Quaternionen und ihre Anwendung mit der Robotic Toolbox

E. G. Kunze

1 Quaternionenoperationen mit der Robotic-Toolbox

Die Robotic Toolbox [1] bietet einige Funktionen für die Ausführung von Drehoperationen mit Quaternionen. Eine Quaternion wird mit Hilfe der Funktion rt_quaternion definiert. Dabei sind fünf Arten der Definition möglich:

q = rt_quaternion(q#)	q# ist eine existierende Quaternion, wird kopiert
q = rt_quaternion(v, theta)	v ist der Vektor der Drehachse, theta der Drewinkel
q = rt_quaternion(R)	R eine 3 x 3 Rotationsmatrix
q = rt_quaternion(T)	T eine 4 x 4 homogene Tranformationsmatrix
q = rt_quaternion(s, v)	s Skalarteil, v Vektorteil, liefert **kein** Einheitsquaternion

Eine homogene Transformation kann, z. B. über die Funktionen rt_rotx(α_rad), rt_roty(β_rad) oder rt_rotx(γ_rad) gefunden werden für eine Drehung um die x-, y- oder z-Achse:

1.1 Beispiel für die Erzeugung von Transformationsmatrizen

α_rad = 45°/180 * %pi
$T_x(\alpha)$ = rt_rotx(α_rad) (auch roty und rotz stehen zur Verfügung)

Aus T kann die Rotationsmatrix entnommen werden:

R_x = rt_tr2rot(T)

Die Programmierung in Scilab lautet:

```
theta = %pi/4
Tx = rt_rotx(theta)
Rx = rt_tr2rot(Tx)
```

Als Ergebnis erhält man:

```
--> theta  =
    0.7853982

 Tx  =
    1.    0.           0.           0.
    0.    0.7071068  - 0.7071068    0.
    0.    0.7071068    0.7071068    0.
    0.    0.           0.           1.

 Rx  =
    1.    0.           0.
    0.    0.7071068  - 0.7071068
    0.    0.7071068    0.7071068
```

Quaternionen können auf verschiedenen Wegen definiert werden. Einer verwendet die homogene Matrix T:

```
q = rt_quaternion(Tx)
```

```
q  =

0.923880 <0.382683, 0.000000, 0.000000>
```

Auf die Quaternion können diverse Methoden angewandt werden (nur lesend):

quat.d	liefert die 4 Elemente
quat.s	liefert den Skalarteil
quat.v	liefert den Vektorteil
quat.t	liefert die homogene Transformationsmatrix T
quat.r	liefert die Rotationsmatrix R

```
Elemente = q.d
Skalarteil = q.s
Vektorteil = q.v
Tx = q.t
Rx = q.r
```

```
Elemente  =
    0.9238795    0.3826834    0.    0.

 Skalarteil  =
    0.9238795

 Vektorteil  =
    0.3826834    0.    0.

Tx  =
    1.    0.           0.           0.
    0.    0.7071068  - 0.7071068    0.
    0.    0.7071068    0.7071068    0.
    0.    0.           0.           1.

 Rx  =
    1.    0.           0.
    0.    0.7071068  - 0.7071068
    0.    0.7071068    0.7071068
```

Das Quaternion läßt sich graphisch darstellen in Form der Matrix **R**, durch die es definiert wurde.

```
// Create a new graphic window and set a
point of view
h0 = scf(0); a0 = h0.children;
a0.tight_limits = "on"; a0.rotation_angles
= [68, 40];

// Plot the Scilab-World coordinate frame.
// X, Y, and Z axes are in black (default
color).
rt_plot(rt_quaternion(eye(3,3)));

// Plot the coordinate frame after the
rotation.
// The Z-axis is in blue whereas the X and
the Y axes are in red und green.
rt_plot(q, "X", "red", "Y", "green", "Z",
"blue");
```

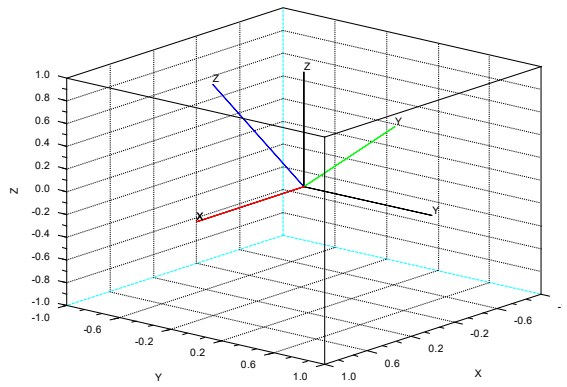

Bild 1: Darstellung der Transformation

Darüberhinaus sind einige Operatoren speziell für Quaternionen konditioniert worden. Die Technik heißt Overload. Bekannte Operatoren nehmen dabei ein auf die Operanden abgestimmtes Verhalten an:

q1 * q2 liefert das Produkt von Quaternionen

Dies wird mit der Overload-Funktion %quat_m_quat realisiert:

```
function [qp] = %quat_m_quat(q1, q2)
// File name:        %quat_m_quat.sci
// Functions:        %quat_m_quat, * (star)
// Description:      multiply quaternion by quaternion
// Annotations:      this code is a Scilab port of corresponding function in the
```

```
//                  Robotics Toolbox for MATLAB(R) written by Peter I. Corke.
// References:      Robotics Toolbox for MATLAB(R),robot7.1/@quaternion/mtimes.m
// Author:          Matteo Morelli, I.R.C. "E. Piaggio", University of Pisa
// Date:            April 2007
//
// $LastChangedDate: 2007-10-05 23:28:34 +0200(ven, 05 ott 2007) $
// Multiply unit-quaternion by unit-quaternion
// decompose into scalar and vector components
    s1 = q1.s;
    v1 = q1.v;
    s2 = q2.s;
    v2 = q2.v;
    // form the product
    qp = rt_quaternion([s1*s2-v1*v2' +s1*v2+s2*v1+rt_cross(v1,v2)'])
endfunction
```

Die Berechnung des Produktes erfolgt dabei nach der bekannten Gleichung [2]
$pq = p_0 q_0 - \mathbf{p}\overline{\mathbf{q}} + \mathbf{p}q_0 + \mathbf{q}p_0 + \mathbf{p} \times \mathbf{q}$.

p und **q** sind also die Vektorteile (v1 und v2) der Quaternionen p und q und $\overline{\mathbf{q}}$ ist die Konjugierte von **q** also v2'. Wenn man q * v programmiert, was äußerlich aussieht wie das Produkt eines Quaternions mit einem Vektor, dann wird mit Hilfe der Overloadtechnik tatsächlich der Drehoperator q *v * q_ realisiert.

Dazu dient die Overloadfunktion %quat_m_s :

```
function [qp] = %quat_m_s(q1, q2)
// File name:       %quat_m_s.sci
// Functions:       %quat_m_s, * (star)
// Description:     multiply quaternion by vector
// Annotations:     this code is a Scilab port of corresponding function in the
//                  Robotics Toolbox for MATLAB(R) written by Peter I. Corke.
// References:      Robotics Toolbox for MATLAB(R), robot7.1/@quaternion/mtimes.m
// Author:          Matteo Morelli, I.R.C. "E. Piaggio", University of Pisa
// Date:            April 2007
// $LastChangedDate: 2007-10-05 23:28:34 +0200(ven, 05 ott 2007) $

   if and(size(q2) == [3 1]) then
        // Multiply vector by unit-quaternion
        qp = q1 * rt_quaternion([0 q2']) * inv(q1);
        qp = qp.v;
    end
endfunction
```

Hier wird also die Quaternion-Multiplikation zweimal angewandt. Dazu wird der Vektor **v** in ein reines Quaternion umgewandelt. Die Funktion gibt schließlich den Vektorteil aus.
Weitere Operationen sind:

q1 / q2	ergibt q1 * (q2)^(-1)
q / s	Division durch einen Skalar
q ^ (j)	Integerpotenz

Weiterhin stehen noch folgende Funktionen bereit:

disp(q)	Ausgabe der Komponenten des Quaternions als Zeile
double(q)	liefert die Koeffizienten als 4-Element-Zeile
inv(q)	gibt die Inverse aus
norm(q)	Berechnet die Norm

rt_plot(q) Zeichnet einen 3D plot, der ein Koordinaten-system nach der Rotation durch q zeigt.
rt_unit(q) Liefert ein Einheitsquaternion

Die Robotic Toolbox muß nach dem Aufruf von Scilab geladen werden. Dazu geht man in das Verzeichnis der Toolbox und führt das Skript loader.sce aus. (loader.sce in den Editor laden und mit *Execute/Load into Scilab* ausführen. Die direkte Ausführung in SciLab über *File/Exec/loader.sce* war nicht erfolgreich.

1.2 Beispiele für die Anwendung der Toolbox

Für eine 45°-Drehung um die z-Achse soll ein Quaternion definiert werden. Dafür dafür benötigt werden der Vektor der Drehachse v = [0 0 1] sowie der Drehwinkel. Alternativ kann auch die homogene Tranformation verwendet werden.

Skript (Editor):

```
//Drehung eines Vektors um die z-
Achse
gamma = 45
gamma_rad = gamma/180*%pi
//z-Achse ist Drehachse
vz = [0 0 1]
//Rotationsmatrix berechnen
c = cos(gamma_rad)
s = sin(gamma_rad)
//Aktive Drehung
Rz = [ c -s  0;
       s  c  0;
       0  0  1]
//Homogene Transformation
Tz = [Rz [0 0 0]'; 0 0 0 1]
//Zwei mögliche Generatoren für q
qz = rt_quaternion(vz, gamma_rad)
qzT = rt_quaternion(Tz)
//Drehung eines Vektor u um die
z-Achse
u = [1; 0; 0]
u1 = qz * u
```

Ausgabe:

```
--> gamma  =
    45.
 gamma_rad  =
    0.7853982
 vz  =
    0.    0.    1.
 c  =
    0.7071068
 s  =
    0.7071068

 Rz  =
    0.7071068  - 0.7071068    0.
    0.7071068    0.7071068    0.
    0.           0.           1.

 qz  =
0.923880 <0.000000, 0.000000, 0.382683>

 qzT  =
0.923880 <0.000000, 0.000000, 0.382683>

 u  =
    1.
    0.
    0.

 u1 =
    0.7071068    0.7071068    0.
```

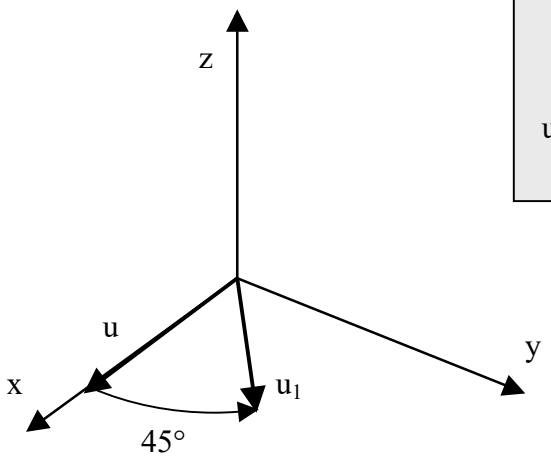

Bild 2: Verdrehter Vektor

Das Ergebnis zeigt einen Vektor u1, der gegen u um +45° in der x-y-Ebene verdreht ist.

Die Drehung kann fortgesetzt werden um weitere Achsen, z. B. um die x-Achse. Man wählt dann v = [1, 0, 0]

```
//weitere Drehung um die x-Achse
alpha_rad = gamma_rad
vx = [1 0 0]
qx = rt_quaternion(vx, alpha_rad)
Rx = qx.r
//
//Drehung von u1 um die x-Achse
u2 = qx * u1'

Rzx = Rx * Rz

// Drehung von u um z- und x-Achse
qzx = qx * qz

Rzx = qzx.r

u2 = qzx*u
```

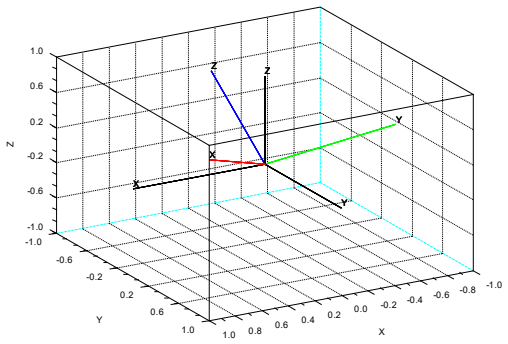

Bild 3: Drehung um z- und x-Achse

```
alpha_rad  =
    0.7853982

 vx  =
    1.    0.    0.

 qx  =
0.923880 <0.382683, 0.000000, 0.000000>

 Rx  =
    1.    0.           0.
    0.    0.7071068  - 0.7071068
    0.    0.7071068    0.7071068

 u2  =
    0.7071068    0.5    0.5

 u2  =
    0.7071068
    0.5
    0.5

 Rzx  =
    0.7071068  - 0.7071068    0.
    0.5          0.5        - 0.7071068
    0.5          0.5          0.7071068

 qzx  =
0.853553 <0.353553, -0.146447, 0.353553>

 Rzx  =
  0.7071068  - 0.7071068   5.551D-17
    0.5          0.5        - 0.7071068
    0.5          0.5          0.7071068

 u2  =
  0.7071068   0.5   0.5
```

Die Graphik zeigt die Darstellung von q_{zx} bzw. R_{zx}.

2 Zusammenfassung

Quaternionen sind vierdimensionale Vektoren mit einem Realteil und drei Imaginärteilen. Der Imaginärteil spannt den dreidimensionalen Raum mit rechtsdrehenden Koordinaten auf. Eine Bedeutung der Quaternionen liegt darin, daß mit Einheitsquaternionen Koordinatentransformationen durchgeführt werden können. Davon wird in der Computergraphik und der Steuerung von Fahrzeugen und Robotern gebrauch gemacht. Der Vorteil liegt in der kleineren Anzahl an Parametern im Vergleich zu Tranformationsmatrizen sowie bei geringerem Aufwand bei der Interpolation zwischen Vektoren. Außerdem haben Quaternionen ein günstigeres Verhalten bei dem sogenannten Gimbal Lock, einer Degeneration der Tranformation bei bestimmten Winkeln [2]. Bei numerischen Berechnungen sind sie daher den Transformationsmatrizen vorzuziehen. Allerdings besteht zwischen einem Einheitsquaternion und einer Transformationsmatrix eine enge Beziehung.

Algemeine Berechnungen mit Quaternionen sind sehr komplex und unübersichtlich und daher für didaktische Aufgabenstellungen überhaupt nicht geeignet. Auf der Anwendungsebene stehen jedoch Programme zur Verfügung, die den Einsatz von Quaternionen ebenso einfach machen, wie den von Transformationsmatrizen. Ein solches Programmpaket ist die *Robotic Toolbox* für *Scilab*.

Die für Quaternionen verfügbaren Funktionen dieser Toolbox wurden analysiert und ihre Anwendung durch Beispiele erläutert. Dabei zeigt sich auch, daß der Übergang von der Transformationsmatrix zum Quaternion und umgekehrt mühelos dargestellt werden kann.

3 Literatur

[1] http://rtss.sourceforge.net
[2] http://www.ekunzeweb.de/PAPERS/Mathematische Grundlagen der Quaternionen.pdf

4 Anhang

In der Robotic Toolbox verfügbare Funktionen:

rt_quaternion	**construct/clone a quaternion object**
rt_q2tr	**convert unit-quaternion to homogeneous transform**
rt_tr2q	**convert homogeneous transform to a unit-quaternion**
rt_qinterp	**interpolate unit-quaternions**
rt_qplot	**display a quaternion as a 3D rotation**
rt_qunit	**unitize a quaternion**
rt_cross	**vector cross product**

rt_quaternion construct/clone a quaternion object

```
function [q] = rt_quaternion(a1, a2)
// File name:       rt_quaternion.sci
//
// Function:        rt_quaternion
//
// Description:     construct/clone a quaternion object
//
// Annotations:     the quaternion data structure is inspired by the one
implemented in the
//                  Robotics Toolbox for MATLAB(R) written by Peter I. Corke.
//
//                  A description of the quaternion data structure (DS) follows.
//
//                  A quaternion is a Scilab mlist with 3 fields. The first
field is the string
//                  vector ["quat"] (the type of DS).
//                  The other fields of the link DS are due to
//
//                      scalar component (field 2)
//                      ==========================
//                          s    (2)
//
//                      vector part (field 3)
//                      =====================
```

```
//                          v   (3)
//
// References:      Robotics Toolbox for MATLAB(R),
robot7.1/@quaternion/quaternion.m
//
// Author:          Matteo Morelli, I.R.C. "E. Piaggio", University of Pisa
//
// Date:            April 2007
//
// $LastChangedDate: 2008-03-26 19:09:34 +0100(mer, 26 mar 2008) $

    [%nargout, %nargin] = argn(0);

    if %nargin > 2 then
        error(77);  // wrong number of rhs arguments
    end

    if %nargin == 0 then

        // create a default quaternion
        s = [];
        v = [];
        q = mlist(["quat"], s, v);

    elseif typeof(a1) == "quat" then

        // clone passed quaternion
        q = a1;

    elseif %nargin == 1 then

        // create quaternion from a 3x3 or 4x4 matrix or from 4-elements row
vector
        if and(size(a1) == [3 3]) then
            q = rt_quaternion(rt_tr2q(a1));
        elseif and(size(a1) == [4 4]) then
            q = rt_quaternion(rt_tr2q(a1(1:3,1:3)));
        elseif and(size(a1) == [1 4]) then
            s = a1(1);
      v = a1(2:4);
            q = mlist(["quat"], s, v);
        else
            error("unknown dimension of input");
        end

    elseif %nargin == 2 then

        // create a quaternion from vector plus angle
        q = rt_unit(rt_quaternion([cos(a2/2) sin(a2/2)*rt_unit(a1(:).')]));

    end

endfunction
```

rt_q2tr convert unit-quaternion to homogeneous transform

```
function [t] = rt_q2tr(q)
// File name:       rt_q2tr.sci
//
// Function:        rt_q2tr
```

```
//
// Description:     convert unit-quaternion to homogeneous transform
//
// Annotations:     this code is a Scilab port of corresponding function in the
//                  Robotics Toolbox for MATLAB(R) written by Peter I. Corke.
//
// References:      Robotics Toolbox for MATLAB(R),
robot7.1/@quaternion/subsref.m
//
// Author:          Matteo Morelli, I.R.C. "E. Piaggio", University of Pisa
//
// Date:            April 2007
//
// $LastChangedDate: 2007-10-05 23:28:34 +0200(ven, 05 ott 2007) $

    q = double(q);
    s = q(1);
    x = q(2);
    y = q(3);
    z = q(4);

    r = [1-2*(y^2+z^2) 2*(x*y-s*z) 2*(x*z+s*y); 2*(x*y+s*z) 1-2*(x^2+z^2)
2*(y*z-s*x); 2*(x*z-s*y) 2*(y*z+s*x) 1-2*(x^2+y^2)];
    t = eye(4,4);
    t(1:3,1:3) = r;
    t(4,4) = 1;

endfunction
```

rt_tr2q convert homogeneous transform to a unit-quaternion

```
function [q] = rt_tr2q(t)
// File name:       rt_tr2q.sci
//
// Function:        rt_tr2q
//
// Description:     convert homogeneous transform to a unit-quaternion
//
// Annotations:     this code is a Scilab port of corresponding function in the
//                  Robotics Toolbox for MATLAB(R) written by Peter I. Corke.
//
// References:      Robotics Toolbox for MATLAB(R),
robot7.1/@quaternion/quaternion.m
//
// Author:          Matteo Morelli, I.R.C. "E. Piaggio", University of Pisa
//
// Date:            April 2007
//
// $LastChangedDate: 2007-10-05 23:28:34 +0200(ven, 05 ott 2007) $

    qs = sqrt(trace(t)+1)/2.0;
    kx = t(3,2) - t(2,3);                           // Oz - Ay
    ky = t(1,3) - t(3,1);                           // Ax - Nz
    kz = t(2,1) - t(1,2);                           // Ny - Ox

    if (t(1,1) >= t(2,2)) & (t(1,1) >= t(3,3)) then
        kx1 = t(1,1) - t(2,2) - t(3,3) + 1;         // Nx - Oy - Az + 1
        ky1 = t(2,1) + t(1,2);                      // Ny + Ox
        kz1 = t(3,1) + t(1,3);                      // Nz + Ax
        add = (kx >= 0);
```

```
    elseif (t(2,2) >= t(3,3)) then
        kx1 = t(2,1) + t(1,2);                      // Ny + Ox
        ky1 = t(2,2) - t(1,1) - t(3,3) + 1;         // Oy - Nx - Az + 1
        kz1 = t(3,2) + t(2,3);                      // Oz + Ay
        add = (ky >= 0);
    else
        kx1 = t(3,1) + t(1,3);                      // Nz + Ax
        ky1 = t(3,2) + t(2,3);                      // Oz + Ay
        kz1 = t(3,3) - t(1,1) - t(2,2) + 1;         // Az - Nx - Oy + 1
        add = (kz >= 0);
    end

    if add then
        kx = kx + kx1;
        ky = ky + ky1;
        kz = kz + kz1;
    else
        kx = kx - kx1;
        ky = ky - ky1;
        kz = kz - kz1;
    end

    nm = norm([kx ky kz]);
    if nm == 0 then
        q = rt_quaternion([1 0 0 0]);
    else
        s = sqrt(1 - qs^2) / nm;
        qv = s*[kx ky kz];

        q = rt_quaternion([qs qv]);

    end

endfunction
```

rt_qinterp interpolate unit-quaternions

```
function q = rt_qinterp(Q1, Q2, r)
// File name:       rt_qinterp.sci
//
// Function:        rt_qinterp
//
// Description:     interpolate unit-quaternions
//
// Annotations:     this code is a Scilab port of corresponding function in the
//                  Robotics Toolbox for MATLAB(R) written by Peter I. Corke.
//
// References:      Robotics Toolbox for MATLAB(R),
robot7.1/@quaternion/qinterp.m
//
// Author:          Matteo Morelli, I.R.C. "E. Piaggio", University of Pisa
//
// Date:            April 2007
//
// $LastChangedDate: 2007-10-05 23:28:34 +0200(ven, 05 ott 2007) $

    if or(r < 0 | r > 1) then
        error("R out of range");
    end
```

```
    q1 = double(Q1);
    q2 = double(Q2);
    theta = acos(q1*q2.');
    q = list();
    count = 1;

    if length(r) == 1 then

        // r is a scalar, returns a unit quaternion
        if theta == 0 then
            q = Q1;
        else
            q = rt_quaternion( (sin((1-r)*theta)*q1 + sin(r*theta)*q2) /
sin(theta) );
        end

    else

        // r is a vector, returns a list of quaternions
        for R = r(:).',
            if theta == 0 then
                qq = Q1;
            else
                qq = rt_quaternion( (sin((1-R)*theta)*q1 + sin(R*theta)*q2) /
sin(theta) );
            end
            q(count) = qq;
            count = count + 1;
        end

    end

endfunction
```

rt_qplot display a quaternion as a 3D rotation

```
function [res] = rt_plot(obj, varargin)
// File name:       rt_plot.sci
//
// Function:      rt_plot
//
// Description:     wrapper function for plotting robot or quaternion object
//
// Annotations:     none
//
// References:      none
//
// Author:          Matteo Morelli, I.R.C. "E. Piaggio", University of Pisa
//
// Date:            April 2007
//
// $LastChangedDate: 2007-10-05 23:28:34 +0200(ven, 05 ott 2007) $

    if typeof(obj) == "robot" then
        res = rt_rplot(obj,varargin);
        return
    end

    if typeof(obj) == "quat" then
        rt_qplot(obj,varargin);
```

```
        res = [];
        return
    end

endfunction
```

rt_qunit unitize a quaternion

```
function [qu] = rt_qunit(q)
// File name:       rt_qunit.sci
//
// Function:        rt_qunit
//
// Description:     unitize a quaternion
//
// Annotations:     this code is a Scilab port of corresponding function in the
//                  Robotics Toolbox for MATLAB(R) written by Peter I. Corke.
//
// References:      Robotics Toolbox for MATLAB(R), robot7.1/@quaternion/unit.m
//
// Author:          Matteo Morelli, I.R.C. "E. Piaggio", University of Pisa
//
// Date:            April 2007
//
// $LastChangedDate: 2007-10-05 23:28:34 +0200(ven, 05 ott 2007) $

    qu = q / norm(q);

endfunction
```

rt_cross vector cross product

```
function [r] = rt_cross(v, w)
// File name:       rt_cross.sci
//
// Function:        rt_cross
//
// Description:     vector cross product
//
// Annotations:     Scilab equivalent for MATLAB(R) function cross is missing.
//                  This code implements a simple emulator of that function.
//                  Extremely simple, input arguments can be only 3-element
vectors.
//
// References:      none
//
// Author:          Matteo Morelli, I.R.C. "E. Piaggio", University of Pisa
//
// Date:            April 2007
//
// $LastChangedDate: 2007-10-05 23:28:34 +0200(ven, 05 ott 2007) $

    [mv, nv] = size(v);
    [mw, nw] = size(w);
    if mv*nv ~= 3 | mw*nw ~= 3 then
        error("cross product can be done only for 3-element vectors");
    end
```

```
    r = [v(2)*w(3) - v(3)*w(2); v(3)*w(1) - v(1)*w(3); v(1)*w(2) - v(2)*w(1)];

endfunction
```

4 **Interne Funktionen**

%quat_e	extract data from quaternion object
%quat_i_st	insert a quaternion object in a Scilab structure
%quat_iconvert	convert a quaternion object to a 4-element vector
%quat_inv	invert a quaternion
%quat_m_quat	multiply quaternion by quaternion
%quat_m_s	multiply quaternion by vector
%quat_n_quat	inequality comparison for quaternion objects
%quat_norm	norm of a quaternion
%quat_o_quat	equality comparison for quaternion objects
%quat_p	displays the quaternion as one-line summary of its coefficients
%quat_p_s	raise quaternion to integer power
%quat_r_quat	divide quaternion by quaternion
%quat_r_s	divide quaternion by scalar
%quat_size	size of quat objects