Appendices

A Installing the Toolboxes

The Toolboxes are freely available from the book's home page

http://www.petercorke.com/RVC

which also has a lot of additional information related to the book such as web links (all those printed in the book and more), code, figures, exercises and errata.

Files and Paths

The files for both Toolboxes reside in a top-level directory called `rvctools` and beneath this are a number of subdirectories:

robot	The Robotics Toolbox.
vision	The Machine Vision Toolbox.
common	Utility functions common to the Robotics and Machine Vision Toolboxes.
simulink	Simulink® blocks for robotics and vision, as well as examples.
contrib	Code written by third-parties.

Downloading

The Toolboxes are packaged together in a single file, either gzipped tar format (`rvtb.gz`) or zip format (`rvtb.zip`). The download site requests some information such as your country, type of organization and application. There is nothing sinister in this, just a means to gauge interest and gather some feedback for the software which is a substantial personal effort.

Installing

Use your favourite unarchiving tool to unpack the files that you downloaded.

To add the Toolboxes to your MATLAB® path execute the command

```
>> addpath( genpath( RVCDIR ));
```

where `RVCDIR` is the full pathname of the directory where you unpacked the top-level toolbox directory `rvctools`. The function `genpath` returns a path string that includes the top level directory and all subdirectories.

This command can be executed interactively or placed in your `startup.m` file to be executed automatically every time you start MATLAB®. Alternatively, for Linux and Mac OS systems, you could add the paths to the environment variable `MATLABPATH` which is a colon-separated list of paths.

MEX-Files

Some functions in the Toolbox are implemented as MEX-files, that is, they are written in C for computational efficiency but are callable from MATLAB® just like any other function. Prebuilt MEX binaries are provided for Ubuntu Linux, MacOS 10.6 (32 bit) and Windows, but the C source code is also provided. Specific details and build instructions for MEX-files can be found on the book's website.

Online Discussion Group

An online discussion group is available via the book's website and provides answers to questions, discussions and bug fixes.

Contributed Code

A number of useful related functions are provided by third-parties and wrappers have been written to make them consistent with other Toolbox functions. All such code resides in the subdirectory `contrib`. To access this functionality you must first download the file `contrib.gz` or `contrib.zip` and unarchive it into the top-level directory `rvctools`.

If you do not download the contributed code but access a function within the contributed code base, you will receive an error message. The contributed code can be downloaded and installed at any time.

Many of these contributed functions are part of active software projects and the downloadable file is a snapshot that has been tested and works as described in this book. These functions are being improved over time and the book's web page has links to the home pages of these various projects.

Licence

All the non third-party code is released under the LGPL licence. This means you are free to distribute it in original or modified form provided that you keep the licence and authorship information intact.

The third-party code modules are provided under various open-source licences. The Toolbox compatibility wrappers for these modules are provided under compatible licences.

MATLAB® Versions

The Toolbox software for this book has been developed and tested using MATLAB® R2010a under Mac OS X (Snow Leopard). A number of recent features of MATLAB® are used so older versions of MATLAB® are increasingly unlikely to work. Please do not report bugs if you are using a MATLAB® version older than R2010a.

Octave

GNU Octave (`www.octave.org`) is an impressive piece of free software that implements a language that is close to, but not the same as, MATLAB®. However the two languages are converging and once massive differences with respect to graphics and object handling are reducing over time. At some point in the future it is conceivable that the Toolboxes could be patched to work with Octave. However it is unlikely that Octave will have GUI or Simulink®-like capability in the near future.

B Simulink®

Simulink® is the block diagram editing and simulation environment for MATLAB®. It is a separately licenced module but the functionality is included in the student version of MATLAB®. Simulink® provides a very convenient way to create and visualize complex dynamic systems, and is particularly applicable to robotics. Users with no previous Simulink® experience are advised to read the relevant Mathworks manuals and experiment with the examples supplied. Experienced Simulink® users should find the use of the Robotics blocks quite straightforward. Generally there is a one-to-one correspondence between Simulink® blocks and Toolbox functions.

Using Simulink®

If you have installed the Toolboxes then the Simulink® blocks and examples will be available for use. The Toolbox block library is loaded and displayed by

```
>> roblocks
```

and the blocks can be dragged and dropped into a model. Example Simulink® models used in this book are included in the directory `rvctools/simulink/examples`. These are all prefixed with `sl_` and are listed in the index of functions on page 557. A model is loaded and displayed in Simulink® by just entering the model name at the prompt, for example

```
>> sl_lanechange
```

To display the underlying model for any block, right-click on it and choose Look under mask.

Signals and Display Format

The wires in a Simulink® model can carry a scalar, vector or matrix. To explicitly show the type of the signal on each wire set the options Format+PortSignal Displays+Signal Dimensions and Format+PortSignal Displays+Wide Nonscalar Lines from the Simulink® toolbar.

Workspace Variables and Callbacks

Most Toolbox Simulink® blocks have parameters and these can be any MATLAB® expression comprising constants, function calls or MATLAB® workspace variables. Some of the provided Simulink® models set their own parameters in the workspace, by using a callback function. Each model has a number of callback functions that are invoked on different events, these can be seen from the menu at File+Model Properties+Callbacks. The PreLoadFcn callback is invoked when the model is being loaded and is used by some models to set the parameters. As a Toolbox convention, this parameter initialization code displays a message in the command window to let you know that it has updated the workspace.

Simulink® Version

The Simulink® models for this book has been developed and tested using MATLAB® R2010a under Mac OS X. A number of recent features of Simulink® are used so older versions are unlikely to work. Please do not report bugs if you are using Simulink® with a MATLAB® version older than R2010a.

Notes on Implementation

Some of the Simulink® blocks are implemented in MATLAB® code as S-files. These are functions written in the MATLAB® M language in a proscribed form in order to interface with the Simulink® simulation engine. While at first sight quite daunting the wrapping of existing functions is quite straightfoward and has the advantage that tried and true functions can be made accessible to the Simulink® environment. See the relevant Mathworks manuals for more information about writing S-files.

Simulink® Blocks

Arm Robots

Robot	represents a serial-link robot, with input of generalized joint force input and output of joint coordinates, velocities and accelerations. The parameters are the robot object to be simulated and the initial joint angles. It computes the forward dynamics of the robot.	`SerialLink.fdyn`
RNE	computes the inverse dynamics using the recursive Newton-Euler algorithm. Inputs are joint coordinates, velocities and accelerations and the output is the generalized joint force. The robot object is a parameter.	`SerialLink.rne`
jacob0	outputs a manipulator Jacobian matrix, with respect to the world frame, based on the input joint coordinate vector. The robot object is a parameter.	`SerialLink.jacob0`
jacobn	outputs a manipulator Jacobian matrix, with respect to the end-effector frame, based on the input joint coordinate vector. The robot object is a parameter.	`SerialLink.jacobn`
fkine	outputs a homogeneous transformation for the pose of the end-effector corresponding to the input joint coordinates. The robot object is a parameter.	`SerialLink.fkine`
plot	creates a graphical animation of the robot in a new window. The robot object is a parameter.	`SerialLink.plot`

Other Robots

Bicycle	is the kinematic model of a mobile robot that uses the bicycle model. The inputs are speed and steer angle and the outputs are position and orientation.
Pose integral	integrates a spatial velocity over time and outputs a homogeneous transformation. The parameter is the initial pose.
Quadcopter	is the dynamic model of a quadcopter. The inputs are rotor speeds and the output is translational and angular position and velocity. Parameter is a quadcopter structure.
ControlMixer	accepts thrust and torque commands and outputs rotor speeds for a quadcopter.

	Quadcopter plot	creates a graphical animation of the quadcopter in a new window. Parameter is a quadcopter structure.

Trajectory

`jtraj`	**jtraj**	outputs coordinates of a point following a quintic polynomial as a function of time, as well as its derivatives. Initial and final velocity are assumed to be zero. The parameters include the initial and final points as well as the overall motion time.
`lspb`	**lspb**	outputs coordinates of a point following an LSPB trajectory as a function of time. The parameters include the initial and final points as well as the overall motion time.
	circle	outputs the *xy*-coordinates of a point around a circle. Parameters are the centre, radius and angular frequency.

Vision

`Camera.project`	**camera**	input is a camera pose and the output is the coordinates of points projected on the image plane. Parameters are the camera object and the point positions.
`Camera2.project`	**camera2**	input is a camera pose and point coordinate frame pose, and the output is the coordinates of points projected on the image plane. Parameters are the camera object and the point positions relative to the point frame.
`CentralCamera.visjac`	**image Jacobian**	input is image points and output is the point feature Jacobian. Parameter is the camera object.
`SphericalCamera.visjac`	**image Jacobian sphere**	input is image points in spherical coordinates and output is the point feature Jacobian. Parameter is a spherical camera object.
`CentralCamera.estpose`	**Pose estimation**	computes camera pose from image points. Parameter is the camera object.

Miscellaneous

	Inverse	outputs the inverse of the input matrix.
	Pre multiply	outputs the input homogeneous transform pre-multiplied by the constant parameter.
	Post multiply	outputs the input homogeneous transform post-multiplied by the constant parameter.
	inv Jac	inputs are a square Jacobian $\boldsymbol{J}$ and a spatial velocity $\boldsymbol{\nu}$ and outputs are $\boldsymbol{J}^{-1}\boldsymbol{\nu}$ and the condition number of $\boldsymbol{J}$.
	pinv Jac	inputs are a Jacobian $\boldsymbol{J}$ and a spatial velocity $\boldsymbol{\nu}$ and outputs are $\boldsymbol{J}^{+}\boldsymbol{\nu}$ and the condition number of $\boldsymbol{J}$.
`tr2delta`	**tr2diff**	computes $\Delta(\cdot)$, the difference between two homogeneous transformations as a 6-vector comprising the translational and rotational difference.
`transl`	**xyz2T**	converts a translational vector to a homogeneous transformation matrix.
`rpy2tr`	**rpy2T**	converts a vector of roll-pitch-yaw angles to a homogeneous transformation matrix.
`eul2tr`	**eul2T**	converts a vector of Euler angles to a homogeneous transformation matrix.

T2xyz	converts a homogeneous transformation matrix to a translational vector.	`transl`
T2rpy	converts a homogeneous transformation matrix to a vector of roll-pitch-yaw angles.	`tr2rpy`
T2eul	converts a homogeneous transformation matrix to a vector of Euler angles.	`tr2eul`
angdiff	computes the difference between two input angles modulo 2π.	`angdiff`

C MATLAB® Objects

The MATLAB® programming language, known as 'M', has syntax and semantics somewhat similar to the classical language Fortran. In particular array indices start from one not zero, and subscripts are indicated by parentheses just like function call arguments. In early versions of MATLAB® the only data type was a two-dimensional matrix of real or complex numbers and a scalar was just a 1×1 matrix. This changed with the release of MATLAB® version 5.0 in 1997 which introduced many features that are part of the language today: structures, cells arrays and classes.

The early computer languages (Fortran, Pascal, C) are imperative languages in which the programmer describes computation in terms of *actions* that change the program's state – its data. The program is a logical procedure that takes input data, processes it, and produces output data. As program size and complexity grew the limitations of imperative programming became evident and new languages were designed to address these shortcomings.

A very powerful idea, dating from the mid 1980s, was object-oriented programming (OOP). The OOP programming model is organized around *objects* rather than *actions*. Each object encapsulates data and the functions, known as *methods*, to manipulate that object's data. The inner details of the object need not be known to the programmer using the object. The object presents a clean interface through its methods which makes large software projects easier to manage.

OOP languages support the concept of object classes. For example, we might define a class that represents a quaternion and which has methods to return the inverse of the quaternion, multiply two quaternions or to display a quaternion in a human-readable form. Our program might have a number of quaternion variables, or *objects*, and each is an *instance* of the quaternion class. Each instance has its own value, the data part of the object, but it shares the methods defined for the class.

Well known OOP languages such as C++, Java, Python and Ruby are still imperative in style but have language features to support objects. MATLAB® shares many features with these other well-known OOP languages and the details are provided in the MATLAB® documentation. The Toolboxes define a number of classes to represent robot arms, robot arm links, quaternions, robot path planners and various types of image feature. Toolbox classes are shown in bold font in the index of functions on page 554.

The use of objects provides a solution to the namespace pollution problem that occurs when using many MATLAB® toolboxes. When a MATLAB® function is invoked it is searched for in a list of directories – the MATLAB® search path. If the search path contains lots of Toolboxes from various sources the chances of two functions having the same name increases and this is problematic. If instead of functions we provide methods for objects then those method names don't occupy the function namespace, and can only be invoked in the context of the appropriate object.

Using a Class

The following illustrates some capabiltiies of the quaternion class provided as part of the Robotics Toolbox. A quaternion object is created by

```
>> q =       ( rotx(0.2) );
```

which invokes the *constructor* method for the class. By convention class names begin with a capital letter. This method checks the types of arguments and computes the equivalent quaternion. The quaternion's scalar and vector components are stored within this particular object or *instance* of the quaternion class. In MATLAB® the data part of an object is referred to as its *properties*. The arguments to the constructor can be a rotation matrix (as in this case), an angle and a vector, a 4-vector comprising the scalar and vector parts, or another quaternion. The result is a new object in the workspace

```
>> about(q)
q [Quaternion] : 1x1 (88 bytes)
```

and it has the type `Quaternion`. In a program we can inquire about the type of an object

```
>> class(q)
ans =
  Quaternion
```

which returns a string containing the name of the object's class. All MATLAB® objects have a class

```
>> x = 3
>> class(x)
ans =
  double
```

and this class `double` is built in, unlike `Quaternion` which is user defined. We can test the class of an object

```
>> isa(q, 'double')
ans =
     0
>> isa(q, 'Quaternion')
ans =
     1
```

We can access the properties of the quaternion object by

```
>> q.s
ans =
    0.9950
```

which returns the value of the scalar part of the quaternion. However the Toolbox implementation of the `Quaternion` does not allow this property to be set

```
>> q.s = 0.5;
??? Setting the 's' property of the 'Quaternion' class is not allowed.
```

since the scalar and vector part should be set together to achieve some consistent quaternion value.

We can compute the inverse of the quaternion by

```
>> qi = inv(q);
```

which returns a new quaternion `qi` equal to the inverse of `q`.

MATLAB® checks the type of the first argument and because it is a `Quaternion` it invokes the `inv` method of the `Quaternion` class. Most object-oriented languages use the *dot* notation which would be

```
>> qi = q.inv();
```

which makes it very clear that we are invoking the `inv` method of the object `q`. Either syntax is permissible in MATLAB® but in this book we use the dot notation for clarity. MATLAB® does not require the empty parentheses either, we could write

```
>> qi = q.inv
```

but for consistency with object-oriented practice in other languages, and to avoid confusion with accessing properties, we will always include them.

Any MATLAB® expression without a trailing semicolon will display the value of the expression. For instance

```
>> qi
qi =
0.995 < -0.099833, 0, 0 >
```

causes the `display` method of the quaternion to be invoked. It is exactly the same as typing

```
>> qi.display()
qi =
0.995 < -0.099833, 0, 0 >
```

This in turn invokes the `char` method to convert the quaternion value to a string

```
>> s = qi.char();
>> about(s)
s [char] : 1x25 (50 bytes)
```

We will create another quaternion

```
>> q2 = quaternion( roty(0.3) );
```

and then compute the product of the two quaternions which we can write concisely as

```
>> q * q2
```

This is an example of operator overloading which is a feature of many object-oriented languages. MATLAB® interprets this as

```
>> q.mtimes(q2)
```

For more complex expressions operator overloading is critical to expressivity, for example we can write

```
>> q*q2*q
ans =
0.96906 < 0.19644, 0.14944, 0 >
```

and MATLAB® does the hardwork of computing the first product `q*q2` into a temporary quaternion, multiplying that by `q` and then deleting the temporary quaternion. To implement this without operator overloading would be the nightmare expression

```
>>  q.mtimes( q2.mtimes(q) )
ans =
0.96906 < 0.19644, 0.14944, 0 >
```

which is both difficult to read and to maintain.

Creating a Class

The quaternion class is defined by the Toolbox file `Quaternion.m` which is over 500 lines long but the basic structure is

```
1   classdef Quaternion
2
3       properties (SetAccess = private)
4           s       % scalar part
5           v       % vector part
6       end
7
8       methods
9
10          function q = Quaternion(a1, a2)
11              % constructor
12          end
13            .
14           % other methods
15            .
16            .
17      end
18  end
```

The `properties` block, lines 3–6, defines the data associated with each quaternion instance, in this case the internal representation is the scalar part in the variable `s` and the vector part in the variable `v`. The methods block, lines 8–17, defines all the methods that the class supports. The name after `classdef` at line 1 must match the name of the file and is the name of the class.

The properties have a `SetAccess` mode `private` which means that the properties can be read directly by programs but not set. If `q` is a quaternion object then `q.s` would be the value of the scalar part of the quaternion. This is a matter of programming style, and some people prefer that all access to object properties is via explicit *getter* functions such as `q.get_s()`.

Every class must have a *constructor* method which is a function with the same name as the class. The constructor is responsible for initialising the data of the object, in this case its properties `s` and `v`. Some object-oriented languages also support a *destructor* function that is invoked when an object is no longer needed, in MATLAB® this is the optional method `delete`.

The quaternion class implements 20 different methods. Each method is written as a MATLAB® function with an `end` statement. The first argument to each method is the quaternion object itself. For example the method that returns the inverse of a quaternion is

```
function qi = inv(q)
    qi = Quaternion( [q.s -q.v] );
end
```

which uses the constructor method `Quaternion` to create the quaternion that it returns.

The method to convert a quaternion to a string is

```
function s = char(q)
    s = [ num2str(q.s), ' < ' num2str(q.v(1)) ...
    ', ' num2str(q.v(2)) ', '   num2str(q.v(3)) ' >' ];
end
```

The method `mtimes` is invoked for operator overloading whenever the operand on either side of an asterisk is a quaternion object.

```
function qp = mtimes(q1, q2)
    if ~isa(q1, 'Quaternion')
        error('left-hand side of * must be a Quaternion');
    end

    if isa(q2, 'Quaternion')
       %Multiply unit-quaternion by unit-quaternion
        s1 = q1.s;  v1 = q1.v;
        s2 = q2.s;  v2 = q2.v;
        qp = Quaternion([s1*s2-v1*v2' s1*v2+s2*v1+cross(v1,v2)]);
    elseif  isa(q2, 'double'),
        if length(q2) == 3
            % Multiply vector by unit-quaternion
            qp = q1 * Quaternion([0 q2(:)']) * inv(q1);
            qp = qp.v(:);
        elseif length(q2) == 1
            % Multiply quaternion by scalar
            qp = Quaternion( double(q1)*q2);
        else
            error('quaternion-vector product: must be a 3-vector
or scalar');
        end
    end
end
```

The method tests the type of the second operand and computes either a quaternion-quaternion, quaternion-vector or quaternion-scalar product.

MATLAB® supports virtual properties which are defined in the quaternion implementation by

```
properties (Dependent = true)
    R   % rotation matrix
    T   % translation matrix
end
```

These two properties can be accessed like normal properties, for example `q.R` or `q.T`. However when accessed they invoke the methods `get.R` and `get.T` respectively. The implementation for `get.R`

```
function r = get.R(q)
    r = t2r( q2tr(q) );
end
```

converts the quaternion to a homogeneous transformation and then to an orthonormal rotation matrix.

MATLAB® classes support inheritance. This is a feature whereby a new class can inherit the properties and methods of an existing class and extend that with additional properties or methods. In Part II the various planners such as `Dstar` and `RRT` inherit from the class `Navigation` and in Part IV the different types of camera such as `CentralCamera` and `FishEyeCamera` inherit from the class `Camera`. Inheritance is indicated at the `classdef` line, for example

```
classdef Dstar < Navigation
```

Inhertitance, particularly multiple inheritance, is a complex topic and the MATLAB® documentation should be referred to for the details.

The MATLAB® functions `methods` and `properties` return the methods and properties of an object. The function `metaclass` returns a data structure that includes all methods, properties and parent classes.

Pass by Reference

One particularly useful application of inheritance is to get around the problem of *pass by value*. Whenever a variable is passed to a function MATLAB® passes its value, that is a copy of it, rather than a reference to it. This is normally quite convenient, but consider now the case of some object which has a method that changes a property. If we write

```
>> myobj.set_x(2);
```

then MATLAB® creates a copy of the object `myobj` and invokes the `set_x()` method on the copy. However since we didn't assign the copied object to anything the change is lost. The correct approach is to write this as

```
>> myobj = myobj.set_x(2);
```

which is cumbersome. If however the object `myobj` belongs to a *reference class* then we can write

```
>> myobj.set_x(2);
```

and the value of `myobj` would change. To create a reference class the class must inherit from the `handle` class

```
classdef MyClass < handle
```

A number of classes within the Toolbox, but not the `Quaternion` class, are reference classes. A possible trap with reference classes is that an assignment of a reference class object

```
>> myobj2 = myobj;
```

means that `myobj2` points to the same object as `myobj`. If `myobj` changes then so does `myobj2`. It is good practice for an object constructor to accept an argument of the class type and to return a copy

```
>> myobj2 = MyClass(myobj);
```

so now changes to `myobj` will not effect `myobj2`

Arrays of Objects

MATLAB® handles arrays or vectors of objects in a very familiar way. Consider the example of an array of SIFT feature objects (from page 384)

```
>> s1 = isurf(im1);
```

which returns a vector of `SurfPointFeature` objects. We can determine the number of objects in the vector

```
>> n = length(s1)
n =
   1288
```

or perform indexing operations such as

```
>> x = s1(1:100);
>> y = s1(1:20:end);
```

Note that the `SurfPointFeature` objects are reference objects so the elements of `x` and `y` are the same objects as referred to by `s1`. We can also delete objects from the vector

```
>> s1(50:end) = [];
```

Invoking a method on an object array, for example the hypothetical method

```
>> z = s1.fewer();
```

results in the entire vector being passed to the method

```
function r = fewer(s)
    r = s(1:20:end);
end
```

so methods can perform operations on single objects or arrays of objects.

A class that supports vectors must have a constructor that handles the case of no passed arguments.

Multi-File Implementation

For a complex class a single file might be too long to be workable and it would be preferable to have multiple files, one per method or group of methods. This would certainly be the case if some of the methods were defined as MEX-files rather than M-files.

In MATLAB® this is handled by creating a directory in the MATLAB® search path with an '@' symbol prefix. The `SerialLink` class which represents a robot arm is defined this way, and all its files are within a directory called `@SerialLink`.

D Linear Algebra Refresher

A taxonomy of matrices is shown in Fig. D.1. In this book we are concerned only with real $m \times n$ matrices

$$\boldsymbol{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \\ a_{m,1} & a_{n,2} & \cdots & a_{m,n} \end{pmatrix}, \quad \boldsymbol{A} \in \mathbb{R}^{m \times n}$$

with m rows and n columns. If $n = m$ the matrix is square.

The transpose is

$$\boldsymbol{B} = \boldsymbol{A}^T, b_{i,j} = a_{j,i} \quad \forall i, j$$

and it can be shown that

$$(\boldsymbol{AB})^T = \boldsymbol{B}^T \boldsymbol{A}^T, (\boldsymbol{ABC})^T = \boldsymbol{C}^T \boldsymbol{B}^T \boldsymbol{A}^T, \text{ etc.}$$

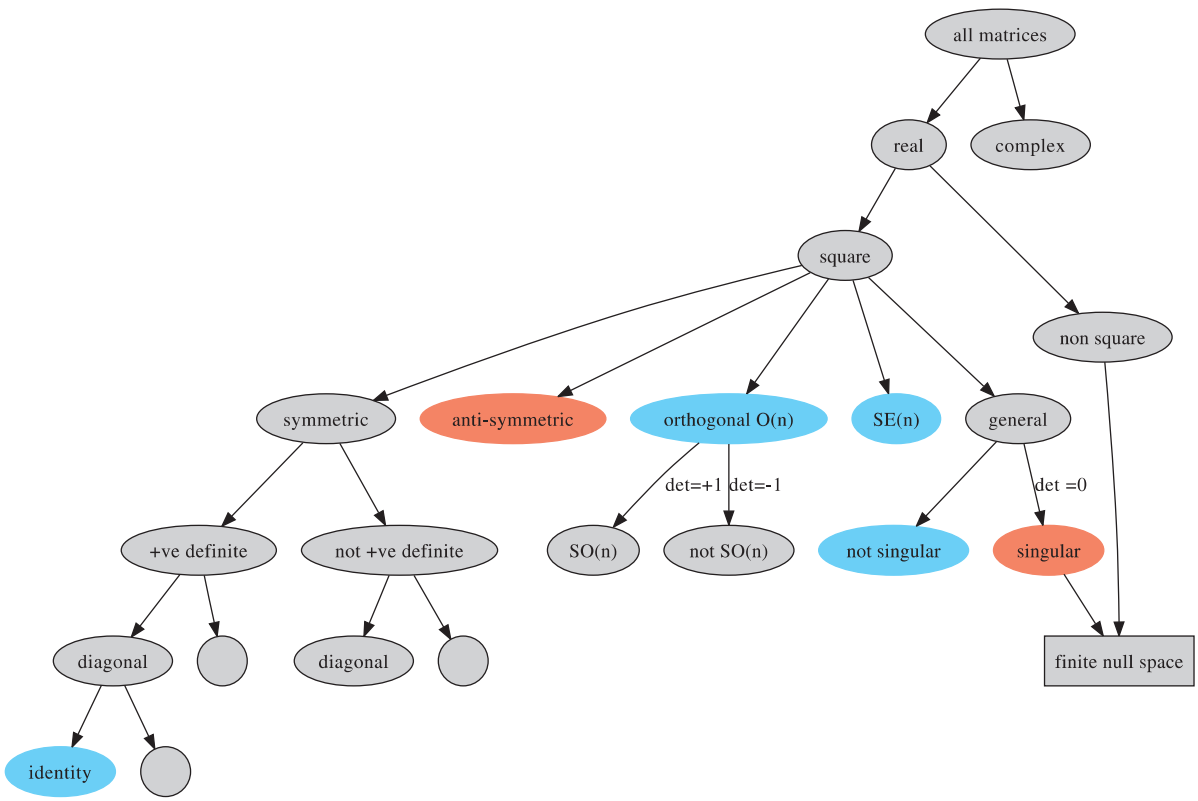


Fig. D.1. Taxonomy of matrices. Classes of matrices that are always singular are shown in red, those that are never singular are shown in blue

A square matrix may have an inverse $\boldsymbol{A}^{-1}$ in which case

```
Ai = inv(A)
```

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}_{n\times n}$$

where

$$\boldsymbol{I}_{n\times n} = \begin{pmatrix} 1 & & 0 \\ & 1 & \\ & \ddots & \\ 0 & & 1 \end{pmatrix} \in \mathbb{R}^{n\times n}$$

is the identity matrix, a unit diagonal matrix. The inverse exists provided that the matrix is non-singular, that is its determinant $\det(A) \neq 0$. If $\boldsymbol{A}$ and $\boldsymbol{B}$ are square and non-singular then

$$(\boldsymbol{A}\boldsymbol{B})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1},\ (\boldsymbol{A}\boldsymbol{B}\boldsymbol{C})^{-1} = \boldsymbol{C}^{-1}\boldsymbol{B}^{-1}\boldsymbol{A}^{-1},\ \text{ etc.}$$

and also

$$(\boldsymbol{A}^T)^{-1} = (\boldsymbol{A}^{-1})^T$$

For a square matrix if

$\boldsymbol{A} = \boldsymbol{A}^T$ the matrix is **symmetric**. The inverse of a symmetric matrix is also symmetric. Many matrices that we encounter in robotics are symmetric, for example covariance matrices and manipulator inertia matrices.

$\boldsymbol{A} = -\boldsymbol{A}^T$ the matrix is **anti-symmetric** or **skew-symmetric**. Such a matrix has a zero diagonal and the property that $\boldsymbol{v}^T\boldsymbol{S}(\boldsymbol{v}) = 0, \forall \boldsymbol{v}$. For the 3×3 case

```
S = skew(v)
```

$$\boldsymbol{S}(\boldsymbol{v}) = \begin{pmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{pmatrix} \tag{D.1}$$

and the inverse operation is

```
v = vex(S)
```

$$\boldsymbol{v} = \text{vex}(\boldsymbol{S})$$

Also $\boldsymbol{v}_1 \times \boldsymbol{v}_2 = \boldsymbol{S}(\boldsymbol{v}_1)\boldsymbol{v}_2$.

$\boldsymbol{A}^{-1} = \boldsymbol{A}^T$ the matrix is **orthogonal**. The matrix is also known as orthonormal since its column vectors (and row vectors) must be of unit length and orthogonal to each other. The product of two orthogonal matrices of the same size is an orthogonal matrix. The set of $n \times n$ orthogonal matrices forms a group $O(n)$, known as the orthogonal group. The determinant of an orthogonal matrix is either $+1$ or -1. The subgroup $SO(n)$ consisting of orthogonal matrices with determinant $+1$ is called the special orthogonal group, and each of its elements is a special orthogonal matrix. The columns (and rows) are orthogonal vectors, that is, their dot product is zero. The product of two orthogonal matrices is also orthogonal.

$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^T$ the matrix is **normal** and can be diagonalized by an orthogonal matrix $\boldsymbol{U}$ so that $\boldsymbol{U}^T\boldsymbol{A}\boldsymbol{U}$ is a diagonal matrix. All symmetric, skew-symmetric and orthogonal matrices are normal matrices.

For a non-square matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ we can determine the left generalized inverse or pseudo inverse or Moore-Penrose pseudo inverse

$$\boldsymbol{A}^{+}\boldsymbol{A} = \boldsymbol{I}_{n\times n}$$

where $\boldsymbol{A}^{+} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$. The right generalized inverse

$$\boldsymbol{A}\boldsymbol{A}^{+} = \boldsymbol{I}_{m\times m}$$

where $\boldsymbol{A}^{+} = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}$.

The square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ can be applied as a linear transformation to a vector $\boldsymbol{x} \in \mathbb{R}^n$

$$\boldsymbol{x}' = \boldsymbol{A}\boldsymbol{x}$$

which results in another vector, generally with a change in its length and direction. However there are some important special cases. If $\boldsymbol{A} \in SO(n)$ the transformation is isometric and the vector's *length* is unchanged $|\boldsymbol{x}'| = |\boldsymbol{x}|$.

In 2-dimensions if $\boldsymbol{x}$ is the set of all points lying on a circle then $\boldsymbol{x}'$ defines points that lie on an ellipse. The MATLAB® builtin demonstration

```
>> eigshow
```

shows this very clearly as you interactively drag the tip of the vector $\boldsymbol{x}$ around the unit circle.

```
[x,e] = eig(A)
```

The eigenvectors of a square matrix are those vectors $\boldsymbol{x}$ such that

$$\boldsymbol{A}\boldsymbol{x} = \lambda_i\boldsymbol{x} \tag{D.2}$$

that is, their direction is unchanged when transformed by the matrix. They are simply scaled by λ_i, the corresponding eigenvalue. The matrix has n eigenvalues which can be real or complex. For an orthogonal matrix the eigenvalues lie on a unit circle in the complex plane, $|\lambda_i| = 1$, and the eigenvectors are all orthogonal to one another.

A symmetric matrix is positive definite if all its eigenvalues are positive

$$\lambda_i > 0, \quad \forall i$$

and is positive semi-definite if

$$\lambda_i \geq 0, \quad \forall i$$

If $\boldsymbol{A}$ is non singular then the eigenvectors of $\boldsymbol{A}^{-1}$ are the same as $\boldsymbol{A}$ and the eigenvalues of $\boldsymbol{A}^{-1}$ are the reciprocal of those of $\boldsymbol{A}$. The eigenvalues of $\boldsymbol{A}^T$ are the same as those of $\boldsymbol{A}$ but the eigenvectors are different.

The matrix form of Eq. D.2 is

$$\boldsymbol{A}\boldsymbol{X} = \boldsymbol{X}\boldsymbol{\Lambda}$$

where $\boldsymbol{X} \in \mathbb{R}^{n\times n}$ is a matrix of eigenvectors of $\boldsymbol{A}$, arranged column-wise, and $\boldsymbol{\Lambda}$ is a diagonal matrix of corresponding eigenvalues. If $\boldsymbol{X}$ is not singular we can rearrange this as

$$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^{-1}$$

which is the eigenvalue or spectral decomposition of the matrix. This implies that the matrix can be diagonalized by a similarity transform

$$\boldsymbol{\Lambda} = \boldsymbol{X}^{-1}\boldsymbol{A}\boldsymbol{X}$$

If $\boldsymbol{A}$ is normal (for example symmetric) then $\boldsymbol{X}$ is orthogonal and we can instead write

$$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^T \tag{D.3}$$

The matrices $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}\boldsymbol{A}^T$ are always symmetric and positive semidefinite. This implies than any symmetric matrix $\boldsymbol{A}$ can be written as

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$$

where $\boldsymbol{L}$ is the Cholesky decomposition of $\boldsymbol{A}$. `L = chol(A)`

The matrix $\boldsymbol{R}$ such that

$$\boldsymbol{A} = \boldsymbol{R}\boldsymbol{R}$$

is the square root of $\boldsymbol{A}$ or $\boldsymbol{A}^{½}$.

If $\boldsymbol{T}$ is any non-singular matrix then

$$\boldsymbol{A} = \boldsymbol{T}\boldsymbol{B}\boldsymbol{T}^{-1}$$

is known as a similarity transform and $\boldsymbol{A}$ and $\boldsymbol{B}$ are said to be similar, and it can be shown that the eigenvalues are unchanged by the transformation.

The determinant of a square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is the factor by which the transformation changes changes volumes in an n-dimensional space. For 2-dimensions imagine a shape defined by points $\boldsymbol{x}_i$ with an enclosed area a. The shape formed by the points $\boldsymbol{A}\boldsymbol{x}_i$ would have an enclosed area $a\det(A)$. If $\boldsymbol{A}$ is singular the points $\boldsymbol{A}\boldsymbol{x}_i$ would lie at a single point or along a line and have zero enclosed area. In a similar way for 3-dimensions, the determinant is a scale factor applied to the volume of a set of points mapped through the transformation $\boldsymbol{A}$. The determinant of a skew-symmetric matrix is always zero $\det(\boldsymbol{S}(\cdot)) = 0$. `det(A)`

The determinant is equal to the product of the eigenvalues

$$\det(\boldsymbol{A}) = \prod_{i=1}^{n} \lambda_i$$

thus a matrix with one or more zero eigenvalues will be singular. A positive definite matrix, $\lambda_i > 0$, therefore has $\det(A) > 0$ and is not singular. The trace of a matrix is the sum of the diagonal elements `trace(A)`

$$\mathrm{tr}(\boldsymbol{A}) = \sum_{i=1}^{n} \boldsymbol{A}_{ii}$$

which is also the sum of the eigenvalues

$$Tr(\boldsymbol{A}) = \sum_{i=1}^{n} \lambda_i$$

The columns of $\boldsymbol{A} = (\boldsymbol{c}_1\boldsymbol{c}_2 \cdots \boldsymbol{c}_n)$ can be considered as a set of vectors that define a space – the column space. Similarly, the rows of $\boldsymbol{A}$ can be considered as a set of vectors that define a space – the row space. The column rank of a matrix is the number of linearly independent columns of $\boldsymbol{A}$. Similarly, the row rank is the number of linearly independent rows of $\boldsymbol{A}$. The column rank and the row rank are always equal and are simply called the rank of $\boldsymbol{A}$ and the rank has an upper bound of $\min(m, n)$. A square matrix for which $\mathrm{rank}(\boldsymbol{A}) < n$ is said to be rank deficient or not of full rank. `rank(A)`

If the matrix $\boldsymbol{A}$ is not of full rank then it has a finite null space or kernel. A vector $\boldsymbol{x}$ lies in the null space of the matrix if

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$$

More precisely this is the right-null space. A vector lies in the left-null space if

$$\boldsymbol{x}\boldsymbol{A} = \boldsymbol{0}$$

The left null space is equal to the right null space of $\boldsymbol{A}^T$.

`null(A)` The null space is defined by a set of orthogonal basis vectors whose dimension is called the nullity of $\boldsymbol{A}$ and is equal to $n - \text{rank}(\boldsymbol{A})$. Any linear combination of these null-space basis vectors lies in the null space.

For a non-square matrix $\boldsymbol{A} \in \mathbf{R}^{m \times n}$ the analog to Eq. D.2 is

$$\boldsymbol{A}\boldsymbol{v}_i = \sigma_i \boldsymbol{u}_i$$

where $\boldsymbol{u}_i \in \mathbf{R}^m$ and $\boldsymbol{v}_i \in \mathbf{R}^n$ are respectively the right- and left-singular vectors of $\boldsymbol{A}$, and σ_i its singular values. The singular values are non-negative real numbers that are the square root of the eigenvalues of $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{u}_i$ are the corresponding eigenvectors. $\boldsymbol{v}_i$ are the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$.

`[U,S,Vt] = svd(A)` The singular value decomposition or SVD of the matrix $\boldsymbol{A}$ is

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

where $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ are both orthogonal matrices comprising, as columns, the corresponding singular vectors $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$. $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times m}$ is a diagonal matrix of the singular values

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & \boldsymbol{0} & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & \boldsymbol{0} & & & \ddots & \\ & & & & & 0 \end{pmatrix}$$

where $r = \text{rank}(\boldsymbol{A})$ is the rank of $\boldsymbol{A}$. For the case where $r < n$ the diagonal will have `cond(A)` zero elements as shown. The condition number of a matrix $\boldsymbol{A}$ is $\max \sigma_i / \min \sigma_i$ and a high value means the matrix is close to singular or "poorly conditioned".

The matrix quadratic form

$$s = \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \tag{D.4}$$

is a scalar. For the case that $\boldsymbol{A}$ is diagonal this can be written

$$s = \sum_{i=1}^{n} A_{ii} x_i^2$$

which is a weighted sum of squares. If $\boldsymbol{A}$ is symmetric then

$$s = \sum_{i=1}^{n} A_{ii} x_i^2 + 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} A_{ij} x_i x_j$$

the result also includes products or correlations between elements of $\boldsymbol{x}$.

Real matrices are a subset of all matrices. For the general case of complex matrices the term Hermitian is the analog of symmetric, and unitary the analog of orthogonal. $\boldsymbol{A}^H$ denotes the Hermitian transpose, the complex conjugate transpose of the complex matrix $\boldsymbol{A}$.

Solving Systems of Equations

We frequently need to solve systems of linear equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

where $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ and $\boldsymbol{b} \in \mathbb{R}^n$ are known, and $\boldsymbol{x} \in \mathbb{R}^m$ is unknown. If $n = m$ then $\boldsymbol{A}$ is square, and if $\boldsymbol{A}$ is non-singular then the solution is obtained using the matrix inverse

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

If $n > m$ the system is over constained we use the pseudo inverse

$$\boldsymbol{x} = \boldsymbol{A}^{+}\boldsymbol{b}$$

which gives $\boldsymbol{x}$ that minimizes the norm of the residual $|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}|$. Using SVD where $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ this is

$$\boldsymbol{x} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T\boldsymbol{b}$$

where $\boldsymbol{\Sigma}^{-1}$ is simply the element-wise inverse of the diagonal elements of $\boldsymbol{\Sigma}^T$.

If the matrix is singular, or the system is under constrained $n < m$, then there infinitely many solutions. We can again use the SVD approach

$$\boldsymbol{x} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T\boldsymbol{b}$$

where this time $\boldsymbol{\Sigma}^{-1}$ is the element-wise inverse of the *non-zero* diagonal elements of $\boldsymbol{\Sigma}$, all other zeros are left in place.

In MATLAB® all these problems can be solved using the backslash operator

```
>> x = A\b
```

Singular value decomposition can also be used to estimate a rotation matrix given a set of vectors $\{(\boldsymbol{p}_i, \boldsymbol{q}_i), i = 1 \cdots N\}$ for which $\boldsymbol{q}_i = \boldsymbol{R}\boldsymbol{p}_i$. We first compute the moment matrix

$$\boldsymbol{M} = \sum_{i=1}^{N} \boldsymbol{q}_i \boldsymbol{p}_i^T$$

and take then compute the SVD $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. The least squares estimate of the rotation matrix is

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^T$$

and is guaranteed to be an orthogonal matrix.

E Ellipses

An ellipse belongs to the family of planar curves known as conics. The simplest form of an ellipse is defined implicitly

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

and is shown in Fig. E.1a. This canonical ellipse is centered at the origin and has its major and minor axes aligned with the x- and y-axes. The radius in the x-direction is a and in the y-direction is b. The longer of the two radii is known as the semi-major axis length and the other is the semi-minor axis length.

We can write the ellipse in matrix quadratic form Eq. D.4 as

$$\begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} 1/a^2 & 0 \\ 0 & 1/b^2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = 1$$

$$\boldsymbol{x}^T \begin{pmatrix} a^2 & 0 \\ 0 & b^2 \end{pmatrix}^{-1} \boldsymbol{x} = 1 \tag{E.1}$$

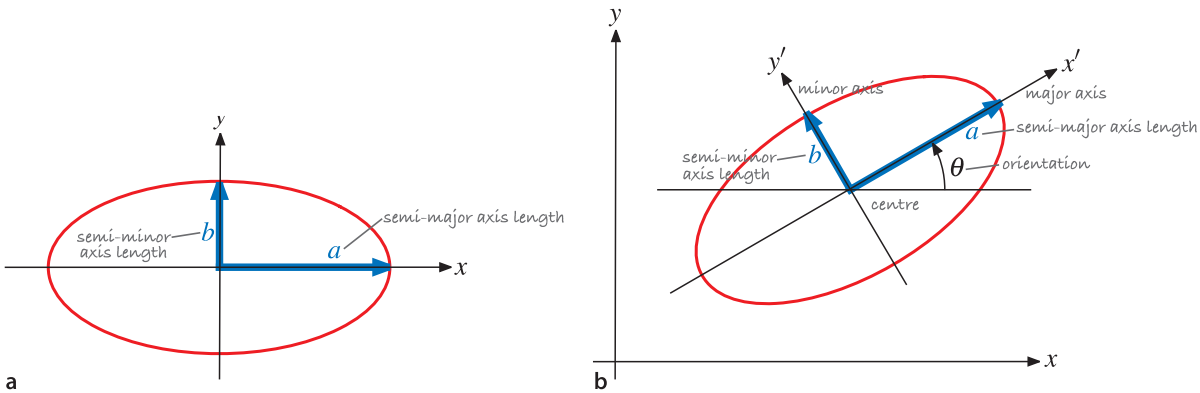
$$\boldsymbol{x}^T \boldsymbol{E}^{-1} \boldsymbol{x} = 1 \tag{E.2}$$

In the most general form $\boldsymbol{E}$ is a symmetric matrix

$$\boldsymbol{E} = \begin{pmatrix} A & C \\ C & B \end{pmatrix} \tag{E.3}$$

and its determinant $\det(E) = AB - C^2$ defines the type of conic

$$\det(\boldsymbol{E}) \begin{cases} > 0 & \text{ellipse} \\ = 0 & \text{parabola} \\ < 0 & \text{hyperbola} \end{cases}$$



a b

Fig. E.1. Ellipses. **a** Canonical ellipse centred at the origin and aligned with the x- and y-axes; **b** general form of ellipse

An ellipse is therefore represented by a positive definite symmetric matrix $\boldsymbol{E}$. Conversely any positive definite symmetric matrix, such as an inertia matrix or covariance matrix, can be represented by an ellipse.

Non-zero values of C change the orientation of the ellipse. The ellipse can be arbitrarily centred at $\boldsymbol{x}_c$ by writing it in the form

$$(\boldsymbol{x} - \boldsymbol{x}_c)^T \boldsymbol{E}^{-1} (\boldsymbol{x} - \boldsymbol{x}_c) = 1$$

which leads to the general ellipse shown in Fig. E.1b.

Since $\boldsymbol{E}$ is symmetric it can be diagonalized by Eq. D.3

$$\boldsymbol{E} = \boldsymbol{X} \boldsymbol{\Lambda} \boldsymbol{X}^T$$

where $\boldsymbol{X}$ is an orthogonal matrix comprising the eigenvectors of $\boldsymbol{E}$. The inverse is

$$\boldsymbol{E}^{-1} = \boldsymbol{X} \boldsymbol{\Lambda}^{-1} \boldsymbol{X}^T$$

so the quadratic form becomes

$$\begin{aligned} \boldsymbol{x}^T \boldsymbol{X} \boldsymbol{\Lambda}^{-1} \boldsymbol{X}^T \boldsymbol{x} &= 1 \\ (\boldsymbol{X}^T \boldsymbol{x})^T \boldsymbol{\Lambda}^{-1} (\boldsymbol{X}^T \boldsymbol{x}) &= 1 \\ \boldsymbol{x}'^T \boldsymbol{\Lambda}^{-1} \boldsymbol{x}' &= 1 \end{aligned}$$

This is similar to Eq. E.2 but with the ellipse defined by the diagonal matrix Λ with respect to the rotated coordinated frame $\boldsymbol{x}' = \boldsymbol{X}^T \boldsymbol{x}$. The major and minor ellipse axes are aligned with the eigenvectors of $\boldsymbol{E}$. The squared radii of the ellipse are the eigenvalues of $\boldsymbol{E}$ or the diagonal elements of $\boldsymbol{\Lambda}$. For the general case of $\boldsymbol{E} \in \mathbb{R}^{n \times n}$ the result is an ellipsoid in n-dimensional space. The Toolbox function `plot_ellipse` will draw an ellipse for the $n = 2$ case and an ellipsoid for the $n = 3$ case.

Alternatively the ellipse can be represented in polynomial form. If we write the ellipse as

$$(\boldsymbol{x} - (x_0, y_0))^T \begin{pmatrix} a & c \\ c & b \end{pmatrix} (\boldsymbol{x} - (x_0, y_0)) = 1$$

and expand we obtain

$$e_1 x^2 + e_2 y^2 + e_3 xy + e_4 x + e_5 y + e_6 = 0$$

where $e_1 = a, e_2 = b, e_3 = 2c, e_4 = -2(ax_0 + cy_0), e_5 = -2(by_0 + cx_0)$ and $e_6 = ax_0^2 + by_0^2 + 2cx_0y_0 - 1$. The ellipse has only five degrees of freedom, its centre coordinate and the three unique elements in $\boldsymbol{E}$. For a non-degenerate ellipse $e_1 \neq 0$ and we rewrite the polynomial in normalized form

$$x^2 + E_1 y^2 + E_2 xy + E_3 x + E_4 y + E_5 = 0 \tag{E.4}$$

with five unique parameters.

Properties

The area of an ellipse is πab and its eccentricity is

$$\varepsilon = \frac{\sqrt{a^2 - b^2}}{a}$$

The eigenvectors of $\boldsymbol{E}$ define the principal directions of the ellipse and the square root of the eigenvalues are the corresponding radii.

Consider the ellipse

$$\boldsymbol{x}\begin{pmatrix}2 & -1\\ -1 & 1\end{pmatrix}^{-1}\boldsymbol{x} = 1$$

which is represented in MATLAB® by

```
>> E = [2 -1; -1 1];
```

We can plot this by

```
>> plot_ellipse(E)
```

which is shown in Fig. E.2.

The eigenvectors and eigenvalues of $\boldsymbol{E}$ are

```
>> [x,e] = eig(E)
x =
   -0.5257   -0.8507
   -0.8507    0.5257
e =
    0.3820         0
         0    2.6180
```

and the ellipse radii are

```
>> r = sqrt(diag(e))
r =
    0.6180
    1.6180
```

which correspond to a and b respectively. If either radius is equal to zero the ellipse is degenerate and becomes a line. If both radii are zero the ellipse is a point.

The eigenvectors are unit vectors in the minor- and major-axis directions and we will scale them by the radii to yield radius vectors which we can plot

```
>> arrow([0 0]', x(:,1)*r(1));
>> arrow([0 0]', x(:,2)*r(2));
```

The orientation of the ellipse is the angle of the major-axis with respect to the horizontal axis and is

$$\theta = \tan^{-1}\frac{\boldsymbol{x}_y}{\boldsymbol{x}_x}$$

For our example this is

```
>> atan2(x(2,2), x(1,2)) * 180/pi
ans =
  148.2825
```

in units of degrees.

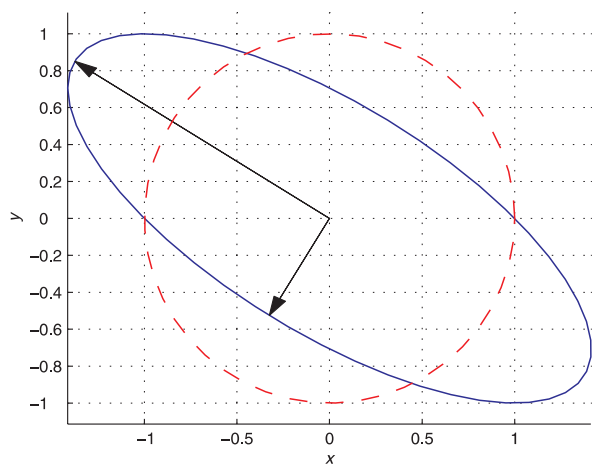


Fig. E.2.
Ellipse corresponding to symmetric 2×2 matrix, and the unit circle shown in red. The arrows indicate the major and minor axes of the ellipse

The ellipse area is $\pi r_1 r_2$ and the ellipsoid volume is $\frac{4}{3}\pi r_1 r_2 r_3$ where the radii $r_i = \sqrt{\lambda_i}$ where λ_i are the eigenvalues of $\boldsymbol{E}$. Since $\det(E) = \Pi\lambda_i$ the area or volume is proportional to $\sqrt{\det(\boldsymbol{E})}$.

Drawing an Ellipse

In order to draw an ellipse we first define a point $\boldsymbol{y} = [x, y]^T$ on the unit circle

$$\boldsymbol{y}^T\boldsymbol{y} = 1$$

and rewrite Eq. E.3 as

$$\boldsymbol{x}^T\boldsymbol{E}^{-\frac{1}{2}}\boldsymbol{E}^{-\frac{1}{2}}\boldsymbol{x} = 1$$

where $\boldsymbol{E}^{\frac{1}{2}}$ is the matrix square root (MATLAB® function `sqrtm`). Equating these two equations we can write

$$\boldsymbol{x}^T\boldsymbol{E}^{-\frac{1}{2}}\boldsymbol{E}^{-\frac{1}{2}}\boldsymbol{x} = \boldsymbol{y}^T\boldsymbol{y}$$

It is clear that

$$\boldsymbol{y} = \boldsymbol{E}^{-\frac{1}{2}}\boldsymbol{x}$$

which we can rearrange as

$$\boldsymbol{x} = \boldsymbol{E}^{-\frac{1}{2}}\boldsymbol{y}$$

which transforms a points on the unit circle to a point on an ellipse. If the ellipse is centered at $\boldsymbol{x}_c$ rather than the origin we can perform a change of coordinates

$$(\boldsymbol{x} - \boldsymbol{x}_c)^T\boldsymbol{E}^{-\frac{1}{2}}\boldsymbol{E}^{-\frac{1}{2}}(\boldsymbol{x} - \boldsymbol{x}_c) = 1$$

from which we write the transformation as

$$\boldsymbol{x} = \boldsymbol{E}^{\frac{1}{2}}\boldsymbol{y} + \boldsymbol{x}_c$$

Continuing the MATLAB® example above

```
>> E = [2 -1; -1 1];
```

We define a set of points on the unit circle

```
>> th = linspace(0, 2*pi, 50);
>> y = [cos(th);  sin(th)];
```

which we transform to points on the perimeter of the ellipse

```
>> x = (sqrtm(E) * y)';
>> plot(x(:,1), x(:,2));
```

which is encapsulated in the Toolbox function

```
>> plot_ellipse(E, [0 0])
```

An ellipsoid is described by a positive-definite symmetric 3×3 matrix. Drawing an ellipsoid is tackled in an analogous fashion and `plot_ellipse` is also able to display a 3-dimensional ellipsoid.

Fitting an Ellipse to Data

From a Set of Interior Points

We wish to find the equation of an ellipse that best fits a set of points that lie within the ellipse boundary. A common approach is to find the ellipse that has the same mass properties as the set of points. From the set of N points $\boldsymbol{x}_i = (x_i, y_i)$ we can compute the moments

$$m_{00} = N$$
$$m_{10} = \sum_{i=1}^{N} x_i$$
$$m_{01} = \sum_{i=1}^{N} y_i$$

The centre of the ellipse is taken to be the centroid of the set of points

$$(x_c, y_c) = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$$

which allows us to compute the central second moments

$$\mu_{20} = \sum_{i=1}^{N} (x_i - x_c)^2$$
$$\mu_{02} = \sum_{i=1}^{N} (y_i - y_c)^2$$
$$\mu_{11} = \sum_{i=1}^{N} (x_i - x_c)(y_i - y_c)$$

The inertia matrix for a general ellipse is the symmetric matrix

$$\boldsymbol{J} = \begin{pmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{pmatrix}$$

where the diagonal terms are the moments of inertia and the off-diagonal terms are the products of inertia. Inertia can be computed more directly by

$$\boldsymbol{J} = \sum_{i=1}^{N} (\boldsymbol{x} - \boldsymbol{x}_c)(\boldsymbol{x} - \boldsymbol{x}_c)^T$$

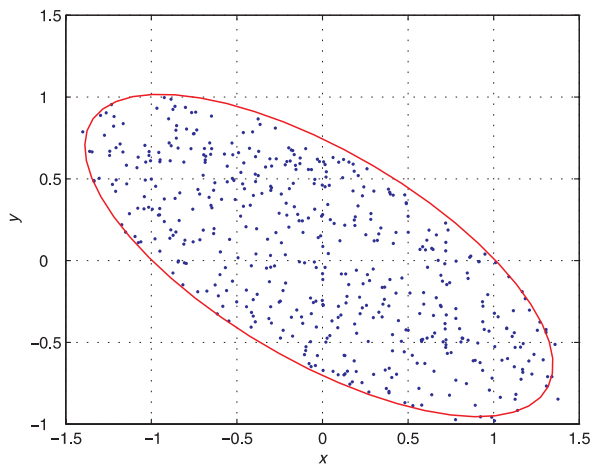


Fig. E.3. Point data

The relationship between the inertia matrix and the symmetric ellipse matrix is

$$\boldsymbol{E} = \frac{4}{m_{00}} \boldsymbol{J}$$

To demonstrate this we can create a set of points that lie within the ellipse used in the example above

```
% generate a set of points within the ellipse
p = [];
while true
    x = (rand(2,1)-0.5)*4;
    if norm(x'*inv(E)*x) <= 1
        p = [p x];
    end
    if numcols(p) >= 500
        break;
    end
end
plot(p(1,:), p(2,:), '.')

% compute the moments
m00 = mpq_point(p, 0,0);
m10 = mpq_point(p, 1,0);
m01 = mpq_point(p, 0,1);
xc = m10/m00; yc = m01/m00;

% compute second moments relative to centroid
pp = bsxfun(@minus, p, [xc; yc]);

m20 = mpq_point(pp, 2,0);
m02 = mpq_point(pp, 0,2);
m11 = mpq_point(pp, 1,1);

% compute the moments and ellipse matrix
J = [m20 m11; m11 m02];
E_est = 4 * J / m00
```

which results in an estimate

```
>> E_est
E_est =
    0.9776    0.9395
    0.9395    1.8976
```

which is similar to the original value of `E`. The point data is shown in Fig. E.3. We can overlay the estimated ellipse on the point data

```
>> plot_ellipse(E_est, [xc yc],  'r')
```

and the result is shown in red in Fig. E.3.

From a Set of Boundary Points

We wish to find the equation of an ellipse given a set of points (x_i, y_i) that define the boundary of an ellipse. Using the polynomial form of the ellipse Eq. E.4 for each point we write this in matrix form

$$\begin{pmatrix} y_1^2 & x_1 y_1 & x_1 & y_1 & 1 \\ y_2^2 & x_2 y_2 & x_2 & y_2 & 1 \\ & & \vdots & & \\ y_N^2 & x_N y_N & x_N & y_N & 1 \end{pmatrix} \begin{pmatrix} E_1 \\ E_2 \\ E_3 \\ E_4 \\ E_5 \end{pmatrix} = \begin{pmatrix} -x_1^2 \\ -x_2^2 \\ \vdots \\ -x_N^2 \end{pmatrix}$$

and for $N \geq 5$ we can solve for the ellipse parameter vector.

F Gaussian Random Variables

The 1-dimensional Gaussian function

$$g(x) = \frac{1}{\sqrt{\sigma^2 2\pi}} e^{-\frac{1}{2}(x-\mu)^2 \frac{1}{\sigma^2}} \tag{F.1}$$

is described by the position of its peak μ and its width σ. The total area under the curve is unity and $g(x) > 0, \forall x$.

The function can be plotted using the Toolbox function `gaussfunc`

```
>> x = linspace(-6, 6, 500);
>> plot(x, gaussfunc(0, 1, x) )
>> hold on
>> plot(x, gaussfunc(0, 2^2, x), '--' )
```

and Fig. F.1 shows two Gaussians with zero mean and $\sigma = 1$ and $\sigma = 2$. Note that the argument to `gaussfunc` is the variance not standard deviation.

If the Gaussian is considered to be a probability density function (PDF) then this is the well known normal distribution and the peak position μ is the mean value and the width σ is the standard deviation. A random variable drawn from a normal distribution is often written as $X \sim N(\mu, \sigma^2)$, and $N(0, 1)$ is referred to as the standard normal distribution. The probability that a random value falls within an interval $x \in [x_1, x_2]$ is obtained by integration

$$P = \int_{x_1}^{x_2} g(x)\mathrm{d}x = \Phi(x_2) - \Phi(x_1)$$

or evaluation of the cumulative distribution function $\Phi(x)$. The marked points in Fig. F.1 at $\mu \pm 1\sigma$ delimit the 1σ confidence interval. The area under the curve over this interval is 0.68, so the probability of a random value being drawn from this interval is 68%.

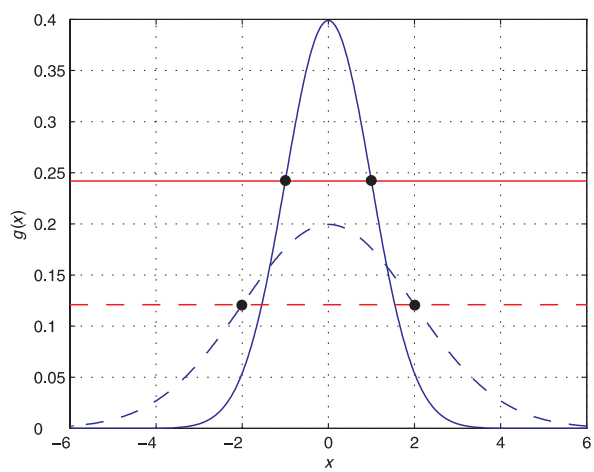


Fig. F.1. Two Gaussian functions, both with with mean $\mu = 0$, and with standard deviation $\sigma = 1$ (solid), and $\sigma = 2$ (dashed). The markers indicate the points $x = \mu \pm 1\sigma$. The dashed curve is wider but less tall, since the total area under the curve is unity

The n-dimensional Gaussian, or multivariate normal distribution, is

$$g(\boldsymbol{x}) = \frac{1}{\sqrt{\det(\boldsymbol{C})(2\pi)^{n/2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\mu)^T \boldsymbol{C}^{-1}(\boldsymbol{x}-\mu)} \tag{F.2}$$

and compared to the scalar case of Eq. F.1 x and μ have become n-vectors, the squared term in the exponent has been replaced by a matrix quadratic form, and σ^2, the variance, has become a covariance matrix $\boldsymbol{C}$. The diagonal elements represent the variance of x_i and the off-diagonal elements $\boldsymbol{C}_{ij}$ are the correlationss between x_i and x_j. If the variables are uncorrelated the matrix $\boldsymbol{V}$ would be diagonal. The covariance matrix is symmetric and positive definite.

We can plot a 2-dimensional Gaussian

```
>> [x,y] = meshgrid(-5:0.1:5, -5:0.1:5);
>> C = diag([1 2^2]);
>> g = gaussfunc([0 0], C, x, y) ;
>> axis([-5 5 -5 5 -.05 .12]);  hold on
>> surfc(x, y, g)
```

as a surface which is shown in Fig. F.2. In this case $\mu = (0, 0)$ and $\boldsymbol{C} = \text{diag}(1^2, 2^2)$ which corresponds to uncorrelated variables with standard deviation of 1 and 2 respectively. Figure F.2 also shows a number of contour lines for the surface which we see are elliptical, and the radii in the y- and x-directions are in the ratio 2:1 as defined by the standard deviations.

Looking at the exponent in Eq. F.2 we see the equation of an ellipse. All the points that satisfy

$$(\boldsymbol{x} - \mu)^T \boldsymbol{C}^{-1}(\boldsymbol{x} - \mu) = 1$$

result in a constant probability density value, that is, a contour line corresponding to the 1σ boundary. For higher order Gaussians, $n > 2$, the corresponding confidence interval is a surface of an ellipsoid in n-dimensional space.

Consider that this 2-dimensional probability density function represents the position of a robot in the xy-plane. The most likely position for the robot is at (0, 0) and we would have a 68% probability of being inside the ellipse corresponding to the 1σ boundary

```
>> plot_ellipse(C, [0 0])
```

The size of the ellipse says something about our spatial certainty. A large ellipse implies we have a 68% probability being anywhere within a large area, whereas a small ellipse means we have the same probability to be within a much smaller area. A useful measure of ellipse size is $\det(C)$ as discussed in Section Appendix E. We can also say that our uncertainty is higher in the y-direction than the x-direction.

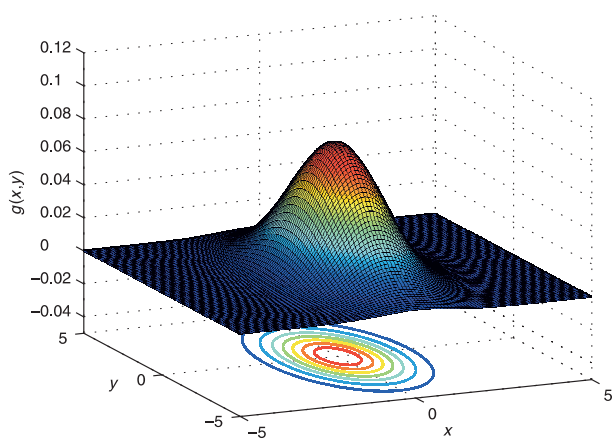


Fig. F.2.
The 2-dimensional Gaussian with covariance $\boldsymbol{C} = \text{diag}(1^2, 2^2)$. Contours lines of constant probability density are shown beneath

In estimation filters for localization, Chap. 6, it is common to represent the robot's uncertainty graphically as an ellipse. If the covariance matrix is diagonal then the ellipse is aligned with the x- and y-axes as we saw in Appendix E. This indicates that the two variables are independent and have zero correlation. Conversely a rotated ellipse indicates that the covariance is not diagonal and the two variables are correlated.

If $\boldsymbol{x} \in \mathbb{R}^n$ is drawn from a multivariate Gaussian its distance from the point μ is

$$d = \sum_{i=1}^{n} (\boldsymbol{x}_i - \boldsymbol{\mu}_i)^2$$

and this scalar has a chi-squared distribution with n degrees of freedom

$$d \sim \chi^2(n)$$

The Mahalanobis distance is a scalar measure

$$d_M = (\boldsymbol{x} - \mu)^T \boldsymbol{C}^{-1} (\boldsymbol{x} - \mu)$$

of the unlikeness of the point $\boldsymbol{x}$ with respect to the distribution μ and $\boldsymbol{C}$.

G Jacobians

A scalar-valued function of a vector $f: \mathbb{R}^n \to \mathbb{R}$ has a derivative with respect to the vector $\boldsymbol{x}$

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

and is itself a vector that points in the direction at which the function $f(\boldsymbol{x})$ has maximal increase. It is often written as $\nabla_x f$ to make explicit that the differentiation is with respect to $\boldsymbol{x}$.

A vector-valued function of a vector $\boldsymbol{f}: \mathbb{R}^m \to \mathbb{R}^n$ can be written as

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{pmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_n(\boldsymbol{x}) \end{pmatrix}$$

where $f_i: \mathbb{R}^m \to \mathbb{R}$ for $i \in \{1, 2, \cdots n\}$. The derivative of $\boldsymbol{f}$ with respect to the vector $\boldsymbol{x}$ can be expressed in matrix form as a Jacobian matrix

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

which can also be written as

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{pmatrix} \nabla f_1^T \\ \nabla f_2^T \\ \vdots \\ \nabla f_n^T \end{pmatrix}$$

This derivative is also known as the tangent map of $\boldsymbol{f}$, denoted $T\boldsymbol{f}$, or the differential of $\boldsymbol{f}$ denoted $D\boldsymbol{f}$. To make explicit that the differentiation is with respect to $\boldsymbol{x}$ this can be denoted as $\boldsymbol{J}_x$, $T_x\boldsymbol{f}$, $D_x\boldsymbol{f}$ or even $\partial \boldsymbol{f} / \partial \boldsymbol{x}$. Jacobians of functions are required for many optimization algorithms as well as for the extended Kalman filter, and can be evaluated numerically or symbolically.

Consider equation Eq. 6.9 for the range and bearing angle of a landmark given the pose of the vehicle and the position of the landmark. We can express this as the very simple MATLAB® function

```
1  function z = zrange(xi, xv, w)
2         z = [ sqrt((xi(2)-xv(2))/(xi(1)-xv(1)))+w(1);
3             atan((xi(2)-xv(2))/(xi(1)-xv(1)))-xv(3)+w(2) ];
```

To estimate the Jacobian $\boldsymbol{H}_{xv} = \partial \boldsymbol{h} / \partial \boldsymbol{x}_v$ for $\boldsymbol{x}_v = (1, 2, \frac{\pi}{3})$ and $\boldsymbol{x}_i = (10, 8)$ we can compute a first order numerical difference

```
>> xv = [1, 2, pi/3]; xi = [10, 8]; w= [0,0];
>> h0 = zrange(xi, xv, w)
```

```
h0 =
    0.9258
    0.5880
>> d = 0.001;
>> J = [ zrange(xi, xv+[1,0,0]*d, w)-h0 ...
         zrange(xi, xv+[0,1,0]*d, w)-h0, ...
         zrange(xi, xv+[0,0,1]*d,w)-h0]  /  d
J =
    0.0454   -0.0680         0
    0.0513   -0.0769   -1.0000
```

which shares the characteristic last column with the Jacobian shown in Eq. 6.4. Note that in computing this Jacobian we have set the measurement noise `w` to zero. The principle difficulty with this approach is choosing `d` which is the difference used to compute the finite-difference approximation to the derivative. Too large and the results will be quite inaccurate if the function is non-linear, too small and numerical problems will lead to reduced accuracy.

In general we can also perform the differentiation symbolically. This particular function is relatively simple and the derivatives can be determined easily using differential calculus. The numerical derivative can be used as a quick check for correctness. To avoid the possibility of error, or for more complex functions we can perform the differentiation symbolically using any of a large number of computer algebra packages. Using the MATLAB® Symbolic Math Toolbox we can declare some symbolic variables

```
>> syms xi yi xv yv thetav wr wb
```

and then evaluate the same function as above

```
>> z = zrange([xi yi], [xv yv thetav], [wr wb])
z =
        wr + ((yi - yv)/(xi - xv))^(1/2)
 wb - thetav + atan((yi - yv)/(xi - xv))
```

which is simply Eq. 6.9 in MATLAB® symbolic form. The Jacobian is computed by a Symbolic Math Toolbox function

```
>> J = jacobian(z, [xv yv thetav])
J =
[ (yi - yv)/(2*((yi - yv)/(xi - xv))^(1/2)*(xi - xv)^2),
    -1/(2*((yi - yv)/(xi - xv))^(1/2)*(xi - xv)),  0]
[ (yi - yv)/((xi - xv)^2*((yi - yv)^2/(xi - xv)^2 + 1)),
    -1/((xi - xv)*((yi - yv)^2/(xi - xv)^2 + 1)), -1]
```

which has the required dimensions

```
>> about(J)
J [sym] : 2x3 (60 bytes)
```

and the characteristic last column. We could cut and paste this code into our program or automatically create a MATLAB® callable function

```
>> Jf = matlabFunction(J);
```

where `Jf` is a MATLAB® function handle. We can evaluate the Jacobian at the operating point given above

```
>> xv = [1, 2, pi/3]; xi = [10, 8]; w= [0,0];
>> Jf( xi(1), xv(1), xi(2), xv(2) )
ans =
    0.0454   -0.0680         0
    0.0513   -0.0769   -1.0000
```

which is similar to the approximation obtained numerically. The function `matlabFunction` can also write the function to an M-file. The functions `ccode` and `fcode` generate C and Fortran representations of the Jacobian.

Another interesting approach is the package ADOL-C which is an open-source tool for the automatic differentiation of C and C++ programs, that is, given a function written in C it will return a Jacobian function written in C. It is available at http://www.coin-or.org/projects/ADOL-C.xml

H Kalman Filter

Consider the discrete-time linear time-invariant system

$$\begin{aligned} \boldsymbol{x}\langle k+1\rangle &= \boldsymbol{F}\boldsymbol{x}\langle k\rangle + \boldsymbol{G}\boldsymbol{u}\langle k\rangle + \boldsymbol{v}\langle k\rangle \\ \boldsymbol{z}\langle k+1\rangle &= \boldsymbol{H}\boldsymbol{x}\langle k\rangle + \boldsymbol{w}\langle k\rangle \end{aligned}$$

with state vector $\boldsymbol{x} \in \mathbb{R}^n$. The vector $\boldsymbol{u} \in \mathbb{R}^m$ is the input to the system at time k, for example a velocity command, or applied forces and torques. The vector $\boldsymbol{z} \in \mathbb{R}^p$ represents the outputs of the system as measured by sensors. The matrix $\boldsymbol{F} \in \mathbb{R}^{n \times n}$ describes the dynamics of the system, that is, how the states evolve with time. The matrix $\boldsymbol{G} \in \mathbb{R}^{n \times m}$ describes how the inputs are coupled to the system states. The matrix $\boldsymbol{H} \in \mathbb{R}^{p \times n}$ describes how the system states are mapped to the observed outputs.

To account for errors in the model (represented by $\boldsymbol{F}$ and $\boldsymbol{G}$) and also unmodeled disturbances we introduce a Gaussian random variable $\boldsymbol{v} \in \mathbb{R}^n$ termed the process noise. $\boldsymbol{v}\langle k\rangle \sim N(0, \boldsymbol{V})$, that is, it has zero-mean and covariance $\boldsymbol{V}$. The sensor measurement model $\boldsymbol{H}$ is not perfect either and this is modelled by measurement noise, another Gaussian random variable $\boldsymbol{w} \in \mathbb{R}^p$ and $\boldsymbol{w}\langle k\rangle \sim N(0, \boldsymbol{W})$. The covariance matrices $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{W} \in \mathbb{R}^{p \times p}$ are symmetric and positive definite.

The general problem that we confront is:

> *given a model of the system, the known inputs $\boldsymbol{u}$ and some noisy sensor measurements $\boldsymbol{z}$, estimate the state of the system $\boldsymbol{x}$.*

In a robotic localization context $\boldsymbol{x}$ is the unknown pose of the robot, $\boldsymbol{u}$ is the commands sent to the motors and $\boldsymbol{z}$ is the output of various sensors on the robot. For a flying robot $\boldsymbol{x}$ could be the attitude, $\boldsymbol{u}$ the known forces applied to the airframe and $\boldsymbol{z}$ are the measured accelerations and angular velocities.

The Kalman filter is an optimal estimator for the case where the process and measurement noise are zero-mean Gaussian noise. The filter has two steps. The first is a prediction of the state based on the previous state and the inputs that were applied.

$$\hat{\boldsymbol{x}}\langle k+1|k\rangle = \boldsymbol{F}\hat{\boldsymbol{x}}\langle k\rangle + \boldsymbol{G}\boldsymbol{u}\langle k\rangle \tag{H.1}$$

$$\hat{\boldsymbol{P}}\langle k+1|k\rangle = \boldsymbol{F}\hat{\boldsymbol{P}}\langle k|k\rangle\boldsymbol{F}^T + \hat{\boldsymbol{V}} \tag{H.2}$$

where $\hat{\boldsymbol{x}}$ is the estimate of the state and $\hat{\boldsymbol{P}} \in \mathbb{R}^{n \times n}$ is the estimated covariance, or uncertainty, in $\hat{\boldsymbol{x}}$. This is an *open-loop* step and its accuracy depends completely on the quality of the model $\boldsymbol{F}$ and $\boldsymbol{G}$ and the ability to measure the inputs $\boldsymbol{u}$. The notation $k+1|k$ makes explicit that the left-hand side is an estimate at time $k+1$ based on information from time k.

The prediction of $\boldsymbol{P}$ involves the addition of two positive-definite matrices so the uncertainty, given no new information and the uncertainty in the process, has increased. To improve things we have to introduce new information and that comes from mea-

surements obtained using sensors. The new information that is added is known as the innovation

$$\boldsymbol{\nu}\langle k+1\rangle = \boldsymbol{z}\langle k+1\rangle - \boldsymbol{H}\hat{\boldsymbol{x}}\langle k+1|k\rangle$$

which is the difference between what the sensors measure and what the sensors are predicted to measure. Some of the difference will be due to the noise in the sensor, the measurement noise, but the remaining discrepancy indicates that the predicted state was in error and does not properly explain the sensor observations.

The second step of the Kalman filter, the *update* step, uses the Kalman gain

$$\boldsymbol{K}\langle k+1\rangle = \hat{\boldsymbol{P}}\langle k+1|k\rangle \boldsymbol{H}^T \underbrace{\left(\boldsymbol{H}\hat{\boldsymbol{P}}\langle k+1|k\rangle \boldsymbol{H}^T + \hat{\boldsymbol{W}}\right)}_{\boldsymbol{S}}{}^{-1} \tag{H.3}$$

to map the innovation into a correction for the predicted state, optimally tweaking the estimate based on what the sensors observed

$$\begin{aligned}\hat{\boldsymbol{x}}\langle k+1|k+1\rangle &= \hat{\boldsymbol{x}}\langle k+1|k\rangle + \boldsymbol{K}\langle k+1\rangle \boldsymbol{\nu}\langle k+1\rangle \\ \hat{\boldsymbol{P}}\langle k+1|k+1\rangle &= \hat{\boldsymbol{P}}\langle k+1|k\rangle - \boldsymbol{K}\langle k+1\rangle \boldsymbol{H}\hat{\boldsymbol{P}}\langle k+1|k\rangle\end{aligned}$$

Importantly we note that the uncertainty is now decreased or *deflated*, since the second term is subtracted from the predicted covariance. The term indicated by $\boldsymbol{S}$ is the estimated covariance of the innovation and comes from the uncertainty in the state and the measurement noise covariance. If the innovation has high uncertainty in relation to some states this will be reflected in the Kalman gain which will make correspondingly small adjustment to those states.

The covariance update can also be written in the Joseph form

$$\hat{\boldsymbol{P}}\langle k+1|k+1\rangle = \left(\boldsymbol{I}_{n\times n} - \boldsymbol{K}\langle k+1\rangle \boldsymbol{H}\right)\hat{\boldsymbol{P}}\langle k+1|k\rangle\left(\boldsymbol{I}_{n\times n} - \boldsymbol{K}\langle k+1\rangle \boldsymbol{H}\right)^T + \boldsymbol{K}\langle k+1\rangle \hat{\boldsymbol{V}}\boldsymbol{K}\langle k+1\rangle^T$$

which has improved numerical properties and keeps the covariance estimate symmetric, but it is computationally more costly.

The equations above constitute the classical Kalman filter which is widely used in applications from aerospace to econometrics. The filter has a number of important characteristics. Firstly it is recursive, the output of one iteration is the input to the next. Secondly, it is asynchronous. At a particular iteration if no sensor information is available we perform just the prediction step and not the update. In the case that there are different sensors, each with their own $\boldsymbol{H}$, and different sample rates, we just apply the update with the appropriate $\boldsymbol{z}$ and $\boldsymbol{H}$. The Kalman-Bucy filter is a continuous-time version of this filter.

The filter must be initialized with some reasonable value of $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{P}}$. The filter also requires our best estimates of the covariance of the process and measurement noise. In general we do not know $\boldsymbol{V}$ and $\boldsymbol{W}$ but we have some estimate $\hat{\boldsymbol{V}}$ and $\hat{\boldsymbol{W}}$ that we use in the filter. From Eq. H.2 we see that if we overestimate $\hat{\boldsymbol{V}}$ our estimate of $\boldsymbol{P}$ will be larger than it really is giving a pessimistic estimate of our certainty in the state. Conversely if we overestimate $\hat{\boldsymbol{V}}$ the filter will be *overconfident* of its estimate.

The covariance matrix $\hat{\boldsymbol{P}}$ is rich in information. The diagonal elements $\hat{\boldsymbol{P}}_{ii}$ are the variance, or uncertainty, in the state $\boldsymbol{x}_i$. The off-diagonal elements $\hat{\boldsymbol{P}}_{ij}$ are the correlations between states $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. The correlations are critical in allowing any piece of new information to *flow through* to adjust multiple states that affect a particular process output.

The term $\boldsymbol{F}\boldsymbol{P}\langle k|k\rangle \boldsymbol{F}^T\langle k\rangle$ in Eq. H.2 is interesting. Consider a one dimensional example where F is a scalar and the state estimate $\hat{x}\langle k\rangle$ has a PDF that is a Gaussian with a mean $\overline{x}\langle k\rangle$ and a variance $\sigma^2\langle k\rangle$. The prediction equation maps the state and its Gaussian dis-

tribution to a new Gaussian distribution with a mean $F\bar{x}\langle k\rangle$ and a variance $F^2\sigma^2\langle k\rangle$. The term $\boldsymbol{FP}\langle k|k\rangle\boldsymbol{F}^T\langle k\rangle$ is the matrix form of this since

$$\text{cov}(\boldsymbol{F}\boldsymbol{x}) = \boldsymbol{F}\,\text{cov}(\boldsymbol{x})\boldsymbol{F}^T$$

and appropriately scales the covariance. The term $\boldsymbol{HP}\langle k+1|k\rangle\boldsymbol{H}^T$ in Eq. H.3 *projects* the covariance of the state estimate into the observed values.

Now consider the case where the system is not linear

$$\begin{aligned}\boldsymbol{x}\langle k+1\rangle &= \boldsymbol{f}\big(\boldsymbol{x}\langle k\rangle, \boldsymbol{u}\langle k\rangle\big) + \boldsymbol{v}\langle k\rangle\\ \boldsymbol{z}\langle k+1\rangle &= \boldsymbol{h}\big(\boldsymbol{x}\langle k\rangle\big) + \boldsymbol{w}\langle k\rangle\end{aligned}$$

where $\boldsymbol{f}$ and $\boldsymbol{h}$ are now functions instead of constant matrices. $\boldsymbol{f}:\mathbb{R}^n, \mathbb{R}^m \to \mathbb{R}^n$ is a function that describes the new state in terms of the previous state and the input to the system. The function $\boldsymbol{h}:\mathbb{R}^n \to \mathbb{R}^p$ maps the state vector to the sensor measurements.

To use the linear Kalman filter with a non-linear system we first make a local linear approximation

$$\begin{aligned}\boldsymbol{x}\langle k+1\rangle &= \boldsymbol{f}\big(\hat{\boldsymbol{x}}\langle k\rangle, \boldsymbol{u}\langle k\rangle\big) + \boldsymbol{F}_x\big(\boldsymbol{x}\langle k\rangle - \hat{\boldsymbol{x}}\langle k|k\rangle\big) + \boldsymbol{F}_u\boldsymbol{u}\langle k\rangle + \boldsymbol{F}_v\boldsymbol{v}\langle k\rangle\\ \boldsymbol{z}\langle k+1\rangle &= \boldsymbol{h}\big(\hat{\boldsymbol{x}}\langle k\rangle\big) + \boldsymbol{H}_x\big(\hat{\boldsymbol{x}}\langle k+1|k\rangle - \boldsymbol{x}\langle k\rangle\big) + \boldsymbol{H}_w\boldsymbol{w}\langle k\rangle\end{aligned}$$

where $\boldsymbol{F}_x \in \mathbb{R}^{n\times n}$, $\boldsymbol{F}_u \in \mathbb{R}^{n\times m}$, $\boldsymbol{F}_v \in \mathbb{R}^{n\times n}$, $\boldsymbol{H}_x \in \mathbb{R}^{p\times n}$ and $\boldsymbol{H}_w \in \mathbb{R}^{p\times p}$ are Jacobians of the functions $\boldsymbol{f}(\cdot)$ and $\boldsymbol{h}(\cdot)$ and are evaluated at each time step.

We define a prediction error

$$\begin{aligned}\tilde{\boldsymbol{x}}\langle k+1|k\rangle &= \boldsymbol{x}\langle k\rangle - \hat{\boldsymbol{x}}\langle k+1|k\rangle\\ &= \boldsymbol{F}_x\tilde{\boldsymbol{x}}\langle k|k\rangle + \boldsymbol{F}_u\boldsymbol{u}\langle k\rangle + \boldsymbol{F}_v\boldsymbol{v}\langle k\rangle\end{aligned}$$

and a measurement residual

$$\begin{aligned}\tilde{\boldsymbol{z}}\langle k+1|k\rangle &= \boldsymbol{z}\langle k+1\rangle - \boldsymbol{h}\langle k+1|k\rangle\\ &= \boldsymbol{H}_x\tilde{\boldsymbol{x}} + \boldsymbol{H}_w\boldsymbol{w}\langle k\rangle\end{aligned}$$

which are linear and the Kalman filter equations above can be applied. The prediction step of the extended Kalman filter is

$$\begin{aligned}\hat{\boldsymbol{x}}\langle k+1|k\rangle &= \boldsymbol{f}\big(\hat{\boldsymbol{x}}\langle k\rangle, \boldsymbol{u}\langle k\rangle\big)\\ \hat{\boldsymbol{P}}\langle k+1|k\rangle &= \boldsymbol{F}_x\hat{\boldsymbol{P}}\langle k|k\rangle\boldsymbol{F}_x^T + \boldsymbol{F}_v\hat{\boldsymbol{V}}\langle k\rangle\boldsymbol{F}_v^T\end{aligned}$$

and the update step is

$$\begin{aligned}\hat{\boldsymbol{x}}\langle k+1|k+1\rangle &= \hat{\boldsymbol{x}}\langle k+1|k\rangle + \boldsymbol{K}\langle k+1\rangle\boldsymbol{\nu}\langle k+1\rangle\\ \hat{\boldsymbol{P}}\langle k+1|k+1\rangle &= \hat{\boldsymbol{P}}\langle k+1|k\rangle - \boldsymbol{K}\langle k+1\rangle\boldsymbol{H}_x\hat{\boldsymbol{P}}\langle k+1|k\rangle\end{aligned}$$

where the innovation is

$$\boldsymbol{\nu}\langle k+1\rangle = \boldsymbol{z}\langle k+1\rangle - \boldsymbol{h}\big(\hat{\boldsymbol{x}}\langle k+1|k\rangle\big)$$

and the Kalman gain is

$$\boldsymbol{K}\langle k+1\rangle = \hat{\boldsymbol{P}}\langle k+1|k\rangle\boldsymbol{H}_x^T\Big(\boldsymbol{H}_x\hat{\boldsymbol{P}}\langle k+1|k\rangle\boldsymbol{H}_x^T + \boldsymbol{H}_w\hat{\boldsymbol{W}}\boldsymbol{H}_w^T\Big)^{-1}$$

A fundamental problem with the extended Kalman filter is that PDFs of the random variables are no longer Gaussian after being operated on by the non-linear

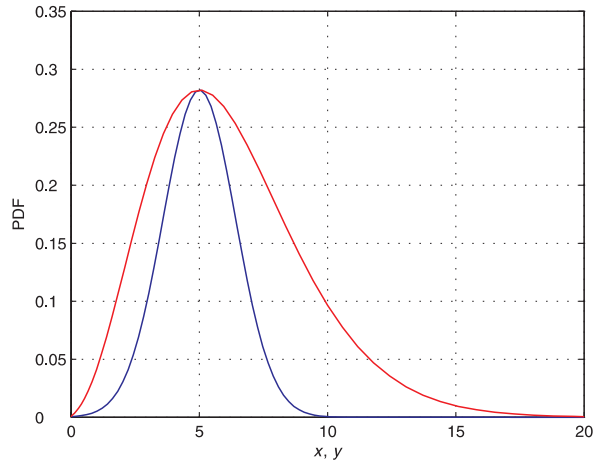


Fig. H.1.
PDF of the state x (blue) which is Gaussian $N(5, 2)$ and the PDF of the non-linear function $x^2/5$ (red)

functions $\boldsymbol{f}(\cdot)$ and $\boldsymbol{h}(\cdot)$. We can easily illustrate this by considering a scalar system with the PDF of the state estimate being the Gaussian $N(5, 2)$

```
>> x = linspace(0, 20, 100);
>> g = gaussfunc(5, 2, x);
>> plot(x, g);
```

Now consider the nonlinear function $y = x^2/5$ and we overlay the PDF of y

```
>> y = x.^2 / 5;
>> plot(y, g, 'r');
```

which is shown in Fig. H.1. We see that the PDF of y has its peak, the mode, at the same location but the distribution is no longer Gaussian. It has lost its symmetry so the mean value will actually be greater than the mode. The Jacobians that appear in the EKF equations appropriately scale the covariance but the resulting non-Gaussian distributions break the assumptions which guarantee that the Kalman filter is an optimal estimator. 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.

Homogeneous Coordinates

A point in n-dimensional Euclidean space $\boldsymbol{x} \in \mathbb{R}^n$ is represented by a coordinate vector $(x_1, x_2 \cdots x_n)$. The corresponding point in homogeneous coordinates, or the projective space $\tilde{\boldsymbol{x}} \in \mathbb{P}^n$ is represented by a coordinate vector $(\tilde{x}_1, \tilde{x}_2 \cdots \tilde{x}_{n+1})$. The Euclidean coordinates are related to the projective coordinates by

$$x_i = \frac{\tilde{x}_i}{\tilde{x}_{n+1}}, \quad i = 1 \cdots n$$

Conversely a homogeneous coordinate vector can be constructed from a Euclidean coordinate vector by

$$\tilde{\boldsymbol{x}} = (x_1, x_2 \cdots x_n, 1)$$

and the tilde is used to indicate that the quantity is homogeneous.

The extra *degree of freedom* offered by projective coordinates has several advantages. It allows points and lines at infinity, known as ideal points and lines, to be represented using only real numbers. It also means that scale is unimportant, that is $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{x}}' = \alpha\tilde{\boldsymbol{x}}$ both represent the same Euclidean point for all $\alpha \neq 0$. We express this as $\tilde{\boldsymbol{x}} \simeq \tilde{\boldsymbol{x}}'$. Points in homogeneous form can also be rotated with respect to a coordinate frame and translated simply by multiplying the homogeneous coordinate by an $(n+1) \times (n+1)$ homogeneous transformation matrix.

Homogeneous vectors are important in computer vision when we consider points and lines that exist in a plane – a camera's image plane. We can also consider that the homogeneous form represents a ray in Euclidean space, and the relationship between points and rays is at the core of the projective transformation.

In $\mathbb{P}^2$ a line is defined by a 3-tuple, $\tilde{\boldsymbol{\ell}} = (\ell_1, \ell_2, \ell_3)^T$, not all zero, and the equation of the line is the set of all points

$$\tilde{\boldsymbol{\ell}}^T \tilde{\boldsymbol{x}} = 0$$

which expands to $\ell_1 x + \ell_2 y + \ell_3 = 0$ and can be manipulated into the more familiar representation of a line. Note that this form can represent a vertical line, parallel to the y-axis, which the familiar form $y = mx + c$ cannot. This is the point equation of a line. The non-homogeneous vector (ℓ_1, ℓ_2) is a normal to the line, and $(-\ell_2, \ell_1)$ is parallel to the line.

A duality exists between points and lines. A point is defined by the intersection of two lines. If we write the point equations for two lines $\tilde{\boldsymbol{\ell}}_1^T \tilde{\boldsymbol{p}} = 0$ and $\tilde{\boldsymbol{\ell}}_2^T \tilde{\boldsymbol{p}} = 0$ their intersection is the point

$$\tilde{\boldsymbol{p}} = \tilde{\ell}_1 \times \tilde{\ell}_2$$

and is known as the line equation of a point. Similarly, a line joining two points $\tilde{\boldsymbol{p}}_1$ and $\tilde{\boldsymbol{p}}_2$ is given by the cross-product

$$\tilde{\ell}_{12} = \tilde{\boldsymbol{p}}_1 \times \tilde{\boldsymbol{p}}_2$$

Consider the case of two parallel lines at 45° to the horizontal axis

```
>> l1 = [1 -1 0]';
>> l2 = [1 -1 -1]';
```

which we can plot

```
>> plot_homline(l1, 'b')
>> plot_homline(l2, 'r')
```

The intersection point of these parallel lines is

```
>> cross(l1, l2)
ans =
     1     1     0
```

This is an *ideal point* since the third coordinate is zero – the equivalent Euclidean point would be at infinity. Projective coordinates allow points and lines at infinity to be simply represented and manipulated without special logic to handle the special case of infinity.

The distance from a point $\tilde{\boldsymbol{p}}$ to a line $\tilde{\boldsymbol{\ell}}$ is

$$d = \frac{\tilde{\boldsymbol{\ell}}^T \tilde{\boldsymbol{p}}}{p_3 \sqrt{\ell_1^2 + \ell_2^2}} \tag{I.1}$$

In the projective space $\mathbb{P}^3$ a duality exists between points and planes: three points define a plane, and the intersection of three planes defines a point.

J Graphs

A graph is an abstract representation of a set of objects connected by links and depicted graphically as shown in Fig. J.1. Mathematically a graph is denoted $G(V, E)$ where V, are called vertices or nodes, and the links, E, that connect some pairs of vertices are called edges or arcs. Edges can be directed (arrows) or undirected as in this case. Edges can have an associated weight or cost associated with moving from one vertex to another. A sequence of edges from one vertex to another is a path, and a sequence that starts and ends at the same vertex is a cycle. An edge from a vertex to itself is a loop. Graphs can be used to represent transport, communications or social networks, and this branch of mathematics is graph theory.

The Toolbox provides a MATLAB® graph class called `PGraph` that supports embedded graphs where the vertices are associated with a point in an n-dimensional space. To create a new graph

```
>> g = PGraph()
g =
  2 dimensions
  0 vertices
  0 edges
  0 components
```

and by default the nodes of the graph exist in a 2-dimensional space. We can add nodes to the graph

```
>> g.add_node( rand(2,1) );
>> g.add_node( rand(2,1) );
>> g.add_node( rand(2,1) );
>> g.add_node( rand(2,1) );
>> g.add_node( rand(2,1) );
```

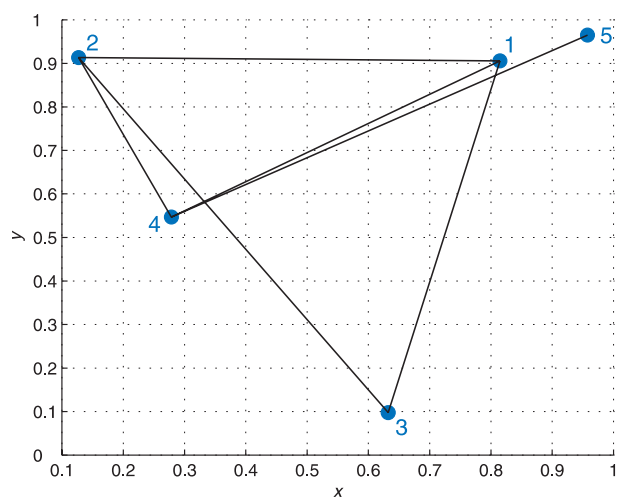


Fig. J.1.
An example graph generated by the `PGraph` class

and each has a random coordinate. A summary of the graph is given with its display method

```
>> g
g =
  2 dimensions
  5 vertices
  0 edges
  4 components
```

and shows that the graph has 5 nodes but no edges. The nodes are numbered 1 to 5 and we add edges between pairs of nodes

```
>> g.add_edge(1, 2);
>> g.add_edge(1, 3);
>> g.add_edge(1, 4);
>> g.add_edge(2, 3);
>> g.add_edge(2, 4);
>> g.add_edge(4, 5);
>> g
g =
  2 dimensions
  5 vertices
  5 edges
  1 components
```

By default the distance between the nodes is the Euclidean distance between the vertices but this can be overridden by a third argument to `add_edge`. This class supports only undirected graphs so the order of the vertices provided to `add_edge` does not matter. The graph has one component, that is all the nodes are connected into one network. The graph can be plotted by

```
>> g.plot('labels')
```

as shown in Fig. J.1. The vertices are shown as blue circles, and the option `'labels'` displays the vertex index next to the circle. Edges are shown as black lines joining vertices. Note that only graphs embedded in 2- and 3-dimensional space can be plotted.

The neighbours of vertex 2 are

```
>> g.neighbours(2)
ans =
     3     4     1
```

which are vertices connected to vertex 2 by edges. Each edge has a unique index and the edges joining vertex 2 are

```
>> e = g.edges(2)
e =
     4     5     1
```

The cost or length of these edges is

```
>> g.cost(e)
ans =
    0.9597    0.3966    0.6878
```

and clearly edge 5 has a lower cost than edges 4 and 1. Edge 5

```
>> g.vertices(5)
ans =
     2
     4
```

joins vertices 2 and 4, and vertex 4 is clearly the closest neighbour of vertex 2. Frequently we wish to obtain a node's neighbouring vertices and their distances at the same time, and this can be achieved conveniently by

```
>> [n,c] = g.neighbours(2)
n =
     3     4     1
c =
    0.9597    0.3966    0.6878
```

To plan a path through the graph we specify the goal vertex

```
>> g.goal(5)
```

which assigns every node in the graph its distance from the goal in a breadth-first fashion. To find a path to the goal from a specified starting vertex is

```
>> g.path(3)
ans =
     3     2     4     5
```

In this case the shortest path from vertex 3 to vertex 5 is via vertices 2 and 4. The vertex closest to the coordinate (0.5, 0.5) is

```
>> g.closest([0.5, 0.5])
ans =
     4
```

Minimum cost path between any two nodes in the graph can be computed using well known algorithms such as A^* (Nilsson 1971) or the earlier method by Dijstrka (1959).

K Peak Finding

A commonly encountered problem is estimating the position of the peak of some discrete signal $y(k)$, $k \in \mathbb{Z}$, see for example Fig. K.1a

```
>> load peakfit1
>> plot(y,  '-o')
```

Finding the peak to the nearest integer is straightforward using MATLAB's `max` function

```
>> [ypk,xpk] = max(y)
ypk =
    0.9905
xpk =
     8
```

which indicates the peak occurs at the eighth element and has a value of 0.9905. In this case there is more than one peak and we can use the Toolbox function `peak` instead

```
>> [ypk,xpk] = peak(y)
ypk =
    0.9905    0.6718   -0.5799
xpk =
     8    25    16
```

which has returned three maxima in descending magnitude. A common test of the quality of a peak is its magnitude and the ratio of the height of the second peak to the first peak

```
>> ypk(2)/ypk(1)
```

which is called the ambiguity ratio and is ideally small.

This signal is a sampled representation of a continuous underlying signal $y(x)$ and the real peak might lie between the samples. If we look at a zoomed version of the signal, Fig. K.1b, we can see that although the eighth point is the maximum the ninth

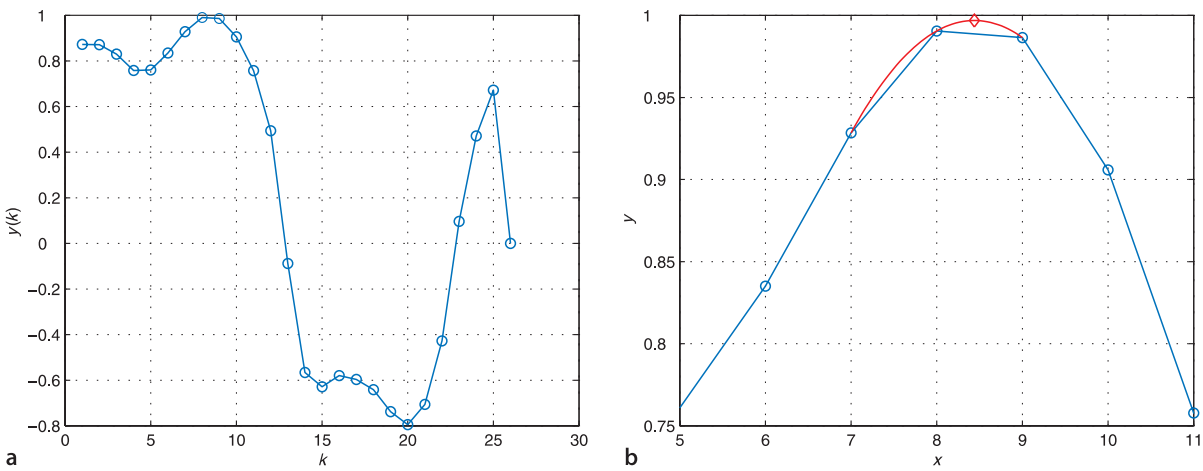


Fig. K.1. Peak fitting. **a** A signal with several local maxima; **b** close-up view of the first maxima with the fitted curve (red) and the estimated peak (red-◊)

point is only slightly lower so the peak lies somewhere between points eight and nine. A common approach is to fit a parabola

$$y = a\delta^2 + b\delta + c, \quad \delta \in \mathbb{R} \tag{K.1}$$

to the points surrounding the peak. For the discrete peak that occurs at $(x_{\text{pk}}, y_{\text{pk}})$ then $\delta = 0$ corresponds to x_{pk} and the discrete x-coordinates on either side correspond to $\delta = -1$ and $\delta = +1$ respectively. Substituting the points $(-1, y(-1))$, $(0, y(0))$ and $(1, y(1))$ into Eq. K.1 we can write three equations

$$\begin{aligned} y(-1) &= a - b + c \\ y(0) &= c \\ y(1) &= a + b + c \end{aligned}$$

or in compact matrix form as

$$\begin{pmatrix} y(-1) \\ y(0) \\ y(1) \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

and then solve for the parabolic coefficients

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}^{-1} \begin{pmatrix} y(-1) \\ y(0) \\ y(1) \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & -2 & 1 \\ -1 & 0 & 1 \\ 0 & 2 & 0 \end{pmatrix} \begin{pmatrix} y(-1) \\ y(0) \\ y(1) \end{pmatrix} \tag{K.2}$$

The maxima of the parabola occurs when its derivative is zero

$$2a\delta + b = 0$$

and substituting the values of a and b from Eq. K.2 we find the displacement of the peak of the fitted parabola with respect to the discrete maxima

$$\delta = \frac{1}{2} \frac{y(-1) - y(1)}{y(-1) - 2y(0) + y(1)}$$

so the refined, or interpolated, position of the maxima is at

$$\hat{x}_{\text{pk}} = x_{\text{pk}} + \delta$$

The coefficient a, which is negative for a maxima, indicates the sharpness of the peak which can be useful in determining whether a peak is *sufficiently* sharp. A large magnitude of a indicates a well defined sharp peak wheras a low value indicates a very broad peak for which estimation of a refined peak detection may not be so accurate.

Continuing the earlier example we can use the Toolbox function `peak` to estimate the refined peak positions

```
>> [ymax,xmax] = peak(y, 'interp', 2)
ymax =
    0.9905    0.6718   -0.5799
xmax =
    8.4394   24.7299   16.2438
```

where the argument after the `'interp'` option indicates that a second order polynomial should be fitted. The fitted parabola is shown in red in Fig. K.1b and is plotted if the option `'plot'` is given.

If the signal has superimposed noise then there are likely to be multiple peaks, many of which are quite minor, and this can be overcome by specifying the *scale* of the peak. For example the peaks that are greater than all other values within ± 5 values in the horizontal direction are

```
>> peak(y, 'scale', 5)
ans =
    0.9905    0.8730    0.6718
```

In this case the result is unchanged since the signal is fairly smooth.

For a 2D signal we follow a similar procedure but instead fit a paraboloid

$$z = ax^2 + by^2 + cx + dy + e \tag{K.3}$$

which has five coefficients that can be calculated from the centre value (the discrete maximum) and its four neighbours (north, south, east and west) using a similar procedure to above. The displacement of the estimated peak with respect to the central point is

$$\delta_x = \frac{1}{2}\frac{z_e - z_w}{2z_c - z_w - z_e}$$
$$\delta_y = \frac{1}{2}\frac{z_s - z_n}{2z_c - z_n - z_s}$$

In this case the coefficients a and b represent the sharpness of the peak in the x- and y-directions, and the quality of the peak can be considered as being $\min a, b$.

A 2D discrete signal was loaded from `peakfit1` earlier

```
>> z
z =
    0.0800    0.2000    0.3202    0.4400    0.5600
    0.0400    0.1717    0.3662    0.4117    0.5200
    0.0002    0.2062    0.8766    0.4462    0.4802
   -0.0400    0.0917    0.2862    0.3317    0.4400
   -0.0800    0.0400    0.1602    0.2800    0.4000
```

In this small example it is clear that the peak is at element (3, 3) but programatically this is

```
>> [zmax,i] = max(z(:))
zmax =
    0.8766
i =
    13
```

Counting the elements, starting with 1 at the top-left down each column then back to the top of the next rightmost column.

and the maximum is at the thirteenth element in row-major order◂ which we convert to array subscripts

```
>> [ymax,xmax] = ind2sub(size(z), i)
xmax =
     3
ymax =
     3
```

We can find this more conveniently using the Toolbox function `peak2`

```
>> [zm,xy]=peak2(z)
zm =
    0.8766
xy =
     3
     3
```

This function will return all non-local maxima where the size of the local region is given by the `'scale'` option. As for the 1-dimensional case we can refine the estimate of the peak

```
>> [zm,xy]=peak2(z, 'interp')
zm =
    0.8839
xy =
    2.9637
    3.1090
```

that is, the peak is at element (2.9637, 3.1090). When this process is applied to image data it is referred to as subpixel interpolation.

Bibliography

Agrawal M, Konolige K, Blas M (2008) CenSurE: Center surround extremas for realtime feature detection and matching. In: Forsyth D, Torr P, Zisserman A (eds) Lecture notes in computer science. Computer Vision – ECCV 2008, vol 5305. Springer-Verlag, Berlin Heidelberg, pp 102–115

Altmann SL (1989) Hamilton, Rodrigues, and the Quaternion scandal. Math Mag 62(5):291–308

Andersen N, Ravn O, Sørensen A (1993) Real-time vision based control of servomechanical systems. In: Chatila R, Hirzinger G (eds) Lecture Notes in Control and Information Sciences. Experimental Robotics II, vol 190. Springer-Verlag, Berlin Heidelberg, pp 388–402

Andersson RL (1989) Dynamic sensing in a ping-pong playing robot. IEEE T Robotic Autom 5(6): 728–739

Antonelli G (2006) Underwater robots: Motion and force control of vehicle-manipulator systems, 2nd ed. Springer Tracts in Advanced Robotics, vol 2. Springer-Verlag, Berlin Heidelberg

Arkin RC (1999) Behavior-based robotics. The MIT Press

Armstrong WW (1979) Recursive solution to the equations of motion of an N-link manipulator. In: Proc. 5th World Congress on Theory of Machines and Mechanisms, Montreal, Jul, pp 1343–1346

Armstrong BS (1988) Dynamics for robot control: Friction modelling and ensuring excitation during parameter identification. Stanford University

Armstrong B (1989) On finding exciting trajectories for identification experiments involving systems with nonlinear dynamics. Int J Robot Res 8(6):28

Armstrong-Hélouvry B, Dupont P, De Wit CC (1994) A survey of models, analysis tools and compensation methods for the control of machines with friction. Automatica 30(7):1083–1138

Arun KS, Huang TS, Blostein SD (1987) Least-squares fitting of 2 3-D point sets. IEEE T Pattern Anal 9(5):699–700

Asada H (1983) A geometrical representation of manipulator dynamics and its application to arm design. J Dyn Syst-T ASME 105:131

Azarbayejani A, Pentland AP (1995) Recursive estimation of motion, structure, and focal length. IEEE T Pattern Anal 17(6):562–575

Baldridge AM, Hook SJ, Grove CI, Rivera G (2009) The ASTER spectral library version 2.0. Remote Sens Environ 113(4):711–715

Ballard DH (1981) Generalizing the Hough transform to detect arbitrary shapes. Pattern Recogn 13(2):111–122

Banks J, Corke PI (2001) Quantitative evaluation of matching methods and validity measures for stereo vision. Int J Robot Res 20(7):512–532

Bar-Shalom Y, Fortmann T (1988) Tracking and data association. Mathematics in Science and Engineering, vol 182. Academic Press

Bar-Shalom Y, Rong Li X, Thiagalingam Kirubarajan (2001) Estimation with applications to tracking and navigation. Wiley-Interscience

Bauer J, Sünderhauf N, Protzel P (2007) Comparing several implementations of two recently published feature detectors. In: IFAC Symposium on Intelligent Autonomous Vehicles (IAV), Toulouse

Bay H, Ess A, Tuytelaars T, Van Gool L (2008) Speeded-up robust features (SURF). Comput Vis Image Und 110(3):346–359

Benosman R, Kang SB (2001) Panoramic vision: Sensors, theory, and applications. Springer-Verlag

Benson KB (ed) (1986) Television engineering handbook. McGraw-Hill

Besl PJ, McKay HD (1992) A method for registration of 3-D shapes. IEEE T Pattern Anal 14(2):239–256

Bhat DN, Nayar SK (2002) Ordinal measures for image correspondence. IEEE T Pattern Anal 20(4): 415–423

Bishop CM (2006) Pattern recognition and machine learning. Information Science and Statistics. Springer-Verlag, New York

Bolles RC, Baker HH, Marimont DH (1987) Epipolar-plane image analysis: An approach to determining structure from motion. Int J Comput Vision 1(1):7–55, Mar

Bolles RC, Baker HH, Hannah MJ (1993) The JISCT stereo evaluation. In: Image Understanding Workshop: proceedings of a workshop held in Washington, DC apr 18–21, 1993. Morgan Kaufmann, pp 263

Borenstein J, Everett HR, Feng L (1996) Navigating mobile robots: Systems and techniques. AK Peters, Ltd. Natick, MA, USA, Out of print and available at http://www-personal.umich.edu/~johannb/Papers/pos96rep.pdf

Borgefors G (1986) Distance transformations in digital images. Comput Vision Graph 34(3):344–371

Bouguet J-Y (2010) Camera calibration toolbox for MATLAB. http://www.vision.caltech.edu/bouguetj/calib_doc

Bradski G, Kaehler A (2008) Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media

Brady M, Hollerbach JM, Johnson TL, Lozano-Pérez T, Mason MT (eds) (1982) Robot motion: Planning and control. The MIT Press

Braitenberg V (1986) Vehicles: Experiments in synthetic psychology. The MIT Press

Brockett RW (1983) Asymptotic stability and feedback stabilization. In: Brockett RW, Millmann RS, Sussmann HJ (eds) Progress in mathematics. Differential geometric control theory, vol 27. pp 181–191

Broida TJ, Chandrashekhar S, Chellappa R (1990) Recursive 3-D motion estimation from a monocular image sequence. IEEE T Aero Elec Sys 26(4):639–656

Brooks R (1986) A robust layered control system for a mobile robot. IEEE T Robotic Autom 2(1):14–23

Brooks RA (1989) A robot that walks: Emergent behaviors from a carefully evolved network. MIT AI Lab, Memo 1091

Brown MZ, Burschka D, Hager GD (2003) Advances in computational stereo. IEEE T Pattern Anal 25(8):993–1008

Buehler M, Iagnemma K, Singh S (eds) (2007) The 2005 DARPA grand challenge: The great robot race. Springer Tracts in Advanced Robotics, vol 36. Springer-Verlag

Buehler M, Iagnemma K, Singh S (eds) (2010) The DARPA urban challenge. Tracts in Advanced Robotics, vol 56. Springer-Verlag

Bukowski R, Haynes LS, Geng Z, Coleman N, Santucci A, Lam K, Paz A, May R, DeVito M (1991) Robot hand-eye coordination rapid prototyping environment. In: Proc ISIR, pp 16.15–16.28

Buttazzo GC, Allotta B, Fanizza FP (1993) Mousebuster: A robot system for catching fast moving objects by vision. In: Proc. IEEE Int. Conf. Robotics and Automation, Atlanta. pp 932–937

Calonder M, Lepetit V, Strecha C, Fua P (2010) BRIEF: Binary robust independent elementary features. In: Daniilidis K, Maragos P, Paragios N (eds) Lecture notes in computer science. Computer Vision – ECCV 2010, vol 6311. Springer-Verlag, Berlin Heidelberg, pp 778–792

Canny JF (1983) Finding edges and lines in images. MIT, Artificial Intelligence Laboratory, AI-TR-720. Cambridge, MA

Canny J (1987) A computational approach to edge detection. Readings in computer vision: issues, problems, principles, and paradigms 184

Chahl JS, Srinivasan MV (1997) Reflective surfaces for panoramic imaging. Appl Optics 31(36): 8275–8285

Chaumette F (1990) La relation vision-commande: Théorie et application et des tâches robotiques. Université de Rennes 1

Chaumette F (1998) Potential problems of stability and convergence in image-based and position-based visual servoing. In: Kriegman DJ, Hager GD, Morse AS (eds) Lecture notes in control and information sciences. The confluence of vision and control, vol 237. Springer-Verlag, pp 66–78

Chaumette F (2004) Image moments: A general and useful set of features for visual servoing. IEEE T Robotic Autom 20(4):713–723

Chaumette F, Hutchinson S (2006) Visual servo control 1: Basic approaches. IEEE T Robotic Autom 13(4):82–90

Chaumette F, Hutchinson S (2007) Visual servo control 2: Advanced approaches. IEEE T Robotic Autom 14(1):109–118

Chaumette F, Rives P, Espiau B (1991) Positioning of a robot with respect to an object, tracking it and estimating its velocity by visual servoing. In: Proc. IEEE Int. Conf. Robotics and Automation, Seoul. pp 2248–2253

Chesi G, Hashimoto K (eds) (2010) Visual servoing via advanced numerical methods. Lecture notes in computer science, 401. Springer-Verlag

Chiaverini S, Sciavicco L, Siciliano B (1991) Control of robotic systems through singularities. Lecture Notes in Control and Information Sciences. Advanced Robot Control, Proceedings of the International Workshop on Nonlinear and Adaptive Control: Issues in Robotics, vol 162. Springer-Verlag, pp 285–295

Chiuso A, Favaro P, Jin H, Soatto S (2002) Structure from motion causally integrated over time. IEEE T Pattern Anal 24(4):523–535

Choset HM, Lynch KM, Hutchinson S, Kantor G, Burgard W, Kavraki LE, Thrun S (2005) Principles of robot motion. The MIT Press

Chum O, Matas J (2005) Matching with PROSAC – Progressive sample consensus. In: IEEE Conf. on Computer Vision and Pattern Recognition, vol 2. San Diego, pp 220–226

Colicchia G, Waltner C, Hopf M, Wiesner H (2009) The scallop's eye – A concave mirror in the context of biology. Physics Education 44(2):175–179

Commission Internationale de L'Éclairage (1987) Colorimetry, 2nd ed. Commission Internationale de L'Eclairage, CIE No 15.2
Corke PI (1994) High-performance visual closed-loop robot control. University of Melbourne, Dept. Mechanical and Manufacturing Engineering. http://eprints.unimelb.edu.au/archive/00000547/01/thesis.pdf
Corke PI (1996a) In situ measurement of robot motor electrical constants. Robotica 14(4):433–436
Corke PI (1996b) Visual control of robots: High-performance visual servoing. Mechatronics, vol 2. Research Studies Press (John Wiley). Out of print and available at http://www.petercorke.com/bluebook
Corke PI (2001) Mobile robot navigation as a planar visual servoing problem. In: Jarvis RA, Zelinsky A (eds) Springer tracts in advanced robotics. Robotics Research: The 10th International Symposium, vol 6. IFRR, Lorne, pp 361–372
Corke PI (2007) A simple and systematic approach to assigning Denavit-Hartenberg parameters. IEEE T Robotic Autom 23(3):590–594
Corke PI (2010) Spherical image-based visual servo and structure estimation. In: Proc. IEEE Int. Conf. Robotics and Automation, Anchorage, pp 5550–5555
Corke PI, Armstrong-Hélouvry BS (1994) A search for consensus among model parameters reported for the PUMA 560 robot. In: Proc. IEEE Int. Conf. Robotics and Automation, San Diego. pp 1608–1613
Corke PI, Armstrong-Hélouvry B (1995) A meta-study of PUMA 560 dynamics: A critical appraisal of literature data. Robotica 13(3):253–258
Corke PI, Good MC (1992) Dynamic effects in high-performance visual servoing. In: Proc. IEEE Int. Conf. Robotics and Automation, Nice. pp 1838–1843
Corke PI, Good MC (1996) Dynamic effects in visual closed-loop systems. IEEE T Robotic Autom 12(5):671–683
Corke PI, Hutchinson SA (2001) A new partitioned approach to image-based visual servo control. IEEE T Robotic Autom 17(4):507–515
Corke PI, Dunn PA, Banks JE (1999) Frame-rate stereopsis using non-parametric transforms and programmable logic. In: Proc. IEEE Int. Conf. Robotics and Automation, Detroit. pp 1928–1933
Corke PI, Strelow D, Singh S (2004) Omnidirectional visual odometry for a planetary rover. In: Proc. Int. Conf on Intelligent Robots and Systems (IROS), Sendai. pp 4007–4012
Corke PI, Spindler F, Chaumette F (2009) Combining Cartesian and polar coordinates in IBVS. In: Proc. Int. Conf on Intelligent Robots and Systems (IROS), St. Louis. pp 5962–5967
Craig JJ (1987) Adaptive control of mechanical manipulators. Addison-Wesley
Craig JJ (2004) Introduction to robotics: Mechanics and control. Prentice Hall
Craig JJ, Hsu P, Sastry SS (1987) Adaptive control of mechanical manipulators. Int J Robot Res 6(2):16
Cummins M, Newman P (2008) FAB-MAP: Probabilistic localization and mapping in the space of appearance. Int J Robot Res 27(6):647
Cutting JE (1997) How the eye measures reality and virtual reality. Behav Res Meth Ins C 29(1):27–36
Daniilidis K, Klette R (eds) (2006) Imaging beyond the pinhole camera. Computational Imaging, vol 33. Springer-Verlag
Davison AJ, Reid ID, Molton ND, Stasse O (2007) MonoSLAM: Real-time single camera SLAM. IEEE T Pattern Anal 29(6):1052–1067
Deguchi K (1998) Optimal motion control for image-based visual servoing by decoupling translation and rotation. In: Proc. Int. Conf on Intelligent Robots and Systems (IROS), Victoria, Canada. Oct, pp 705–711
Dellaert F, Seitz SM, Thorpe CE, Thrun S (2000) Structure from motion without correspondence. In: IEEE Conf. on Computer Vision and Pattern Recognition, Hilton Head Island, SC. pp 557–564
DeMenthon D, Davis LS (1992) Exact and approximate solutions of the perspective-three-point problem. IEEE T Pattern Anal 14(11):1100–1105
Denavit J, Hartenberg RS (1955) A kinematic notation for lower-pair mechanisms based on matrices. J Appl Mech-T ASME 22(1):215–221
Deo AS, Walker ID (1995) Overview of damped least-squares methods for inverse kinematics of robot manipulators. J Intell Robot Syst 14(1):43–68
Deriche R, Giraudon G (1993) A computational approach for corner and vertex detection. Int J Comput Vision 10(2):101–124
DeWitt BA, Wolf PR (2000) Elements of photogrammetry (with applications in GIS). McGraw-Hill Higher Education
Dickmanns ED (2007) Dynamic vision for perception and control of motion. Springer-Verlag, London
Dickmanns ED, Graefe V (1988a) Applications of dynamic monocular machine vision. Mach Vision Appl 1:241–261
Dickmanns ED, Graefe V (1988b) Dynamic monocular machine vision. Mach Vision Appl 1(4): 223–240
Dickmanns ED, Zapp A (1987) Autonomous high speed road vehicle guidance by computer vision. In: Tenth Triennial World Congress of the International Federation of Automatic Control, vol 4. Munich, pp 221–226
Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1(1):269–271

Dougherty ER, Lotufo RA (2003) Hands-on morphological image processing. Society of Photo-Optical Instrumentation Engineers (SPIE)
Duda RO, Hart PE (1972) Use of the Hough transformation to detect lines and curves in pictures. Commun ACM 15(1):11–15
Espiau B, Chaumette F, Rives P (1992) A new approach to visual servoing in robotics. IEEE T Robotic Autom 8(3):313–326
Everett HR (1995) Sensors for mobile robots: Theory and application. AK Peters, Ltd.
Faugeras OD (1993) Three-dimensional computer vision: A geometric viewpoint. The MIT Press
Faugeras OD, Lustman F (1988) Motion and structure from motion in a piecewise planar environment. Int J Pattern Recogn 2(3):485–508
Faugeras O, Luong QT, Papadopoulou T (2001) The geometry of multiple images: The laws that govern the formation of images of a scene and some of their applications. The MIT Press
Featherstone R (1987) Robot dynamics algorithms. Kluwer Academic
Feddema JT (1989) Real time visual feedback control for hand-eye coordinated robotic systems. Purdue University
Feddema JT, Mitchell OR (1989) Vision-guided servoing with feature-based trajectory generation. IEEE T Robotic Autom 5(5):691–700
Feddema JT, Lee CSG, Mitchell OR (1991) Weighted selection of image features for resolved rate visual feedback control. IEEE T Robotic Autom 7(1):31–47
Felzenszwalb PF, Huttenlocher DP (2004) Efficient graph-based image segmentation. Int J Comput Vision 59(2):167–181
Ferguson D, Stentz A (2006) Using interpolation to improve path planning: The Field D* algorithm. J Field Robotics 23(2):79–101
Fischler MA, Bolles RC (1981) Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM 24(6):381–395
Fisher RB (2004) The PETS04 surveillance ground-truth data sets. In: Proc. 6th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, Prague. pp 1–5
Fomena R, Chaumette F (2007) Visual servoing from spheres using a spherical projection model. In: Proc. IEEE Int. Conf. Robotics and Automation, Rome, pp 2080–2085
Förstner W (1994) A framework for low level feature extraction. In: Ecklundh J-O (ed) Lecture notes in computer science. Computer Vision – ECCV 1994, vol 800. Springer-Verlag, Berlin Heidelberg, pp 383–394
Förstner W, Gülch E (1987) A fast operator for detection and precise location of distinct points, corners and centres of circular features. In: ISPRS Intercommission Workshop, Interlaken, pp 149–155
Forsyth DA, Ponce J (2002) Computer vision: A modern approach. Prentice Hall
Freeman H (1974) Computer processing of line-drawing images. ACM Comput Surv 6(1):57–97
Friedman DP, Felleisen M, Bibby D (1987) The little LISPer. MIT Press
Funda J, Taylor RH, Paul RP (1990) On homogeneous transforms, quaternions, and computational efficiency. IEEE T Robotic Autom 6(3):382–388
Gans NR, Hutchinson SA, Corke PI (2003) Performance tests for visual servo control systems, with application to partitioned approaches to visual servo control. Int J Robot Res 22(10–11):955
Gautier M, Khalil W (1992) Exciting trajectories for the identification of base inertial parameters of robots. Int J Robot Res 11(4):362
Geraerts R, Overmars MH (2004) A comparative study of probabilistic roadmap planners. In: Boissonnat J-D, Burdick J, Goldberg K, Hutchinson S (eds) Springer Tracts in Advanced Robotics. Algorithmic Foundations of Robotics V, vol 7. Springer-Verlag, pp 43–58
Geyer C, Daniilidis K (2000) A unifying theory for central panoramic systems and practical implications. In: Vernon D (ed) Lecture notes in computer science. Computer vision – ECCV 2000, vol 1843. Springer-Verlag, pp 445–461
Goldberg K (ed) (2001) The robot in the garden: Telerobotics and telepistemology in the age of the internet. The MIT Press
Goldberg K, Siegwart R (eds) (2001) Beyond webcams: An introduction to online robots. The MIT Press
Gonzalez R, Woods R (2008) Digital image processing, 3rd ed. Prentice Hall
Gonzalez R, Woods R, Eddins S (2009) Digital image processing using MATLAB, 2nd ed. Gatesmark
Groves PD (2008) Principles of GNSS, inertial, and multisensor integrated navigation systems. Artech House
Hager GD, Toyama K (1998) X Vision: A portable substrate for real-time vision applications. Comput Vis Image Und 69(1):23–37
Hamel T, Mahony R (2002) Visual servoing of an under-actuated dynamic rigid-body system: An image based approach. IEEE T Robotic Autom 18(2):187–198
Hamel T, Mahony R, Lozano R, Ostrowski J (2002) Dynamic modelling and configuration stabilization for an X4-flyer. IFAC World Congress 1(2), p 3. Available at: http://citeseerx.ist.psu.edu/viewdoc/download
Hansen P, Corke PI, Boles W (2010) Wide-angle visual feature matching for outdoor localization. Int J Robot Res 29(1–2):267–297

Harris CG, Stephens MJ (1988) A combined corner and edge detector. In: Proceedings of the Fourth Alvey Vision Conference, Manchester. pp 147–151
Hart PE (2009) How the Hough transform was invented [DSP history]. IEEE Signal Proc Mag 26(6): 18–22
Hartenberg RS, Denavit J (1964) Kinematic synthesis of linkages. McGraw-Hill New York, available online at http://kmoddl.library.cornell.edu/bib.php?m=23
Hartley R, Zisserman A (2003) Multiple view geometry in computer vision. Cambridge University Press, New York
Harvey P (nd) ExifTool. http://www.sno.phy.queensu.ca/~phil/exiftool
Hashimoto K (ed) (1993) Visual servoing. In: Robotics and automated systems, vol 7. World Scientific
Hashimoto K, Kimoto T, Ebine T, Kimura H (1991) Manipulator control with image-based visual servo. In: Proc. IEEE Int. Conf. Robotics and Automation, Seoul. pp 2267–2272
Hellerstein JL, Diao Y, Parekh S, Tilbury DM (2004) Feedback control of computing systems. IEEE Press – Wiley
Hill J, Park WT (1979) Real time control of a robot with a mobile camera. In: Proc. 9th ISIR, SME, Washington, DC. Mar, pp 233–246
Hoag D (1963) Consideration of Apollo IMU gimbal lock. MIT Instrumentation Laboratory, E–1344, http://www.hq.nasa.gov/alsj/e-1344.htm
Hollerbach JM (1980) A recursive Lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity. IEEE T Syst Man Cyb 10(11):730–736, Nov
Hollerbach JM (1982) Dynamics. In: Brady M, Hollerbach JM, Johnson TL, Lozano-Perez T, Mason MT (eds) Robot motion – Planning and control. The MIT Press, pp 51–71
Horaud R, Canio B, Leboullenx O (1989) An analytic solution for the perspective 4-point problem. Comput Vision Graph 47(1):33–44
Horn BKP (1987) Closed-form solution of absolute orientation using unit quaternions. J Opt Soc Am A 4(4):629–642
Horn BKP, Hilden HM, Negahdaripour S (1988) Closed-form solution of absolute orientation using orthonormal matrices. J Opt Soc Am A 5(7):1127–1135
Hosoda K, Asada M (1994) Versatile visual servoing without knowledge of true Jacobian. In: Proc. Int. Conf on Intelligent Robots and Systems (IROS), Munich. Sep, pp 186–193
Howard TM, Green CJ, Kelly A, Ferguson D (2008) State space sampling of feasible motions for high-perfor-mance mobile robot navigation in complex environments. J Field Robotics 25(6–7):325–345
Hu MK (1962) Visual pattern recognition by moment invariants. IRE T Inform Theor 8:179–187
Huang TS, Netravali AN (1994) Motion and structure from feature correspondences: A review. P IEEE 82(2):252–268
Humenberger M, Zinner C, Kubinger W (2009) Performance evaluation of a census-based stereo matching algorithm on embedded and multi-core hardware. In: Proc. 19th Int. Symp. on Image and Signal Processing and Analysis (ISPA). Sep, pp 388–393
Hunt RWG (1987) The reproduction of colour, 4th ed. Fountain Press
Hunter RS, Harold RW (1987) The measurement of appearance. John Wiley
Hutchinson S, Hager G, Corke PI (1996) A tutorial on visual servo control. IEEE T Robotic Autom 12(5):651–670
Iwatsuki M, Okiyama N (2002a) A new formulation of visual servoing based on cylindrical coordinate system with shiftable origin. In: Proc. Int. Conf on Intelligent Robots and Systems (IROS), Lausanne, pp 354–359
Iwatsuki M, Okiyama N (2002b) Rotation-oriented visual servoing based on cylindrical coordinates. In: Proc. IEEE Int. Conf. Robotics and Automation, Washington, DC. May, pp 4198–4203
Izaguirre A, Paul RP (1985) Computation of the inertial and gravitational coefficients of the dynamics equations for a robot manipulator with a load. In: Proc. IEEE Int. Conf. Robotics and Automation. Mar, pp 1024–1032
Jägersand M, Fuentes O, Nelson R (1996) Experimental evaluation of uncalibrated visual servoing for pre-cision manipulation. In: Proc. IEEE Int. Conf. Robotics and Automation, Albuquerque, NM. pp 2874–2880
Jarvis RA, Byrne JC (1988) An automated guided vehicle with map building and path finding capabilities. In: Robotics Research: The Fourth international symposium. The MIT Press, pp 497–504
Jazwinski AH (1970) Stochastic processes and filtering theory. Academic Press
Jebara T, Azarbayejani A, Pentland A (1999) 3D structure from 2D motion. IEEE Signal Proc Mag 16(3):66–84
Julier SJ, Uhlmann JK (2004) Unscented filtering and nonlinear estimation. P IEEE 92(3):401–422
Kahn ME (1969) The near-minimum time control of open-loop articulated kinematic linkages. Stanford University, AIM-106
Kálmán RE (1960) A new approach to linear filtering and prediction problems. J Basic Eng-T Asme 82(1): 35–45
Kane TR, Levinson DA (1983) The use of Kane's dynamical equations in robotics. Int J Robot Res 2(3):3–21
Kavraki LE, Svestka P, Latombe JC, Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE T Robotic Autom 12(4):566–580

Kelly R (1996) Robust asymptotically stable visual servoing of planar robots. IEEE T Robotic Autom 12(5):759–766
Kelly R, Carelli R, Nasisi O, Kuchen B, Reyes F (2002a) Stable visual servoing of camera-in-hand robotic systems. IEEE-ASME T Mech 5(1):39–48
Kelly R, Shirkey P, Spong MW (2002b) Fixed-camera visual servo control for planar robots. In: Proc. IEEE Int. Conf. Robotics and Automation. IEEE, Washington, DC, pp 2643–2649
Khalil W, Creusot D (1997) SYMORO+: A system for the symbolic modelling of robots. Robotica 15(2): 153–161
Khalil W, Dombre E (2002) Modeling, identification and control of robots. Kogan Page Science
King-Hele D (2002) Erasmus Darwin's improved design for steering carriages and cars. Notes and Records of the Royal Society of London 56(1):41–62
Klafter RD, Chmielewski TA, Negin M (1989) Robotic engineering – An integrated approach. Prentice-Hall
Klein CA, Huang CH (1983) Review of pseudoinverse control for use with kinematically redundant manipulators. IEEE T Syst Man Cyb 13:245–250
Klette R, Kruger N, Vaudrey T, Pauwels K, van Hulle M, Morales S, Kandil F, Haeusler R, Pugeault N, Rabe C (2011) Performance of correspondence algorithms in vision-based driver assistance using an online image sequence database. IEEE T Veh Technol 60(5):2012–2026
Koenderink JJ (1984) The structure of images. Biol Cybern 50(5):363–370
Koenig S, Likhachev M (2002) D* Lite. In: Proceedings of the National Conference on Artificial Intelligence, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, pp 476–483
Koenig S, Likhachev M (2005) Fast replanning for navigation in unknown terrain. IEEE T Robotic Autom 21(3):354–363
Kriegman DJ, Hager GD, Morse AS (eds) (1998) The confluence of vision and control. Lecture Notes in Control and Information Sciences, vol 237. Springer-Verlag
Kuipers JB (1999) Quaternions and rotation sequences: A primer with applications to orbits, aeroespace and virtual reality. Princeton University Press
Lamport L (1994) LATEX: A document preparation system. User's guide and reference manual. Addison-Wesley Publishing Company, Reading, Ma
LaValle SM (1998) Rapidly-exploring random trees: A new tool for path planning. Computer Science Dept., Iowa State University, TR 98–11
LaValle SM (2006) Planning algorithms. Cambridge Univ Press
LaValle SM, Kuffner JJ (2001) Randomized kinodynamic planning. Int J Robot Res 20(5):378–400
Leavers VF (1993) Which Hough transform? Comput Vis Image Und 58(2):250–264
Lee CSG, Lee BH, Nigham R (1983) Development of the generalized D'Alembert equations of motion for mechanical manipulators. In: Proc. 22nd CDC, San Antonio, Texas. pp 1205–1210
Lepetit V, Moreno-Noguer F, Fua P (2009) EPnP: An accurate O(n) solution to the PnP problem. Int J Comput Vision 81(2):155–166
Li H, Hartley R (2006) Five-point motion estimation made easy. In: 18th Int. Conf. on Pattern Recognition ICPR 2006, Hong Kong, pp 630–633
Lin Z, Zeman V, Patel RV (1989) On-line robot trajectory planning for catching a moving object. In: Proc. IEEE Int. Conf. Robotics and Automation. pp 1726–1731
Lindeberg T (1993) Scale-space theory in computer vision. Springer-Verlag
Lloyd J, Hayward V (1991) Real-time trajectory generation using blend functions. In: Proc. IEEE Int. Conf. Robotics and Automation, Seoul. pp 784–789
Longuet-Higgins H (1981) A computer algorithm for reconstruction of a scene from two projections. Nature 293:133–135
Lourakis MIA, Argyros AA (2009) SBA: A software package for generic sparse bundle adjustment. ACM T Math Software 36(1):1–30
Lovell J, Kluger J (1994) Apollo 13. Coronet Books
Lowe DG (1991) Fitting parametrized three-dimensional models to images. IEEE T Pattern Anal 13(5): 441–450
Lowe DG (2004) Distinctive image features from scale-invariant keypoints. Int J Comput Vision 60(2): 91–110
Lucas SM (2005) ICDAR 2005 text locating competition results. In: Proceedings of the Eighth International Conference on Document Analysis and Recognition, ICDAR05, pp 80–84
Lucas BD, Kanade T (1981) An iterative image registration technique with an application to stereo vision. In: International joint conference on artificial intelligence (IJCAI), Vancouver, vol 2. http://ijcai.org/Past%20Proceedings/IJCAI-81-VOL-2/PDF/017.pdf, pp 674–679
Luh JYS, Walker MW, Paul RPC (1980) On-line computational scheme for mechanical manipulators. J Dyn Syst-T ASME 102(2):69–76
Lumelsky V, Stepanov A (1986) Dynamic path planning for a mobile automaton with limited information on the environment. IEEE T Automat Contr 31(11):1058–1063
Luong QT (1992) matrice fondamentale et autocalibration en vision par ordinateur. Universitéde Paris-Sud, Orsay, France
Ma Y, Kosecka J, Soatto S, Sastry S (2003) An invitation to 3D. Springer-Verlag

Maimone M, Cheng Y, Matthies L (2007) Two years of visual odometry on the Mars exploration rovers. J Field Robotics 24(3):169–186
Makhlin AG (1985) Stability and sensitivity of servo vision systems. In: Proc 5th Int Conf on Robot Vision and Sensory Controls – RoViSeC 5. IFS (Publications), Amsterdam, pp 79–89
Malis E, Vargas M (2007) Deeper understanding of the homography decomposition for vision-based control. INRIA, 6303
Malis E, Chaumette F, Boudet S (1999) 2-1/2D visual servoing. IEEE T Robotic Autom 15(2):238–250
Marey M, Chaumette F (2008) Analysis of classical and new visual servoing control laws. In: Proc. IEEE Int. Conf. Robotics and Automation, Pasadena, pp 3244–3249
Mariottini GL, Prattichizzo D (2005) EGT for multiple view geometry and visual servoing: Robotics vision with pinhole and panoramic cameras. IEEE T Robotic Autom 12(4):26–39
Mariottini GL, Oriolo G, Prattichizzo D (2007) Image-based visual servoing for nonholonomic mobile robots using epipolar geometry. IEEE T Robotic Autom 23(1):87–100
Marr D (2010) Vision: A computational investigation into the human representation and processing of visual information. The MIT Press
Martin D, Fowlkes C, Tal D, Malik J (2001) A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Proc. 8th Int'l Conf. Computer Vision, vol 2, pp 416–423
Masutani Y, Mikawa M, Maru N, Miyazaki F (1994) Visual servoing for non-holonomic mobile robots. In: Proc. Int. Conf on Intelligent Robots and Systems (IROS), Munich, pp 1133–1140
Matarić MJ (2007) The robotics primer. MIT Press
Matas J, Chum O, Urban M, Pajdla T (2004) Robust wide-baseline stereo from maximally stable extremal regions. Image Vision Comput 22(10):761–767
Matthews ND, An PE, Harris CJ (1995) Vehicle detection and recognition for autonomous intelligent cruise control. Image, Speech and Intelligent Systems 6
Matthies L (1992) Stereo vision for planetary rovers: Stochastic modeling to near real-time implementation. Int J Comput Vision 8(1):71–91
Mayeda H, Yoshida K, Osuka K (1990) Base parameters of manipulator dynamic models. IEEE T Robotic Autom 6(3):312–321
McLauchlan PF (1999) The variable state dimension filter applied to surface-based structure from motion. University of Surrey, VSSP-TR-4/99
Merlet JP (2006) Parallel robots. Kluwer Academic
Mettler B (2003) Identification modeling and characteristics of miniature rotorcraft. Kluwer Academic
Mičušík B, Pajdla T (2003) Estimation of omnidirectional camera model from epipolar geometry. In: IEEE Conf. on Computer Vision and Pattern Recognition, Madison, vol 1. pp 485–490
Middleton RH, Goodwin GC (1988) Adaptive computed torque control for rigid link manipulations. Syst Control Lett 10(1):9–16
Mikolajczyk K, Schmid C (2004) Scale and affine invariant interest point detectors. Int J Comput Vision 60(1):63–86
Mikolajczyk K, Schmid C (2005) A performance evaluation of local descriptors. IEEE T Pattern Anal 27(10):1615–1630
Mindell DA (2008) Digital Apollo. MIT Press
Molton N, Brady M (2000) Practical structure and motion from stereo when motion is unconstrained. Int J Comput Vision 39(1):5–23
Montemerlo M, Thrun S, Koller D, Wegbreit B (2002) FastSLAM: A factored solution to the simultaneous localization and mapping problem. In: Proceedings of the AAAI National Conference on Artificial Intelligence. AAAI, Edmonton, Canada
Moravec H (1980) Obstacle avoidance and navigation in the real world by a seeing robot rover. Stanford University
Morel G, Liebezeit T, Szewczyk J, Boudet S, Pot J (2000) Explicit incorporation of 2D constraints in vision based control of robot manipulators. In: Corke PI, Trevelyan J (eds) Lecture notes in control and information sciences. Experimental robotics VI, vol 250. Springer-Verlag, pp 99–108
NASA (1970) Apollo 13: Technical air-to-ground voice transcription. Test Division, Apollo Spacecraft Program Office, http://www.hq.nasa.gov/alsj/a13/AS13_TEC.PDF
Nayar SK (1997) Catadioptric omnidirectional camera. In: IEEE Conf. on Computer Vision and Pattern Recognition, Los Alamitos, CA, pp 482–488
Nethery JF, Spong MW (1994) Robotica: A mathematica package for robot analysis. IEEE T Robotic Autom 1(1):13–20
Ng J, Bräunl T (2007) Performance comparison of bug navigation algorithms. J Intell Robot Syst 50(1):73–84
Niblack W (1985) An introduction to digital image processing. Strandberg Publishing Company Birkeroed, Denmark
Nilsson NJ (1971) Problem-solving methods in artificial intelligence. McGraw-Hill
Nistér D (2003) An efficient solution to the five-point relative pose problem. In: IEEE Conf. on Computer Vision and Pattern Recognition, vol. 2. Madison, pp 195–202

Nistér D, Naroditsky O, Bergen J (2006) Visual odometry for ground vehicle applications. J Field Robotics 23(1):3–20
Nixon MS, Aguado AS (2008) Feature extraction and image processing. Academic Press
Noble JA (1988) Finding corners. Image Vision Comput 6(2):121–128
Okutomi M, Kanade T (1993) A multiple-baseline stereo. IEEE T Pattern Anal 15(4):353–363
Ollis M, Herman H, Singh S (1999) Analysis and design of panoramic stereo vision using equi-angular pixel cameras. Robotics Institute, Carnegie Mellon University, CMU-RI-TR-99-04, Citeseer, Pittsburgh, PA
Orin DE, McGhee RB, Vukobratovic M, Hartoch G (1979) Kinematics and kinetic analysis of open-chain linkages utilizing newton-euler methods. Math Biosci 43(1/2):107–130
Ortega R, Spong MW (1989) Adaptive motion control of rigid robots: A tutorial. Automatica 25(6):877–888
Otsu N (1975) A threshold selection method from gray-level histograms. Automatica 11:285–296
Papanikolopoulos NP, Khosla PK (1993) Adaptive robot visual tracking: Theory and experiments. IEEE T Automat Contr 38(3):429–445
Papanikolopoulos NP, Khosla PK, Kanade T (1993) Visual tracking of a moving target by a camera mounted on a robot: A combination of vision and control. IEEE T Robotic Autom 9(1):14–35
Park FC (1994) Computational aspects of the product-of-exponentials formula for robot kinematics. Automatic Control, IEEE Transactions on 39(3):643–647
Paul R (1972) Modelling, trajectory calculation and servoing of a computer controlled arm. Stanford University
Paul R (1979) Manipulator Cartesian path control. IEEE T Syst Man Cyb 9:702–711
Paul RP (1981) Robot manipulators: Mathematics, programming, and control. MIT Press, Cambridge, Massachusetts
Paul RP, Shimano B (1978) Kinematic control equations for simple manipulators. In: IEEE Conference on Decision and Control, vol 17. pp 1398–1406
Paul RP, Zhang H (1986) Computationally efficient kinematics for manipulators with spherical wrists based on the homogeneous transformation representation. Int J Robot Res 5(2):32–44
Piepmeier JA, McMurray G, Lipkin H (1999) A dynamic quasi-Newton method for uncalibrated visual servoing. In: Proc. IEEE Int. Conf. Robotics and Automation, Detroit. pp 1595–1600
Pilu M (1997) A direct method for stereo correspondence based on singular value decomposition. In: Proc. Computer Vision and Pattern Recognition, IEEE Computer Society, San Juan, pp 261–266
Pollefeys M, Nistér D, Frahm JM, Akbarzadeh A, Mordohai P, Clipp B, Engels C, Gallup D, Kim SJ, Merrell P, et al. (2008) Detailed real-time urban 3D reconstruction from video. Int J Comput Vision 78(2):143–167, Jul
Pomerleau D, Jochem T (1995) No hands across America Journal. http://www.cs.cmu.edu/~tjochem/nhaa/Journal.html
Pomerleau D, Jochem T (1996) Rapidly adapting machine vision for automated vehicle steering. IEEE Expert 11(1):19–27
Pounds P (2007) Design, construction and control of a large quadrotor micro air vehicle. Australian National University
Pounds P, Mahony R, Gresham J, Corke PI, Roberts J (2004) Towards dynamically-favourable quad-rotor aerial robots. In: Proc. Australasian Conf. on Robotics and Automation, Canberra
Pounds P, Mahony R, Corke PI (2006) A practical quad-rotor robot. In: Proc. Australasian Conf. on Robotics and Automation, Auckland
Pounds P, Mahony R, Corke PI (2007) System identification and control of an aerobot drive system. In: Information, Decision and Control. pp 154–159
Poynton CA (2003) Digital video and HDTV: Algorithms and interfaces. Morgan Kaufmann
Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007) Numerical recipes, 3rd ed. Cambridge University Press
Prouty RW (2002) Helicopter performance, stability, and control. Krieger
Pynchon T (2006) Against the day. Jonathan Cape
Rabaud V (nd) Vincent's structure from motion toolbox. http://vision.ucsd.edu/~vrabaud/toolbox
Rives P, Chaumette F, Espiau B (1989) Positioning of a robot with respect to an object, tracking it and estimating its velocity by visual servoing. In: Hayward V, Khatib O (eds) Lecture Notes in Control and Information Sciences. Experimental Robotics I, vol 139. Springer-Verlag, pp 412–428
Rizzi AA, Koditschek DE (1991) Preliminary experiments in spatial robot juggling. In: Chatila R, Hirzinger G (eds) Lecture Notes in Control and Information Sciences. Experimental Robotics II, vol 190. Springer-Verlag, pp 282–298
Roberts LG (1963) Machine perception of three-dimensional solids. MIT Lincoln Laboratory, TR 315, http://www.packet.cc/files/mach-per-3D-solids.html
Rosenfield GH (1959) The problem of exterior orientation in photogrammetry. Photogramm Eng 25(4):536–553
Rosten E, Porter R, Drummond T (2010) FASTER and better: A machine learning approach to corner detection. IEEE T Pattern Anal 32:105–119
Sakaguchi T, Fujita M, Watanabe H, Miyazaki F (1993) Motion planning and control for a robot performer. In: Proc. IEEE Int. Conf. Robotics and Automation, Atlanta. May, pp 925–931

Salvi J, Matabosch C, Fofi D, Forest J (2007) A review of recent range image registration methods with accuracy evaluation. Image Vision Comput 25(5):578–596
Samson C, Espiau B, Le Borgne M (1990) Robot control: The task function approach. Oxford University Press
Sanderson AC, Weiss LE, Neuman CP (1987) Dynamic sensor-based control of robots with visual feedback. IEEE T Robotic Autom RA-3(5):404–417
Scharstein D, Pal C (2007) Learning conditional random fields for stereo. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), Minneapolis, MN
Scharstein D, Szeliski R (2002) A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. Int J Comput Vision 47(1):7–42
Serra J (1983) Image analysis and mathematical morphology. Academic Press
Shi J, Tomasi C (1994) Good features to track. In: Proc. Computer Vision and Pattern Recognition. IEEE Computer Society, Seattle, pp 593–593
Shirai Y (1987) Three-dimensional computer vision. Springer-Verlag, New York
Shirai Y, Inoue H (1973) Guiding a robot by visual feedback in assembling tasks. Pattern Recogn 5(2):99–106
Shoemake K (1985) Animating rotation with quaternion curves. In: Proceedings of ACM SIGGRAPH, San Francisco, pp 245–254
Siciliano B, Khatib O (eds) (2008) Springer handbook of robotics. Springer-Verlag, New York
Siciliano B, Sciavicco L, Villani L, Oriolo G (2008) Robotics: Modelling, planning and control. Springer-Verlag
Siegwart R, Nourbakhsh IR, Scaramuzza D (2011) Introduction to autonomous mobile robots. The MIT Press
Silver WM (1982) On the equivalance of Lagrangian and Newton-Euler dynamics for manipulators. Int J Robot Res 1(2):60–70
Sivic J, Zisserman A (2003) Video Google: A text retrieval approach to object matching in videos. In: Proc. Ninth IEEE Int. Conf. on Computer Vision, pp 1470–1477
Skaar SB, Brockman WH, Hanson R (1987) Camera-space manipulation. Int J Robot Res 6(4):20–32
Skofteland G, Hirzinger G (1991) Computing position and orientation of a freeflying polyhedron from 3D data. In: Proc. IEEE Int. Conf. Robotics and Automation, Seoul. pp 150–155
Slama CC (ed) (1980) Manual of photogrammetry, 4th ed. American Society of Photogrammetry
Sobel D (1996) Longitude: The true story of a lone genius who solved the greatest scientific problem of his time. Fourth Estate London
Soille P (2003) Morphological image analysis: Principles and applications. Springer-Verlag
Spong MW (1989) Adaptive control of flexible joint manipulators. Syst Control Lett 13(1):15–21
Spong MW, Hutchinson S, Vidyasagar M (2006) Robot modeling and control. Wiley
Srinivasan VV, Venkatesh S (1997) From living eyes to seeing machines. Oxford University Press
Stentz A (1994) The D* algorithm for real-time planning of optimal traverses. The Robotics Institute, Carnegie-Mellon University, CMU-RI-TR-94-37
Strelow D, Singh S (2004) Motion estimation from image and inertial measurements. Int J Robot Res 23(12):1157–1195
Sussman GJ, Wisdom J, Mayer ME (2001) Structure and interpretation of classical mechanics. The MIT Press
Sutherland IE (1974) Three-dimensional data input by tablet. P IEEE 62(4):453–461
Svoboda T, Pajdla T (2002) Epipolar geometry for central catadioptric cameras. Int J Comput Vision 49(1):23–37
Szeliski R (2011) Computer vision: Algorithms and applications. Springer-Verlag
Tahri O, Chaumette F (2005) Point-based and region-based image moments for visual servoing of planar objects. IEEE T Robotic Autom 21(6):1116–1127
Tahri O, Mezouar Y, Chaumette F, Corke PI (2009) Generic decoupled image-based visual servoing for cameras obeying the unified projection model. In: Proc. IEEE Int. Conf. Robotics and Automation, Kobe, pp 1116–1121
Taylor RA (1979) Planning and execution of straight line manipulator trajectories. IBM J Res Dev 23(4):424–436
ter Haar Romeny BM (1996) Introduction to scale-space theory: Multiscale geometric image analysis. Utrecht Univ.
Thrun S, Burgard W, Fox D (2005) Probabilistic robotics. The MIT Press
Tissainayagam P, Suter D (2004) Assessing the performance of corner detectors for point feature tracking applications. Image Vision Comput 22(8):663–679
Tomasi C, Kanade T (1991) Detection and tracking of point features. Carnegie Mellon University, CMU-CS-91-132
Torr PHS (2002) A structure and motion toolkit in MATLAB – Interactive adventures in S and M. Microsoft Research. MSR-TR-2002-56, Cambridge, UK
Triggs B, McLauchlan P, Hartley R, Fitzgibbon A (2000) Bundle adjustment – A modern synthesis. Lecture notes in computer science. Vision algorithms: theory and practice, vol 1883. Springer-Verlag, pp 153–177

Tsakiris D, Rives P, Samson C (1998) Extending visual servoing techniques to nonholonomic mobile robots. In: Kriegman DJ, Hager GD, Morse AS (eds) Lecture Notes in Control and Information Sciences. The confluence of vision and control, vol 237. Springer-Verlag, pp 106–117

Uicker JJ (1965) On the dynamic analysis of spatial linkages using 4 by 4 matrices. Dept. Mechanical Engineering and Astronautical Sciences, NorthWestern University

Usher K (2005) Visual homing for a car-like vehicle. Queensland University of Technology

Usher K, Ridley P, Corke PI (2003) Visual servoing of a car-like vehicle – An application of omnidirectional vision. In: Proc. IEEE Int. Conf. Robotics and Automation, Taipai. Sep, pp 4288–4293

Vedaldi A, Fulkerson B (2008) VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org

Walker MW, Orin DE (1982) Efficient dynamic computer simulation of robotic mechanisms. J Dyn Syst-T ASME 104(3):205–211

Walter WG (1950) An imitation of life. Sci Am 182(5):42–45

Walter WG (1951) A machine that learns. Sci Am 185(2):60–63

Walter WG (1953) The living brain. Duckworth London

Weiss LE (1984) Dynamic visual servo control of robots: An adaptive image-based approach. Carnegie-Mellon University

Weiss L, Sanderson AC, Neuman CP (1987) Dynamic sensor-based control of robots with visual feedback. IEEE T Robotic Autom 3(1):404–417

Westmore DB, Wilson WJ (1991) Direct dynamic control of a robot using an end-point mounted camera and Kalman filter position estimation. In: Proc. IEEE Int. Conf. Robotics and Automation, Seoul. Apr, pp 2376–2384

Whitney DE (1969) Resolved motion rate control of manipulators and human prostheses. IEEE T Man Machine 10(2):47–53

Wiener N (1965) Cybernetics or control and communication in the animal and the machine. The MIT Press

Wolf PR (1974) Elements of photogrammetry. McGraw-Hill

Woodfill J, Von Herzen B (1997) Real-time stereo vision on the PARTS reconfigurable computer. In: Proc. IEEE Symposium on FPGAs for Custom Computing Machines, Grenoble. pp 201–210

Xu G, Zhang Z (1996) Epipolar geometry in stereo, motion, and object recognition: A unified approach. Springer-Verlag

Ying X, Hu Z (2004) Can we consider central catiodioptric cameras and fisheye cameras within a unified imaging model. In: Pajdla T, Matas J (eds) Lecture notes in computer science. Computer vision – ECCV 2004, vol 3021. Springer-Verlag, pp 442–455

Yoshikawa T (1984) Analysis and control of robot manipulators with redundancy. In: Brady M, Paul R (eds) Robotics Research: The First International Symposium. The MIT Press, pp 735–747

Zabih R, Woodfill J (1994) Non-parametric local transforms for computing visual correspondence. In: Ecklundh J-O (ed) Lecture notes in computer science. Computer Vision – ECCV 1994, vol 800. Springer-Verlag, Berlin Heidelberg, pp 151–158

Zarchan P, Musoff H (2005) Fundamentals of Kalman filtering: A practical approach. Progress in Astronautics and Aeronautics, vol 208. American Institute of Aeronautics and Astronautics

Zhang Z, Faugeras O, Kohonen T, Hunag TS, Schroeder MR (1992) Three D-dynamic scene analysis: A stereo based approach. Springer-Verlag, New York

Index

Index of People

Index of Functions, Classes and Methods

Classes are shown in bold, Simulink® models in italics, and methods are prefixed by a dot. All others are Toolbox functions.

General Index