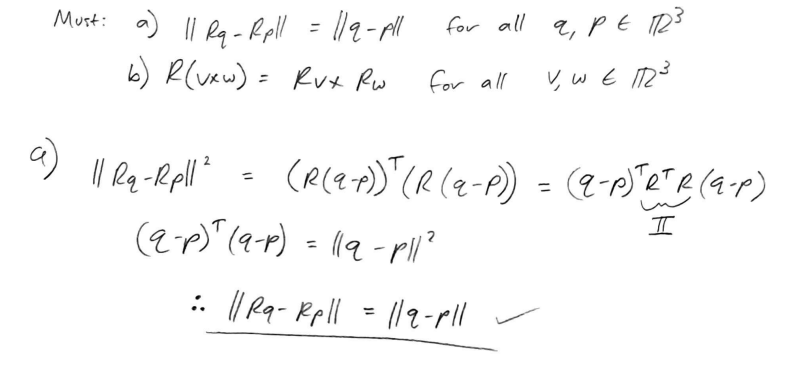
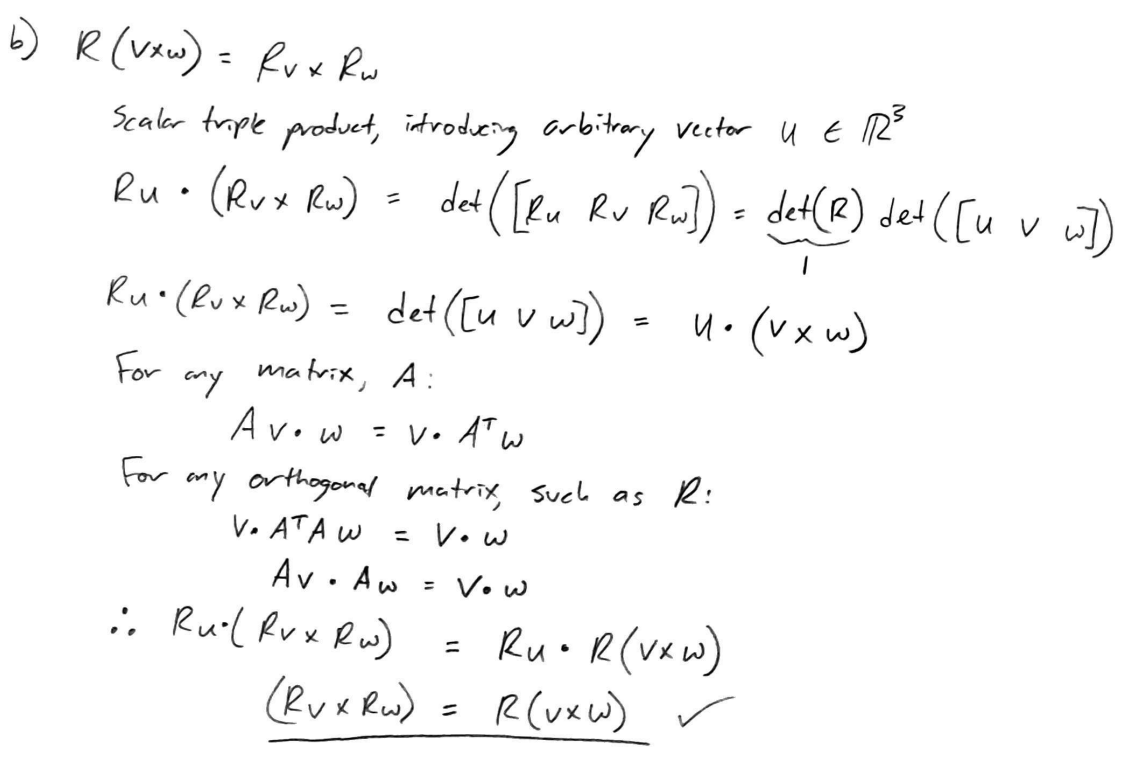
**HOMEWORK ASSIGNMENT (Q1-9)**

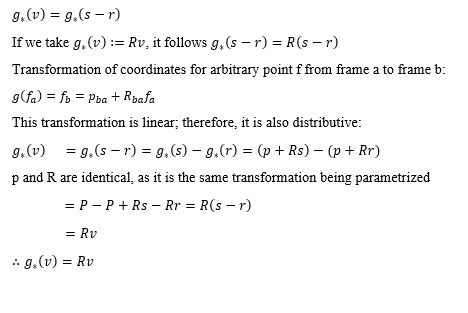
Jared Rosenbaum & Steven Swanbeck

1. Prove that a rotation matrix is a rigid body transformation.





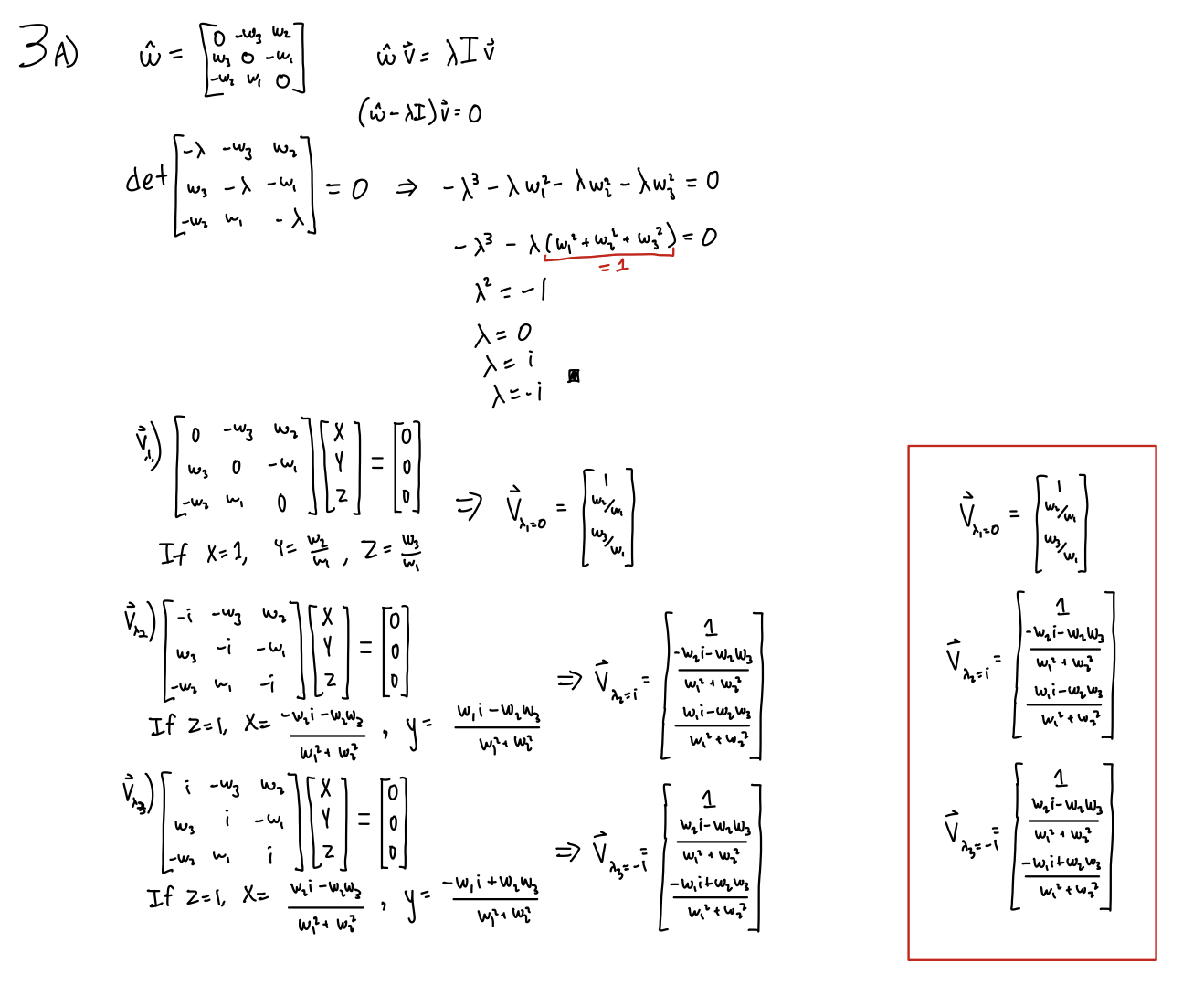
1. Prove that rigid transformation g = (p,R) on a vector v = s - ris .



1. **(MLS Book [1])** *Properties of rotation matrices*

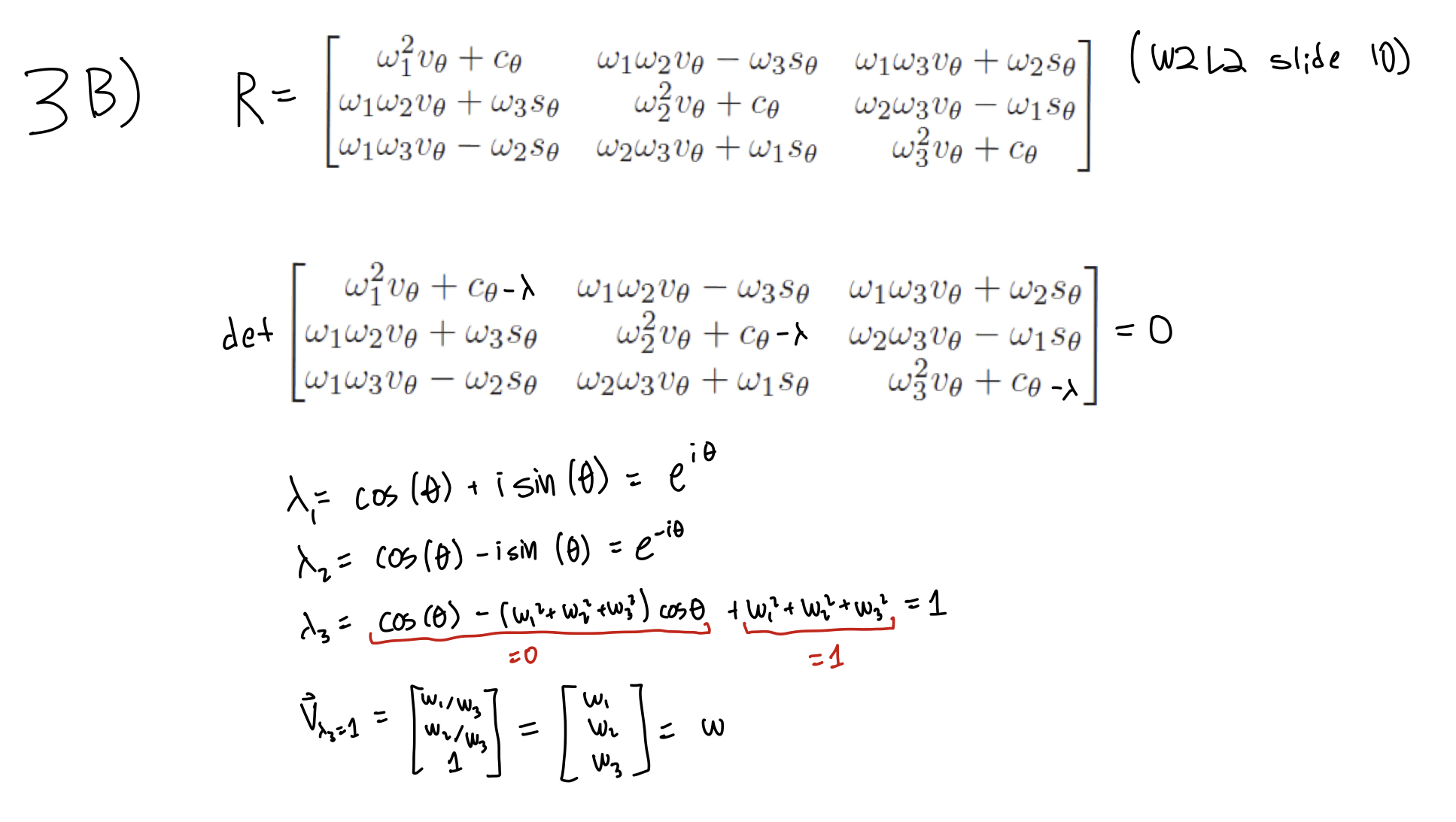
Let 𝑅 ∈ 𝑆𝑂(3) be a rotation matrix generated by rotating about a unit vector ω by θ radians. That is, R satisfies 𝑅 = exp(𝜔̂𝜃).

1. Show that the eigenvalues of 𝜔̂ are 0, i, and −i, where i = √−1. What are the corresponding eigenvectors?



1. Show that the eigenvalues of R are 1, 𝑒 𝑖𝜃, and 𝑒 −𝑖𝜃. What is the eigenvector whose eigenvalue is 1? What is the physical interpretation of this eigenvector?

*\*

**

Eigenvalues and vectors solved for with the following MATLAB script:

syms w1 w2 w3 th L;

ct = cos(th);

st = sin(th);

vt = 1-cos(th);

R = [w1^2\*vt+ct, w1\*w2\*vt-w3\*st, w1\*w3\*vt+w2\*st;

w1\*w2\*vt+w3\*st, w2^2\*vt+ct, w2\*w3\*vt-w1\*st;

w1\*w3\*vt-w2\*st, w2\*w3\*vt+w1\*st, w3^2\*vt+ct];

% simplify(det(R));

[V,D] = eig(R)

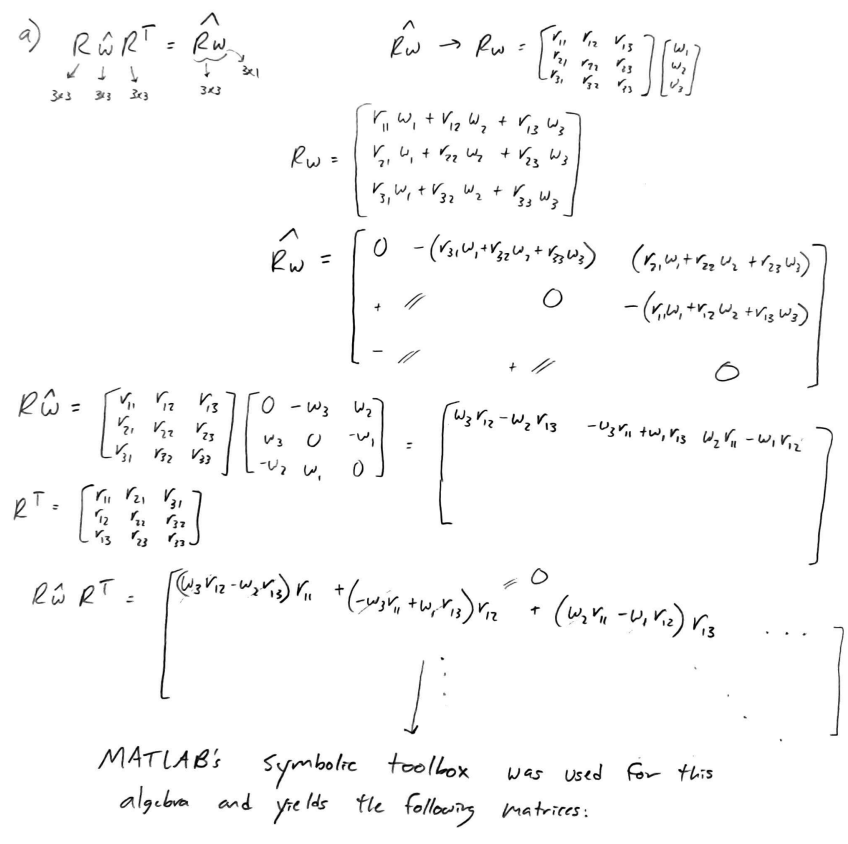
The physical interpretation of the Lambda=1 eigenvector of R is the axis of rotation; by definition,

When =1, , meaning the application of the rotation matrix on v has no result - v lies on the axis of rotation.

1. **(MLS Book [1])** *Properties of skew-symmetric matrices*

Show That the following properties of skew- symmetric matrices are true:

1. If 𝑅 ∈ 𝑆𝑂(3) and 𝜔 ∈ ℝ3 , then 𝑅𝜔̂𝑅 𝑇 = 𝑅𝜔̂ .
2. If 𝑅 ∈ 𝑆𝑂(3) and 𝑣, 𝜔 ∈ ℝ3 , then 𝑅(𝜈 × 𝜔) = (𝑅𝜈) × (𝑅𝜔).



MATLAB Code used to calculate matrix multiplications:

—----------------------------------------------------------------------------------------------------------------------------

syms w1 w2 w3 r11 r12 r13 r21 r22 r23 r31 r32 r33

R = [r11 r12 r13; r21 r22 r23; r31 r32 r33];

w = [w1; w2; w3];

R\_w = R \* w;

R\_w\_hat = [0 -1\*R\_w(3) R\_w(2); R\_w(3) 0 -1\*R\_w(1); -1\*R\_w(2) R\_w(1) 0]

w\_hat = [0 -1\*w3 w2; w3 0 -1\*w1; -1\*w2 w1 0];

R\*w\_hat\*transpose(R);

R\_w\_hat\_R\_T = simplify(R\*w\_hat\*transpose(R))

—----------------------------------------------------------------------------------------------------------------------------

Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------

R\_w\_hat =

[ 0, - r31\*w1 - r32\*w2 - r33\*w3, r21\*w1 + r22\*w2 + r23\*w3]

[ r31\*w1 + r32\*w2 + r33\*w3, 0, - r11\*w1 - r12\*w2 - r13\*w3]

[- r21\*w1 - r22\*w2 - r23\*w3, r11\*w1 + r12\*w2 + r13\*w3, 0]

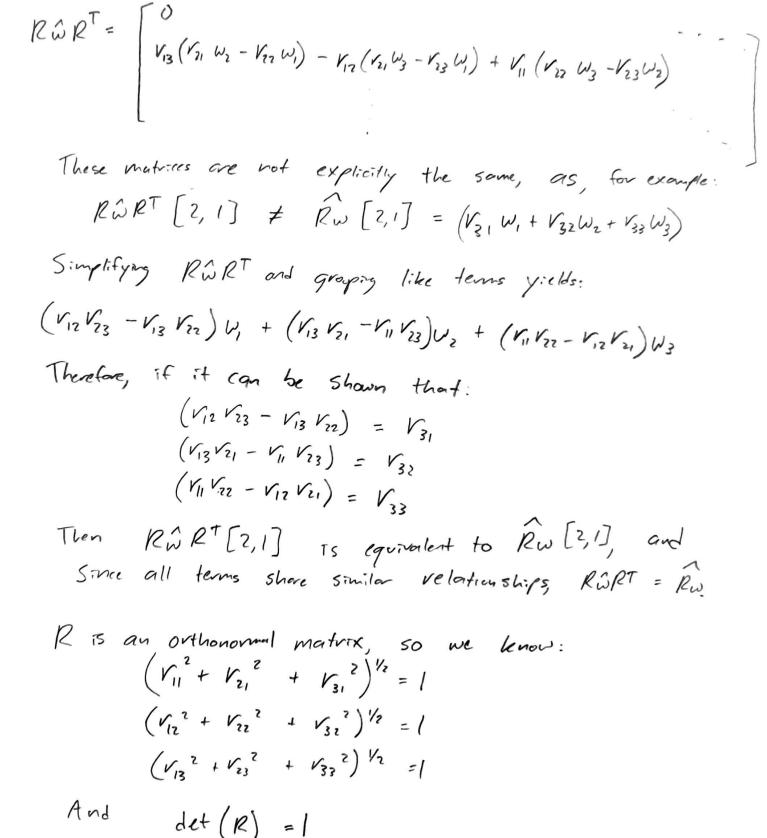
R\_w\_hat\_R\_T =

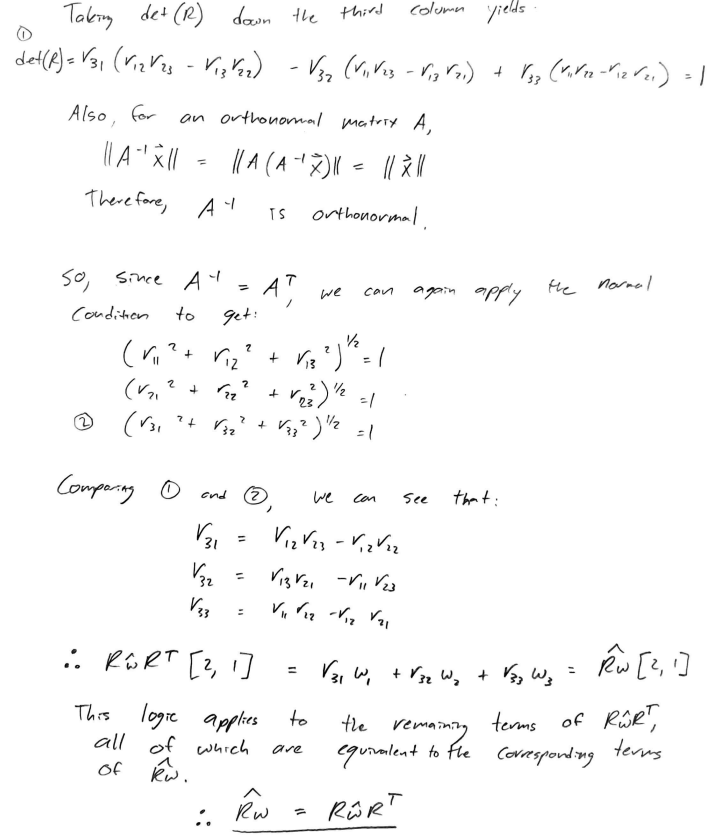
[ 0, r23\*(r11\*w2 - r12\*w1) - r22\*(r11\*w3 - r13\*w1) + r21\*(r12\*w3 - r13\*w2), r33\*(r11\*w2 - r12\*w1) - r32\*(r11\*w3 - r13\*w1) + r31\*(r12\*w3 - r13\*w2)]

[r13\*(r21\*w2 - r22\*w1) - r12\*(r21\*w3 - r23\*w1) + r11\*(r22\*w3 - r23\*w2), 0, r33\*(r21\*w2 - r22\*w1) - r32\*(r21\*w3 - r23\*w1) + r31\*(r22\*w3 - r23\*w2)]

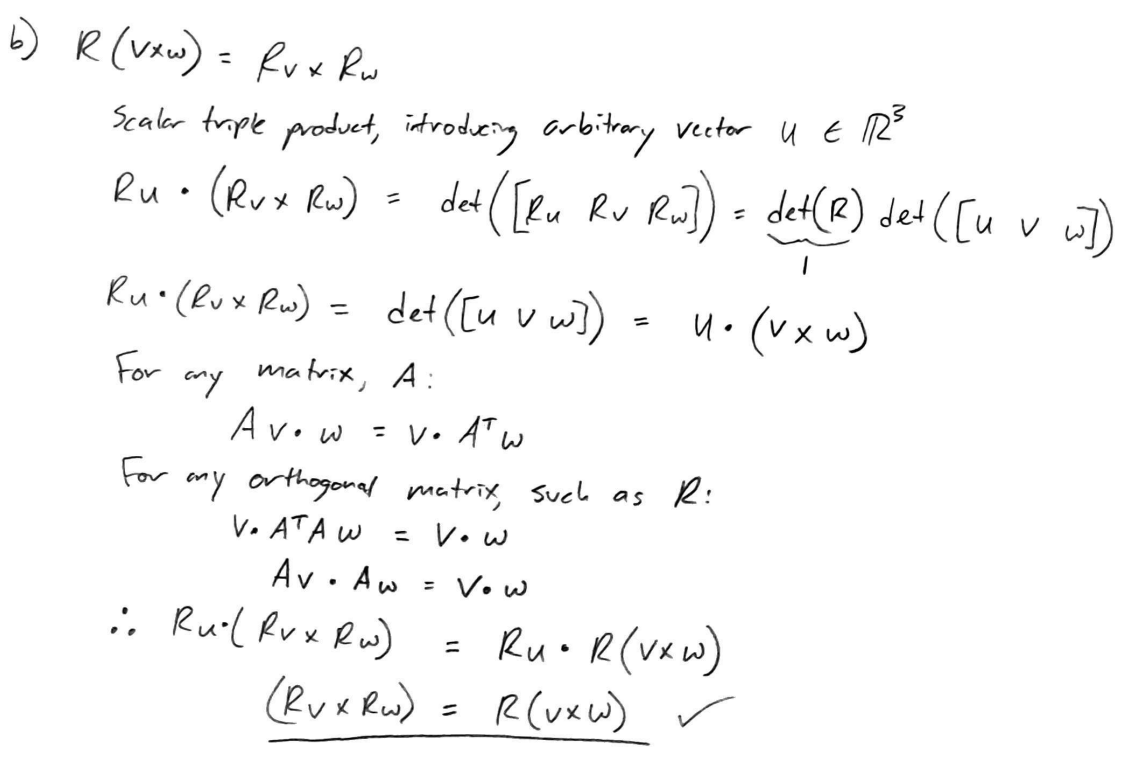
[r13\*(r31\*w2 - r32\*w1) - r12\*(r31\*w3 - r33\*w1) + r11\*(r32\*w3 - r33\*w2), r23\*(r31\*w2 - r32\*w1) - r22\*(r31\*w3 - r33\*w1) + r21\*(r32\*w3 - r33\*w2), 0]

—----------------------------------------------------------------------------------------------------------------------------





b) (As taken from Problem 1 b))



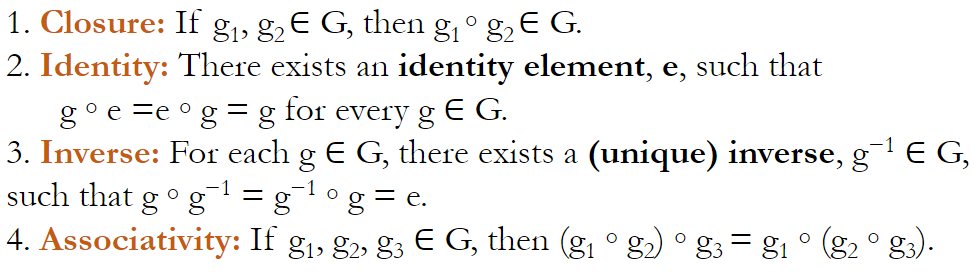
1. A
2. A
3. A
4. **(MLS Book [1])** *Unit quaternions*

Let 𝑄 = (𝑞𝑜, 𝑞⃑) and 𝑃 = (𝑝𝑜, 𝑝⃑) be quaternions, where 𝑞𝑜, 𝑝𝑜 ∈ ℝ are the scalar parts of Q and P and 𝑞⃑, 𝑝⃑ are the vector parts.

* 1. Show that the set of unit quaternions satisfies the axioms of the group
  2. Let x be a point and let X be a quaternion whose scalar part is zero and whose vector part is equal to x (such a quaternion is called a pure quaternion). Show that if Q is a unit quaternion, the product QXQ\* is a pure quaternion and the vector part of QXQ\* satisfies

( − 𝑞⃑ ∙ 𝑞⃑)𝑥⃑ + 2(( × 𝑥⃑) + (𝑥 ∙ 𝑞⃑)𝑞⃑

**a)**



**Using multiplication () as the binary operator:**

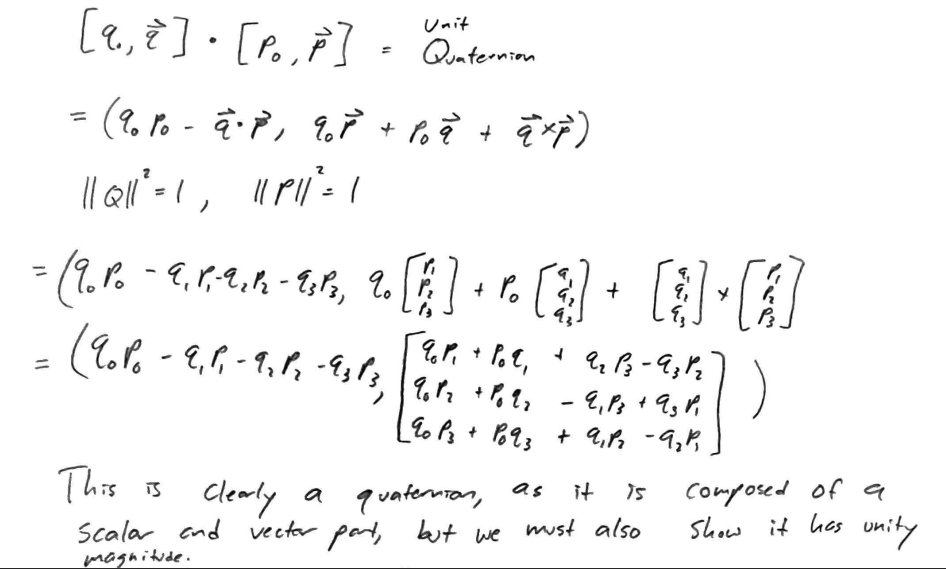
1. Closure is proved:

QP = (---,

++-,

-++,

+-+)



MATLAB Code used to check resultant magnitude:

—----------------------------------------------------------------------------------------------------------------------------

syms q0 q1 q2 q3 p0 p1 p2 p3;

c0 = q0\*p0 - q1\*p1 - q2\*p2 - q3\*p3;

c1 = q0\*p1 + p0\*q1 + q2\*p3 - q3\*p2;

c2 = q0\*p2 + p0\*q2 - q1\*p3 + q3\*p1;

c3 = q0\*p3 + p0\*q3 + q1\*p2 - q2\*p1;

simplify(expand(c0^2 + c1^2 + c2^2 + c3^2))

—----------------------------------------------------------------------------------------------------------------------------

Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------

ans =

(p0^2 + p1^2 + p2^2 + p3^2)\*(q0^2 + q1^2 + q2^2 + q3^2)

—----------------------------------------------------------------------------------------------------------------------------



Because QP takes the form of a quaternion [q0, q^3x1] and has unity magnitude when both Q and P are unit quaternions, the closure axiom is satisfied for unit quaternions under quaternion multiplication.

1. The identity element is (1, 0, 0, 0), as proved below.

QI = (---,

++-,

-++,

+-+) =

1. The unique inverse is defined as , where ||Q|| = 1 &

Q = ,

= (+++,

--+,

-++-,

--++) = (, 0, 0, 0) = (1, 0, 0, 0) = I

1. Associativity of unit quaternions is proved below.

QP = (---,

++-,

-++,

+-+)

PW = (---,

++-,

-++,

+-+)

(QP)W = [(---)-(++-)-(-++)-(+-+),

(++-)+(---)+(-++)-(+-+),

(---)-(++-)+(-++)+(+-+),

(---)+(++-)-(-++)+(+-+)]

Q(PW) = [---,

++-,

-++,

+-+]

Expanding the above vectors, we find (QP)W=Q(PW). This math was done symbolically using MATLAB.

MATLAB Code used to calculate and compare :

—----------------------------------------------------------------------------------------------------------------------------

q0 = sym('q0','real');

q1 = sym('q1','real');

q2 = sym('q2','real');

q3 = sym('q3','real');

p0 = sym('p0','real');

p1 = sym('p1','real');

p2 = sym('p2','real');

p3 = sym('p3','real');

w0 = sym('w0','real');

w1 = sym('w1','real');

w2 = sym('w2','real');

w3 = sym('w3','real');

Q = [q0 q1 q2 q3];

P = [p0 p1 p2 p3];

W = [w0 w1 w2 w3];

PW = quatmultiply(P, W);

Q\_PW = quatmultiply(Q, PW);

test1 = simplify(expand(Q\_PW))

QP = quatmultiply(Q, P);

QP\_W = quatmultiply(QP, W);

test2 = simplify(expand(QP\_W))

tf = zeros(4,1,'logical');

for i = 1:4

tf(i) = isequal(test1(i), test2(i));

end

tf

—----------------------------------------------------------------------------------------------------------------------------

Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------

test1 =

[p0\*q0\*w0 - p0\*q1\*w1 - p1\*q0\*w1 - p1\*q1\*w0 - p0\*q2\*w2 - p2\*q0\*w2 - p2\*q2\*w0 - p0\*q3\*w3 + p1\*q2\*w3 - p1\*q3\*w2 - p2\*q1\*w3 + p2\*q3\*w1 - p3\*q0\*w3 + p3\*q1\*w2 - p3\*q2\*w1 - p3\*q3\*w0, p0\*q0\*w1 + p0\*q1\*w0 + p1\*q0\*w0 - p1\*q1\*w1 + p0\*q2\*w3 - p0\*q3\*w2 + p1\*q2\*w2 + p2\*q0\*w3 - p2\*q1\*w2 - p2\*q2\*w1 - p2\*q3\*w0 - p3\*q0\*w2 + p3\*q2\*w0 + p1\*q3\*w3 - p3\*q1\*w3 - p3\*q3\*w1, p0\*q0\*w2 + p0\*q2\*w0 + p2\*q0\*w0 - p0\*q1\*w3 + p0\*q3\*w1 - p1\*q0\*w3 - p1\*q1\*w2 - p1\*q2\*w1 + p1\*q3\*w0 + p2\*q1\*w1 + p3\*q0\*w1 - p3\*q1\*w0 - p2\*q2\*w2 + p2\*q3\*w3 - p3\*q2\*w3 - p3\*q3\*w2, p0\*q0\*w3 + p0\*q1\*w2 - p0\*q2\*w1 + p0\*q3\*w0 + p1\*q0\*w2 - p1\*q2\*w0 - p2\*q0\*w1 + p2\*q1\*w0 + p3\*q0\*w0 - p1\*q1\*w3 - p1\*q3\*w1 + p3\*q1\*w1 - p2\*q2\*w3 - p2\*q3\*w2 + p3\*q2\*w2 - p3\*q3\*w3]

test2 =

[p0\*q0\*w0 - p0\*q1\*w1 - p1\*q0\*w1 - p1\*q1\*w0 - p0\*q2\*w2 - p2\*q0\*w2 - p2\*q2\*w0 - p0\*q3\*w3 + p1\*q2\*w3 - p1\*q3\*w2 - p2\*q1\*w3 + p2\*q3\*w1 - p3\*q0\*w3 + p3\*q1\*w2 - p3\*q2\*w1 - p3\*q3\*w0, p0\*q0\*w1 + p0\*q1\*w0 + p1\*q0\*w0 - p1\*q1\*w1 + p0\*q2\*w3 - p0\*q3\*w2 + p1\*q2\*w2 + p2\*q0\*w3 - p2\*q1\*w2 - p2\*q2\*w1 - p2\*q3\*w0 - p3\*q0\*w2 + p3\*q2\*w0 + p1\*q3\*w3 - p3\*q1\*w3 - p3\*q3\*w1, p0\*q0\*w2 + p0\*q2\*w0 + p2\*q0\*w0 - p0\*q1\*w3 + p0\*q3\*w1 - p1\*q0\*w3 - p1\*q1\*w2 - p1\*q2\*w1 + p1\*q3\*w0 + p2\*q1\*w1 + p3\*q0\*w1 - p3\*q1\*w0 - p2\*q2\*w2 + p2\*q3\*w3 - p3\*q2\*w3 - p3\*q3\*w2, p0\*q0\*w3 + p0\*q1\*w2 - p0\*q2\*w1 + p0\*q3\*w0 + p1\*q0\*w2 - p1\*q2\*w0 - p2\*q0\*w1 + p2\*q1\*w0 + p3\*q0\*w0 - p1\*q1\*w3 - p1\*q3\*w1 + p3\*q1\*w1 - p2\*q2\*w3 - p2\*q3\*w2 + p3\*q2\*w2 - p3\*q3\*w3]

tf =

4×1 logical array

1

1

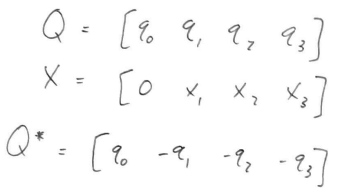
1

1

—----------------------------------------------------------------------------------------------------------------------------

Each of the logical comparisons between components of the resulting quaternions returns true, meaning they are equivalent and the associativity axiom is satisfied for unit quaternions under quaternion multiplication.

**b)**



MATLAB’s symbolic toolbox was again used to perform the calculation of QXQ\*, calculation of ( − 𝑞⃑ ∙ 𝑞⃑)𝑥⃑ + 2(( × 𝑥⃑) + (𝑥 ∙ 𝑞⃑)𝑞⃑, and compare the results :

MATLAB Code used to calculate QXQ\* and ( − 𝑞⃑ ∙ 𝑞⃑)𝑥⃑ + 2(( × 𝑥⃑) + (𝑥 ∙ 𝑞⃑)𝑞⃑ and compare results :

—----------------------------------------------------------------------------------------------------------------------------

q0 = sym('q0','real');

q1 = sym('q1','real');

q2 = sym('q2','real');

q3 = sym('q3','real');

x1 = sym('x1','real');

x2 = sym('x2','real');

x3 = sym('x3','real');

Q = [q0 q1 q2 q3];

X = [0 x1 x2 x3];

q = [q1 q2 q3]';

x = [x1 x2 x3]';

Q\_star = [q0 -1\*q'];

m1 = quatmultiply(Q, X);

m2 = quatmultiply(m1, Q\_star);

test1 = simplify(expand(m2))

% show vector part equals formula

vector\_part = (q0^2 - dot(q, q))\*x + 2\*(q0\*cross(q, x) + dot(x, q)\*q);

test2 = simplify(expand(vector\_part))

tf = zeros(3,1,'logical');

for i = 1:3

tf(i) = isequal(test1(i + 1), test2(i));

end

tf

—----------------------------------------------------------------------------------------------------------------------------

Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------

test1 =

[0, x1\*q0^2 + 2\*x3\*q0\*q2 - 2\*x2\*q0\*q3 + x1\*q1^2 + 2\*x2\*q1\*q2 + 2\*x3\*q1\*q3 - x1\*q2^2 - x1\*q3^2, x2\*q0^2 - 2\*x3\*q0\*q1 + 2\*x1\*q0\*q3 - x2\*q1^2 + 2\*x1\*q1\*q2 + x2\*q2^2 + 2\*x3\*q2\*q3 - x2\*q3^2, x3\*q0^2 + 2\*x2\*q0\*q1 - 2\*x1\*q0\*q2 - x3\*q1^2 + 2\*x1\*q1\*q3 - x3\*q2^2 + 2\*x2\*q2\*q3 + x3\*q3^2]

test2 =

x1\*q0^2 + 2\*x3\*q0\*q2 - 2\*x2\*q0\*q3 + x1\*q1^2 + 2\*x2\*q1\*q2 + 2\*x3\*q1\*q3 - x1\*q2^2 - x1\*q3^2

x2\*q0^2 - 2\*x3\*q0\*q1 + 2\*x1\*q0\*q3 - x2\*q1^2 + 2\*x1\*q1\*q2 + x2\*q2^2 + 2\*x3\*q2\*q3 - x2\*q3^2

x3\*q0^2 + 2\*x2\*q0\*q1 - 2\*x1\*q0\*q2 - x3\*q1^2 + 2\*x1\*q1\*q3 - x3\*q2^2 + 2\*x2\*q2\*q3 + x3\*q3^2

tf =

3×1 logical array

1

1

1

—----------------------------------------------------------------------------------------------------------------------------

The calculation of QXQ\* indeed yields a pure quaternion with zero scalar component. Calculating the vector result of the provided formula and comparing it elementwise to the vector component of the calculated quaternion shows that each corresponding element is equivalent. Therefore, the product of QXQ\* by quaternion multiplication is a pure quaternion and its vector component is equivalent to ( − 𝑞⃑ ∙ 𝑞⃑)𝑥⃑ + 2(( × 𝑥⃑) + (𝑥 ∙ 𝑞⃑)𝑞⃑.

1. Recall from notes or the book that the explicit matrix representations for the rigid body rotations:

* (𝜙) ∶ ℝ ⟼ 𝑆𝑂(3) corresponding to a rotation of 𝜙 radians about the x-axis — i.e. a 3 × 3 matrix whose elements contain expressions such as sin(𝜙) and cos(𝜙)
* (𝜃) ∶ ℝ ⟼ 𝑆𝑂(3) corresponding to a rotation of 𝜃 radians about the y-axis.
* (𝜓) ∶ ℝ ⟼ 𝑆𝑂(3) corresponding to a rotation of 𝜓 radians about the z-axis.

In the class, we have learned that the ZXZ and ZYZ are the most common (the explicit form of the ZYZ Euler angles is in MLS). Here, construct the explicit representation for 𝑅𝑥𝑦𝑧(𝜓) ∶ ℝ ⟼ 𝑆𝑂(3) given by

Construct the inverse function ∶ 𝑆𝑂(3) ⟼ such that ∀𝑅 ∈ 𝑆𝑂(3) if 𝑦 = (𝑅) then 𝑅 = (𝑦), i.e., derive the formulas such as in (2.20) in MLS.

MATLAB Code used to calculate :

—----------------------------------------------------------------------------------------------------------------------------

syms phi theta psi;

Rx = [1 0 0;

0 cos(phi) -sin(phi);

0 sin(phi) cos(phi)];

Ry = [cos(theta) 0 sin(theta);

0 1 0;

-sin(theta) 0 cos(theta)];

Rz = [cos(psi) -sin(psi) 0;

sin(psi) cos(psi) 0;

0 0 1];

R = Rx \* Ry \* Rz

—----------------------------------------------------------------------------------------------------------------------------

Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------

R =

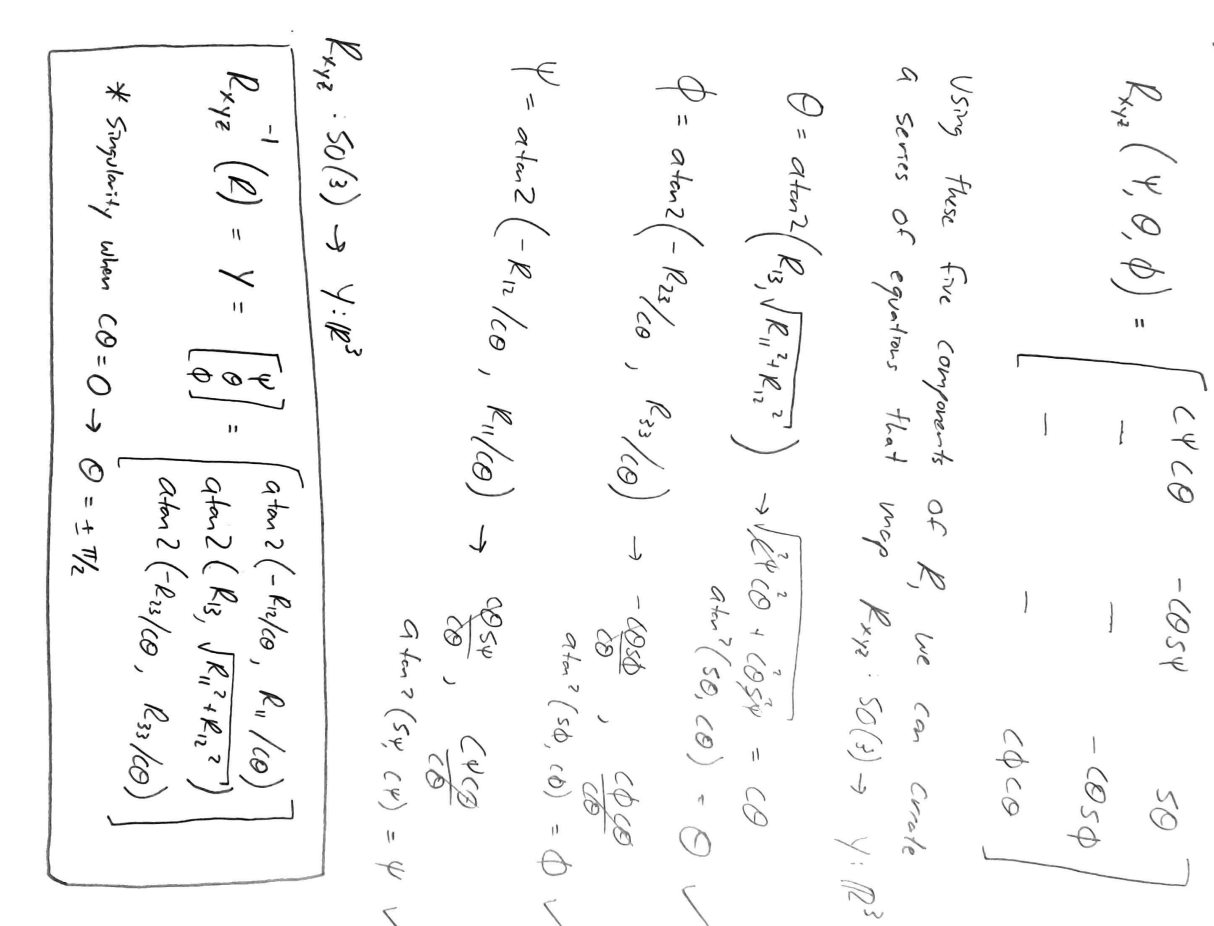
[cos(psi)\*cos(theta), -cos(theta)\*sin(psi), sin(theta) ]

[cos(phi)\*sin(psi) + cos(psi)\*sin(phi)\*sin(theta), cos(phi)\*cos(psi) - sin(phi)\*sin(psi)\*sin(theta), -cos(theta)\*sin(phi)]

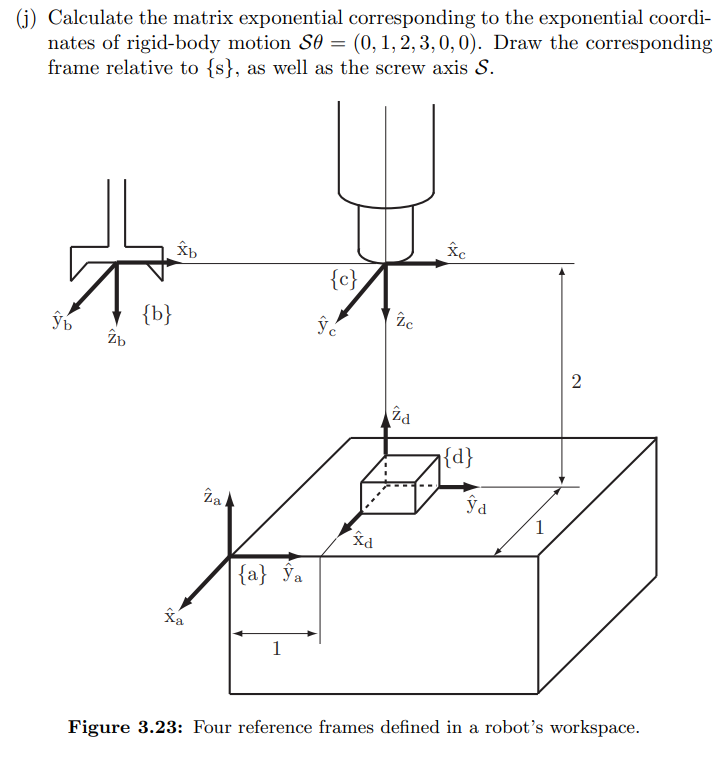
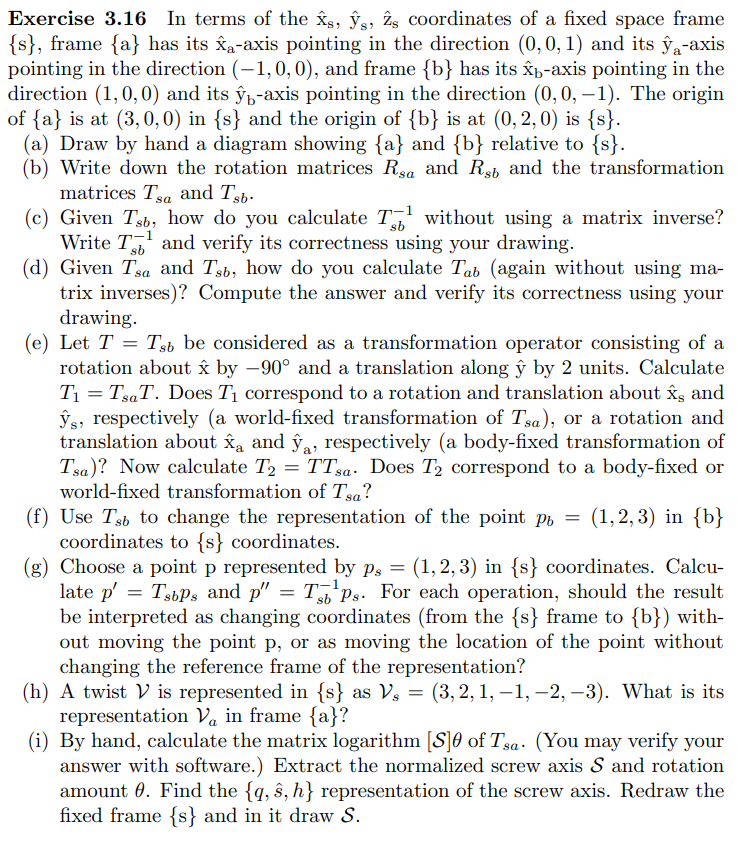
[sin(phi)\*sin(psi) - cos(phi)\*cos(psi)\*sin(theta), cos(psi)\*sin(phi) + cos(phi)\*sin(psi)\*sin(theta), cos(phi)\*cos(theta)]

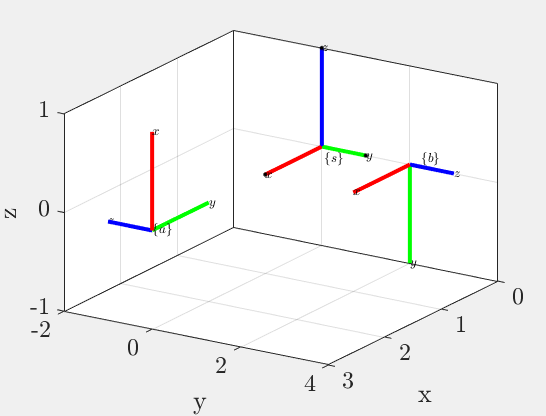
—----------------------------------------------------------------------------------------------------------------------------

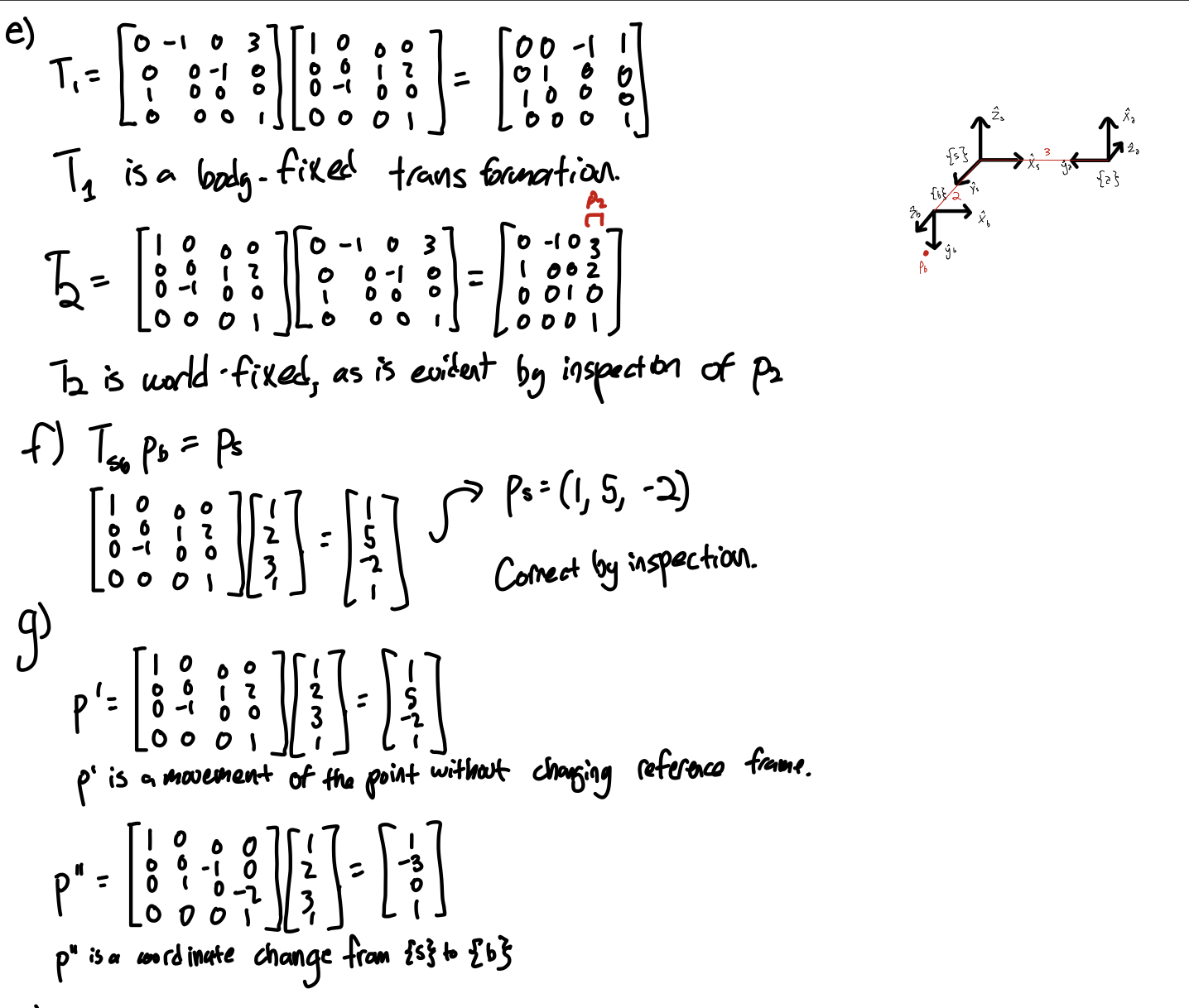
Calculation of inverse function ∶ 𝑆𝑂(3) ⟼ :

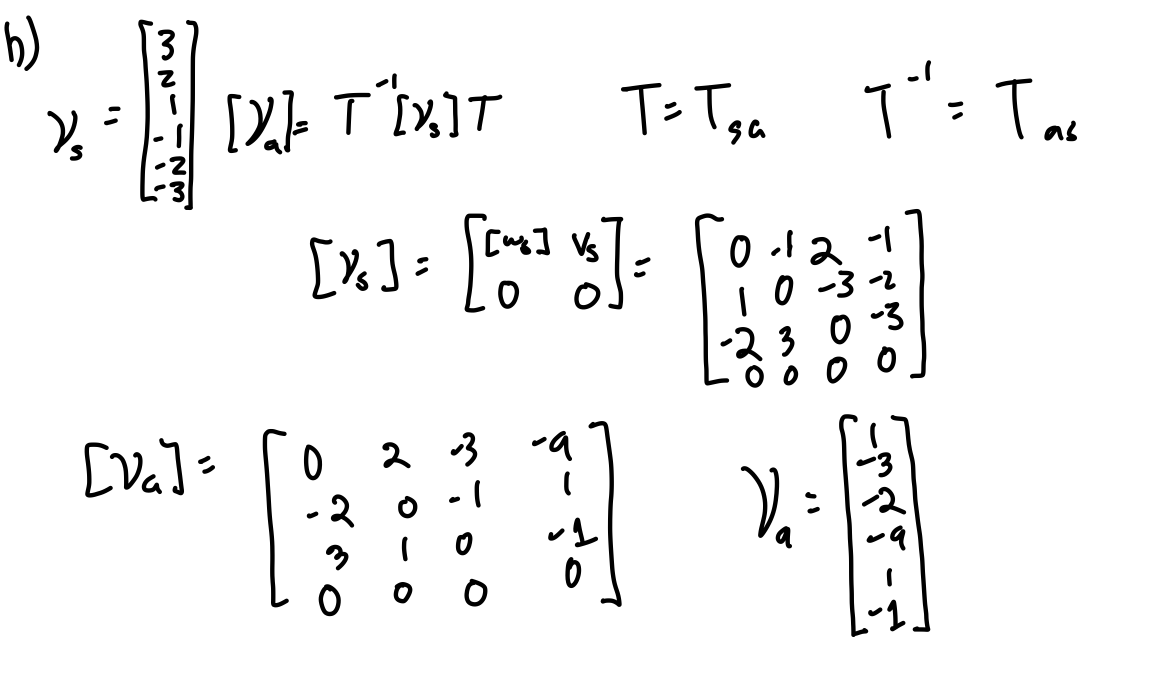


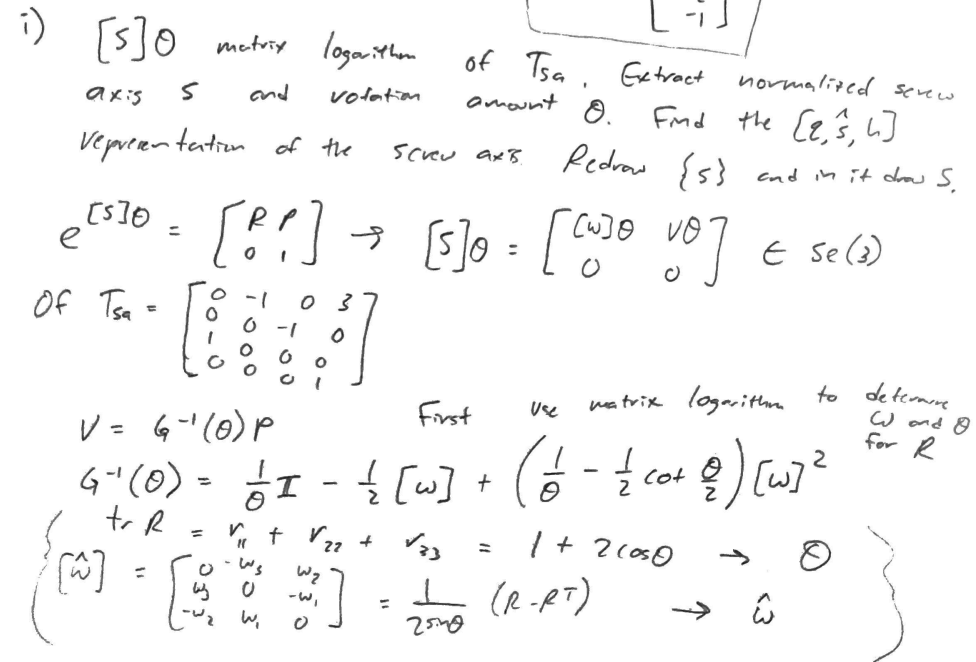
1. **Exercise 3.16** (pg.121) in Modern Robotics: Mechanics, Planning, and Control (Lynch et al.) [2].

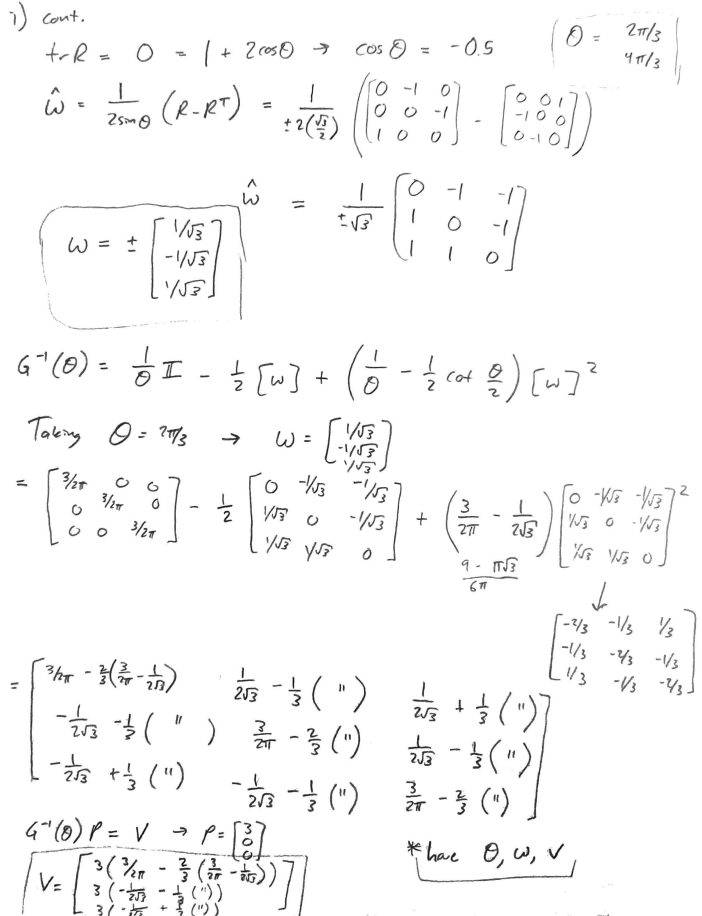


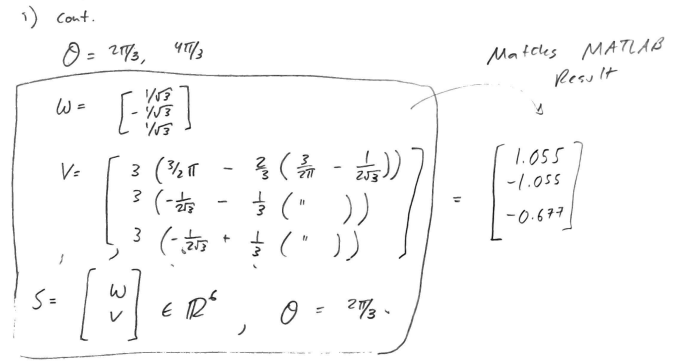












This matches the results obtained running these calculations in MATLAB as well:

MATLAB Code used to calculate S and Theta

—----------------------------------------------------------------------------------------------------------------------------

Ti = [0 -1 0 3;

0 0 -1 0;

1 0 0 0;

0 0 0 1];

Ri = Ti(1:3, 1:3);

Pi = Ti(1:3, 4);

theta = acos((trace(Ri) - 1) / 2)

w\_hat = 1 / (2\*sin(theta)) \* (Ri - Ri')

w = [w\_hat(3,2), w\_hat(2,3), w\_hat(2,1)]'

G = (1/theta) \* eye(3) - 1/2 \* w\_hat + (1/theta - 1/2 \* cot(theta/2)) \* w\_hat^2

v = G \* Pi

S = [w;v]

—----------------------------------------------------------------------------------------------------------------------------

Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------

S =

0.5774

-0.5774

0.5774

1.0548

-1.0548

-0.6772

—----------------------------------------------------------------------------------------------------------------------------

Further, converting these results back into a transformation matrix using the matrix exponential also yields the original matrix:

MATLAB Code used to recalculate T

—----------------------------------------------------------------------------------------------------------------------------

R = eye(3) + sin(theta) \* Axis2SkewSymmetricMatrix(w) + (1 - cos(theta)) \* Axis2SkewSymmetricMatrix(w)^2;

P = (eye(3) \* theta + (1 - cos(theta)) \* Axis2SkewSymmetricMatrix(w) + (theta - sin(theta)) \* Axis2SkewSymmetricMatrix(w)^2) \* v;

T = [R P; 0 0 0 1]

—----------------------------------------------------------------------------------------------------------------------------

Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------

T =

-0.0000 -1.0000 0.0000 3.0000

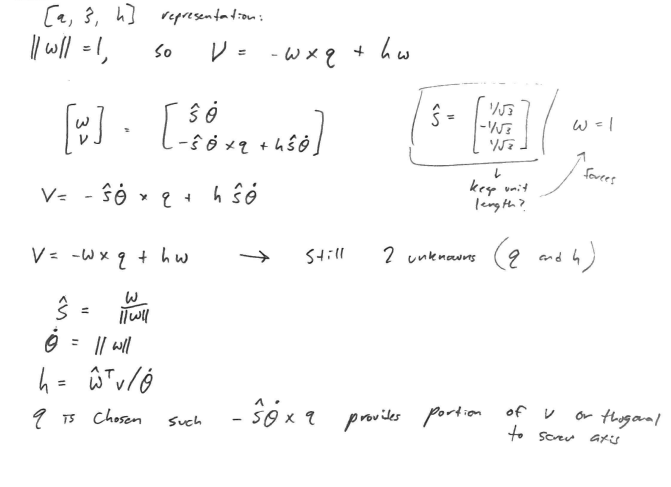
-0.0000 -0.0000 -1.0000 0.0000

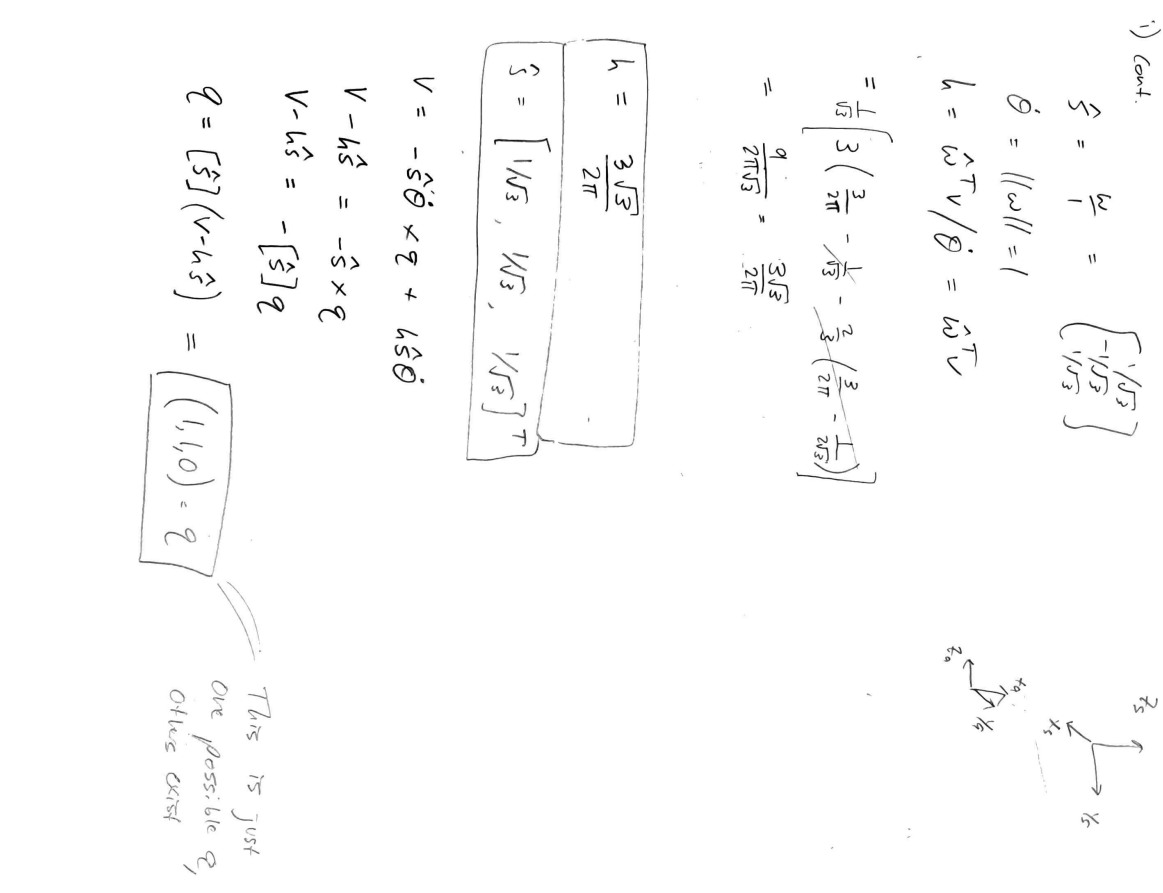
1.0000 -0.0000 -0.0000 -0.0000

0 0 0 1.0000

—----------------------------------------------------------------------------------------------------------------------------

Now finding q,s,h representation:





MATLAB Code used to confirm calculations

—----------------------------------------------------------------------------------------------------------------------------

T = [R P; 0 0 0 1];

h = w' \* v

s\_hat = w

q = Axis2SkewSymmetricMatrix(s\_hat) \* (v - h \* s\_hat)

% check to make sure configuration is equivalent

w = s\_hat;

v = cross(-s\_hat \* 1, q) + h \* s\_hat \* 1;

S = [w; v]

—----------------------------------------------------------------------------------------------------------------------------

Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------

h =

0.8270

s\_hat =

0.5774

-0.5774

0.5774

q =

1.0000

1.0000

-0.0000

S =

0.5774

-0.5774

0.5774

1.0548

-1.0548

-0.6772

—----------------------------------------------------------------------------------------------------------------------------

This code also calculated S according to the formula using qsh components to confirm that this combination of q, s, and h parameterized the same screw axis as S as calculated and presented earlier in part i.

The drawing of this screw axis was also done using MATLAB, with the code:

MATLAB Code used to draw frame {s} and screw axis

—----------------------------------------------------------------------------------------------------------------------------

figure

plot3(0, 0, 0, '.k', 1, 0, 0,'.k', 0, 1, 0,'.k', 0, 0, 1,'.k', markersize=10)

view(3)

box on

hold on

grid on

xlabel('x'), ylabel('y'), zlabel('z');

% {s}

plotFrame\_T(eye(4), 's');

% S

plotScrewAxis\_qsh(q, s\_hat, h)

function plotFrame\_T(T, label)

origin = T(1:3, 4)';

% x-axis

line('XData', [origin(1) origin(1) + T(1,1)], 'YData', [origin(2) origin(2) + T(2,1)],...

'ZData', [origin(3) origin(3) + T(3,1)], 'Color','r','LineWidth',3);

% y-axis

line('XData', [origin(1) origin(1) + T(1,2)], 'YData', [origin(2) origin(2) + T(2,2)],...

'ZData', [origin(3) origin(3) + T(3,2)], 'Color','g','LineWidth',3);

% z-axis

line('XData', [origin(1) origin(1) + T(1,3)], 'YData', [origin(2) origin(2) + T(2,3)],...

'ZData', [origin(3) origin(3) + T(3,3)], 'Color','b','LineWidth',3);

% frame label

text(origin(1) - 0.2 \* T(1,1), origin(2) - 0.2 \* T(2,2), origin(3) - 0.2 \* T(3,3), horzcat('$\lbrace ', label, ' \rbrace$'));

% axes labels

text(origin(1) + T(1,1), origin(2) + T(2,1), origin(3) + T(3,1), '$x$');

text(origin(1) + T(1,2), origin(2) + T(2,2), origin(3) + T(3,2), '$y$');

text(origin(1) + T(1,3), origin(2) + T(2,3), origin(3) + T(3,3), '$z$');

end

function plotScrewAxis\_qsh(q, s\_hat, h, C)

origin = q';

% length of frame vectors

if nargin < 4

C = 1;

end

% x-axis

line('XData', [origin(1) origin(1) + C \* s\_hat(1)], 'YData', [origin(2) origin(2) + C \* s\_hat(2)],...

'ZData', [origin(3) origin(3) + C \* s\_hat(3)], 'Color','k','LineWidth',3);

line('XData', [origin(1) origin(1) - C \* s\_hat(1)], 'YData', [origin(2) origin(2) - C \* s\_hat(2)],...

'ZData', [origin(3) origin(3) - C \* s\_hat(3)], 'Color','k','LineWidth',3);

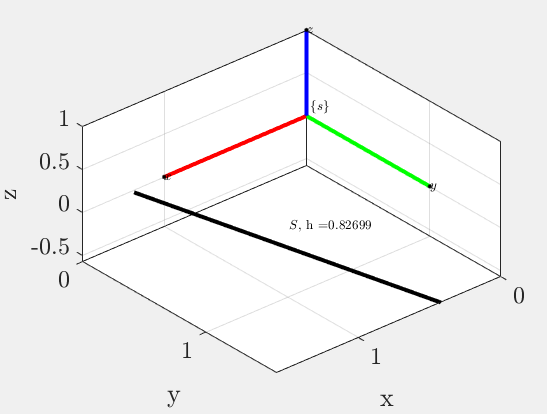
text(origin(1) - 0.1, origin(2) - 0.1, origin(3) + 0.1, strcat('$S$', ', h = ', string(h)));

end

—----------------------------------------------------------------------------------------------------------------------------

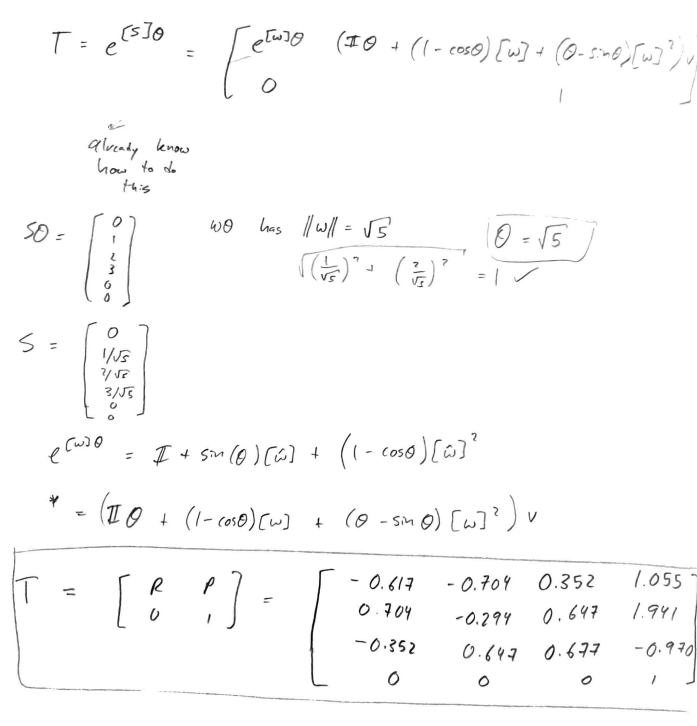
Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------



—----------------------------------------------------------------------------------------------------------------------------

j)



MATLAB Code used to calculate T

—----------------------------------------------------------------------------------------------------------------------------

S\_theta = [0, 1, 2, 3, 0, 0]';

theta = sqrt(5);

S = S\_theta ./ theta;

w = S(1:3)

v = S(4:6)

R = eye(3) + sin(theta) \* Axis2SkewSymmetricMatrix(w) + (1 - cos(theta)) \* Axis2SkewSymmetricMatrix(w)^2;

P = (eye(3) \* theta + (1 - cos(theta)) \* Axis2SkewSymmetricMatrix(w) + (theta - sin(theta)) \* Axis2SkewSymmetricMatrix(w)^2) \* v;

T = [R P; 0 0 0 1]

—----------------------------------------------------------------------------------------------------------------------------

Corresponding output:

—----------------------------------------------------------------------------------------------------------------------------

T =

-0.6173 -0.7037 0.3518 1.0555

0.7037 -0.2938 0.6469 1.9407

-0.3518 0.6469 0.6765 -0.9704

0 0 0 1.0000

—----------------------------------------------------------------------------------------------------------------------------

The drawing of this screw axis was also done using MATLAB, with the code:

MATLAB Code used to draw frame {s}, transformed frame {b}, and screw axis

—----------------------------------------------------------------------------------------------------------------------------

figure

plot3(0, 0, 0, '.k', 1, 0, 0,'.k', 0, 1, 0,'.k', 0, 0, 1,'.k', markersize=10)

view(3)

box on

hold on

grid on

xlabel('x'), ylabel('y'), zlabel('z');

% {s}

plotFrame\_T(eye(4), 's');

% {b}

plotFrame\_T(T, 'b');

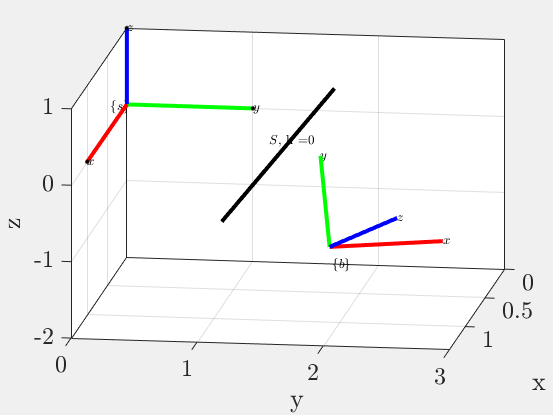
% S

plotScrewAxis\_qsh(q, s\_hat, h)

—----------------------------------------------------------------------------------------------------------------------------

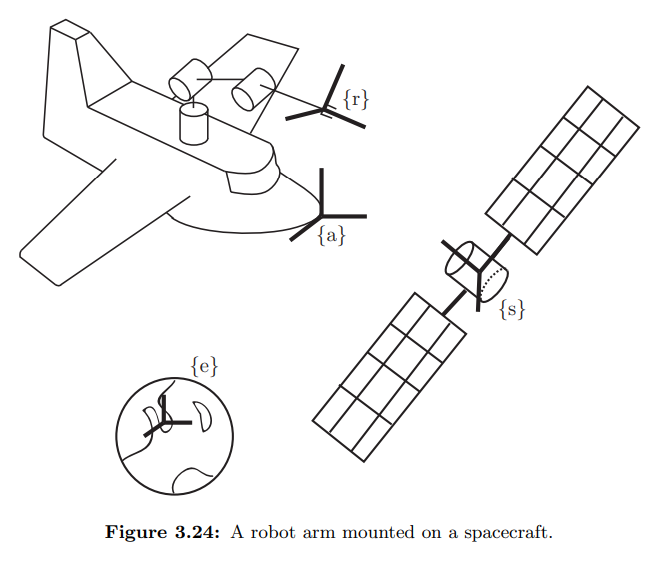
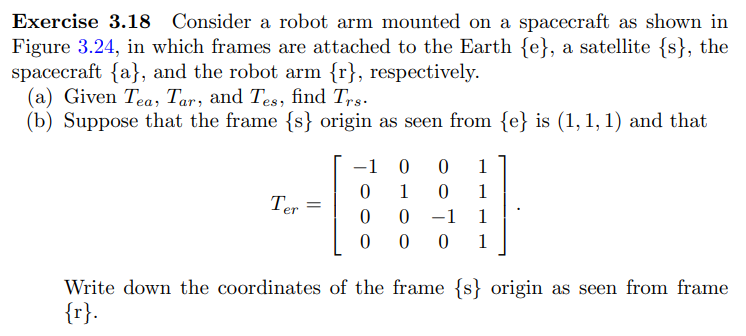
Corresponding output:

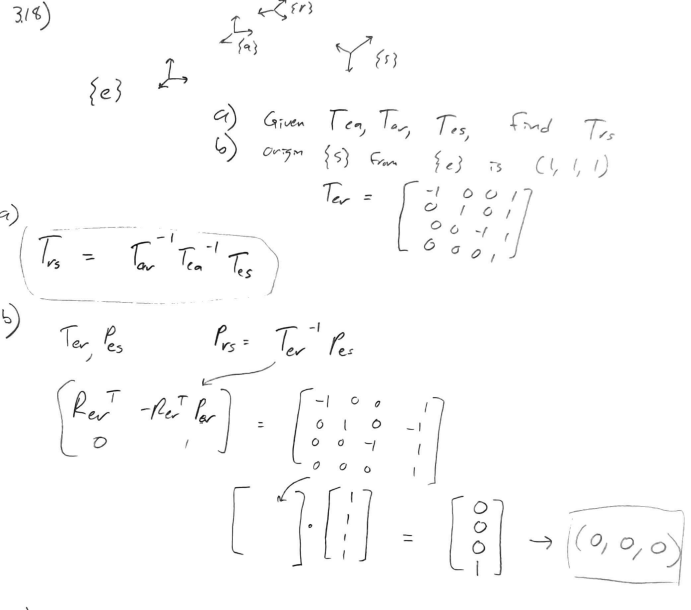
—----------------------------------------------------------------------------------------------------------------------------



—----------------------------------------------------------------------------------------------------------------------------

1. **Exercise 3.18** (pg.123) in Modern Robotics: Mechanics, Planning, and Control (Lynch et al.) [2].





1. **Consider** the pelvic osteotomy situation illustrated in Fig. 1. Here we assume that three locating pins have been inserted into the patient’s pelvis, and that a CT scan of the pelvis with the pins inserted has been produced. The patient has been placed onto the operating table. Also, a magnetic navigation system (here, the Northern Digital Aurora) is present in the room.

Two surgical tools are available:

- A probe/pointer device

- An osteotome (essentially a fancy chisel) that will be used to cut the pelvis. 6-DOF Aurora tracking sensors have been attached to the handle of each tool and an additional 6-DOF sensor has been affixed rigidly to the pelvis. The Aurora is capable of determining the position and orientation of each sensor relative to the Aurora base unit.

We will define the following coordinate systems and relationships:

= Coordinate system of tracking system base unit

= Coordinate system of tracking device on pointer handle

= Coordinate system of tracking device on osteotome handle

= Coordinate system of tracking device attached to pelvis

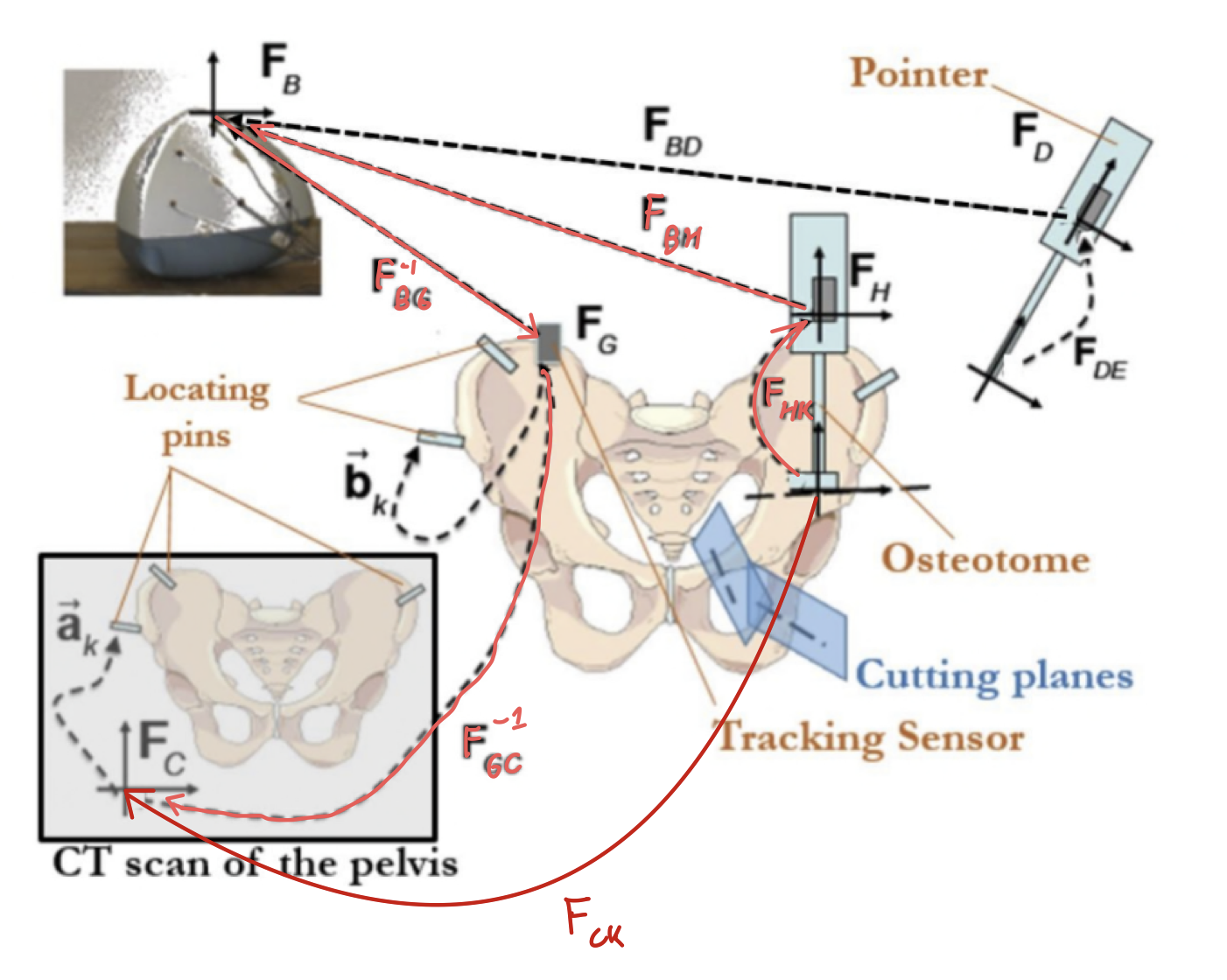
= Coordinate system of CT image

We also have the following relationships

=Measured 6 DOF pose of tracking device x relative to base unit

= 6 DOF pose of osteotome blade relative to osteotome handle tracking device

= 6 DOF pose of pointer tip relative to pointer handle tracking device



**Fig. 1: Computer-Assisted Osteotomy (edited with solution)**

=Position of the top of pin *k* in CT coordinates

=Position of the top of pin *k* relative to tracking device G

Suppose that we have touched the tops of the three fiducial pins and used the results to compute a registration transformation such that =. Give an expression for computing the position and orientation of the osteotome blade in CT coordinates, based on the available tracking system measurements [3].

