MROVER

MICHIGAN MARS ROVER TEAM

Spatial Geometry for Robotics

2/4/2024 | Riley Bridges (rlybrdgs)

What is Spatial Geometry?

- Geometry/math of where things are and how they move in space
- Useful for math and algorithms involving physical things in the real world
- Seems look it should be simple, but actually is a very large field
    - We are just going to scratch the surface of some useful concepts

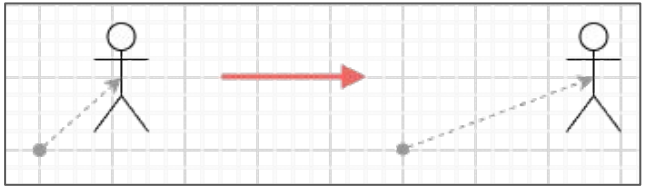
Why Does This Matter?

Some applications include:

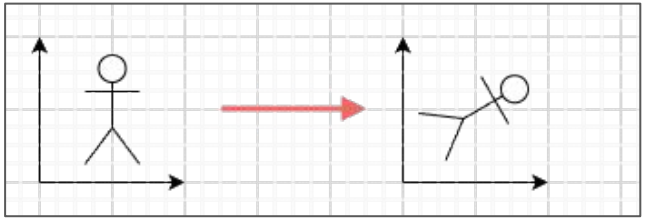
- Pretty much all of robotics
- Computer vision
- Virtual/augmented reality
- Computer graphics/rendering
- Physics simulation
- Video games
- Airplanes and satellites
- Kinematics & dynamics modeling
- Control theory

Pose

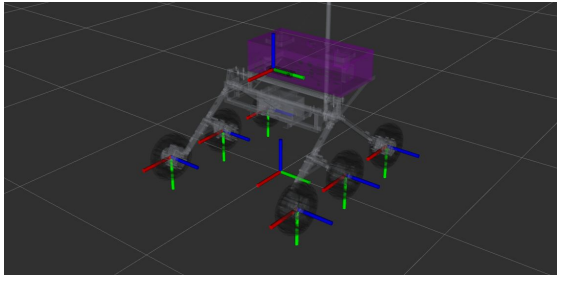
- **Position** and **orientation** of an object in space
  - Position: where the object is
  - Orientation: how the object is rotated
- Defined relative to a **coordinate frame**
  - “Pose of object *in* frame A”
  - Can attach frame B to object
  - -> “Pose of frame B *in* frame A”
- Fully describes how an object is placed
- We care about 2D and 3D



Change in position



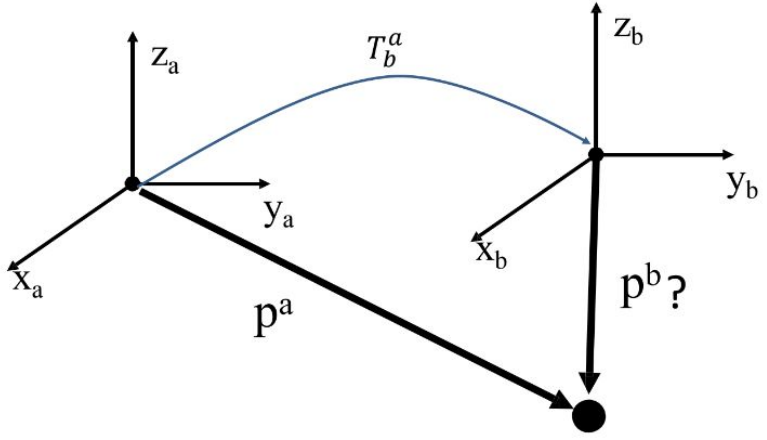
Change in orientation



Coordinate frames attached to rover

Transform

- **Translation** and **Rotation** to move between two coordinate frames
  - “Transform from frame B *to* frame A”
  - Maps point in frame B -> point in frame A
- Contains same information as pose
- Subtle difference:
  - Pose: frame B in frame A
  - Transform: frame B to frame A
- This can get confusing!
  - Rule of thumb: **B in A = B to A**



Pose Representations

What is a pose representation?

- Numerical way to define an object's pose
- Necessary to represent pose in software

What do we want from a pose representation? - Talk to your neighbor

Pose Representations

What is a pose representation?

- Numerical way to define an object's pose
- Necessary to represent pose in software

What do we want from a pose representation?

- Easy to understand/visualize
- Supports common pose operations
- Common operations are mathematically and computationally nice
  - Simple, easy to implement, fast to compute, numerically stable
- Compact: requires little memory to store

Common Pose Operations

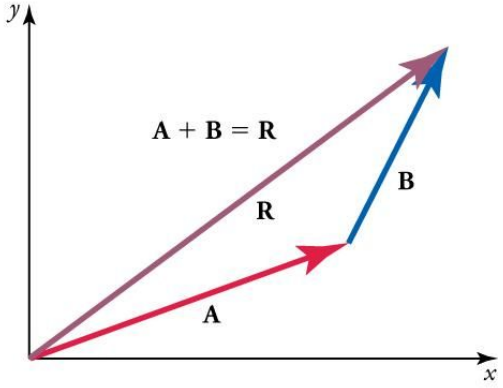
- Composition: $P_C^A = P_C^B \cdot P_B^A$
- Inverse: $P_A^B = \left(P_B^A\right)^{-1}$
- Act on a point/vector: $v_B = T_B^A \cdot v_A$
- Conversion to other representations

P_B^A = pose of frame A in frame B = T_B^A = transform from frame A to frame B

Representing Position

- We can just use a 2D or 3D vector: $\begin{bmatrix} x \\ y \end{bmatrix}$ $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$
  - Vector directly represents position
  - Any real vector -> valid, unique position

- Common operations are simple
  - Composition: vector addition $p_C^A = p_C^B + p_B^A$
  - Action on a point: vector addition $p_B = p_B^A + p^A$
  - Inverse: multiply by -1 $p_A^B = -p_B^A$



2D Orientation

Properties

- 1 degree of freedom: direction object is facing in 2D plane
- Composition order doesn't matter (commutative)

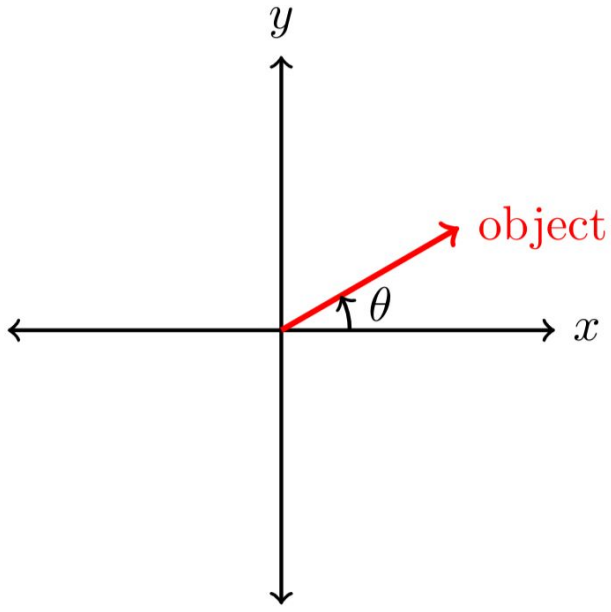
$$R_1 \cdot R_2 = R_2 \cdot R_1$$

Representations

- Angle
- 2D rotation matrix
- Unit complex number

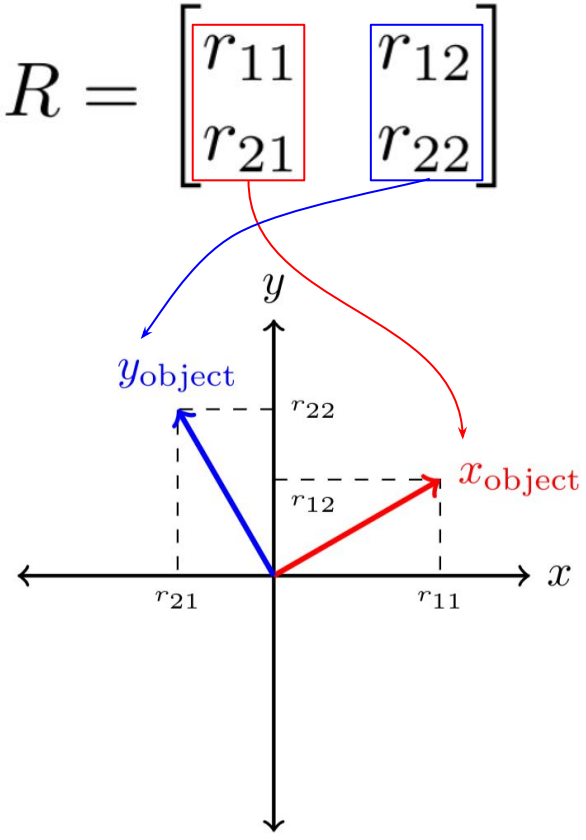
2D Angle of Rotation

- Angle from reference axis
- Simple and intuitive
- Operations aren’t great
  - Angle wrap around
  - Trig functions



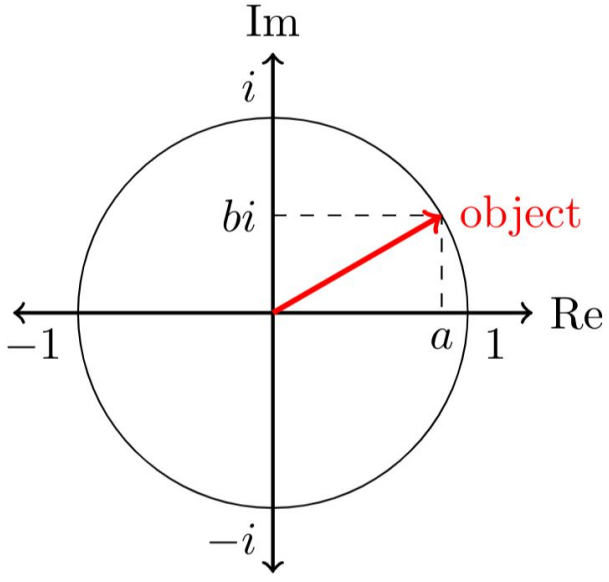
2D Rotation Matrices

- 2x2 orthonormal matrix
- Each column = axis of rotated frame
- 2D special orthogonal group: **SO(2)**



Unit Complex Number

- $z = a + bi, \quad \sqrt{a^2 + b^2} = 1$
- vector on the unit circle in the complex plane
- Operations are nice because of complex multiplication



3D Orientation

Properties

- 3 degrees of freedom: rotation about X, Y, and Z axes
- Composition order **does** matter (not commutative):

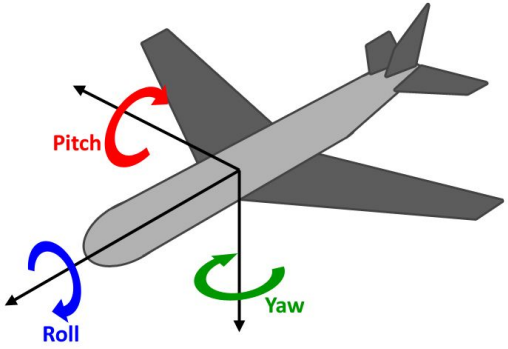
$$R_1 \cdot R_2 \neq R_2 \cdot R_1$$

Representations

- Euler Angles
- Rotation Matrix
- Axis Angle/Rotation Vector
- Quaternion

Euler Angles

- Rotation by 3 angles, each about a different axis
- Rotations applied sequentially
- Often called roll (x axis), pitch (y axis), yaw (z axis)

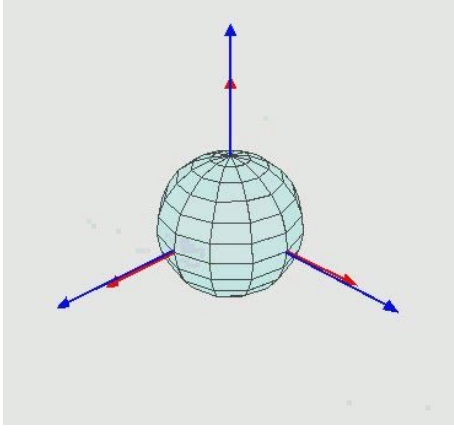


Pros

- Easy to understand
- Compact

Cons

- Many different conventions
- Operations are terrible
- Singularities (gimbal lock)

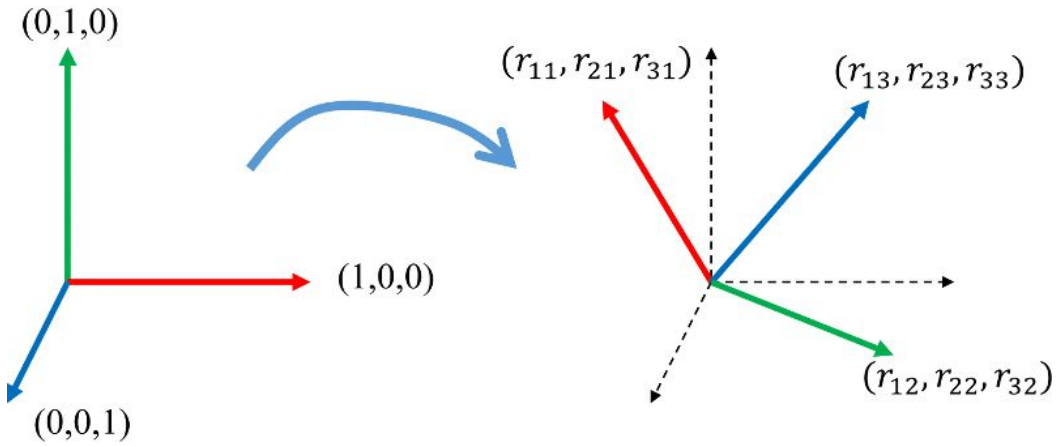


3D Rotation Matrices

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

x vector y vector z vector

- 3x3 orthonormal matrix
- Each column = axis of rotated frame
- 3D Special orthogonal group: **SO(3)**
- Pros
  - somewhat intuitive
  - no singularities
  - Operations are simple
- Cons
  - Less compact
  - Operations are slower



Axis Angle/Rotation Vectors

- Euler's Theorem: can reach any orientation from a single rotation about some axis
- unit vector representing axis of rotation, multiplied by angle of rotation:

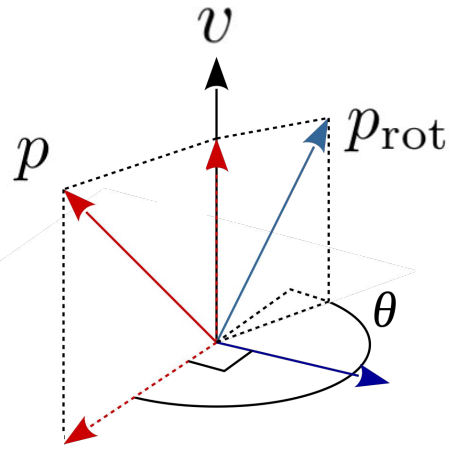
$$\begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}^\top \cdot \theta$$

Pros

- compact
- fairly intuitive
- operations aren't too bad

Cons

- singularities (theta = 0, v = 0)



Quaternions

- unit vector on 4d complex hypersphere (yikes!)
- Rough intuition: [x y z] = axis, w = angle
- $\mathbb{S}^3$ group: double coverage of SO(3)

$$\mathbf{q} = \begin{bmatrix} x & y & z & w \end{bmatrix}^\top$$
$$= w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$$

Pros

- no singularities
- operations are simple and fast

Cons

- very unintuitive
- have to be careful about double cover

Putting Position and Rotation Back Together

Pose is position and orientation: $P = (R, t)$

Combined operations

- Composition: $(R_1, t_1) \cdot (R_2, t_2) = (R_1 \cdot R_2, R_1 \cdot t_2 + t_1)$
- Act on a point: $(R, t) \cdot v = R \cdot v + t$
- Inverse: $(R, t)^{-1} = (R^{-1}, -R^{-1}t)$

We can put this in matrix form…

Homogeneous Transform Matrices

When using rotation matrices, can represent entire pose with one matrix:

$$M = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

…in the **SE(3)** group

Common operations become matrix operations:

- Composition: $M_1 \cdot M_2 = M_1 M_2$ (matrix multiplication)
- Act on a point: $M \cdot v = Mv$
- Inverse: $M^{-1} = M^{-1}$ (matrix inverse)

Using This Stuff in Practice

What representation should you use?

- Quaternions: common for high performance applications
- Rotation matrices: good for experimentation and less speed critical things
- Or… whatever your library uses

What library should you use?

- manif - simple, fast, supports C++ and Python… coming soon to mrover-ros!
- Sophus - C++ and Python
- smooth - C++20 features, autodiff, integration, fancy optimization tools

Example

- We have:
  - Pose of AR tag in camera frame: `SE3 tagInCam`
  - Transform from world frame to camera frame: `SE3 worldToCam`

- We want to know pose of AR tag in world frame: `SE3 tagInWorld`
  … how do we get it?
  - Remember operations: composition, inverse
  - B in A = B to A

Example

- We have:
  - Pose of AR tag in camera frame: `SE3 tagInCam`
  - Transform from world frame to camera frame: `SE3 worldToCam`

- We want to know pose of AR tag in world frame: `SE3 tagInWorld`
  … how do we get it?
  - Remember operations: composition, inverse
  - B in A = B to A

- Answer:

```
SE3 CamToWorld = WorldToCam.inverse();
SE3 tagInWorld = camToWorld * tagInCam;
```

Further Reading

- Groups, exponential coordinates, screw theory:
  A Mathematical Introduction to Robotic Manipulation
- Lie groups/Lie theory:
  A micro Lie theory for state estimation in robotics
- 3D geometry and transforms:
  Dmitry Berenson's slides
- Various wikipedia pages:
  - https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula
  - https://en.wikipedia.org/wiki/Orientation_(geometry)
  - https://en.wikipedia.org/wiki/Euler_angles
  - https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation

2D Pose Representations: Common Operations

Euler Angle Operations

- Composition
-

Rotation Matrix Operations