Contents

Chapter 1

Unconstrained Optimization

1.1 Newton's and Gauss-Newton's Methods

Following the discussion on gradient descent method, we still consider the unconstrained optimization problem (restated here in E.q. (1.1)).

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) \tag{1.1}$$

First we discuss the second-order method (Newton's method), which utilizes information in the second-order derivatives. Then we focus on a certain type of objective function, and approximate the hessian matrix in Newton's method by first-order derivatives.

Here we compare the three optimization methods:

Table 1.1: Comparison between three unconstrained optimization methods

Optimization methods	Step size	Comments
Gradient descent	$\delta\mathbf{x}_k = -\nabla f(\mathbf{x}_k)$	cell6
Newton's method	$\delta\mathbf{x}_k = -[\nabla^2 f(\mathbf{x}_k)]^{-1}\nabla f(\mathbf{x}_k)$	cell9
Gauss-Newton's method	$\delta\mathbf{x}_k = -\hat{H}^{-1}g$ $f(\mathbf{x}) = \frac{1}{2}\mathbf{e}(\mathbf{x})^T\mathbf{e}(\mathbf{x})$ $g = (\frac{\partial\mathbf{e}(\mathbf{x}_k)}{\partial\mathbf{x}}\|_{\mathbf{x}=\mathbf{x}_k})^T$ $\hat{H} = g\cdot g^T$	cell12

1.1.1 Newton's Method

We approximate the objective function $f(\mathbf{x})$ using Taylor's expansion:

$$f(\mathbf{x}_k + \delta\mathbf{x}) \approx f(\mathbf{x}_k) + (\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}_k})\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^T(\frac{\partial^2 f(\mathbf{x})}{\partial\mathbf{x}\partial\mathbf{x}^T}|_{\mathbf{x}=\mathbf{x}_k})\delta\mathbf{x} \tag{1.2}$$

where $\delta\mathbf{x}$ is a small change on $\mathbf{x} = \mathbf{x}_k$. Here we define:

$$q(\delta\mathbf{x}, \mathbf{x}_k) := f(\mathbf{x}_k) + (\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}|_{\mathbf{x}=\mathbf{x}_k})\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^T(\frac{\partial^2 f(\mathbf{x})}{\partial\mathbf{x}\partial\mathbf{x}^T}|_{\mathbf{x}=\mathbf{x}_k})\delta\mathbf{x} \tag{1.3}$$

$q(\delta\mathbf{x}, \mathbf{x}_k)$ is a quadratic approximation of $f(\mathbf{x})$ around $\mathbf{x} = \mathbf{x}_k$. Minimizing $q(\delta\mathbf{x}, \mathbf{x}_k)$ leads us to a descent direction (with step size indicated as well). Then we can write the optimization problem of minimizing q:

$$\min_{\delta\mathbf{x}\in\mathbb{R}^d} q(\delta\mathbf{x}, \mathbf{x}_k) \tag{1.4}$$

This is an unconstrained optimization problem and the objective function q is quadratic and thus convex. $\delta\mathbf{x}$ can be determined by setting the derivative of q with respect to $\delta\mathbf{x}$ to zero:

$$0 = \frac{\partial q\left(\delta\mathbf{x}, \mathbf{x}_k\right)}{\partial\delta\mathbf{x}} = \nabla f\left(\mathbf{x}_k\right)^\top + \delta\mathbf{x}^\top\nabla^2 f\left(\mathbf{x}_k\right) \tag{1.5}$$

Recall the properties of a hessian matrix $H := \nabla^2 f(\mathbf{x})$:

1. (For all twice differentiable functions,) H is symmetric since $\frac{\partial f}{\partial x_i \partial x_j} = \frac{\partial f}{\partial x_j \partial x_i}$;
2. (For convex functions,) H is positive semi-definite; (See the proof here.)
3. (For strictly convex functions,) H is positive definite.

Then we can solve E.q. 1.5 uniquely for $f(\mathbf{x})$ being strictly convex (i.e., $\nabla^2 f(\mathbf{x}_k) \succ 0$):

$$\delta\mathbf{x} = -\left[\nabla^2 f\left(\mathbf{x}_k\right)\right]^{-1}\nabla f\left(\mathbf{x}_k\right) \tag{1.6}$$

Finally, we have the update function in Newton's method:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \left[\nabla^2 f(\mathbf{x}_k)\right]^{-1} \nabla f(\mathbf{x}_k), \quad \alpha_k > 0 \tag{1.7}$$

Fig. 1.1 is a graphic illustration on how we apply Newton's method. In each iteration $\mathbf{x} = \mathbf{x}^{(k)}$, we approximate $f(\mathbf{x})$ locally by a quadratic function (drawn in the black line). In each step of applying the update equation (E.q. 1.7), we decide $\mathbf{x}^{(k+1)}$ by the minimizer given from $q(\delta\mathbf{x}, \mathbf{x}^{(k)})$. Finally, it converges to a local minimum $f(\mathbf{x}^*)$.

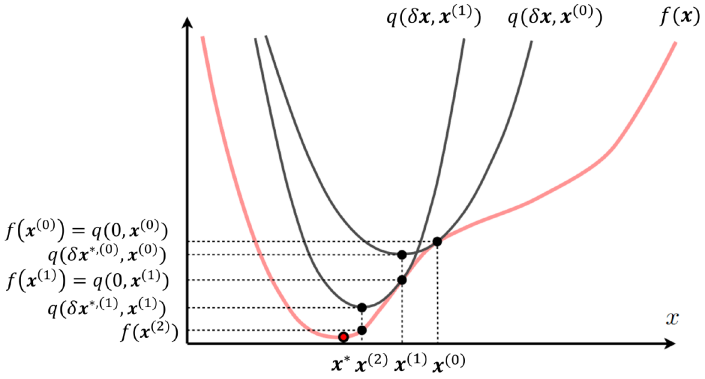


Figure 1.1: Graphic illustration of the quadratic approximation in Newton's method

Here are some facts about Newton's method covered in the lecture slides.

- Like other descent methods, Newton's method converges to a local minimum
- **Damped Newton phase**: when the iterates are "far away" from the optimum, the function value is decreased sublinearly, i.e., the step sizes α_k are small
- **Quadratic convergence phase**: when the iterates are "sufficiently close" to the optimum, full Newton steps are taken, i.e., $\alpha_k = 1$, and the function value converges quadratically to the optimum
- A **disadvantage** of Newton's method is the need to form the Hessian $\nabla^2 f(\mathbf{x}_k)$, which can be numerically ill-conditioned or computationally expensive in high-dimensional problems

Now Gauss-Newton's method comes into the picture as we want to bypass the above disadvantage of Newton's method (needs to compute H and its inverse).

1.1.2 Gauss-Newton's Method

Gauss-Newton is an *approximation* to Newton's method that avoids computing the Hessian. It is applicable when the objective function has the following quadratic form:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{e}(\mathbf{x})^\top \mathbf{e}(\mathbf{x}) \quad \mathbf{e}(\mathbf{x}) \in \mathbb{R}^m \tag{1.8}$$

i.e., $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{e}(\mathbf{x})\|_2^2$.

To derive the Gauss-Newton's method, we first take a look at the derivative and hessian of $f(\mathbf{x})$:

- Derivative and Hessian:

$$\text{Jacobian:} \quad \left.\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k} = \mathbf{e}(\mathbf{x}_k)^\top \left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right)$$

$$\text{Hessian:} \quad \left.\frac{\partial^2 f(\mathbf{x})}{\partial \mathbf{x}\partial \mathbf{x}^\top}\right|_{\mathbf{x}=\mathbf{x}_k} = \left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right)^\top \left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right) + \sum_{i=1}^{m} e_i(\mathbf{x}_k) \left(\left.\frac{\partial^2 e_i(\mathbf{x})}{\partial \mathbf{x}\partial \mathbf{x}^\top}\right|_{\mathbf{x}=\mathbf{x}_k}\right)$$

Now, the second-order derivative only exists in the second term in hessian. We assume the second term is small relative to the first and approximate hessian without second derivatives:

$$\left.\frac{\partial^2 f(\mathbf{x})}{\partial \mathbf{x}\partial \mathbf{x}^\top}\right|_{\mathbf{x}=\mathbf{x}_k} \approx \left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right)^\top \left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right) \tag{1.9}$$

To derive the update function, the rest procedure is basically the same as Newton's method, except for we replace hessian in E.q. 1.3 by E.q. 1.9. This leads to the update function in Gauss-Newton's method:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \delta \mathbf{x}, \quad \alpha_k > 0 \tag{1.10}$$

where $\delta \mathbf{x}$ can be determined from:

$$\left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right)^\top \left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right) \delta \mathbf{x} = -\left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right)^\top \mathbf{e}\left(\mathbf{x}_k\right) \tag{1.11}$$

Levenberg-Marquardt's Method

The Levenberg-Marquardt's Method is a modified version of the Gauss-Newton's method to compensate the missing hessian term. The modification is based on E.q. 1.11, which uses a positive diagonal matrix D:

$$\left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right)^{\top}\left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}+\lambda D\right)\delta\mathbf{x}=-\left(\left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}\right)^{\top}\mathbf{e}\left(\mathbf{x}_k\right) \tag{1.12}$$

When $\lambda \geq 0$ is large, the descent direction $\delta\mathbf{x}$ corresponds to a small step in the direction of the steepest descent. This helps when the Hessian approximation is poor or poorly conditioned by providing a meaningful direction. An example choice of D:

$$D = \mathbf{diag}(\sum_j J_j(\mathbf{x}_k)^T J_j(\mathbf{x}_k)) \tag{1.13}$$

where $J_j(\mathbf{x}) := \frac{\partial \mathbf{e}_j(\mathbf{x})}{\partial \mathbf{x}} \in \mathbb{R}^{m_j \times n}, \lambda \geq 0, D \in \mathbb{R}^{n \times n}$.

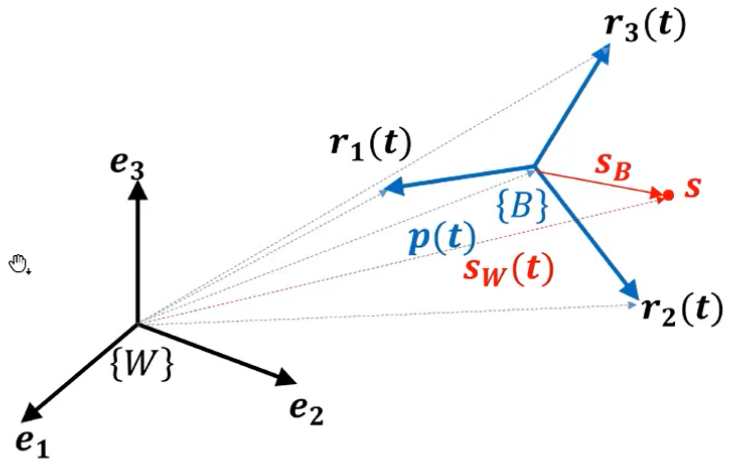
1.1.3 Example

Chapter 2

Rotation

In this lecture, we will discuss how to handle orientation. Robots are typically rigid bodies, that have position and orientation. Positions are simply x, y, z coordinates but rotation is more complex. We will then discuss how we can model the motion and sensing of a robot and bring it back to the optimization theory we discussed. Hence, we have to be able to optimise over position and rotation.

We consider two frames - the world or the map $\{W\}$ and the body reference frame $\{B\}$, for the robot itself. Since the robot remains the same, we can specify the motion of one point $p(t) \in R^3$ and three coordinate axes $r_1(t), r_2(t), r_3(t)$, attached to that point.



In the above figure, we can see the black axes as the world frame, and the blue lines as the body frame. The point s will have constant coordinates with respect to the body frame (it is some point

on the robot), but different coordinates in the world frame (since the body is constantly moving). Alternatively,

A point s on the rigid body has fixed coordinates $s_B \in R^3$ in the body frame $\{B\}$ but time-varying coordinates $s_w(t) \in R^3$ in $\{W\}$.

Pose

The pose $T(t) \in SE(3)$ of a rigid body reference frame $\{B\}$ at time t in a fixed world frame $\{W\}$ is determined by:

1. The position $p(t) \in R^3$ of $\{B\}$ relative to $\{W\}$.
2. The orientiation $R(t) \in SO(3)$ of $\{B\}$ relative to $\{W\}$, determined by the 3 coordinate axes $r_1(t), r_2(t), r_3(t)$.

Now, how do we describe the space of orientations of orientations $SO(3)$ and the space of poses $SE(3)$.

2.1 Special Euclidian Group

Rigid body motion is described by a sequence of functions that describe how the coordinates of 3-D points on the object change with time.

Rigid body motion preserves distances (vector norms) and does not allow reflection of the coordinate system (vector cross products).

For instance, the distance between our eyes should not change when we move around. Further, it shouldn't be possible to get a mirror image of a body when it moves.

The **Euclidian Group** $E(3)$ is a set of functions $g : R^3 \to R^3$ that preserves the norm of any two vectors.

The **Special Euclidian Group** $SE(3)$ is a set of functions $g : R^3 \to R^3$ that preserve the norm and the cross product of any two vectors.

1. Norm: $\|g_*(u) - g_*(v)\| = \|v - u\|, \forall u, v \in R^3$.
2. Cross product: $g_*(u) \times g_*(v) = g_*(u \times v), \forall u, v \in R^3$

where $g_*(x) := g(x) - g(0)$.

Corollary: $SE(3)$ elements g also preserve:

1. Angle: $u^T v = \frac{1}{4}(\|u+v\|^2 - \|u-v\|^2) \implies u^T v = g_*(u)^T g_*(v), \forall u, v \in R^3$.
2. Volume: $\forall u, v, w \in R^3, g_*(u)^T (g_*(v) \times g_*(w)) = u^T (v \times w)$

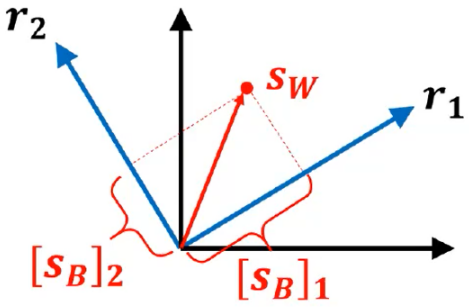
Pure rotation is a special case of rigid body motion. The orientation of the body frame $\{B\}$ in the world frame $\{W\}$ is determined by the coordinates of the three orthogonal vectors $r_1 = g(e_1), r_2 = g(e_2), r_3 = g(e_3)$, transformed from $\{B\}$ to $\{W\}$.

The vectors organized in a 3×3 matrix specify the orientation of $\{B\}$ in $\{W\}$:

$$ {}_{\{W\}}R_{\{B\}} = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix} \in \mathbb{R}^{3\times 3} $$

Consider a point with coordinates $s_B \in R^3$ in $\{B\}$. Its coordiantes s_W in $\{W\}$ are:

$$ \begin{aligned} s_W &= [s_B]_1 r_1 + [s_B]_2 r_2 + [s_B]_3 r_3 \\ &= R s_B \end{aligned} $$



Not sure if this is correct This is because the transformation from the body frame to world frame is the same as the position of the body frame in the world frame.

The rotation transformation g from $\{B\}$ to $\{W\}$ is:

$$ g(s) = Rs $$

2.2 Special Orthogonal Group $SO(3)$

r_1, r_2, r_3 from an orthonormal basis: $r_i^T r_j = 1$ (when $i = j$), else 0.

Since r_1, r_2, r_3 form an orthonormal basis, the inverse of R is its transpose $R^T R = I$ and $R^{-1} = R^T$.

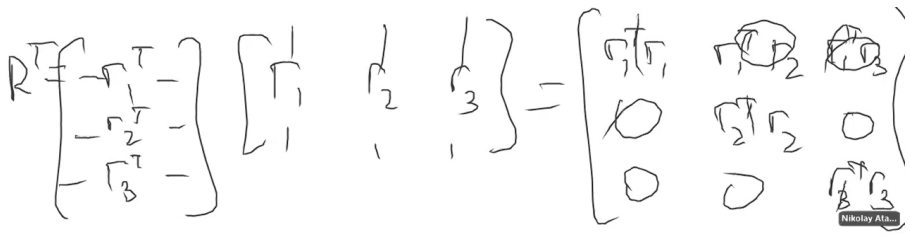


Figure 2.1: A quick illustration to see why $R^T R = I$

R belongs to the orthogonal group:

$$O(3) := \{R \in \mathbb{R}^{3\times 3} | R^T R = R R^T = I\}$$

Distance preserving: Let's look at -

$$\|Rx - Ry\|^2 = \|R(x-y)\|^2 = (x-y)^T R^T R (x-y) = (x-y)^T (x-y) = \|x-y\|^2$$

Reflections are not allowed since $\det(R) = r_1^T (r_2 \times r_3) = 1$:

$$R(x \times y) = R(x \times (R^T R y)) = (R\hat{x}R^T)Ry = \frac{1}{det(R)}(Rx) \times (Ry)$$

Hence, we can say that R belongs to the special orthogonal group:

$$SO(3) := \{R \in \mathbb{R}^{3\times 3} | R^T R = I, det(R) = 1\}$$

2.3 2D Rotations

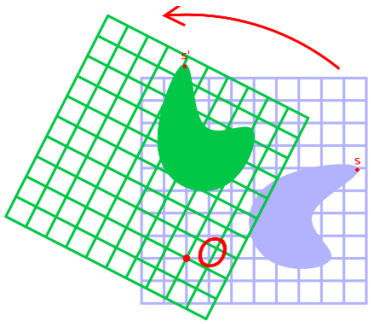
2.3.1 Angles

In two dimensions, we can just use an angle around Z axis to represent rotation.

- **Rotation angle**: a 2-D rotation of a point $\mathbf{s}_B \in \mathbb{R}^2$ can be parametrized by an angle θ around the z-axis:

$$\mathbf{s}_W = R(\theta)\mathbf{s}_B := \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \mathbf{s}_B$$

- $\theta > 0$: counterclockwise rotation



The angle works out this way because of right hand thumb rule (fingers curl in direction of rotation - axis coincides with thumb)

The matrix values can be calculated by simply making the columns the positions of the X and Y axis after rotation.

2.3.2 Unit-norm Complex Number

We can represent a rotation by a complex number, something like $e^{i\theta}$, which can be expanded by euler's formula:

$$e^{i\theta} = \cos\theta + i\sin\theta$$

Or, if we have a 2D rotation of $[s_B]_1 + i[s_B]_2 \in \mathbb{C}$

$$e^{i\theta}([s_B]_1 + i[s_B]_2) = ([s_B]_1\cos\theta - [s_B]_2\sin\theta) + i([s_B]_1\sin\theta + [s_B]_2\cos\theta)$$

We'll revisit this in the 3D scenario.

2.4 3D Rotation

Euler Angles

This is our 3D extension of the 2D angle representation. When in 2D, we had an angle around one axis, in 3D, we have it about 3 different axes.

Or, Euler angles: an extension of the rotation angle parametrization of 2-D rotations that specifies rotation angles around the three principal axes.

Axis Angle

An extension of the rotation angle parametrization of 2-D rotations that allows the axis of rotation to be chosen freely instead of being a fixed principal axis.

Unit Quaternion

An extension of the unit-norm complex number parametrization of 2-D rotations.

2.4.1 Euler Angles

This uses three angles to specify rotation around three principal axes.

There are 24 different ways to apply these rotations - depending on the order, which axes we come back to, whether we rotate around original axes, or we keep using the updated one.

1. Extrinsic axes: The rotation axes remain static
2. Intrinsic Axes: THe rotation axes move with the rotations.
3. Further, there are two groups that they can be divided into:
   (a) Euler angles: Rotat ion about one axis, then a second, and then the first
   (b) Tait-bryan Angles: Rotation about all three axes.

The Euler and Tait-Bryan Angles each have 6 possible choices for each of the extrinsic/intrinsic groups leading to 2 * 2 * 6 = 24 possible conventions to specify a rotation sequence with three given angles.

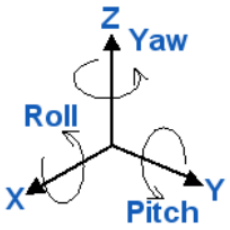
This is a mess - too many options!

Hence, when we deal with euler, we must specify rotations explicitly:

1. r (rotating = intrinsic) or s (static = extrinsic)
2. xyz or zyx or zxz, etc. (order of rotation axes)

The most widely used one is row-pitch-yaw convention

Row-Pitch-Yaw Convention



Roll (ϕ), pitch (θ), yaw (ψ) angles are used in aerospace engineering to specify rotation of an aircraft around the x, y, and z axes, respectively.

Now, elementary rotations can be represented as follows:

A rotation by an angle ϕ around the x-axis is represented by:

$$R_x(\phi) := \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

A rotation by an angle θ around the y-axis is represented by:

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

A rotation by an angle ψ around the z-axis is represented by:

$$R_z(\psi) := \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If we are given some information about the roll, pitch, yaw, we can compute the R matrix as follows:

Intrinsic yaw (ψ), pitch (θ), roll (ϕ) rotation ($rzyx$):

- A rotation ψ about the original z-axis
- A rotation θ about the intermediate y-axis
- A rotation ϕ about the transformed x-axis

Extrinsic roll (ϕ), pitch (θ), yaw (ψ) rotation ($sxyz$):

- A rotation ϕ about the global x-axis
- A rotation θ about the global y-axis
- A rotation ψ about the global z-axis

Both conventions define the following body-to-world rotation:

$$
\begin{aligned}
R &= R_z(\psi)R_y(\theta)R_x(\phi) \\
&= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}
\end{aligned}
$$

Gimbal Lock

Ideally, with three variables, and three degrees of freedom, we should be able to represent all rotations but this is not true. This is gimbal lock. Alternatively, angle paramterizations are not one-to-one.

Example: if the pitch becomes $\theta = 90$ deg, the roll and yaw become associated with the same degree of freedom and cannot be uniquely determined.

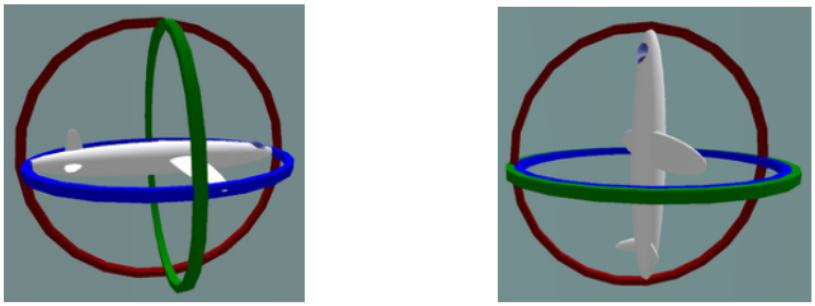


Figure 2.2: Look at what happens after the pitch - two axes align and we can't distinguish between roll and yaw.

Hence, two big problems with euler - it's confusing and gimbal lock.

2.4.2 Axis Angle Representation

Cross Product Review

The **cross product** of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$ is also a vector in $\mathbb{R}^3$:

$$\mathbf{x} \times \mathbf{y} := \begin{bmatrix} x_2y_3 - x_3y_2 \\ x_3y_1 - x_1y_3 \\ x_1y_2 - x_2y_1 \end{bmatrix} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \hat{\mathbf{x}}\mathbf{y}$$

$\hat{x}$ is the skew symmetric matrix used to represent the x that is used for cross product. It is also called the **hat map**.

The **hat map** $\hat{\cdot} : \mathbb{R}^3 \to \mathfrak{so}(3)$ transforms a vector $\mathbf{x} \in \mathbb{R}^3$ to a skew-symmetric matrix:

$$\hat{\mathbf{x}} := \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \qquad \hat{\mathbf{x}}^\top = -\hat{\mathbf{x}}$$

The vector space $\mathbb{R}^3$ and the space of skew-symmetric 3×3 matrices $so(3)$ are isomorphic, i.e., there exists a one-to-one map (the hat map) that preserves their structure.

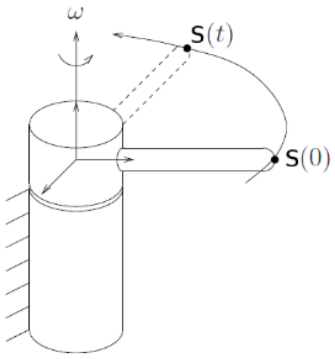
The **vee map** (v because its the upside down version of the hat) simply obtains x back from the skew-symmetric matrix $\hat{x}$.

Hat map properties: for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$, $A \in \mathbb{R}^{3\times3}$:

- $\hat{\mathbf{x}}\mathbf{y} = \mathbf{x} \times \mathbf{y} = -\mathbf{y} \times \mathbf{x} = -\hat{\mathbf{y}}\mathbf{x}$
- $\hat{\mathbf{x}}^2 = \mathbf{x}\mathbf{x}^\top - \mathbf{x}^\top\mathbf{x}\, I$
- $\hat{\mathbf{x}}^{2k+1} = (-\mathbf{x}^\top\mathbf{x})^k\hat{\mathbf{x}}$
- $-\frac{1}{2}\,\mathrm{tr}(\hat{\mathbf{x}}\hat{\mathbf{y}}) = \mathbf{x}^\top\mathbf{y}$
- $\hat{\mathbf{x}}A + A^\top\hat{\mathbf{x}} = ((\mathrm{tr}(A)I - A)\mathbf{x})^\wedge$
- $\mathrm{tr}(\hat{\mathbf{x}}A) = \frac{1}{2}\,\mathrm{tr}(\hat{\mathbf{x}}(A - A^\top)) = -\mathbf{x}^\top(A - A^\top)^\vee$
- $(A\mathbf{x})^\wedge = \det(A)A^{-\top}\hat{\mathbf{x}}A^{-1}$

Coming back to axis-angle.

Let's assume that we have a stick on a cylinder that is rotating about the cylinder. Let us consider a point s at the end of this stick.



Let the point have a constant unit velocity, such that:

$$\dot{s}(t) = \eta \times s(t) = \hat{\eta} s(t)$$

Now, $s(t) = R_z(t)s(0)$.

This is simply a linear time-invariant system of ordinary differential equations determined by the skew symmetric matrix $\hat{\eta}$. The solution for this can be written as (this is a standard solution):

$$s(t) = \exp(\hat{\eta} t s(0))$$

$$s(t) = R_n(t)s(0)$$

We can just replace the t with θ,

$$s(t) = \exp(\hat{\eta} \theta s(0))$$

$$s(t) = R_n(\theta)s(0)$$

This is the exponential map:

Look at Hao Su's description of axis angle and the conversions there.

Exponential map $\exp : \mathfrak{so}(3) \mapsto SO(3)$ maps a skew-symmetric matrix $\hat{\boldsymbol{\theta}}$ obtained from an axis-angle vector $\boldsymbol{\theta}$ to a rotation matrix R:

$$R = \exp(\hat{\boldsymbol{\theta}}) := \sum_{n=0}^{\infty} \frac{1}{n!} \hat{\boldsymbol{\theta}}^n = I + \hat{\boldsymbol{\theta}} + \frac{1}{2!}\hat{\boldsymbol{\theta}}^2 + \frac{1}{3!}\hat{\boldsymbol{\theta}}^3 + \dots$$

The skwigly so is simply the set of skew symmetric matrices. The exponential map maps every skew symmetric matix to a special orthogonal group rotation.

exp is not the same as taking exponential of every element, but rather:

$$\exp(A) = I + A + \frac{1}{2}AA + \frac{1}{3!}AAA$$

The exponential map is **surjective** but **not injective**: every element of $SO(3)$ can be generated from multiple elements of $\mathfrak{so}(3)$, e.g., any vector $(\|\boldsymbol{\theta}\| + 2\pi k)\,\frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}$ for integer k leads to the same R

The exponential map is **not commutative**: $e^{\hat{\boldsymbol{\theta}}_1} e^{\hat{\boldsymbol{\theta}}_2} \neq e^{\hat{\boldsymbol{\theta}}_2} e^{\hat{\boldsymbol{\theta}}_1} \neq e^{\hat{\boldsymbol{\theta}}_1 + \hat{\boldsymbol{\theta}}_2}$, unless $\hat{\boldsymbol{\theta}}_1\hat{\boldsymbol{\theta}}_2 - \hat{\boldsymbol{\theta}}_2\hat{\boldsymbol{\theta}}_1 = 0$

Rodriguez Formula

It is the closed form expression for the exponential map from the skew symmetric space to special orthogonal group:

$$R = \exp(\hat{\boldsymbol{\theta}}) = I + \left(\frac{\sin\|\boldsymbol{\theta}\|}{\|\boldsymbol{\theta}\|}\right)\hat{\boldsymbol{\theta}} + \left(\frac{1 - \cos\|\boldsymbol{\theta}\|}{\|\boldsymbol{\theta}\|^2}\right)\hat{\boldsymbol{\theta}}^2$$

An important property to derive this is that $\hat{x}\hat{x} = xx^T - x^T x I$. Further, any odd power of $\hat{x}$ can be written as some product of x.

The formula is derived using that $\hat{\boldsymbol{\theta}}^{2n+1} = (-\boldsymbol{\theta}^\top \boldsymbol{\theta})^n \hat{\boldsymbol{\theta}}$:

$$\begin{aligned}
\exp(\hat{\boldsymbol{\theta}}) &= I + \sum_{n=1}^{\infty} \frac{1}{n!} \hat{\boldsymbol{\theta}}^n \\
&= I + \sum_{n=0}^{\infty} \frac{1}{(2n+1)!} \hat{\boldsymbol{\theta}}^{2n+1} + \sum_{n=0}^{\infty} \frac{1}{(2n+2)!} \hat{\boldsymbol{\theta}}^{2n+2} \\
&= I + \left(\sum_{n=0}^{\infty} \frac{(-1)^n \|\boldsymbol{\theta}\|^{2n}}{(2n+1)!} \right) \hat{\boldsymbol{\theta}} + \left(\sum_{n=0}^{\infty} \frac{(-1)^n \|\boldsymbol{\theta}\|^{2n}}{(2n+2)!} \right) \hat{\boldsymbol{\theta}}^2 \\
&= I + \left(\frac{\sin \|\boldsymbol{\theta}\|}{\|\boldsymbol{\theta}\|} \right) \hat{\boldsymbol{\theta}} + \left(\frac{1 - \cos \|\boldsymbol{\theta}\|}{\|\boldsymbol{\theta}\|^2} \right) \hat{\boldsymbol{\theta}}^2
\end{aligned}$$

In the above equation, we split the infinite sum into odd powers and even powers respectively (look at the superscript).

Then, we subtitute the odd power of $\hat{\theta}$ with the equation that expresses it as a product of $\hat{\theta}$.

The final line is simply the expansion of sin and cos.

Why do we need it?

This is faster, closed form. Practically, doesn't matter but this is better obviously.

Log Map

There is a log map that simply inverts the exponential - it goes from SO group to the skew symmetric group.

$\forall R \in SO(3)$, there exists a (non-unique) $\boldsymbol{\theta} \in \mathbb{R}^3$ such that $R = \exp(\hat{\boldsymbol{\theta}})$

Logarithm map $\log : SO(3) \to \mathfrak{so}(3)$ is the inverse of $\exp(\hat{\boldsymbol{\theta}})$:

$$\theta = \|\boldsymbol{\theta}\| = \arccos\left(\frac{\mathrm{tr}(R) - 1}{2}\right)$$

$$\boldsymbol{\eta} = \frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|} = \frac{1}{2\sin(\|\boldsymbol{\theta}\|)} \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix}$$

$$\hat{\boldsymbol{\theta}} = \log(R) = \frac{\|\boldsymbol{\theta}\|}{2\sin\|\boldsymbol{\theta}\|}(R - R^\top)$$

- If $R = I$, then $\theta = 0$ and $\boldsymbol{\eta}$ is undefined
- If $\mathrm{tr}(R) = -1$, then $\theta = \pi$ and for any $i \in \{1, 2, 3\}$:

$$\boldsymbol{\eta} = \frac{1}{\sqrt{2(1 + R_{ii})}}(I + R)e_i$$

Figure 2.3: All this is derived from the rodriguez formula

The log map is not unique - so it's not a true inverse. R is not truly invertible.

The matrix exponential "integrates" $\hat{\theta} \in se(3)$ for one second; the matrix logarithm "differentiates" $R \in SO(3)$ to obtain $\hat{\theta} \in se(3)$.

2.4.3 Quaternion

We now look for a representation that doesn't have singularity. Quaternions are a solution for this, and are an extension of the complex number idea that we had disucssed earlier.

Quaternions: $\mathbb{H} = \mathbb{C} + \mathbb{C}j$ generalize complex numbers $\mathbb{C} = \mathbb{R} + \mathbb{R}i$

$$\mathbf{q} = q_s + q_1 i + q_2 j + q_3 k = [q_s,\ \mathbf{q}_v] \qquad ij = -ji = k,\ i^2 = j^2 = k^2 = -1$$

Here, $\mathbb{H}$ is the quaternion space. $\mathbb{H}_*$ is the space of unit-norm quaternions:

$$\mathbb{H}_* := q \in \mathbb{H} | q_s^2 + q_v^T q_v = 1$$

A vector with a unit-norm is essentially a sphere, so quaternions are 4-D spheres. The surface of the sphere is 3-D dimensional. Hence, every surface of the sphere is a rotation matrix. More accurately, only half of the space is a rotation matrix.

A rotation matrix $R \in SO(3)$ can be obtained from a unit quaternion $\mathbf{q}$:

$$R(\mathbf{q}) = E(\mathbf{q})G(\mathbf{q})^\top \qquad \begin{aligned} E(\mathbf{q}) &= [-\mathbf{q}_v,\ q_s I + \hat{\mathbf{q}}_v] \\ G(\mathbf{q}) &= [-\mathbf{q}_v,\ q_s I - \hat{\mathbf{q}}_v] \end{aligned}$$

The fact that only half the sphere is relevant is called double covering of quaternions - $R(q) = R(-q)$.

We can also convert quaternion from axis-angle representation.

A rotation around a unit axis $\boldsymbol{\eta} := \frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|} \in \mathbb{R}^3$ by angle $\theta := \|\boldsymbol{\theta}\|$ can be represented by a unit quaternion:

$$\mathbf{q} = \left[\cos\left(\frac{\theta}{2}\right),\ \sin\left(\frac{\theta}{2}\right)\boldsymbol{\eta}\right] \in \mathbb{H}_*$$

A rotation around a unit axis $\boldsymbol{\eta} \in \mathbb{R}^3$ by angle θ can be recovered from a unit quaternion $\mathbf{q} = [q_s,\ \mathbf{q}_v] \in \mathbb{H}_*$:

$$\theta = 2\arccos(q_s) \qquad \boldsymbol{\eta} = \begin{cases} \frac{1}{\sin(\theta/2)}\mathbf{q}_v, & \text{if } \theta \neq 0 \\ 0, & \text{if } \theta = 0 \end{cases}$$

Quaternion Operations

Addition	$\mathbf{q} + \mathbf{p} := [q_s + p_s,\ \mathbf{q}_v + \mathbf{p}_v]$
Multiplication	$\mathbf{q} \circ \mathbf{p} := \left[q_s p_s - \mathbf{q}_v^T \mathbf{p}_v,\ q_s \mathbf{p}_v + p_s \mathbf{q}_v + \mathbf{q}_v \times \mathbf{p}_v\right]$
Conjugation	$\bar{\mathbf{q}} := [q_s,\ -\mathbf{q}_v]$
Norm	$\|\mathbf{q}\| := \sqrt{q_s^2 + \mathbf{q}_v^T \mathbf{q}_v} \qquad \|\mathbf{q} \circ \mathbf{p}\| = \|\mathbf{q}\|\|\mathbf{p}\|$
Inverse	$\mathbf{q}^{-1} := \frac{\bar{\mathbf{q}}}{\|\mathbf{q}\|^2}$
Rotation	$[0,\ \mathbf{x}'] = \mathbf{q} \circ [0,\ \mathbf{x}] \circ \mathbf{q}^{-1} = [0,\ R(\mathbf{q})\mathbf{x}]$
Velocity	$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \circ [0,\ \boldsymbol{\omega}] = \frac{1}{2} G(\mathbf{q})^\top \boldsymbol{\omega}$
Exp	$\exp(\mathbf{q}) := e^{q_s} \left[\cos \|\mathbf{q}_v\|,\ \frac{\mathbf{q}_v}{\|\mathbf{q}_v\|} \sin \|\mathbf{q}_v\|\right]$
Log	$\log(\mathbf{q}) := \left[\log \|\mathbf{q}\|,\ \frac{\mathbf{q}_v}{\|\mathbf{q}_v\|} \arccos \frac{q_s}{\|\mathbf{q}\|}\right]$