Linear Algebra Solutions

1. Rotation Matrix Properties

(a) Prove that $\|r_i\| \leq 1$ for $i = 1, 2, \ldots, 9$: A 3D rotation matrix R satisfies the following properties:

- **Orthogonality:** $R^T R = I$.
- **Determinant:** $\det(R) = 1$.

From the orthogonality property, each row (and column) vector of R forms an orthonormal set:

- The dot product of any two distinct rows or columns is zero (orthogonality).
- The norm of each row or column vector is 1 (unit length).

Thus, to prove that the elements r_i of any 3D rotation matrix R satisfy $||r_i|| \leq 1$ (where $i = 1, 2, \ldots, 9$), we can use some key properties of rotation matrices to complete the proof.

Step 1: Properties of 3D Rotation Matrices

A 3D rotation matrix R is a special orthogonal matrix, which satisfies two important properties: 1. **Orthogonality**: $R^T R = I$, where I is the identity matrix. 2. **Determinant**: $\det(R) = 1$.

The matrix R is usually represented as:

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}$$

where each row and column of matrix R forms an orthonormal basis in 3D space.

Step 2: Norm of the Column Vectors

Let the column vectors of R be represented as:

$$\mathbf{c_1} = \begin{bmatrix} r_1 \\ r_4 \\ r_7 \end{bmatrix}, \quad \mathbf{c_2} = \begin{bmatrix} r_2 \\ r_5 \\ r_8 \end{bmatrix}, \quad \mathbf{c_3} = \begin{bmatrix} r_3 \\ r_6 \\ r_9 \end{bmatrix}$$

Since R is a rotation matrix, its column vectors are unit orthogonal vectors. Therefore:

$$||\mathbf{c_1}|| = ||\mathbf{c_2}|| = ||\mathbf{c_3}|| = 1$$

and they are mutually orthogonal:

$$\mathbf{c_1} \cdot \mathbf{c_2} = \mathbf{c_2} \cdot \mathbf{c_3} = \mathbf{c_3} \cdot \mathbf{c_1} = 0$$

Step 3: Upper Bound on the Elements $|r_i| \leq 1$

Since each column vector is a unit vector, the sum of the squares of its components equals 1. For example, for the vector $\mathbf{c_1}$, we have:

$$r_1^2 + r_4^2 + r_7^2 = 1$$

Similarly:

$$r_2^2 + r_5^2 + r_8^2 = 1$$
$$r_3^2 + r_6^2 + r_9^2 = 1$$

Since the square of each term is non-negative, this implies that each $r_i^2 \leq 1$, and hence $|r_i| \leq 1$.

Conclusion

Therefore, each element r_i of the rotation matrix R satisfies $|r_i| \leq 1$. This means that for the norm of each element (i.e., its absolute value), we have:

$$||r_i|| \leq 1$$

where $i = 1, 2, \ldots, 9$, and the proof is complete.

(b) Prove that $R_{k,\theta} = R_{-k,-\theta}$: The rotation matrix $R_{k,\theta}$ for a rotation by angle θ about a unit vector $k = [k_x, k_y, k_z]$ is given by:

$$R_{k,\theta} = I + \sin\theta[K] + (1 - \cos\theta)[K]^2,$$

where $[K]$ is the skew-symmetric matrix representing k:

$$[K] = \begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix}.$$

For $-k$ and $-\theta$, the skew-symmetric matrix becomes $[-K]$, and the rotation matrix is:

$$R_{-k,-\theta} = I + \sin(-\theta)([-K]) + (1 - \cos(-\theta))([-K])^2.$$

Using $\sin(-\theta) = -\sin\theta$ and $\cos(-\theta) = \cos\theta$, we see:

$$R_{-k,-\theta} = I - \sin\theta[K] + (1 - \cos\theta)[K]^2.$$

Since the terms match $R_{k,\theta}$, we have:

$$R_{k,\theta} = R_{-k,-\theta}.$$

(c) What does each row in ${}^aR_b$ represent? Each row of the rotation matrix ${}^aR_b$ represents the coordinates of one of the basis vectors of frame b expressed in terms of the basis of frame a: For a 3×3 rotation matrix ${}^aR_b$, we can write:

$${}^aR_b = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Each row has the following meaning:
1. **First Row**: Represents the X-axis of the source coordinate system b (denoted as $\mathbf{X}_b$) expressed in the target coordinate system a:

First row: $[r_{11}, r_{12}, r_{13}] = \mathbf{X}_b$ in the a coordinate system,corresponding to weight components in x, y, z respec

2. **Second Row**: Represents the Y-axis of the source coordinate system b (denoted as $\mathbf{Y}_b$) expressed in the target coordinate system a:

First row: $[r_{21}, r_{22}, r_{23}] = \mathbf{Y}_b$ in the a coordinate system,corresponding to weight components in x, y, z respec

3. **Third Row**: Represents the Z-axis of the source coordinate system b (denoted as $\mathbf{Z}_b$) expressed in the target coordinate system a:

First row: $[r_{31}, r_{32}, r_{33}] = \mathbf{Z}_b$ in the a coordinate system,corresponding to weight components in x, y, z respec

2. Gimbal Lock and Representations

(a) Example of Gimbal Lock:

Explanation of Gimbal Lock with x-y-z (Tait-Bryan) Intrinsic Rotations

In the x-y-z intrinsic rotation representation, the object undergoes three rotations in the following order:

1. A rotation about the x-axis (roll).
2. A rotation about the y-axis (pitch).
3. A rotation about the z-axis (yaw).

These rotations are applied one after the other in the sequence:

$$R_z(\psi) \cdot R_y(\theta) \cdot R_x(\phi)$$

where θ is the pitch, ϕ is the roll, and ψ is the yaw.

Gimbal Lock Condition

The issue arises when the pitch angle θ reaches $\pm 90°$. At this point, the second and third rotation axes become aligned, effectively reducing the system to two degrees of freedom rather than three. This leads to gimbal lock, and the system loses the ability to perform rotations about the yaw and pitch axes independently.

Why does this happen?

When $\theta = \pm 90°$, the rotation about the y-axis (pitch) will bring the x-axis and z-axis of the local coordinate frame into alignment. This results in two axes becoming parallel, and as a consequence, you cannot distinguish between rotations about the yaw and roll axes anymore. The system can no longer represent arbitrary rotations, since further rotation around one of these axes will no longer lead to a distinct new orientation but rather a reconfiguration of existing angles.

Mathematical Explanation

In terms of the rotation matrices, the problem occurs because of the nature of the trigonometric functions used to describe rotations. When $\theta = \pm 90°$, the sine and cosine terms involved in the rotation matrices become zero or one, leading to degenerate cases where the two rotational axes collapse into one.

For example, the matrix for $R_y(\theta)$ looks like this:

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

At $\theta = \pm 90°$, the matrix becomes:

$$R_y(90°) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

This means that the rotation matrix will cause the x- and z-axes to become aligned, and further rotations about the x- and z-axes will no longer produce unique orientations.

Why avoid gimbal lock in robotic arms?

- Gimbal lock reduces the controllable degrees of freedom, making it impossible to orient the arm in certain configurations.
- To avoid it, use quaternion or rotation matrix representations, which are not susceptible to gimbal lock.

(b) Quaternion to Rotation Matrix: Given a quaternion $q = [w, x, y, z]$ and target form of Rotation Matrix expressed in Quaternion Terms

$$R = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix}.$$

Steps:

1. Start with $q = w + xi + yj + zk$ (original form).

2. Axis-Angle to Quaternion To convert from an axis-angle representation to a quaternion, we can follow a standard mathematical procedure. The axis-angle representation defines a rotation by specifying:

   - A unit vector $\mathbf{u} = (u_x, u_y, u_z)$, which represents the axis of rotation. - A scalar θ, which represents the angle of rotation (in radians).

   The quaternion $q = w + xi + yj + zk$ corresponding to this axis-angle representation is given by:

$$q = \cos\left(\frac{\theta}{2}\right) + \left(u_x \sin\left(\frac{\theta}{2}\right)\right) i + \left(u_y \sin\left(\frac{\theta}{2}\right)\right) j + \left(u_z \sin\left(\frac{\theta}{2}\right)\right) k$$

   Steps to Convert Axis-Angle to Quaternion:

   1. **Normalize the axis vector** (if it's not already normalized). The axis of rotation should be a unit vector. If the axis vector $\mathbf{u} = (u_x, u_y, u_z)$ is not a unit vector, normalize it by dividing each component by its magnitude:

$$\mathbf{u} = \frac{\mathbf{u}}{|\mathbf{u}|}$$

   2. **Compute the scalar and vector parts of the quaternion**: - The scalar part w is:

$$w = \cos\left(\frac{\theta}{2}\right)$$

   - The vector part $\mathbf{v} = (x, y, z)$ is:

$$x = u_x \sin\left(\frac{\theta}{2}\right)$$
$$y = u_y \sin\left(\frac{\theta}{2}\right)$$
$$z = u_z \sin\left(\frac{\theta}{2}\right)$$

   where the sum of the squares of x, y, and z given by:

$$x = u_x \sin\left(\frac{\theta}{2}\right), \quad y = u_y \sin\left(\frac{\theta}{2}\right), \quad z = u_z \sin\left(\frac{\theta}{2}\right)$$

   is indeed equal to $\sin^2\left(\frac{\theta}{2}\right)$.

   Let's square each of the terms and add them:

$$x^2 + y^2 + z^2 = \left(u_x \sin\left(\frac{\theta}{2}\right)\right)^2 + \left(u_y \sin\left(\frac{\theta}{2}\right)\right)^2 + \left(u_z \sin\left(\frac{\theta}{2}\right)\right)^2$$

This simplifies to:

$$x^2 + y^2 + z^2 = \sin^2\left(\frac{\theta}{2}\right)\left(u_x^2 + u_y^2 + u_z^2\right)$$

Since $u_x^2 + u_y^2 + u_z^2 = 1$ (because $\mathbf{u} = (u_x, u_y, u_z)$ is a unit vector representing the axis of rotation), we get:

$$x^2 + y^2 + z^2 = \sin^2\left(\frac{\theta}{2}\right)$$

So, indeed:

$$x^2 + y^2 + z^2 = \sin^2\left(\frac{\theta}{2}\right)$$

3. **Combine the scalar and vector parts** to form the quaternion:

$$q = w + xi + yj + zk$$

where $w = \cos\left(\frac{\theta}{2}\right)$, and $x = u_x \sin\left(\frac{\theta}{2}\right)$, $y = u_y \sin\left(\frac{\theta}{2}\right)$, $z = u_z \sin\left(\frac{\theta}{2}\right)$.
Final Formula:

$$q = \cos\left(\frac{\theta}{2}\right) + \left(u_x \sin\left(\frac{\theta}{2}\right)\right) i + \left(u_y \sin\left(\frac{\theta}{2}\right)\right) j + \left(u_z \sin\left(\frac{\theta}{2}\right)\right) k$$

3. Substitution on Axis-Angle Matrix with relationship between x,y,z in quaternion and u_x, u_y, u_z in Axis-Angle (

$$x = u_x \sin\left(\frac{\theta}{2}\right)$$

$$y = u_y \sin\left(\frac{\theta}{2}\right)$$

$$z = u_z \sin\left(\frac{\theta}{2}\right)$$

) and $\cos\theta = 1 - 2\sin^2\left(\frac{\theta}{2}\right)$, $\sin(\theta) = 2\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\theta}{2}\right)$ and $x^2 + y^2 + z^2 = \sin^2\left(\frac{\theta}{2}\right)$

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) + u_z\sin\theta & u_x u_z(1-\cos\theta) - u_y\sin\theta \\ u_x u_y(1-\cos\theta) - u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) + u_x\sin\theta \\ u_x u_z(1-\cos\theta) + u_y\sin\theta & u_y u_z(1-\cos\theta) - u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}$$

4. With these equation substitution in the original Axis-Angle Rotation Matrix, the new Rotation Matrix is expressed in the Quaternion terms of w,x,y,z

$$R = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}.$$

(c) Rotation Representation Suggestions:

- **Nano-robot with limited memory:** Use axis-angle representation (compact).
- **Nano-robot with limited computation:** Use rotation matrix (direct application).
- **iPhone navigation system:** Use quaternion (avoids gimbal lock, efficient interpolation).
- **Robotic arm with 6 DOF:** Use Denavit-Hartenberg (DH) parameters or quaternions for flexibility.

When selecting a rotation representation for different systems, the choice should be based on the specific constraints and requirements of each case. Below are the suggested rotation representations for each of the listed scenarios:

1. Nano-robot with very limited memory storage:

Recommended Rotation Representation: Axis-Angle (or Euler Angles)

- **Reasoning:** Euler Angles (pitch, roll, yaw) or Axis-Angle representations are very compact and use only 3 parameters, making them memory efficient.
- **Pros:**
  - Simple and compact (only 3 numbers needed).
  - Low memory usage.
- **Cons:**
  - Susceptible to gimbal lock (loss of one degree of freedom in certain orientations).
  - May be less intuitive for certain operations like interpolating rotations or composing multiple rotations.

2. Nano-robot with very limited computational power:

Recommended Rotation Representation: Axis-Angle (or Quaternion if computation allows)

- **Reasoning:** Axis-Angle representation is computationally light because it requires only 4 values (the axis of rotation and the angle).
- **Pros:**
  - Axis-Angle is computationally efficient as it requires simple arithmetic (sin, cos) and avoids the complexity of matrix operations.
  - Quaternions, though a bit more computationally expensive, avoid gimbal lock and are suitable if the system can afford slightly more computation.
- **Cons:**
  - Quaternions, while very efficient computationally in terms of performance, may require some learning and understanding of their operations (e.g., conjugation, normalization).
  - Axis-Angle, while simple, can require more work for interpolating rotations and can suffer from other limitations compared to quaternions in some applications.

3. iPhone navigation system:

Recommended Rotation Representation: Quaternions

- **Reasoning:** iPhones (and most smartphones) have powerful processors and sufficient memory for quaternion computations. Quaternions avoid gimbal lock and provide smooth interpolation for rotation operations.
- **Pros:**
  - Avoids gimbal lock, leading to more robust and stable performance in 3D rotations.
  - Compact (4 values).
  - Efficient for interpolating and composing rotations.
- **Cons:**
  - Slightly more complex than Euler angles, but modern computational power can handle it.

4. Robotic arm with 6 Degrees of Freedom (DOF):

Recommended Rotation Representation: Rotation Matrices or Quaternions

- **Reasoning:** For a robotic arm with multiple degrees of freedom, you typically need to handle both orientation and position in 3D space. Rotation matrices are often used because they are straightforward for transforming between coordinate frames and can be directly multiplied for successive rotations. Quaternions can be used for smooth and efficient rotation without gimbal lock, and they are easier to interpolate compared to matrices.
- **Pros of Rotation Matrices:**
  - Direct and intuitive for composing rotations.
  - Commonly used in kinematics and dynamics of robotic systems.
- **Pros of Quaternions:**
  - Avoid gimbal lock.
  - Better for interpolating between orientations, which is useful for smooth motion control of the robot.
- **Cons:**
  - Rotation matrices are larger (9 values compared to 4 for quaternions).
  - Quaternions are more computationally intensive than rotation matrices in some cases (though much less than matrices for large systems).

Summary of Recommendations:

- **Nano-robot with very limited memory storage:** Euler Angles or Axis-Angle.
- **Nano-robot with very limited computational power:** Axis-Angle or Quaternions.
- **iPhone navigation system:** Quaternions.
- **Robotic arm with 6 DOF:** Rotation Matrices or Quaternions.

3. Quaternion and Matrix Properties

(a) Prove q and $-q$ are equivalent: Quaternions q and $-q$ represent the same rotation because:

1. q defines a rotation as:
$$v' = qvq^{-1},$$
where v is a vector.

2. Using $-q$:

$$v' = (-q)v(-q)^{-1} = qvq^{-1}.$$

Thus, q and $-q$ yield identical transformations.

(b) When do R_a and R_b commute? Two rotation matrices R_a and R_b commute if:

$$R_a R_b = R_b R_a.$$

This occurs if:

- Both rotations are about the same axis.
- Two rotations are about orthogonal axes with angles that are integer multiples of π.
- One of the rotation matrices with angles that are integer of 2π