create a local linear approximation or linearization of the function $\hat{\boldsymbol{x}}\langle k\rangle$ by

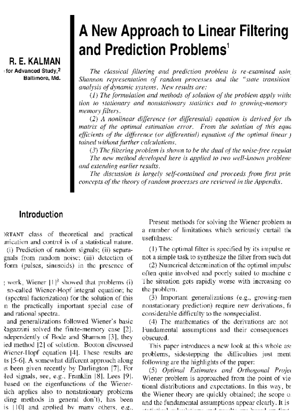
A truncated Taylor series.

$$\hat{\boldsymbol{x}}\langle k+1\rangle = \hat{\boldsymbol{x}}\langle k\rangle + \boldsymbol{F}_x\big(\boldsymbol{x}\langle k\rangle - \hat{\boldsymbol{x}}\langle k\rangle\big) + \boldsymbol{F}_v \boldsymbol{v}\langle k\rangle$$

with respect to the current state estimate $\hat{\boldsymbol{x}}\langle k\rangle$. The terms $\boldsymbol{F}_x$ and $\boldsymbol{F}_v$ are Jacobians which are vector versions of derivatives which are sometimes written as $\partial \boldsymbol{f} / \partial \boldsymbol{x}$ and $\partial \boldsymbol{f} / \partial \boldsymbol{v}$ respectively. This approach to estimation of a non-linear system is known as the extended Kalman filter or EKF. Jacobians are reviewed in Appendix G.

The Jacobians are obtained by differentiating Eq. 6.2 and evaluating them for $\boldsymbol{v} = 0$ giving

$$\boldsymbol{F}_x = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{v}=0} = \begin{pmatrix} 1 & 0 & -\delta_d\langle k\rangle - \sin\big(\theta\langle k\rangle + \delta_\theta\big) \\ 0 & 1 & \delta_d\langle k\rangle \cos\big(\theta\langle k\rangle + \delta_\theta\big) \\ 0 & 0 & 1 \end{pmatrix} \qquad (6.4)$$


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

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

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

(1) The formulation and methods of solution of the problem apply withc tion to stationary and nonstationary statistics and to growing-memory memory filters.

(2) A nonlinear difference (or differential) equation is derived for th matrix of the optimal estimation error. From the solution of this equa efficients of the difference (or differential) equation of the optimal linear tained without further calculations.

(3) The filtering problem is shown to be the dual of the noise-free regulat The new method developed here is applied to two well-known problem and extending earlier results.

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

Introduction

RTANT class of theoretical and practical nication and control is of a statistical nature. (i) Prediction of random signals; (ii) separagnals from random noise; (iii) detection of form (pulses, sinusoids) in the presence of

work, Wiener [1][3] showed that problems (i) so-called Wiener-Hopf integral equation; he (spectral factorization) for the solution of this n the practically important special case of and rational spectra.

and generalizations followed Wiener's basic Ragazzini solved the finite-memory case [2]. ndependently of Bode and Shannon [3], they ied method [2] of solution. Booton discussed Wiener-Hopf equation [4]. These results are ts [5-6]. A somewhat different approach along s been given recently by Darlington [7]. For led signals, see, e.g., Franklin [8], Lees [9]. based on the eigenfunctions of the Wienerich applies also to nonstationary problems ding methods in general don't), has been is [10] and applied by many others, e.g.,

Present methods for solving the Wiener problem a a number of limitations which seriously curtail th usefulness:

(1) The optimal filter is specified by its impulse re not a simple task to synthesize the filter from such dat

(2) Numerical determination of the optimal impuls often quite involved and poorly suited to machine c The situation gets rapidly worse with increasing co the problem.

(3) Important generalizations (e.g., growing-men nonstationary prediction) require new derivations, f considerable difficulty to the nonspecialist.

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

This paper introduces a new look at this whole as problems, sidestepping the difficulties just ment following are the highlights of the paper:

(5) *Optimal Estimates and Orthogonal Proje* Wiener problem is approached from the point of vie tional distributions and expectations. In this way, b the Wiener theory are quickly obtained; the scope o and the fundamental assumptions appear clearly. It is

Rudolf Kálmán (1930–) is a mathematical system theorist born in Budapest. He obtained his bachelors and masters degrees in electrical engineering from MIT, and PhD 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 amongst 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 is 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 non-linear version now known as the extended Kalman filter. This lead to its incorporation in the Apollo navigation computer for trajectory estimation. (Extract from Kálmán's famous paper (1960) on the left reprinted with permission of ASME)

$$\boldsymbol{F}_v = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{v}}\right|_{v=0} = \begin{pmatrix} \cos(\theta\langle k\rangle + \delta_\theta) & -\delta_d\langle k\rangle \sin(\theta\langle k\rangle + \delta_\theta) \\ \sin(\theta\langle k\rangle + \delta_\theta) & \delta_d\langle k\rangle \cos(\theta\langle k\rangle + \delta_\theta) \\ 0 & 1 \end{pmatrix} \tag{6.5}$$

The `Vehicle` object provides methods `Fx` and `Fv` to compute these Jacobians, for example

```
>> 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 about which the Jacobian is computed and the second is the odometry.

Now we can write the EKF prediction equations

The Kalman filter, Appendix H, 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.

$$\hat{\boldsymbol{x}}\langle k+1|k\rangle = \boldsymbol{f}(\hat{\boldsymbol{x}}\langle k\rangle, \delta\langle k\rangle, 0) \tag{6.6}$$

$$\hat{\boldsymbol{P}}\langle k+1|k\rangle = \boldsymbol{F}_x\langle k\rangle \hat{\boldsymbol{P}}\langle k|k\rangle \boldsymbol{F}_x\langle k\rangle^T + \boldsymbol{F}_v\langle k\rangle \hat{\boldsymbol{V}} \boldsymbol{F}_v\langle k\rangle^T \tag{6.7}$$

that describe how the state and covariance evolve with time. The term $\hat{\boldsymbol{x}}\langle k+1|k\rangle$ is read as the estimate of $\boldsymbol{x} = (\hat{x}, \hat{y}, \hat{\theta})$ at the time $k+1$ based on information up to, and including, time k. $\hat{\boldsymbol{P}} \in \mathbb{R}^{3\times 3}$ is a covariance matrix representing uncertainty in the estimated vehicle configuration. The second term in Eq. 6.7 is positive definite which means that $\hat{\boldsymbol{P}}$, the position uncertainty, can never decrease. $\hat{\boldsymbol{V}}$ is our estimate of the covariance of the odometry noise which in reality we do not know.

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);
```

Runing 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` object.

Error ellipses. If the position of the robot (ignoring orientation) is considered as a PDF such as shown in Fig. 6.3 then a horizontal cross-section will be an ellipse. The 2-dimensional Gaussian probability density function is

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

where $\mu_x \in \mathbb{R}^2$ is the mean of $\boldsymbol{x}$ and $\boldsymbol{P} \in \mathbb{R}^{2\times 2}$ is the covariance matrix. The 1σ boundary is defined by the points $\boldsymbol{x}$ such that

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

It is useful to plot such an ellipse, as shown in Fig. 6.4, to represent the positional uncertainty. A large ellipse corresponds to a wider PDF peak and less certainty about position.

A handy scalar measure of total 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 $\boldsymbol{P}$. Since $\det(\boldsymbol{P}) = \Pi\lambda_i$ the ellipse area – the scalar uncertainty – is proportional to $\sqrt{\det(\boldsymbol{P})}$. See also Appendices E and F.

Stored within the `Vehicle` object.

We can plot the true path taken by the vehicle

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

Stored within the `EKF` object.

and the filter's estimate of the path

```
>> 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 700$^{\text{th}}$ time step is

```
>> P700 = ekf.history(700).P
P700 =
    0.4674    0.0120   -0.0295
    0.0120    0.7394    0.0501
   -0.0295    0.0501    0.0267
```

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 of the PDF associate with the x-coordinate is

```
>> sqrt(P700(1,1))
ans =
   0.6837
```

There is a 95% chance that the robot's x-coordinate is within the $\pm 2\sigma$ bound or ± 1.37 m in this case. We can consider 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 $P_{2,3} = P_{3,2} = 0.0501$ indicates that the uncertainties in x and θ are related as we would expect – changes in heading angle will affect the x-position. 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 1σ confidence bound. 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 elements of $\boldsymbol{P}$ have different units: m^2 and rad. The uncertainty is therefore a mixture of spatial and angular uncertainty with an implicit weighting. Typically $x, y \gg \pi$ so positional uncertainty dominates.

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

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

as shown in Fig. 6.5. We observe that it never decreases.

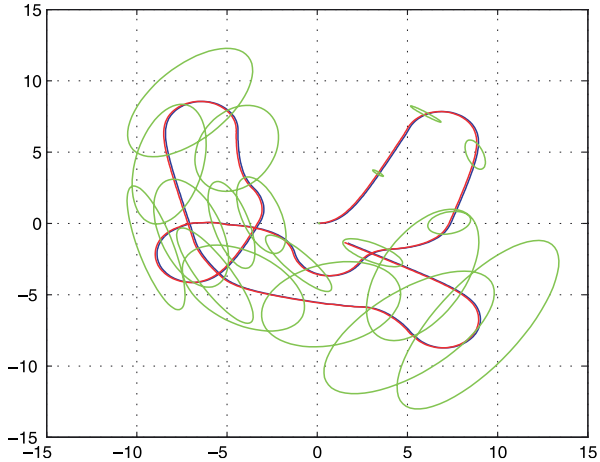


Fig. 6.4. Deadreckoning using the EKF. The true path of the robot, blue, and the path estimated from odometry in red. The robot starts at the origin, uncertainty ellipses are indicated in green

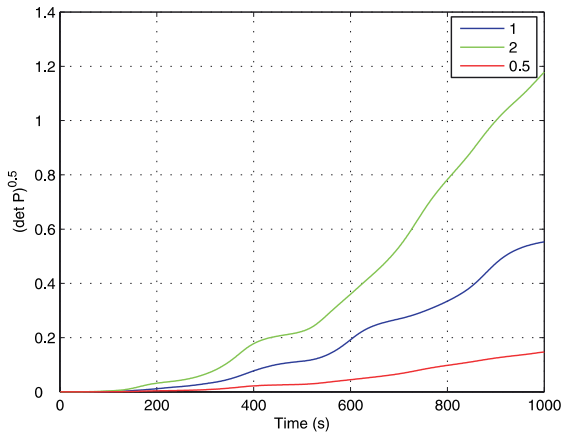


Fig. 6.5.
Overall uncertainty is given by $\sqrt{\det(\boldsymbol{P})}$ which shows monotonically increasing uncertainty (blue). The effect of changing the magnitude of $\boldsymbol{V}$ is to change the rate of uncertainty growth. Curves are shown for $\boldsymbol{V} = \alpha \boldsymbol{V}^*$ where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

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 that is *actually added* to the true odometry to simulate odometry error in Eq. 6.3. In a real application this noise process would be *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. 6.7. 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. 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. That is, 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 Using 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 new 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 position is known.

The Toolbox supports a `Map` object

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

that in this case contains $N = 20$ features 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 features *with respect to the robot* as described by

$$\boldsymbol{z} = \boldsymbol{h}(\boldsymbol{x}_v, \boldsymbol{x}_f, \boldsymbol{w}) \tag{6.8}$$

where $\boldsymbol{x}_v$ the vehicle state, $\boldsymbol{x}_f$ is the known location of the observed feature in the world frame and $\boldsymbol{w}$ is a random variable that models errors in the sensor.

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. 6.6. The sensor is mounted onboard the robot so the observation of the i^{th} feature is $\boldsymbol{x}_{fi} = (x_i, y_i)$ is

Fig. 6.6. A scanning laser range finder. The sensor has a rotating assembly that emits pulses of infra-red laser light and measures the time taken for the reflection to return. This sensor has a maximum range of 30 m and an angular range of 270 deg. Angular resolution is 0.25 deg and the sensor makes 40 scans per second (Courtesy of Hokuyo Automatic Co. Ltd.)

$$\boldsymbol{z} = \begin{pmatrix} \sqrt{(y_i - y_v)^2 + (x_i - x_v)^2} \\ \tan^{-1}(y_i - y_v)/(x_i - x_v) - \theta_v \end{pmatrix} + \begin{pmatrix} w_r \\ w_\beta \end{pmatrix} \tag{6.9}$$

where $\boldsymbol{z} = (r, \beta)^T$ and r is the range, β the bearing angle, and the measurement noise is

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

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 ($1/d^4$) with distance (d) to the target.

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

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);
```

A subclass of `Sensor`.

In the Toolbox we model this type of sensor with a `RangeBearingSensor` object

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

which is connected to the vehicle and the map, and the sensor covariance matrix `W` is specified. The `reading` method provides the range and bearing to a randomly selected map feature along with the identity of the map feature it has sensed

If the `interval` property is set to N then the method returns a reading on every N^{th} call. A non-measurement is indicated by `i` having a value of `NaN`.

```
>> [z,i] = sensor.reading()
z =
   31.9681
    1.6189
i =
    18
```

The identity is an integer $i \in [1, 20]$ since the map was created with 20 features. We have avoided the data association problem by assuming that we know the identity of the sensed feature. The position of feature 18 can be looked up in the map

```
>> map.feature(18)
   -8.0574
    6.4692
```

Using Eq. 6.9 the robot can estimate the range and bearing angle to the feature based on its own estimated position and the known position of the feature from the map. Any difference between the observation and the estimated observation indicates an error in the robot's position estimate – it isn't where it thought it was. This *difference*

$$\boldsymbol{\nu}\langle k+1\rangle = \boldsymbol{z}\langle k+1\rangle - \boldsymbol{h}\left(\hat{\boldsymbol{x}}(k+1|k), x_f, 0\right)$$

See Appendix H.

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 $\boldsymbol{P}\langle k\rangle$.

As we did previously on page 113 we linearize the observation Eq. 6.8 and write

$$\boldsymbol{z}\langle k\rangle = \hat{\boldsymbol{h}} + \boldsymbol{H}_x\left(\boldsymbol{x}\langle k\rangle - \hat{\boldsymbol{x}}\langle k\rangle\right) + \boldsymbol{H}_w \boldsymbol{w}\langle k\rangle \tag{6.10}$$

where the Jacobians are obtained by differentiating Eq. 6.9 yielding

$$\boldsymbol{H}_{x_i} = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}_v}\right|_{\boldsymbol{w}=0} = \begin{pmatrix} -\frac{x_i - x_v\langle k\rangle}{r} & -\frac{y_i - y_v\langle k\rangle}{r} & 0 \\ \frac{x_i - x_v\langle k\rangle}{r^2} & -\frac{y_i - y_v\langle k\rangle}{r^2} & -1 \end{pmatrix} \tag{6.11}$$

$$\boldsymbol{H}_w = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=0} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{6.12}$$

The RangeBearingSensor object above includes methods h to implement Eq. 6.9 and H_x and H_w to compute these Jacobians respectively.

Now we use the innovation to *update* the predicted state computed earlier using Eq. 6.6 and Eq. 6.7

$$\hat{\boldsymbol{x}}\langle k+1|k+1\rangle = \hat{\boldsymbol{x}}\langle k+1|k\rangle + \boldsymbol{K}\langle k+1\rangle \nu\langle k+1\rangle \tag{6.13}$$

$$\hat{\boldsymbol{P}}\langle k+1|k+1\rangle = \hat{\boldsymbol{P}}\langle k+1|k\rangle \boldsymbol{F}_x\langle k\rangle^T - \boldsymbol{K}\langle k+1\rangle \boldsymbol{H}_x\langle k+1\rangle \hat{\boldsymbol{P}}\langle k+1|k\rangle \tag{6.14}$$

which are the Kalman filter update equations. These take the *predicted* values for the next time step denoted $k+1|k$ and apply information from time step $k+1$ to compute values denoted $k+1|k+1$. The innovation has been added to the estimated state after multiplying by the Kalman gain matrix $\boldsymbol{K}$ which is defined as

$$\boldsymbol{S}\langle k+1\rangle = \boldsymbol{H}_x\langle k+1\rangle \hat{\boldsymbol{P}}\langle k+1|k\rangle \boldsymbol{H}_x\langle k+1\rangle^T + \boldsymbol{H}_w\langle k+1\rangle \hat{\boldsymbol{W}}\langle k+1\rangle \boldsymbol{H}_w\langle k+1\rangle^T \tag{6.15}$$

$$\boldsymbol{K}\langle k+1\rangle = \hat{\boldsymbol{P}}\langle k+1|k\rangle \boldsymbol{H}_x\langle k+1\rangle^T \boldsymbol{S}\langle k+1\rangle^{-1} \tag{6.16}$$

where $\hat{\boldsymbol{W}}$ is the estimated covariance of the sensor noise. Note that the second term in Eq. 6.14 is *subtracted* from the covariance and this provides a means for covariance to decrease which was not possible for the dead-reckoning case of Eq. 6.7.

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

```
>> map = Map(20);
>> veh = Vehicle(V);
>> veh.add_driver( RandomPath(map.dim) );
>> sensor = RangeBearingSensor(veh, map, W);
>> ekf = EKF(veh, V, P0, sensor, W, map);
```

The Map 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')
```

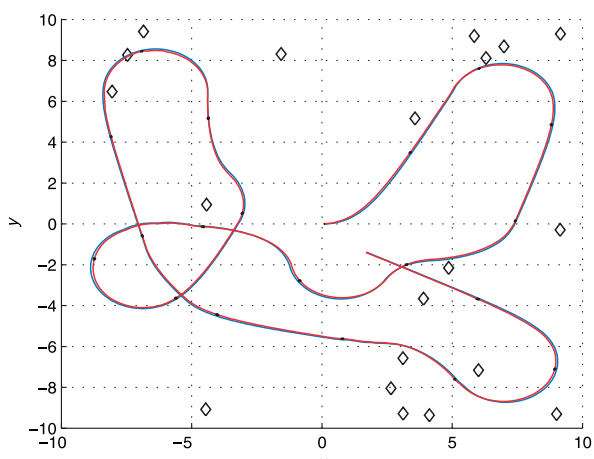


Fig. 6.7.
EKF localization showing the true path of the robot (blue) and the path estimated from odometry and landmarks (red). The robot starts at the origin

Reverend Thomas Bayes (1702–1761) was a non-conformist 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.

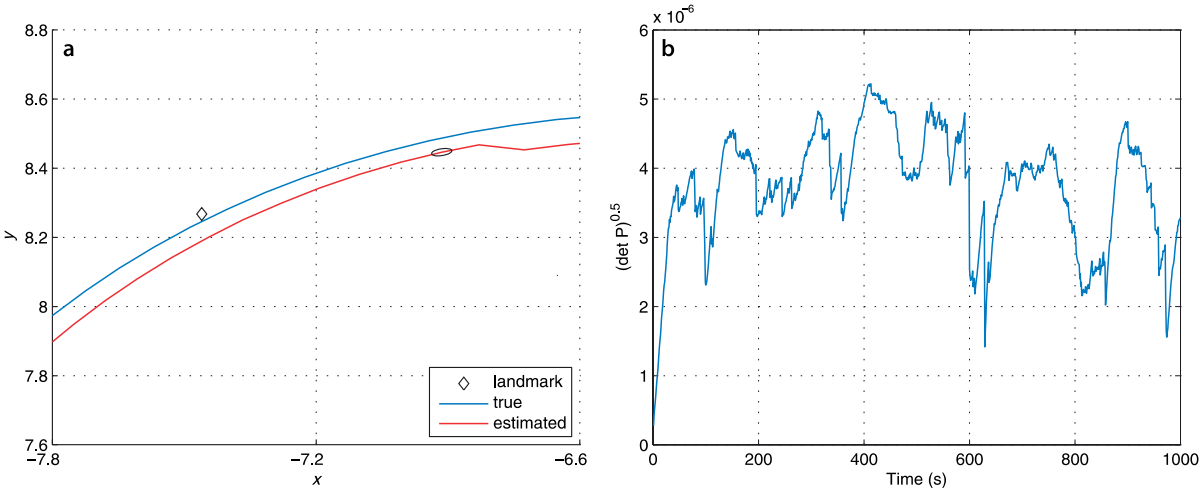


Fig. 6.8. a Closeup of the robot's true and estimated path; **b** the covariance magnitude as a function of time. Overall uncertainty is given by det(***P***) and shows that uncertainty is not increasing with time

which are shown in Fig. 6.7. We are hard pressed to see the error ellipses since they are now so small.

Figure 6.8a shows a zoomed view of the robot's actual and estimated path. We can see a small error ellipse and we can also see a *jag* in the estimated path. The vehicle state evolves smoothly with time according to the bicycle model of Eq. 4.2 but new information from a sensor reading updates the state and can sometimes cause noticeable changes, jumping the state estimate either forwards, backwards or sideways. Figure 6.8b shows that the uncertainty is no longer growing monotonically, new information is reducing the uncertainty through Eq. 6.14.

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 *actually added* to the computed range and bearing to simulate sensor error as in Eq. 6.9. The second usage, $\hat{\boldsymbol{W}}$ is our best estimate of the sensor covariance which is used by the Kalman filter Eq. 6.15.

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 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 output covariance $\boldsymbol{W}$. The function $\boldsymbol{h}(\cdot)$ can be non-linear and even non-invertible – the EKF will do the rest.

In this example, and in the next two sections, we assume that the sensor provides information about the position of the target with respect to the robot and also the *identity* of the target. In practice most landmarks are anonymous, that is, we do not know their identity and hence do not know their true location in the world. We therefore need to solve the correspondence or target association problem – given the map and the observation, determine which feature is the *most likely* to have been seen. Errors in this step can lead rapidly to failure of the estimator – the system sees target i but thinks it is target j. The state vector is updated incorrectly making it more likely to incorrectly associate targets and a downward spiral ensues. The covariance may not necessarily increase greatly and this is a dangerous thing – a confident but wrong robot. In indoor robotic problems the sensor may detect spurious targets such as people moving in the environment and the people will also obscure real landmarks.

Bar codes could be used to provide distinct target identity in some applications such as indoor mobile robots.

An alternative is a multi-hypothesis estimator, such as the particle filter that we will discuss in Sect. 6.5, 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 a single hypothesis about location.

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 maps. 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 features with respect to the robot. There are a total of N features in the environment and as for the previous example we assume that the sensor can determine the identity of each observed feature. 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 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

$$\hat{\boldsymbol{x}} = (x_1, y_1, x_2, y_2, \cdots x_M, y_M)^T$$

and has $2M$ elements. 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 feature is observed.

The prediction equation is straightforward in this case since the features do not move

$$\hat{\boldsymbol{x}}\langle k+1|k\rangle = \hat{\boldsymbol{x}}\langle k|k\rangle \tag{6.17}$$

$$\hat{\boldsymbol{P}}\langle k+1|k\rangle = \hat{\boldsymbol{P}}\langle k|k\rangle \tag{6.18}$$

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

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

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 $\boldsymbol{y}(\cdot)$

$$\boldsymbol{x}\langle k|k\rangle^{*} = \boldsymbol{y}\big(\boldsymbol{x}\langle k|k\rangle, \boldsymbol{z}\langle k\rangle, \boldsymbol{x}_v\langle k|k\rangle\big) \tag{6.19}$$

$$= \begin{pmatrix} \boldsymbol{x}\langle k|k\rangle \\ \boldsymbol{g}\big(\boldsymbol{x}_v\langle k|k\rangle, \boldsymbol{z}\langle k\rangle\big) \end{pmatrix} \tag{6.20}$$

which appends the estimate of the new feature's coordinates to the coordinates 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

$$\hat{\boldsymbol{P}}\langle k|k\rangle^{*} = \boldsymbol{Y}_z \begin{pmatrix} \hat{\boldsymbol{P}}\langle k|k\rangle & 0 \\ 0 & \hat{\boldsymbol{W}} \end{pmatrix} \boldsymbol{Y}_z^T$$

where $\boldsymbol{Y}_z$ is another Jacobian

$$\boldsymbol{Y}_z = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}} = \begin{pmatrix} \boldsymbol{I}_{n\times n} & & \boldsymbol{0}_{n\times 2} \\ \boldsymbol{G}_x & \boldsymbol{0}_{2\times n-3} & \boldsymbol{G}_z \end{pmatrix} \tag{6.21}$$

$$\boldsymbol{G}_x = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}_v} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{6.22}$$

$$\boldsymbol{G}_z = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}} = \begin{pmatrix} \cos(\theta_v + \theta_z) & -r_z \sin(\theta_v + \theta_z) \\ \sin(\theta_v + \theta_z) & r_z \cos(\theta_v + \theta_z) \end{pmatrix} \tag{6.23}$$

where n is the dimension of $\hat{\boldsymbol{P}}$ prior to it being extended.

An additional Jacobian for $\boldsymbol{h}(\cdot)$ is

$$\boldsymbol{H}_{x_i} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}_i} = \begin{pmatrix} \frac{x_i - x_v}{r} & \frac{y_i - y_v}{r} \\ -\frac{x_i - x_v}{r^2} & \frac{y_i - y_v}{r^2} \end{pmatrix} \tag{6.24}$$

which relates change in map features to change in observation and is implemented by the `Sensor` class method `H_xf`.

The Jacobian $\boldsymbol{H}_x$ used in Eq. 6.14 describes how the feature observation changes with respect to the state vector. However in this case, the observation depends only on a single observed feature so this Jacobian is mostly zeros

$$\boldsymbol{H}_x = (0 \cdots \boldsymbol{H}_{x_i} \cdots 0) \tag{6.25}$$

where $\boldsymbol{H}_{xi}$ is at the location corresponding to the state $\boldsymbol{x}_i$.

The Toolbox implementation is

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

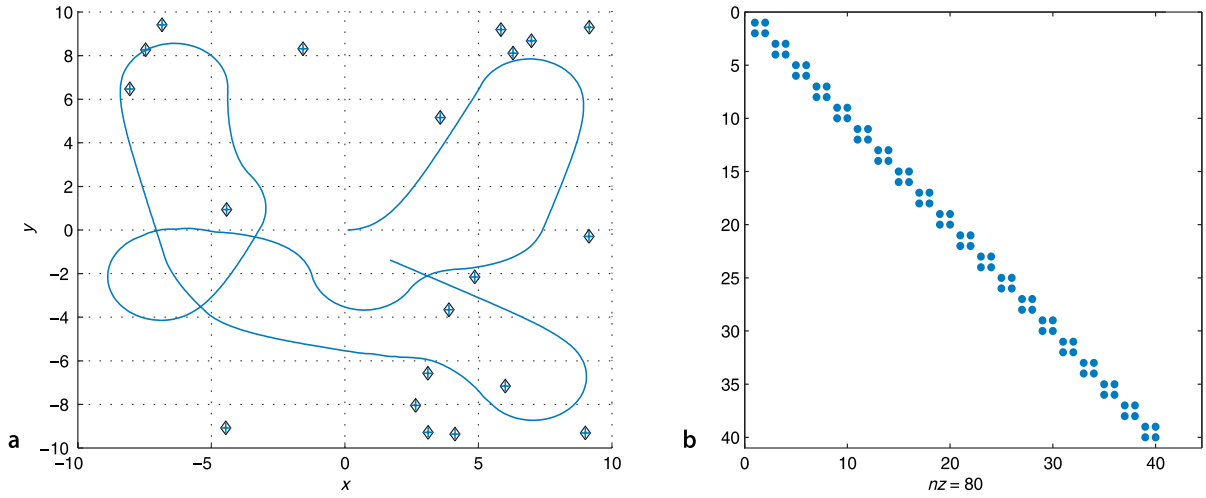


Fig. 6.9. EKF mapping results. **a** The estimated landmarks are indicated by +-markers with 5σ confidence ellipses (green), the true location (black ◊-marker) and the robot's path (blue); **b** the non-zero elements of the final covariance matrix

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);
```

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(5, 'g');
>> veh.plot_xy('b');
```

are shown in Fig. 6.9a as 5σ confidence ellipses (in order to be visible) along with the true landmark positions and the path taken by the robot. The covariance matrix has a block diagonal structure which is displayed graphically

```
>> spy( ekf.P_est );
```

in Fig. 6.9b. The blue dots represent non-zero elements and each 2×2 block represents the covariance of the position of a map feature. The correlations, the off-diagonal elements are zero, which implies that the feature estimates are uncorrelated or independent. This is to be expected since observing feature i provides no new information about feature $j \neq i$.

This is known as the sparsity structure of the matrix, and a large class of numerical algorithms exist that work efficiently on matrices that are predominantly zero. MATLAB's `spy` function shows the sparsity structure of matrices graphically. See `help sparse` for more details.

Internally the `EKF` object maintains a table to relate the feature identity, returned by the `RangeBearingSensor`, to the position of that feature's coordinates in the state vector. For example the landmark with identity 10

```
>> ekf.features(:,10)
ans =
    19
    51
```

was seen a total of 51 times during the simulation and comprises elements 19 and 20 of $\hat{\boldsymbol{x}}$

```
>> ekf.x_est(19:20)'
ans =
    5.8441    9.1898
```

which is its estimated location. Its estimated covariance is

```
>> ekf.P_est(19:20,19:20)
ans =
   1.0e-03 *
    0.2363   -0.0854
   -0.0854    0.2807
```

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. In robotics this problem is known as simultaneous localization and mapping (SLAM) or concurrent mapping and localization (CML). This is sometimes referred to as a "chicken and egg" problem but based on what we have learnt in the previous sections this problem is now quite straightforward to solve.

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

$$\hat{\boldsymbol{x}} = (x_v, y_v, \theta_v, x_1, y_1, x_2, y_2, \cdots x_M, y_M)$$

and has $2M + 3$ elements. The 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 state, $\hat{\boldsymbol{P}}_{mm}$ the covariance of the map features, and $\hat{\boldsymbol{P}}_{vm}$ is the correlation between vehicle and map states.

The predicted vehicle state and covariance are given by Eq. 6.6 and Eq. 6.7 and the sensor update is given by Eq. 6.13 to 6.16. When a new feature is observed the state vector is updated using the Jacobian $\boldsymbol{Y}_z$ given by Eq. 6.21 but in this case $\boldsymbol{G}_x$ is non-zero

$$\boldsymbol{G}_x = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}_v} = \begin{pmatrix} 1 & 0 & -r_z \sin(\theta_v + \theta_z) \\ 0 & 1 & r_z \cos(\theta_v + \theta_z) \end{pmatrix}$$

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

The Jacobian $\boldsymbol{H}_x$ describes how the feature 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 feature and is

$$\boldsymbol{H}_x = (\boldsymbol{H}_{x_v} \cdots 0 \cdots \boldsymbol{H}_{x_i} \cdots 0) \tag{6.26}$$

where $\boldsymbol{H}_{xi}$ is at the location corresponding to the state $\boldsymbol{x}_i$. This is similar to Eq. 6.25 but with the non-zero block $\boldsymbol{H}_{xv}$ at the left to account for the effect of vehicle position.

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

The Toolbox implementation is by now quite familiar

```
>> P0 = diag([.01, .01, 0.005].^2);
>> map = Map(20);
>> veh = Vehicle(W);
>> veh.add_driver( RandomPath(map.dim) );
>> sensor = RangeBearingSensor(veh, map, 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);
```

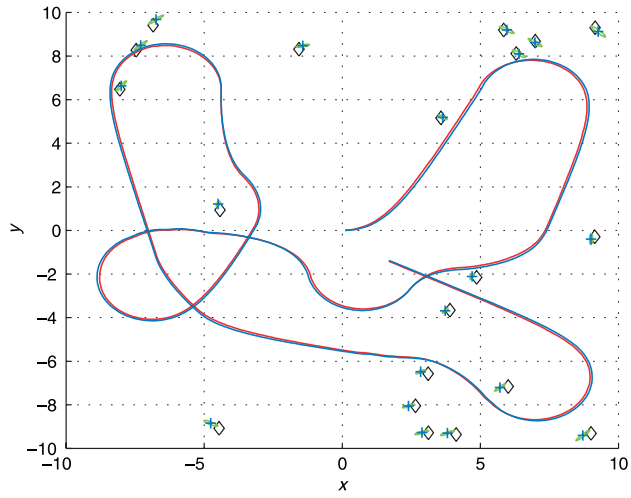


Fig. 6.10.
Simultaneous localization and mapping showing the true (blue) and estimated (red) robot path superimposed on the true map (black ◊-marker). The estimated map features are indicated by +-markers and the 5σ confidence ellipses (green)

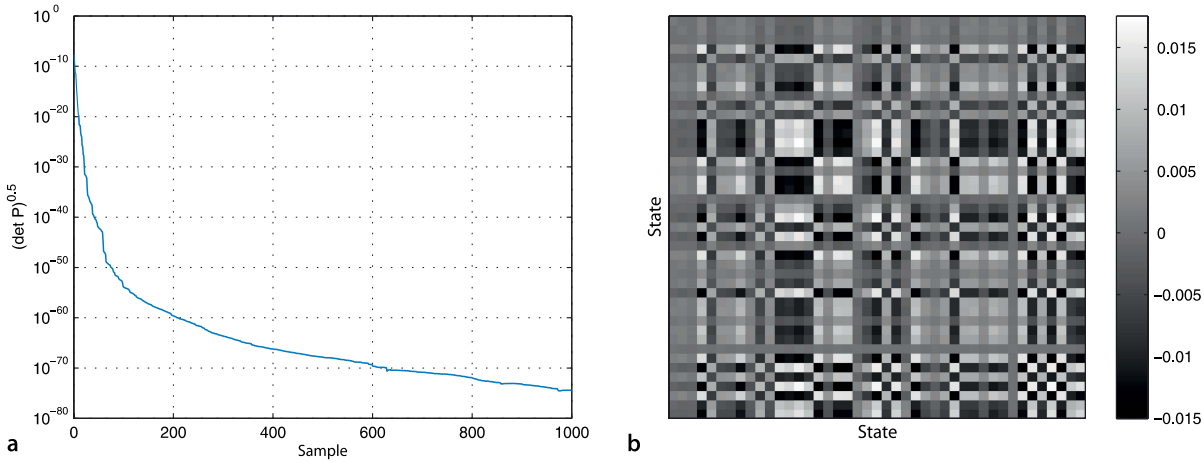


Fig. 6.11. Simultaneous localization and mapping. **a** Covariance versus time; **b** the final covariance matrix (values have been scaled in the interval 0 to 255)

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(5, 'g');
>> ekf.plot_xy('r');
>> veh.plot_xy('b');
```

which are shown in Fig. 6.10.

Figure 6.11a shows that uncertainty is decreasing over time. Figure 6.11b shows the final covariance matrix as an image and we see a complex structure. Unlike the mapping case $\hat{\boldsymbol{P}}_{mm}$ is not block diagonal, and the finite off-diagonal terms represent correlation *between* the features in the map. The feature uncertainties never increase, the prediction model is that they don't change, but they also never drop below the initial uncertainty of the vehicle. The block $\hat{\boldsymbol{P}}_{vm}$ is the correlation between errors in the vehicle and the map features. A feature's location estimate is a function of the vehicle's location and errors in the vehicle location appear as errors in the feature location – and vice versa.

The correlations cause information about the observation of any feature to affect the estimate of every other feature in the map and the vehicle pose. It is as if all the states were connected by springs and the movement of any one affects all the others.

6.5 Monte-Carlo Localization

The estimation examples so far have assumed that the error in sensors such 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 multimodal distribution with several peaks. The functions we used in the Kalman filter such as Eq. 6.3 and Eq. 6.8 are strongly non-linear 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* versions of the vehicle's state vector. When a new measurement is available we score how well each version of the state explains the data. We keep the best fitting states and randomly perturb them to form a new generation of states. Collectively these many possible states and their scores approximate a probability density function for 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 standard desktop computers. If we plot these state vectors as points 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 (x, y, θ) is computationally tractable to solve with straightforward MATLAB® code. The estimator is initialized by creating N particles $\boldsymbol{x}_{v,i}$, $i \in [1, N]$ 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

$$\boldsymbol{x}_{v,i}\langle k+1\rangle = \boldsymbol{f}\left(\boldsymbol{x}_{v,i}\langle k\rangle, \delta\langle k\rangle\right) + \boldsymbol{q}\langle k\rangle$$

moving each particle according to the measured odometry. We also add a random vector $\boldsymbol{q}\langle k\rangle$ which represents uncertainty in the position of the vehicle. Often $\boldsymbol{q}$ is drawn from a Gaussian random variable with covariance $\boldsymbol{Q}$ but any physically meaningfull distribution can be used.

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 $h / n = 2l / 2\pi$ 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 Neuman, Stanislaw Ulam and Nicholas Metropolis. The name Monte Carlo alludes to games of chance and was the code name for the secret project.

2. We make an observation $\boldsymbol{z}$ of feature i which has, according to the map, coordinate $\boldsymbol{x}_f$. For each particle we compute the innovation

$$\nu_i = \boldsymbol{h}(\boldsymbol{x}_{v,i}, \boldsymbol{x}_f) - \boldsymbol{z}$$

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 Gaussian likelihood function

$$w = \exp(-0.5\nu^T \boldsymbol{L}\nu) + w_0$$

where w is referred to as the *importance* or *weight* of the particle, $\boldsymbol{L}$ is a covariance matrix, and $w_0 > 0$ ensures that there is a finite but small probability associated with any point in the state space. We use a Gaussian function only for convenience, the function does not need to be smooth or invertible but only to adequately describe the likelihood of an observation.

3. Select the particles that best explain the observation, a process known as resampling. 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 $w_i' = w_i / \sum_{i=1}^{N} w_i$ and construct a cumulative histogram $c_j = \sum_{i=1}^{j} w_i'$. We then draw a uniform random number $r \in [0, 1]$ and find $\arg_j \min |c_j - r|$ where particle j 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.

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

The Toolbox implementation is broadly similar to the previous examples. We create a map

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

and a robot with noisy odometry and an initial condition

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

and then a sensor with noisy readings

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

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 $\boldsymbol{W}$

```
>> 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);
```

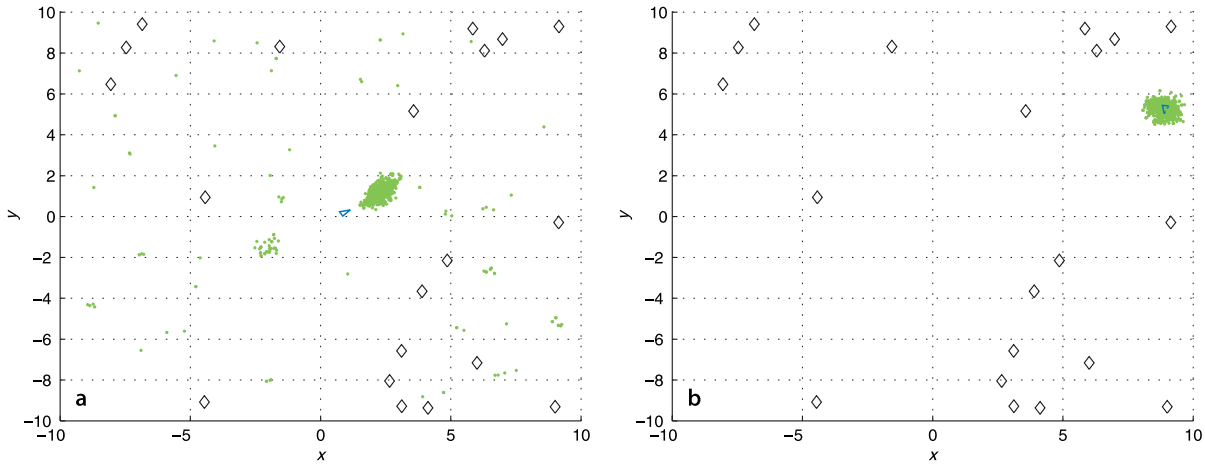


Fig. 6.12. Particle filter results showing the evolution of the evolution of the particle cloud (green dots) over time. The vehicle is shown as a blue triangle

and watch the animation, two snapshots of which are shown in Fig. 6.12. 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 plot is 3-dimensional, so you can rotate the graph while the animation is running to see the heading angle state which is the height (z-coordinate) of the particles.

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. As usual we plot the results of the simulation

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

and overlay the mean of the particle cloud

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

which is shown in Fig. 6.13. 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. 6.13b. 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.

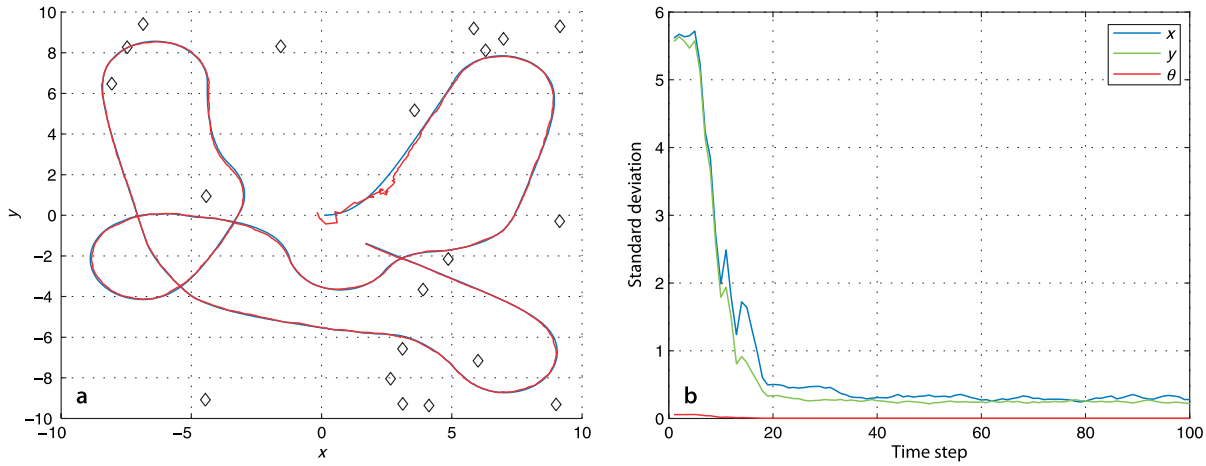


Fig. 6.13. Particle filter results. **a** True (blue) and estimated (red) robot path; **b** standard deviation of the particles versus time

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.

6.6 Wrapping Up

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

We modelled 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 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 is non-linear and we approximated it with a linear model which requires some Jacobians to be computed, an approach known as extended Kalman filtering.

The Kalman filter also estimates uncertainty associated with the configuration 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, whose location are known, 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.

Finally we brought all this together and estimated the vehicle's position, the position of the landmarks and their uncertainties. 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.

Finally we learnt about 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.

Further Reading

The book by Borenstein et al. (1996) has an excellent discussion of robotic sensors in general and odomery in particular. Although out of print it is available 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. The handbook (Siciliano and Khatib 2008, § 20, § 22) provides a brief but more modern treatment of these sensors.

The books of Borenstein et al. (1996) and Everett (1995) were published *before* GPS became operational. The principles of GPS and other radio-based localization systems are covered in some detail in the book by Groves (2008). The Robotics Handbook (Siciliano and Khatib 2008, § 20) also briefly describes GPS, and a number of links to GPS technical data are provided from this book's web site.

There are many published and online resources for Kalman filtering. Kálmán's original paper, Kálmán (1960), is now over 50 years old. The book by Zarchan and Musoff (2005) is a very clear and readable introduction to Kalman filtering. I have always found the classic 1970 book by Jazwinski (1970) to be very readable and it has recently been republished. Bar-Shalom et al. (2001) provide comprehensive coverage of estimation theory and also the use of GPS. An excellent reference for EKF and Monte-Carlo localization is the book by Thrun et al. (2005). Data association is an important topic and is covered in detail in, the now very old, book by Bar-Shalom and Fortmann (1988). The Robotics Handbook (Siciliano and Khatib 2008, § 4) covers Kalman filter and data association. 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 non-linearity. Alternatives include the iterated EKF described by Jazwinski (1970) or the Unscented Kalman Filter (UKF) (Julier and Uhlmann 2004) which uses discrete sample points to approximate the PDF. Some of these topics are covered in the Handbook (Siciliano and Khatib 2008, § 25) as multi-sensor fusion. The book by Siegwart et al. (2011) also has a good treatment of robot localization.

There is a very large literature on SLAM and this chapter has only touched the surface with a very classical EKF-based approach which does not scale well for large numbers of features. FastSLAM (Montemerlo et al. 2002) is a state-of-the-art algorithm for large-scale applications. There are a lot of online resources related to SLAM. Many of the SLAM summer schools have websites that host excellent online resources such as lecture notes and practicals.

A collection of open-source SLAM implementations such as gmapping and iSam is available from OpenSLAM at http://www.openslam.org. MATLAB® implementations include the CAS Robot Navigation Toolbox for planar SLAM at http://www.cas.kth.se/toolbox and a 6DOF SLAM system at http://homepages.laas.fr/jsola/JoanSola/eng/toolbox.html.

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

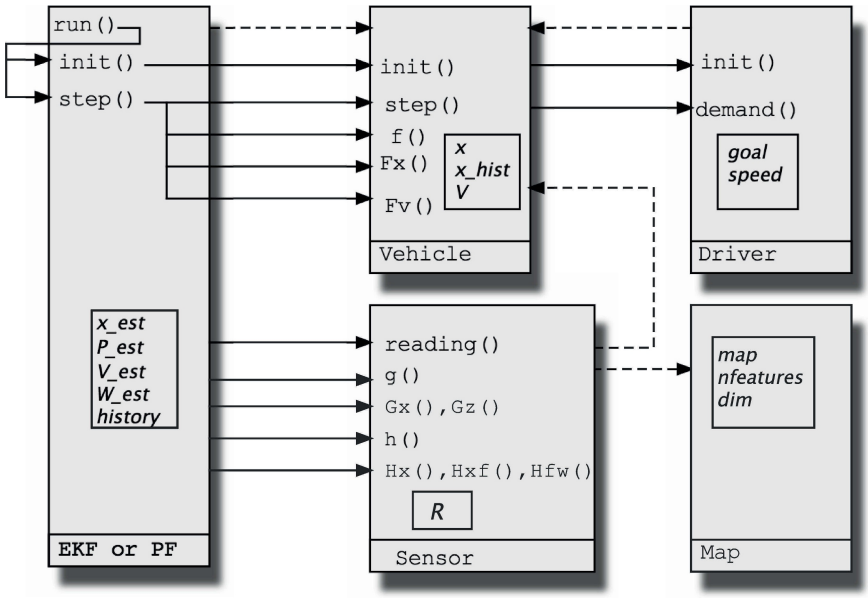


Fig. 6.14.
Toolbox class relationship for localization and mapping. Each class is shown as a rectangle, method calls are shown as arrows from caller to callee, properties are boxed, and dashed lines represent object references

Notes on Toolbox Implementation

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. 6.14. As always more documentation is available through the online help system or comments in the code.

Exercises

1. What is the value of the Longitude Prize in todays currency?
2. Implement a driver object (page 113) that drives the robot around inside a circle with specified centre and radius.
3. Derive an equation for heading odometry in terms of the rotational rate of the left and right wheels.
4. Dead-reckoning (page 111)
   a) Experiment with different values of $\boldsymbol{P}_0$, $\boldsymbol{V}$ and $\hat{\boldsymbol{V}}$.
   b) Fig. 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?
5. Using a map (page 116)
   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 120)
   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 123)
   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) Fig. 6.10 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. Particle filter (page 125)
   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.