اصول علم ربات – جلسه پنجم

Fundamentals of Robotics – Lecture 06

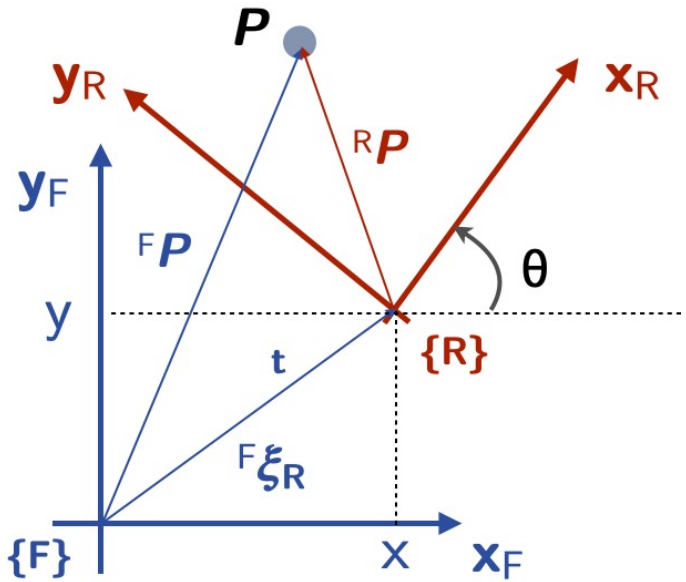
Pose Representations and Transformations-3D

دکتر مهدی جوانمردی

زمستان ۱۴۰۰

[slides adapted from Gianni Di Caro, @CMU with permission]

Recap: Roto-translation of a frame (coordinate transformation)



$$ {}^{F}\mathbf{P} = \begin{bmatrix} {}^{F}\mathrm{x} \\ {}^{F}\mathrm{y} \end{bmatrix} = \underbrace{\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} {}^{R}\mathrm{x} \\ {}^{R}\mathrm{y} \end{bmatrix}}_{Rotation\ {}^{R}\mathbf{P}} + \underbrace{\begin{bmatrix} \mathrm{x} \\ \mathrm{y} \end{bmatrix}}_{Translation} $$

$${}^{F}\mathbf{P}$$

✓ In a more compact form, using **homogeneous vectors**

$$ \begin{bmatrix} {}^{F}\mathrm{x} \\ {}^{F}\mathrm{y} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & \mathrm{x} \\ \sin(\theta) & \cos(\theta) & \mathrm{y} \end{bmatrix} \begin{bmatrix} {}^{R}\mathrm{x} \\ {}^{R}\mathrm{y} \\ 1 \end{bmatrix} $$

$$ {}^{F}\tilde{\mathbf{P}} = \underbrace{\begin{bmatrix} {}^{F}\mathbf{R}_{R} & \mathbf{t} \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix}}_{{}^{F}\mathbf{T}_{R}} {}^{R}\tilde{\mathbf{P}} $$

$$ \begin{bmatrix} {}^{F}\mathrm{x} \\ {}^{F}\mathrm{y} \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{F}\mathbf{R}_{R} & \mathbf{t} \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix} \begin{bmatrix} {}^{R}\mathrm{x} \\ {}^{R}\mathrm{y} \\ 1 \end{bmatrix} $$

Recap: Homogeneous transformation matrix for pose transformation

$$ {}^{F}\tilde{\mathbf{P}} = {}^{F}\mathbf{T}_{R}\,{}^{R}\tilde{\mathbf{P}} \qquad {}^{F}\mathbf{T}_{R} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & \mathrm{x} \\ \sin(\theta) & \cos(\theta) & \mathrm{y} \\ 0 & 0 & 1 \end{bmatrix} $$

The roto-translation matrix **T** is a ***homogenous transformation*** and belongs to SE(2)

The displacement of a robot / rigid body is fully described by a *homogeneous transformation matrix* ${}^{F}\mathbf{T}_{R}$

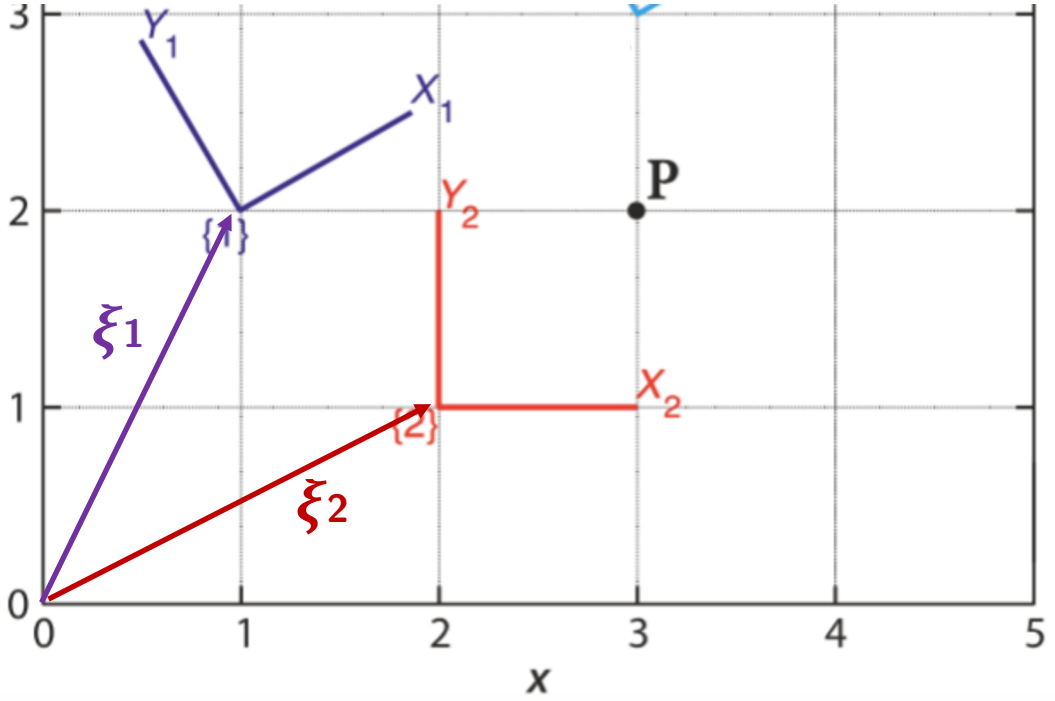
- Representation of **Relative pose** $\boldsymbol{\xi}$ of Frame {R} with respect to a Frame {F} $\quad {}^{F}\boldsymbol{\xi}_{R} \sim {}^{F}\mathbf{T}_{R}$
  - Pose composition: ${}^{A}\boldsymbol{\xi}_{R} \oplus {}^{R}\boldsymbol{\xi}_{B} \rightarrow$ Matrix multiplication: ${}^{A}T_{R} \cdot {}^{R}T_{B}$
  - Point coordinate transformation: ${}^{A}\tilde{\boldsymbol{P}} = {}^{A}\boldsymbol{\xi}_{B} \cdot {}^{B}\tilde{\boldsymbol{P}} \rightarrow$ Matrix-vector multiplication: ${}^{A}T_{B} \cdot {}^{B}\tilde{\boldsymbol{P}}$

Pose transformation example, no commutativity for poses

$\xi_1 = \xi_0 \oplus (1,\ 2,\ 30^o)$

$\xi_1 = I \cdot T_1$

$$T_1 = \begin{bmatrix} 0.866 & -0.5 & 1 \\ 0.5 & 0.866 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$



ξ_0 ~ (0,0,0)

$\xi_0 = (0, 0, 0) \oplus I$

$\xi_0 = T_0 \cdot I$

$$T_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$\xi_2 = \xi_0 \oplus (2,1,0^o)$

$$T_2 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

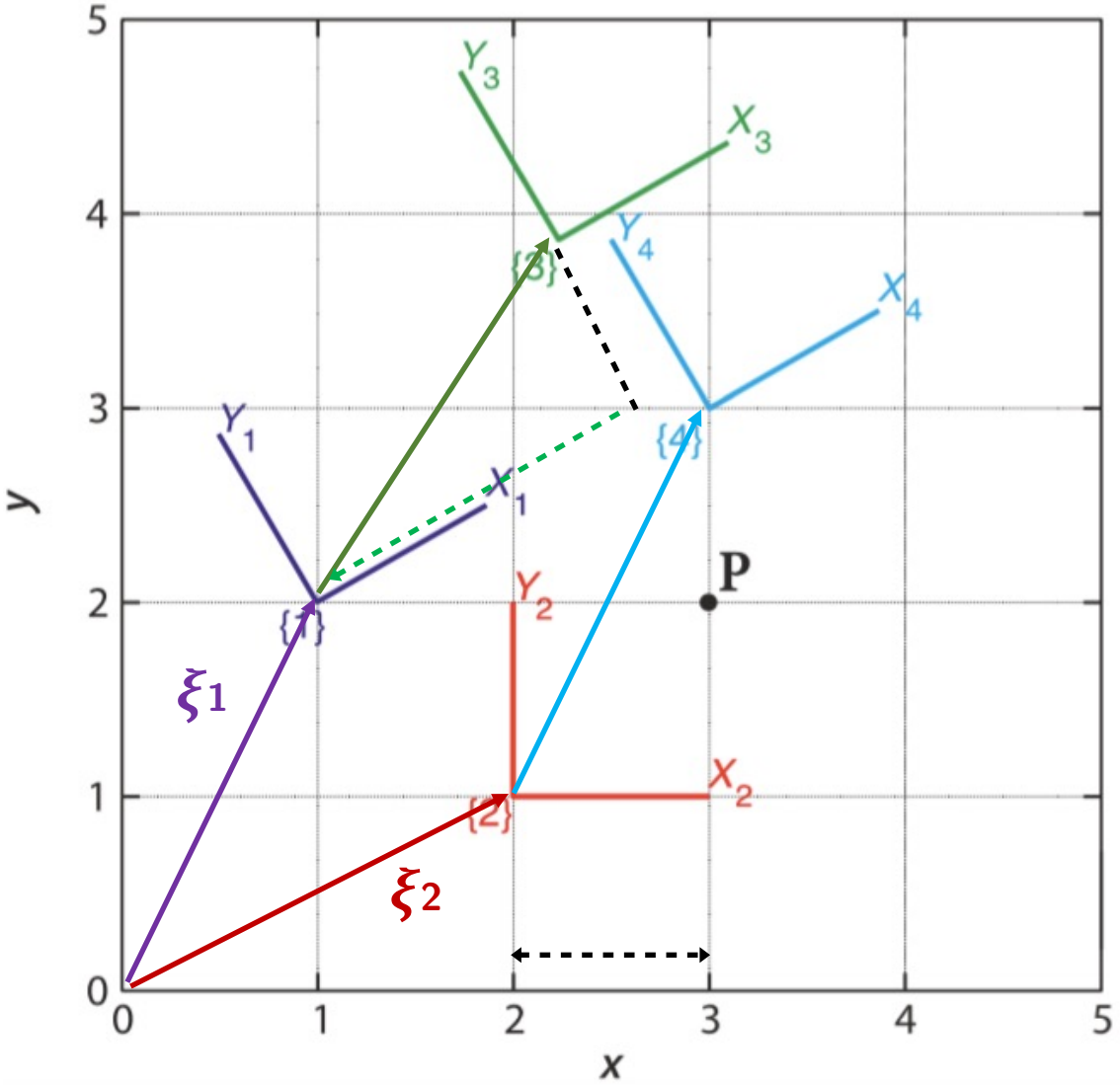
$\xi_2 = T_2 \cdot I$

? $\xi_3 = \xi_1 \oplus \xi_2$

? $\xi_4 = \xi_2 \oplus \xi_1$

Abusing notation, a bit ...

Pose transformation example, no commutativity for poses



$$\xi_3 = \xi_1 \oplus \xi_2$$

- ξ_2 represents a transformation T_2 consisting of a translation of 2 long *x*, 1 long *y*, plus a counterclockwise rotation of 0 degrees
- $\xi_3 = T_1 T_2$ means that T_2 is applied wrt T_1 : the origin of {3} has coordinates (2,1) in {1} and no further rotation is applied to the coordinate axes.
  - Note that the green dashed line is the projection of the origin of {3} along the *x* axis of {1}, which has lenght 2 has expected

$$\xi_4 = \xi_2 \oplus \xi_1$$

- $\xi_4 = T_2 T_1$ means that T_1 is applied wrt T_2 : the origin of {4} has coordinates (1,2) in {2} and is rotated of 30 degrees wrt {2}'s coordinate axes

Relative Poses in 3D? → 4×4 Homogeneous transformation matrices

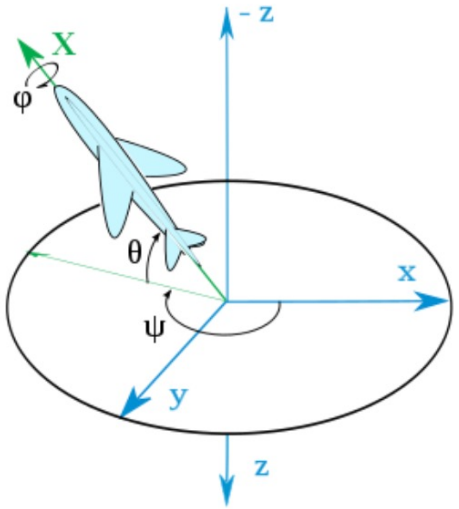
$$ {}^{F}\tilde{\mathbf{P}} = \underbrace{\begin{bmatrix} {}^{F}\mathbf{R}_{R} & \mathbf{t} \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix}}_{{}^{F}\mathbf{T}_{R}} {}^{R}\tilde{\mathbf{P}} $$

$$ {}^{F}\tilde{\mathbf{P}} = \underbrace{\begin{bmatrix} {}^{F}\mathbf{R}_{R}^{3\times 3} & \mathbf{t}_{3\times 1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}}_{{}^{F}\mathbf{T}_{R}} {}^{R}\tilde{\mathbf{P}} $$

$$ {}^{W}\boldsymbol{\xi} = [x\ y\ \theta] $$

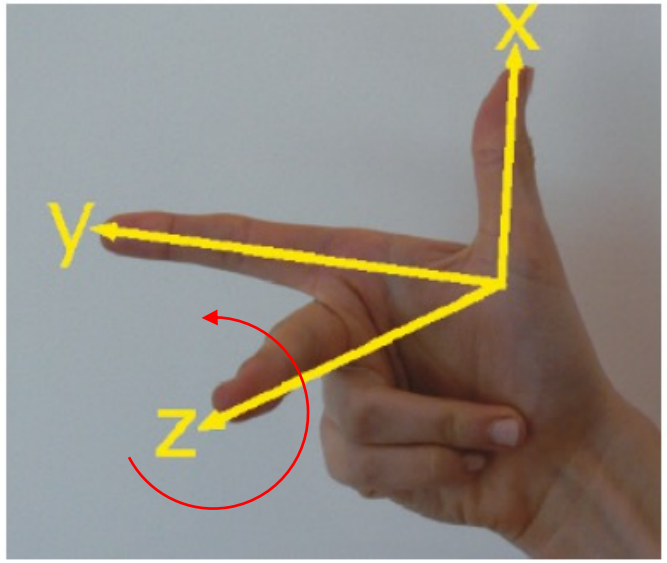
$$ {}^{W}\boldsymbol{\xi} = [x\ y\ z\ \theta\ \varphi\ \psi] $$

- ✓ Homogeneous transformation matrices are **redundant / implicit representations of a pose**
  - We use more parameters than strictly necessary
    - **2D**: 6 vs. 3
    - **3D**: 12 vs. 6
- ➢ Useful to avoid singularities in the configuration space related to the angular part of the poses
- ➢ Easier to manipulate and make calculations



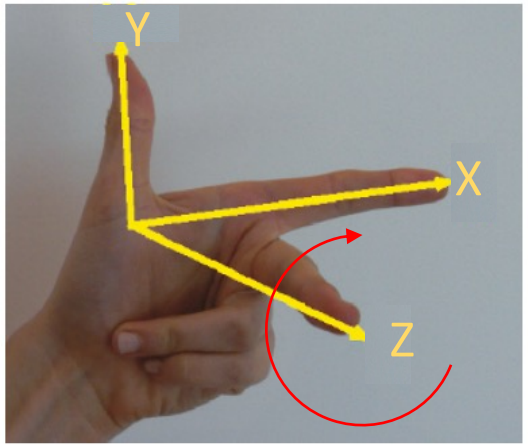
(Only) need 3 angles for orientation + 3 coordinates for positions

Relative orientation of coordinate axes in 3D



Right-hand rule for the relative orientation of the coordinate axes

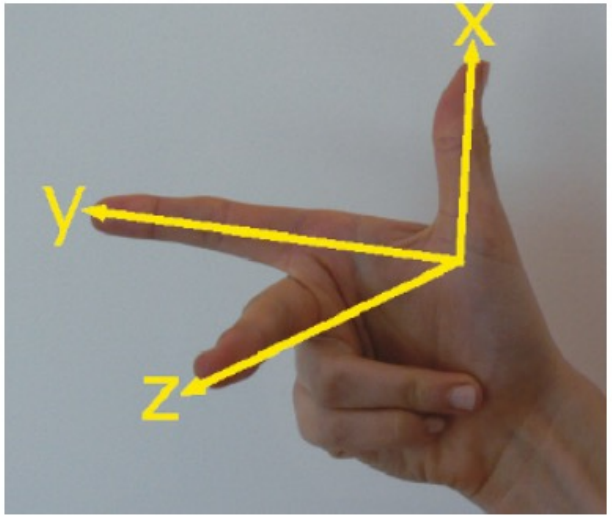
Positive rotation is **counterclockwise** about the axis of rotation.



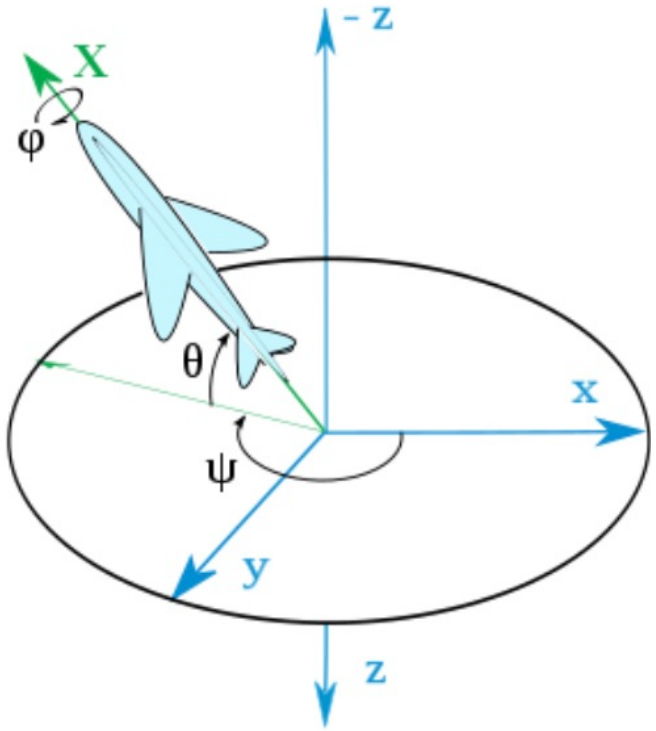
Left-hand rule for the relative orientation of the coordinate axes

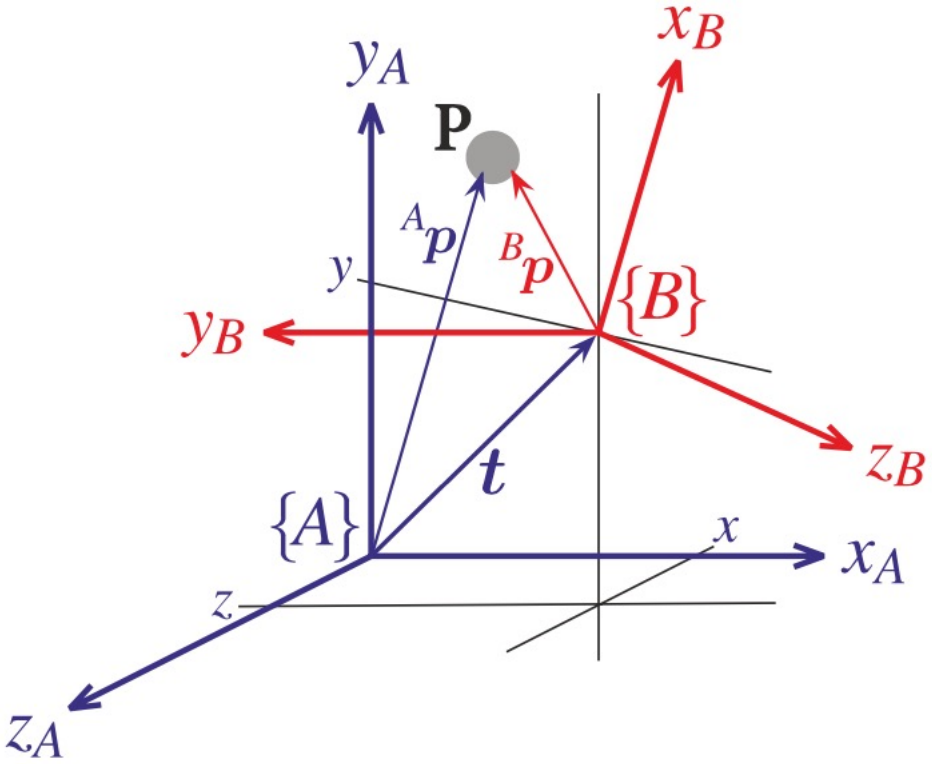
Positive rotation is **clockwise** about the axis of rotation.

Orientation of a rigid body in 3D



Right-hand rule for the relative orientation of the coordinate axes

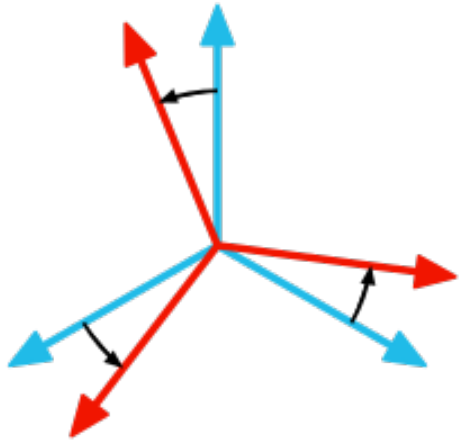


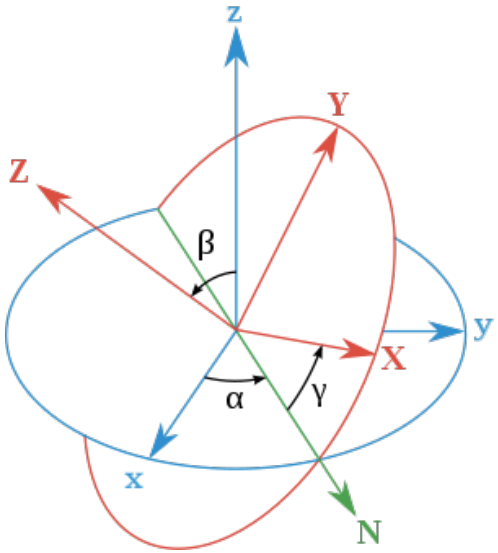


Transformation between coordinate frames: Euler's Theorem

- **Euler's Theorem**: Any two independent orthonormal coordinate frames can be related by a **sequence of rotations** (not more than **three**) about coordinate axes, where no two successive rotations may be about the same axis.

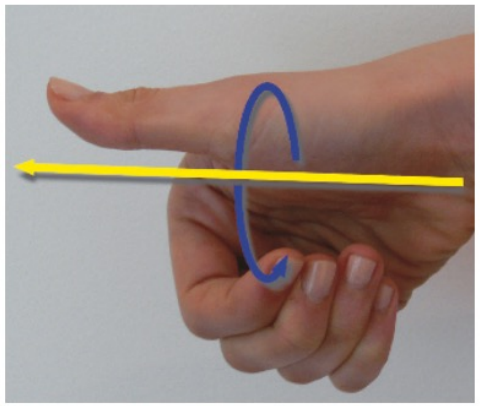
  - Any target orientation can be reached by composing three ***elemental*** rotations around the three coordinate axes



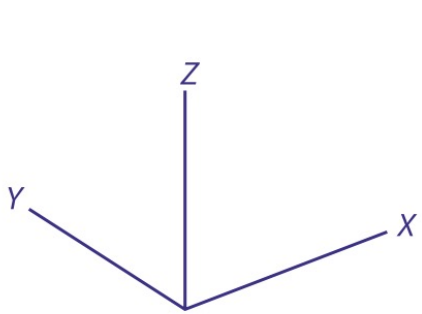


In 3-dimensions rotation is not commutative – the order in which rotations are applied makes a difference to the result.

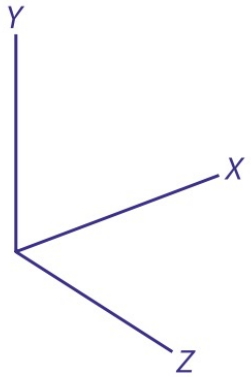
Transformation between coordinate frames: Euler's Theorem



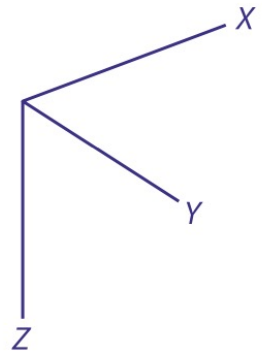
Rotation about a vector. Wrap your right hand around the vector with your thumb (your x-finger) in the direction of the arrow. The curl of your fingers indicates the direction of increasing angle.



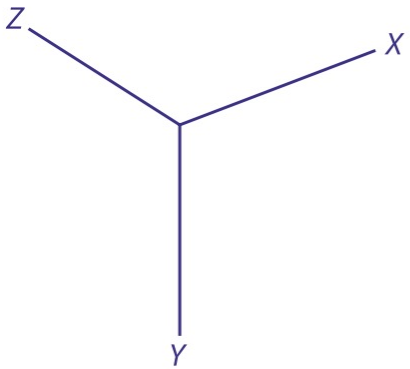
a Original



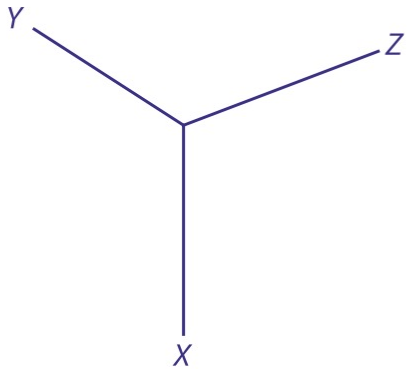
b $\frac{\pi}{2}$ about x-axis



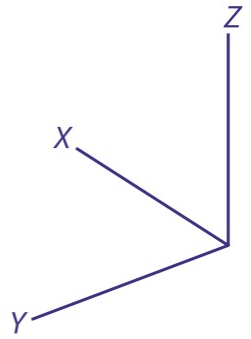
c π about x-axis



d $-\frac{\pi}{2}$ about x-axis



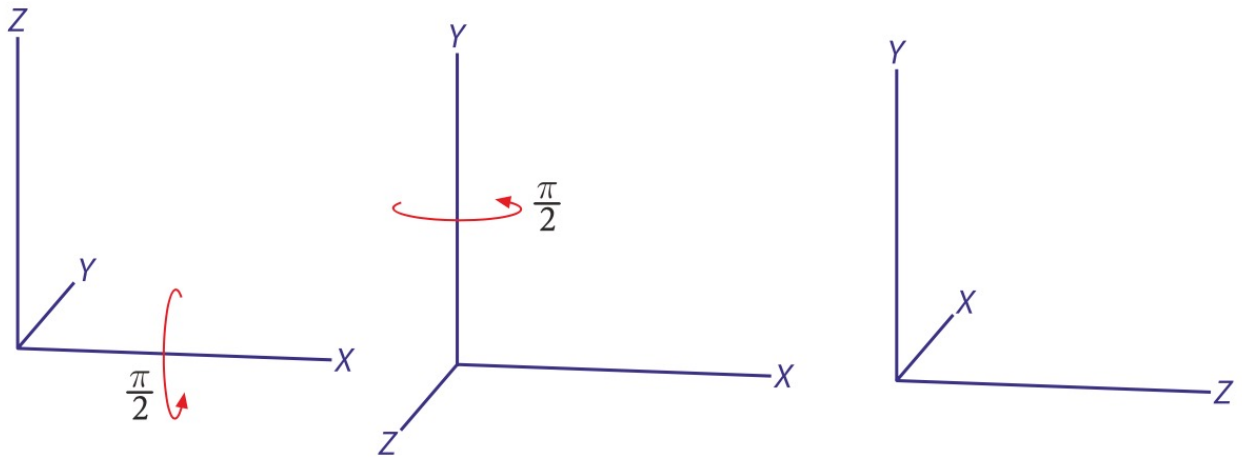
e $\frac{\pi}{2}$ about y-axis



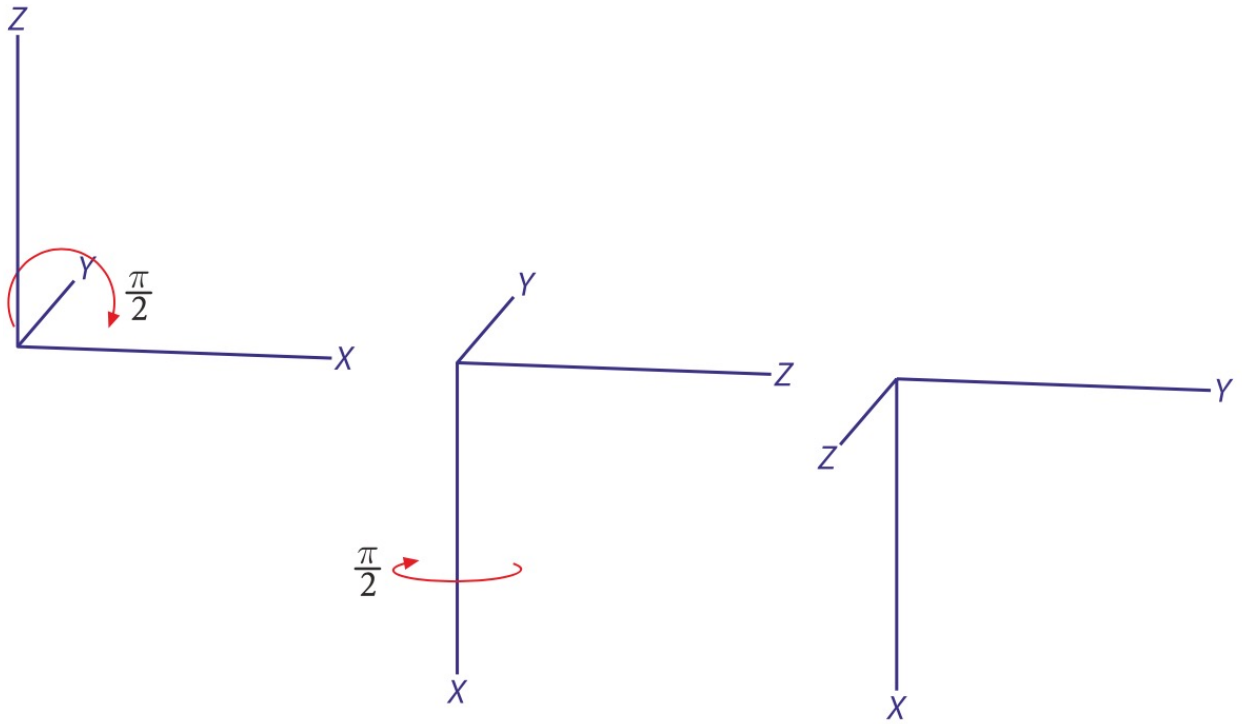
f $\frac{\pi}{2}$ about z-axis

Non-commutativity

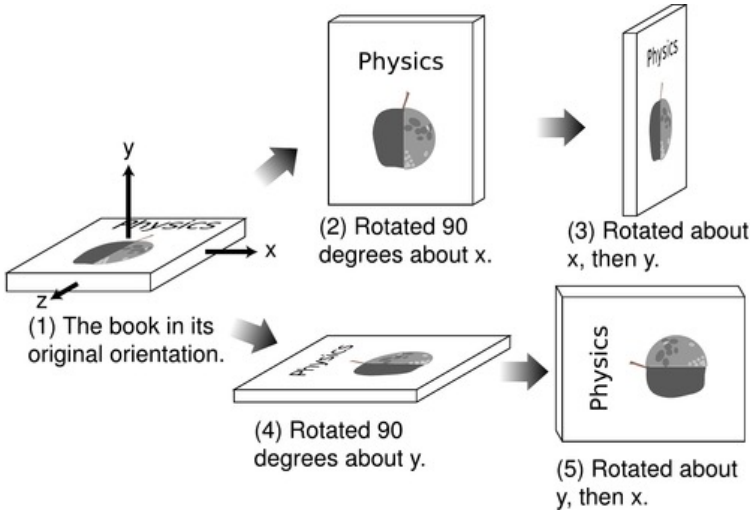
$X_{90} \oplus Y_{90}$

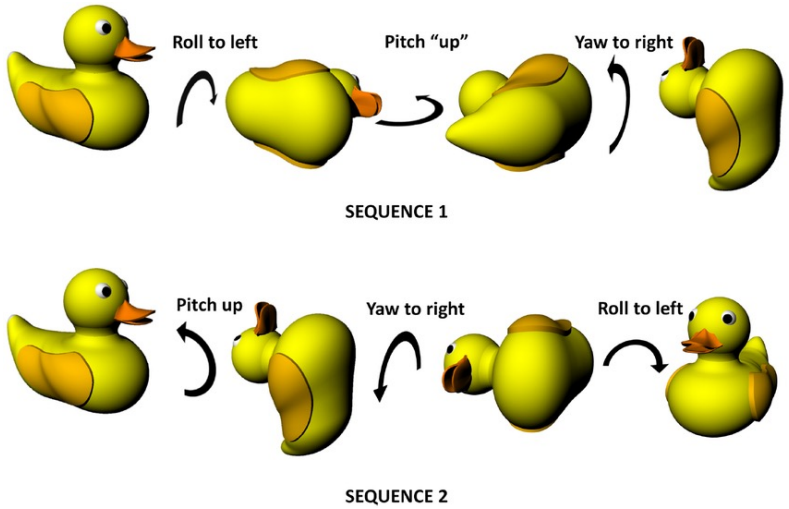


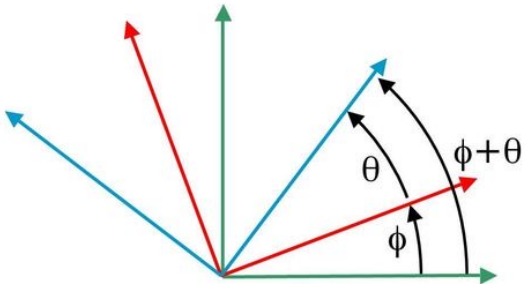
$Y_{90} \oplus X_{90}$



Rotations are not commutative in more than 2 dimensions



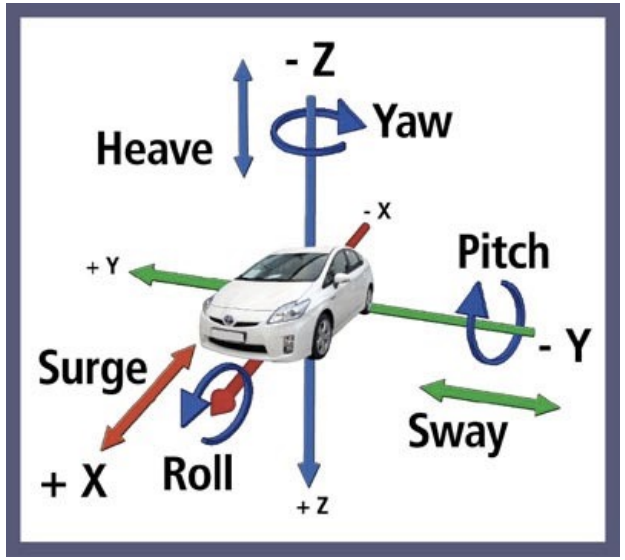


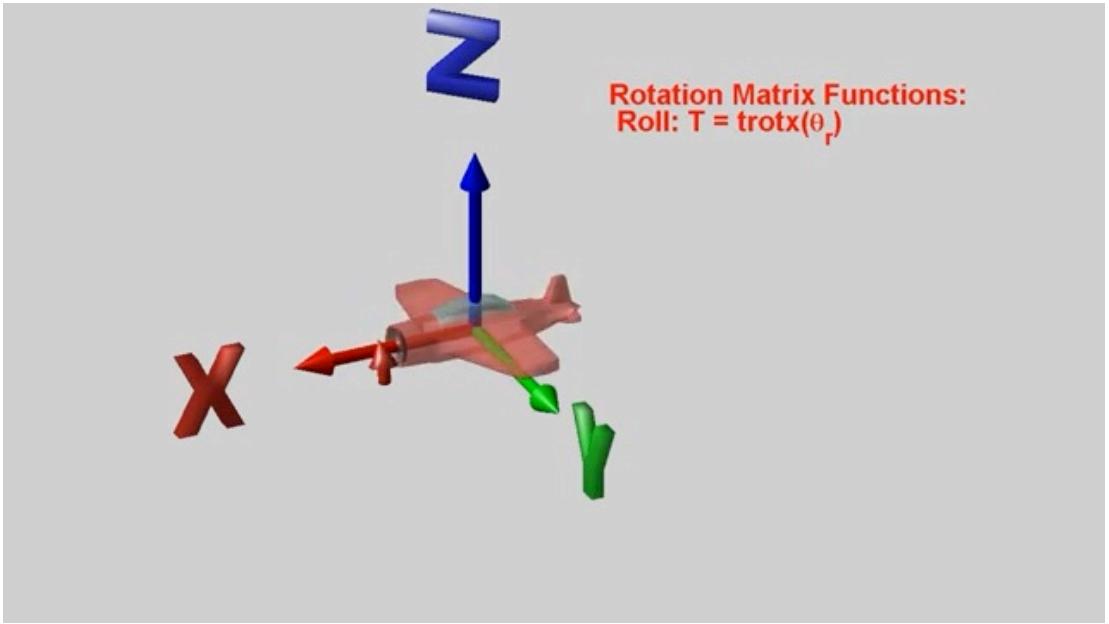


$$R_x(\frac{\pi}{4})\cdot R_y(\frac{\pi}{4}) = \begin{bmatrix} 0.707 & 0 & -0.707 \\ -0.5 & 0.707 & -0.5 \\ 0.5 & 0.707 & 0.5 \end{bmatrix},\ R_x(\frac{\pi}{4})\cdot R_y(\frac{\pi}{4})\cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -1.414 \\ 0.586 \\ 3.414 \end{bmatrix}$$

$$R_y(\frac{\pi}{4})\cdot R_x(\frac{\pi}{4}) = \begin{bmatrix} 0.707 & -0.5 & -0.5 \\ 0 & 0.707 & -0.707 \\ 0.707 & 0.5 & 0.5 \end{bmatrix},\ R_y(\frac{\pi}{4})\cdot R_x(\frac{\pi}{4})\cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -1.793 \\ 0.707 \\ 3.207 \end{bmatrix}$$

3D angles


- Z
Heave
Yaw
- X
+ Y
Pitch
- Y
Surge
Sway
+ X
Roll
+ Z

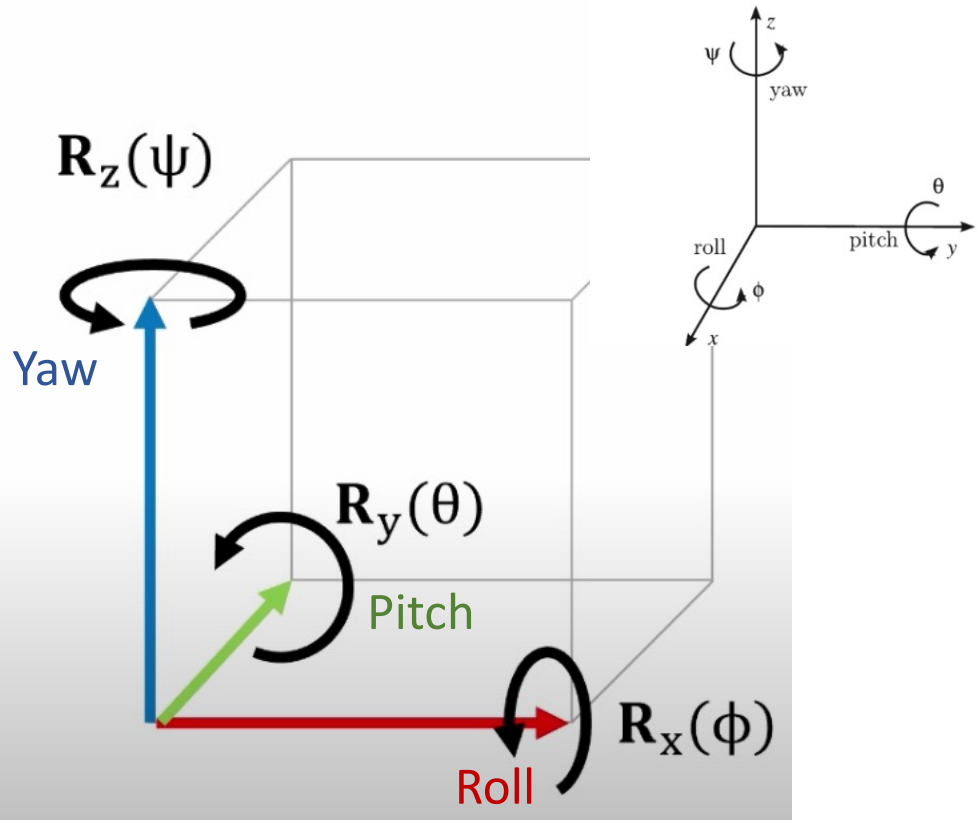

Z
Rotation Matrix Functions:
Roll: T = trotx(θr)
X
Y

3D rotations about the axes (elementary rotation matrices)

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

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

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

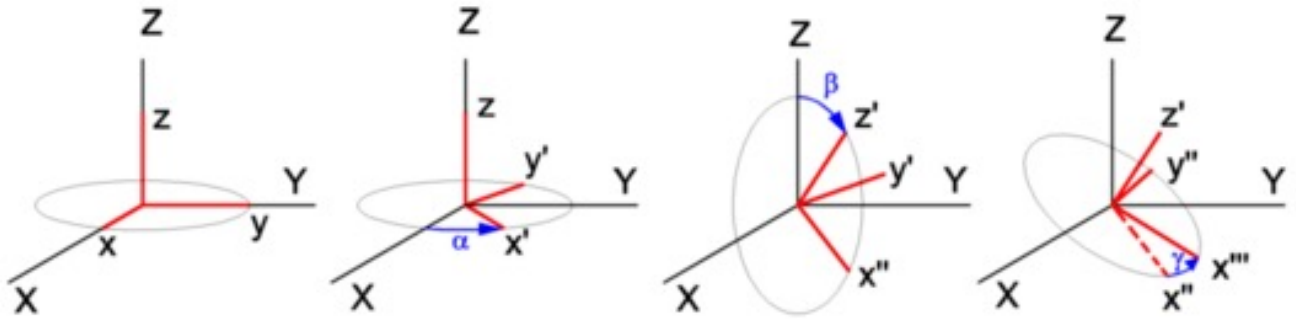


Rotations in 3D - SO(3)

Sequences of elemental rotations: Eulerian and Cardanian

- Euler's rotation theorem only requires successive rotation about three axes such that no two successive rotations are about the same axis.

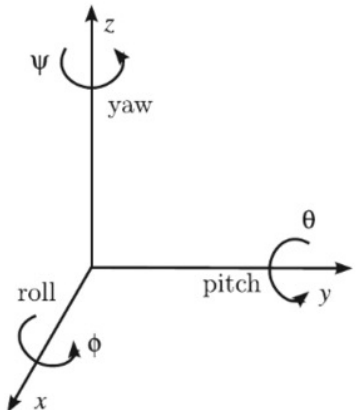
It is common practice to refer to all 3-angle representations as Euler angles but this is underspecified since there are twelve different types to choose from. The particular angle sequence is often a convention within a particular technological field.



- There are two classes of rotation sequence: **Eulerian** and **Cardanian**
  - The **Eulerian** type involves repetition, but not successive, of rotations about one particular axis: XYX, XZX, YXY, YZY, ZXZ, ZYZ.
  - The **Cardanian** type is characterized by rotations about all three axes: XYZ, XZY, YZX, YXZ, ZXY, ZYX.

Homogeneous transformations in 3D

Sequence matter! R-P-Y (Roll-Pitch-Yaw) Cardanian sequence



$$\mathbf{R} = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi) = \begin{pmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta s_\phi + c_\psi c_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{pmatrix}$$

General rotation matrix in 3D

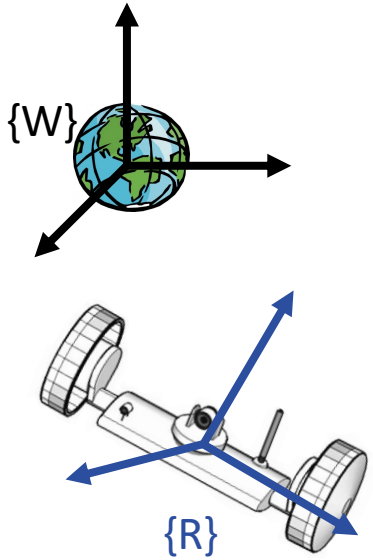
$$c_x \triangleq \cos(x)$$
$$s_x \triangleq \sin(x)$$

$$\boldsymbol{T} = \begin{pmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi & t_x \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta s_\phi + c_\psi c_\phi & t_y \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

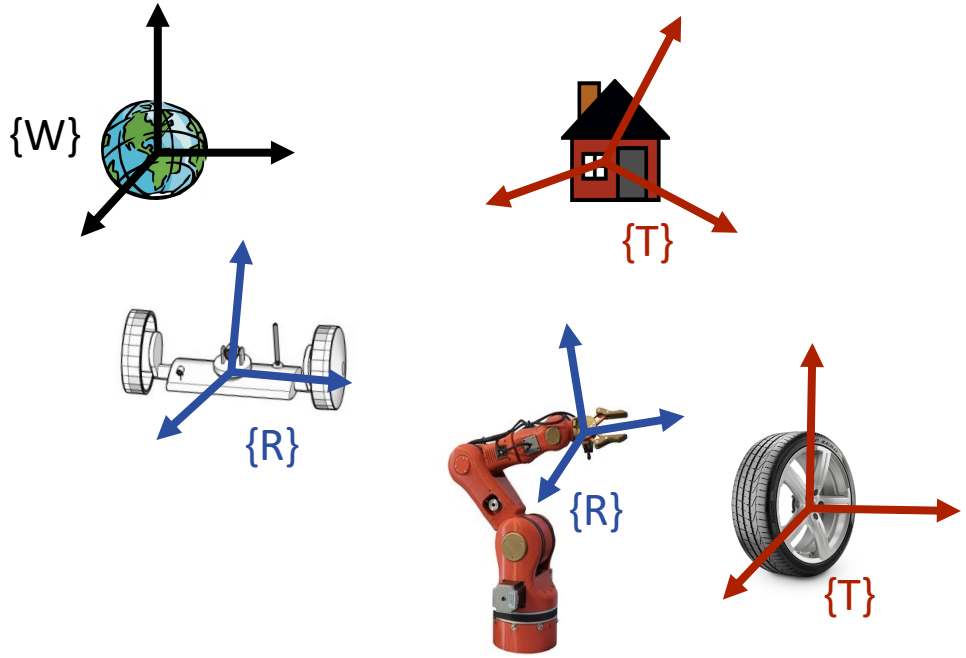
Homogeneous transformation matrix in 3D

Now we can deal with these problems requiring pose transformations...

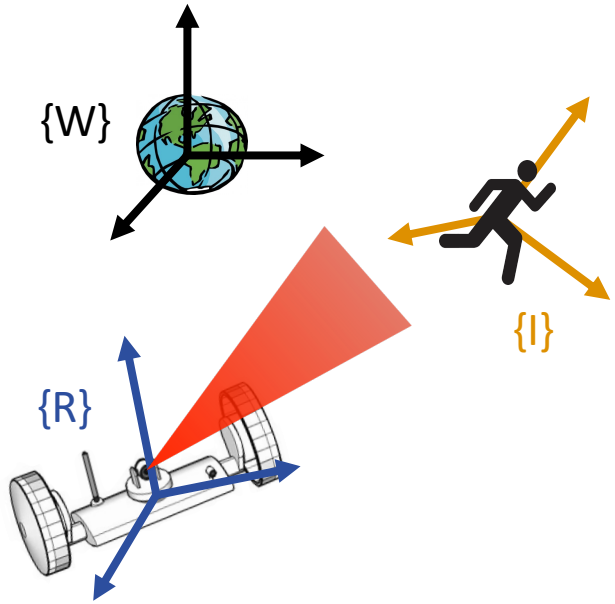
Where's the robot? What is robot's pose with respect to the world reference frame {W}?



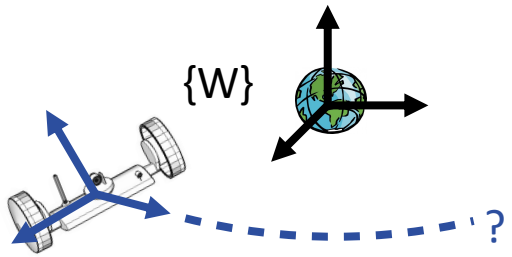
Robot's pose with respect to the external frame {T} (e.g., a target) ?



What is intruder's pose, observed using robot's lateral camera (local frame), in the world frame {W}?



Predict: What is robot's pose in {W} after moving at a velocity v for 1 minute?



Plan: What is the velocity profile that allows to reach a pose ξ in {W}?

Calculations….

$$\begin{pmatrix} {}^{W}x \\ {}^{W}y \\ {}^{W}z \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi & t_x \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta s_\phi + c_\psi c_\phi & t_y \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{{}^{W}\mathbf{T}_R} \begin{pmatrix} {}^{R}x \\ {}^{R}y \\ {}^{R}z \\ 1 \end{pmatrix}$$

Coordinates in {W} of a point P expressed in the local (robot) coordinate frame {R} , where ${}^{W}\mathbf{T}_R$ is the relative pose of {R} with respect to {W}

$$\xi_R(t+1) = \xi_R(t) \oplus \mathbf{T}$$

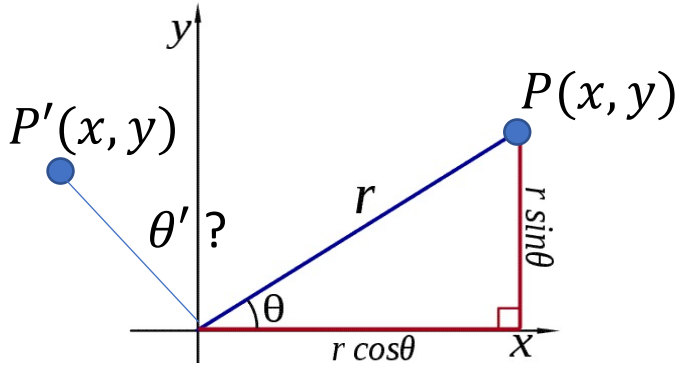
Robot's 3D pose in {W} after applying a homogeneous transformation **T from an initial pose** $\xi_R(t) \sim (0\ 0\ 0\ 0\ 0\ 0)$

$$\mathbf{T} = \begin{pmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi & t_x \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi & t_y \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} := \begin{pmatrix} t_{1,1} & t_{1,2} & t_{1,3} & t_{1,4} \\ t_{2,1} & t_{2,2} & t_{2,3} & t_{2,4} \\ t_{3,1} & t_{3,2} & t_{3,3} & t_{3,4} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\xi_R(t+1) \sim \begin{pmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{pmatrix}$$

$$\begin{aligned} x &= t_{1,4} \\ y &= t_{2,4} \\ z &= t_{3,4} \\ \phi &= \operatorname{atan2}(t_{2,1}, t_{1,1}) \\ \theta &= \operatorname{atan2}(-t_{3,1}, \cos(\phi)\, t_{1,1} + \sin(\phi)\, t_{2,1}) \\ \psi &= \operatorname{atan2}(\sin(\phi)\, t_{1,3} - \cos(\phi)\, t_{2,3}, -\sin(\phi)\, t_{1,2} + \cos(\phi)\, t_{2,2}) \end{aligned}$$

Angle calculations and atan2(y, x) function

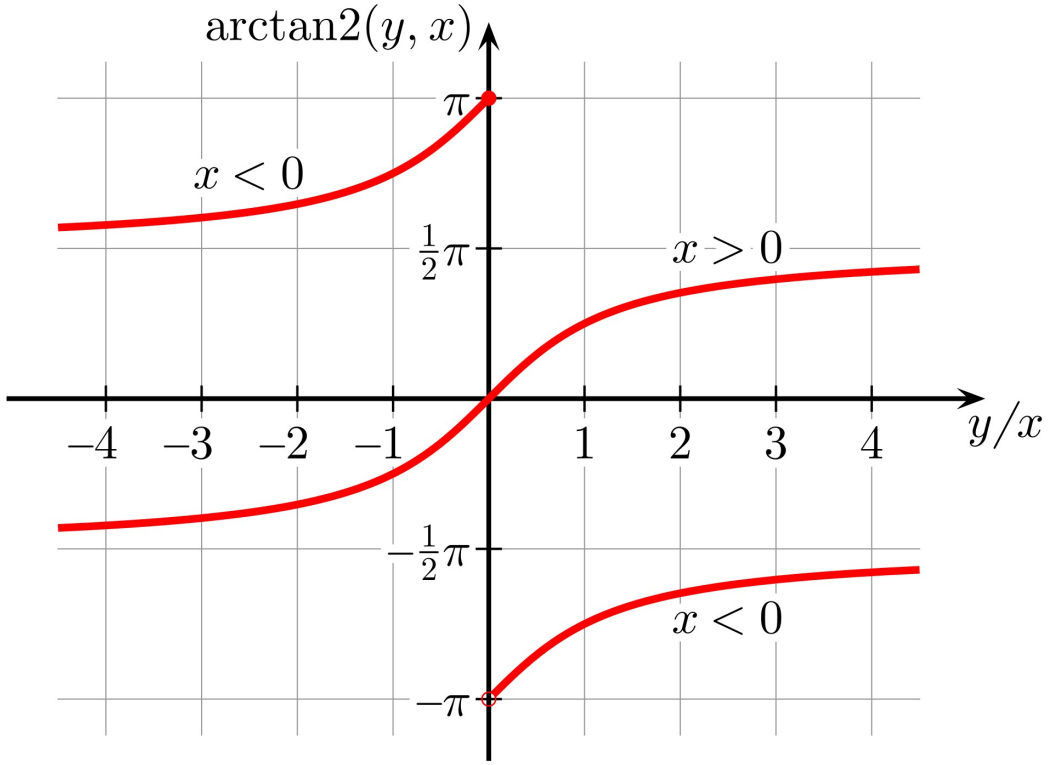


- Moved to coordinate y and x: What is my angle w.r.t. to origin?
- Move $(\Delta y, \Delta x)$ w.r.t. previous point → how much did the robot rotate?

$atan2(y, x)$ returns a value of an angle θ, such that:

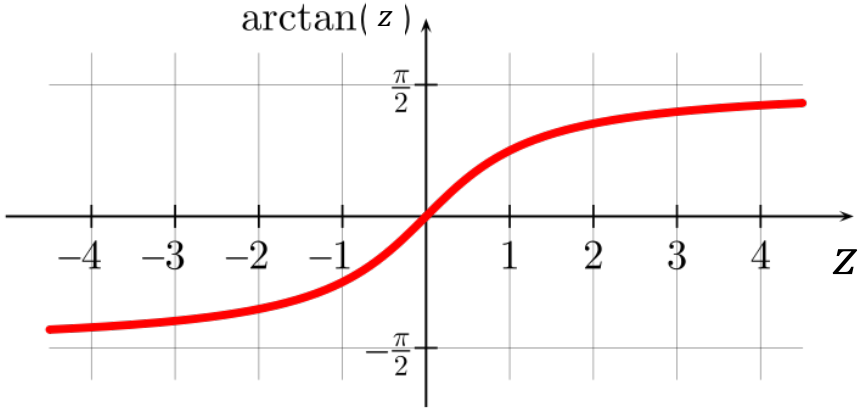
$$-\pi \leq \theta \leq \pi$$

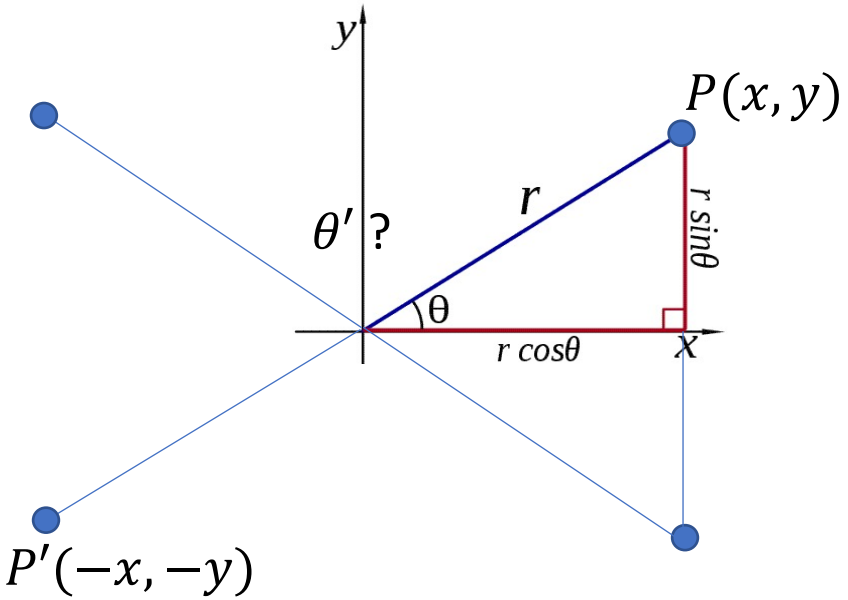
$$\text{atan2}(y,x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0, \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$



Angle calculations and atan2(y, x) function

- Why do we need an $\text{atan2}(y, x)$ function given that we already have an $\arctan\left(\frac{y}{x}\right)$ function from trigonometry?
- Note that we want to specify the argument as $z = \frac{y}{x}$





If the argument is specified as z, then: $\frac{y}{x} = \frac{-y}{-x}$

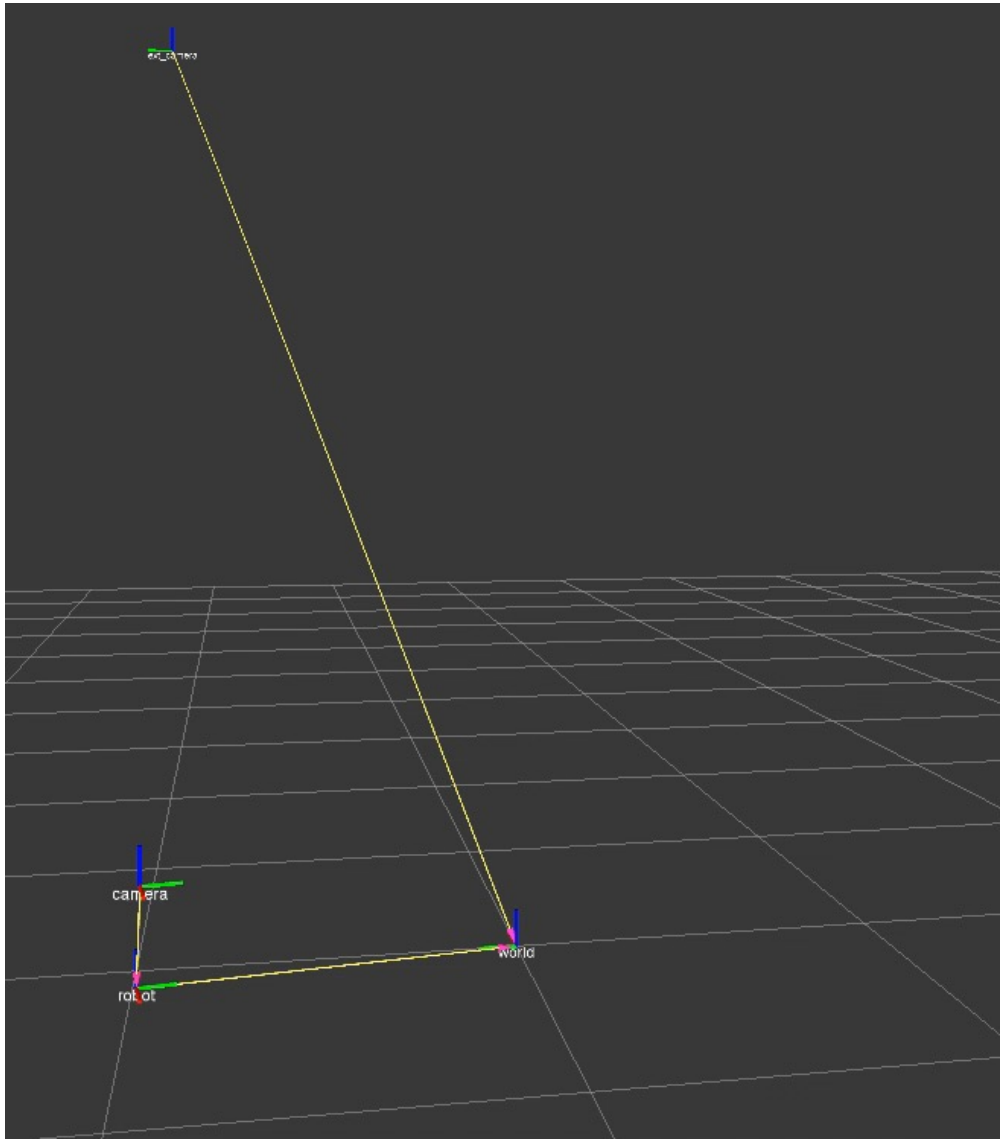
→ $\arctan(P)$ is the same as $\arctan(P')$ but angles differ by π !

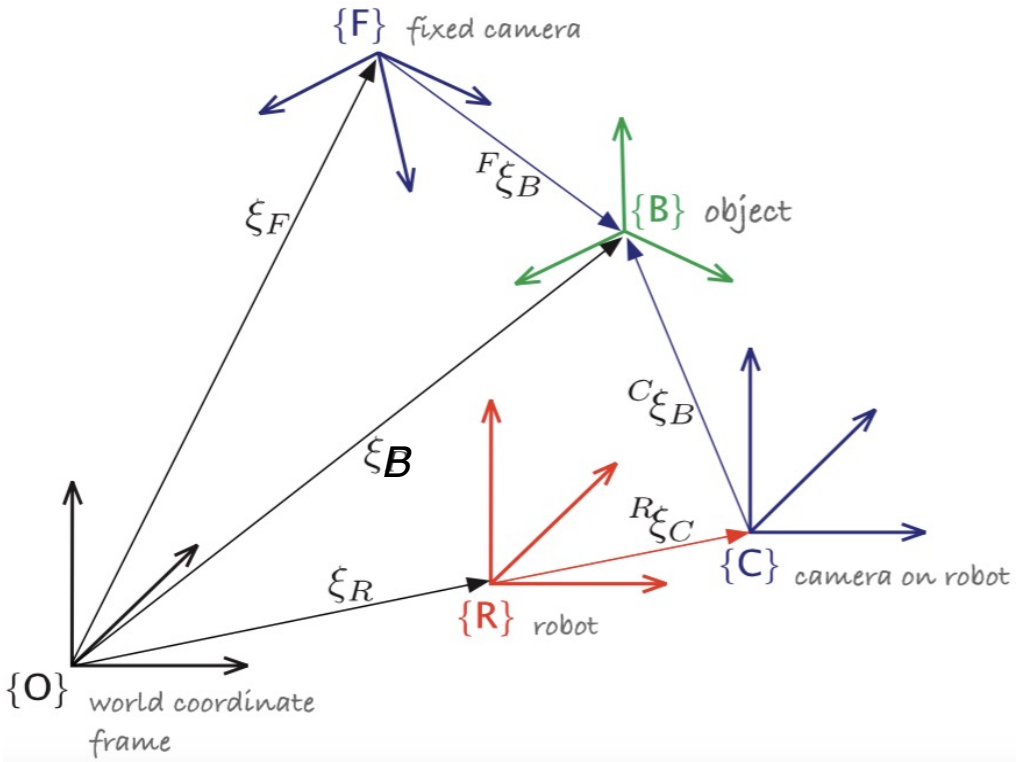
- $\arctan\left(\frac{y}{x}\right)$ cannot distinguish between these cases because it provides an angle $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$

$atan2(y, x)$ deals with the problem:

- for $y > 0$ the returned angle is $0 \leq \theta \leq \pi$,
- while for $y < 0$, angle is $-\pi \leq \theta \leq 0$

ROS helps: once set the relative poses, it can keep track of relative changes among frames



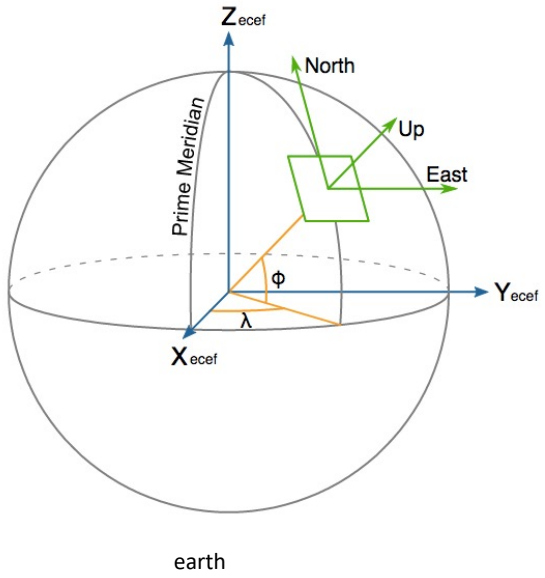


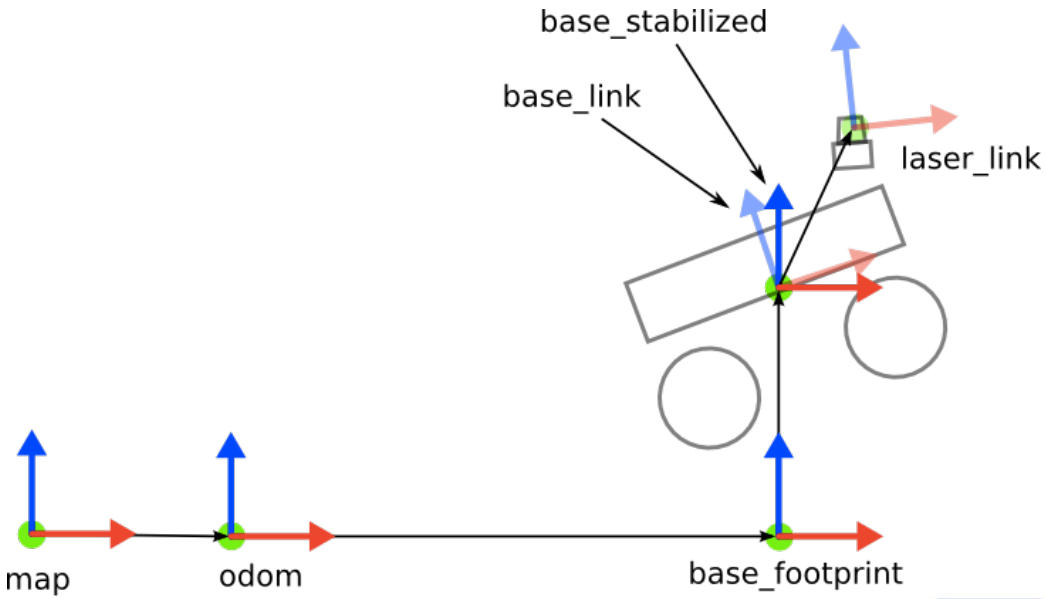
ROS helps! `tf` package

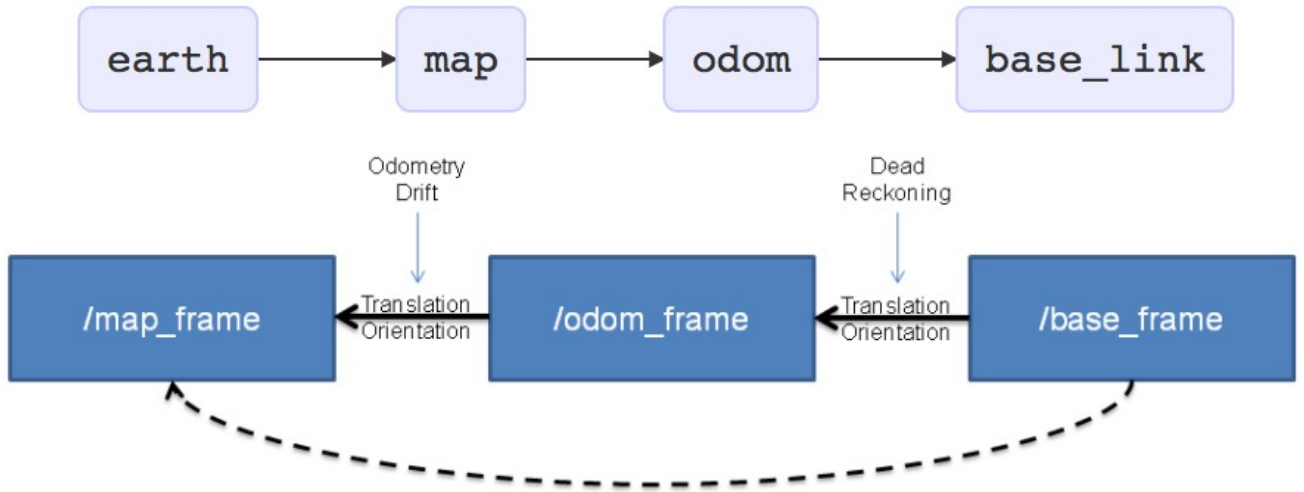
```cpp
#include <stdio.h>
#include <math.h>
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
#include <tf/transform_listener.h>

int main (int argc, char** argv)
{
  ros::init(argc, argv, "tf_example");

  ros::NodeHandle node;

  // to publish and read the topics regarding frame transformation
  tf::TransformListener tf_ls;
  tf::TransformBroadcaster tf_br;

  // transformation objects for the robot, the camera on board of the robot,
  // the external camera
  tf::Transform robot_tf;
  tf::Transform camera_tf;
  tf::Transform ext_camera_tf;


  // set a displacement (translation + rotation) for external camera
  ext_camera_tf.setOrigin(tf::Vector3(3.0, 1.0, 3.0));
  ext_camera_tf.setRotation(tf::Quaternion(0.0, 0.0, 0.0, 1.0));

  // set a displacement (translation + rotation) for external camera
  camera_tf.setOrigin(tf::Vector3(0.15, 0.0, 0.30));
  camera_tf.setRotation(tf::Quaternion(0.0, 0.0, 0.0, 1.0));

  // robot start position, it will then follow a circular trajectory
  float R = 1.0;
  float x = 0.0;
  float y = 0.0;
  float phi = 0.0;

  // orientation for robot trajectory
  tf::Quaternion robot_ori;
  tf::Vector3 pt_on_robot(0.0, 0.0, 0.0);


  // where to store/publish time-stamped information about frame transformation
  tf::StampedTransform relative_pose_tf;
```

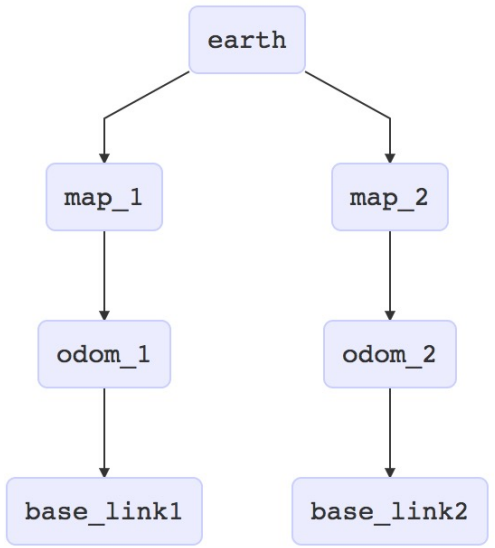
```cpp
  // main control cycle, with a defined frequency
  ros::Rate loop_rate(100);
  while(ros::ok())
  {
    // set robot's pose to follow a circular trajectory
    x = R * cos(phi);
    y = R * sin(phi);
    phi += 0.01;
    ROS_INFO("Ground truth: %3.2f, %3.2f, %3.2f", x, y, 0.0);

    // to make the robot curving on the trajectory
    robot_ori.setRPY(0.0, 0.0, phi + M_PI / 2.0);

    // robot's relative pose
    robot_tf.setOrigin(tf::Vector3(x, y, 0.0));
    robot_tf.setRotation(robot_ori);

    ros::Time stamp = ros::Time::now();

    // parent-child relations between the frames attached to the considered entities are defined
    tf_br.sendTransform(tf::StampedTransform(robot_tf, stamp, "/world", "/robot"));
    tf_br.sendTransform(tf::StampedTransform(camera_tf, stamp, "/robot", "/camera"));
    tf_br.sendTransform(tf::StampedTransform(ext_camera_tf, stamp, "/world", "/ext_camera"));

    try
    {
      // computing the homogeneous transformation from robot frame to the external camera frame
      tf_ls.lookupTransform("/ext_camera", "/robot", ros::Time(0), relative_pose_tf);

      // a point in the robot frame is transformed in a point in the external camera frame
      tf::Vector3 pt_in_ext_camera = relative_pose_tf * pt_on_robot;
      // the point the external camera frame is transformed in a point in the world frame
      tf::Vector3 pt_in_world = ext_camera_tf * pt_in_ext_camera;

      ROS_INFO("Point in /ext_camera: %3.2f, %3.2f, %3.2f",
               pt_in_ext_camera.getX(), pt_in_ext_camera.getY(), pt_in_ext_camera.getZ());
      ROS_INFO("Point in /world: %3.2f, %3.2f, %3.2f",
               pt_in_world.getX(), pt_in_world.getY(), pt_in_world.getZ());
    }
    catch(tf::TransformException &e)
    {
      ROS_ERROR("%s", e.what());
      ros::Duration(1.0).sleep();
      continue;
    }
    loop_rate.sleep();
    ros::spinOnce();
  }
  return 0;
}
```

In which frame poses / odometry measures are represented



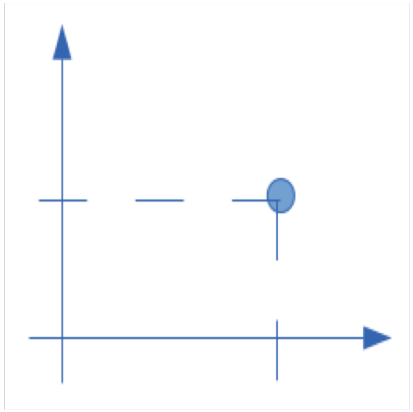




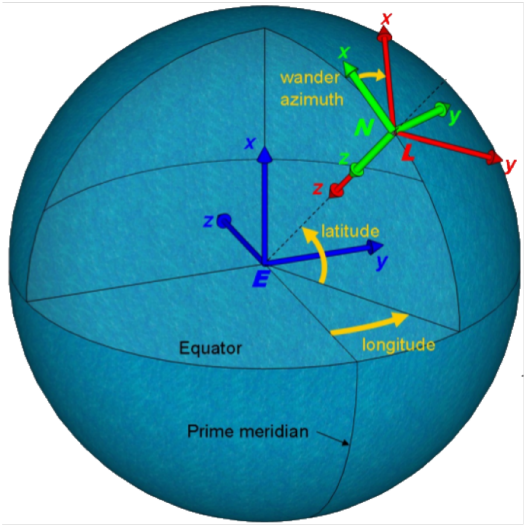


Maps….

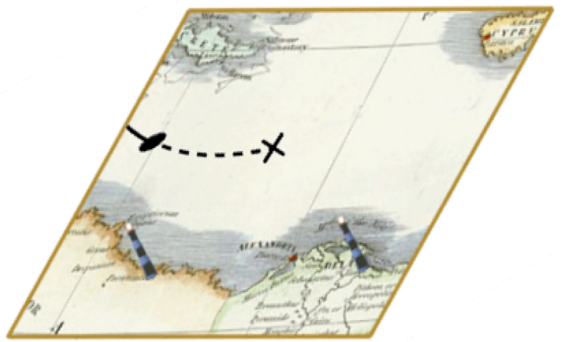
Cartesian, 2D

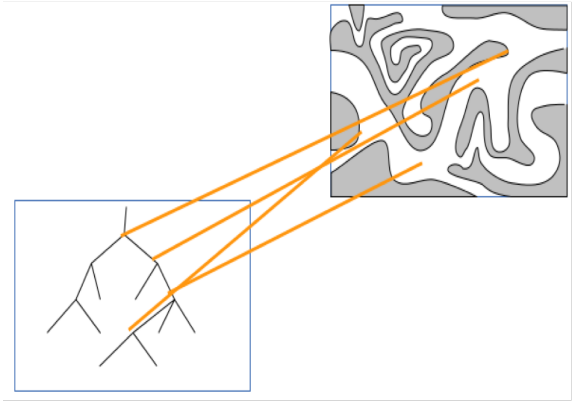


3D earth coordinates

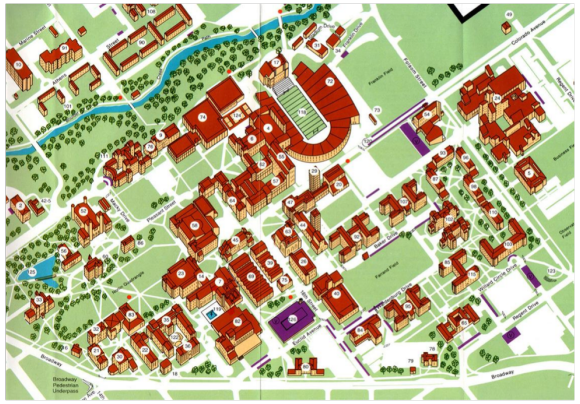


Geographical map, landmarks

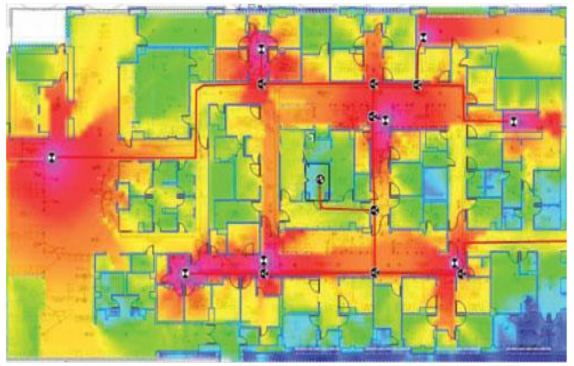




Toplogical map



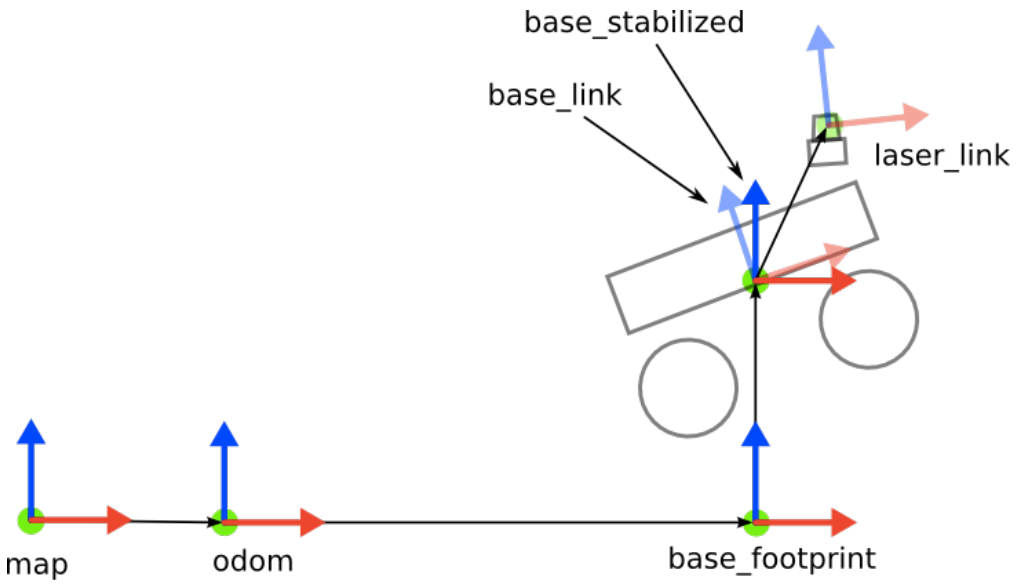
Metric map



Sensor map

Basic Ros frames

- **base_link** is rigidly attached to the mobile robot base. It can be attached to the base in any arbitrary position or orientation
- **odom** is a world-fixed frame. The pose of a mobile platform in the odom frame can drift over time, without any bounds. This drift makes the odom frame useless as a long-term global reference.
- The pose of a robot in the odom frame is guaranteed to be continuous, meaning that the pose of a mobile platform in the odom frame always evolves in a smooth way, without discrete jumps.
- In a typical setup the odom frame is computed based on an odometry source, such as wheel odometry, visual odometry or an IMU

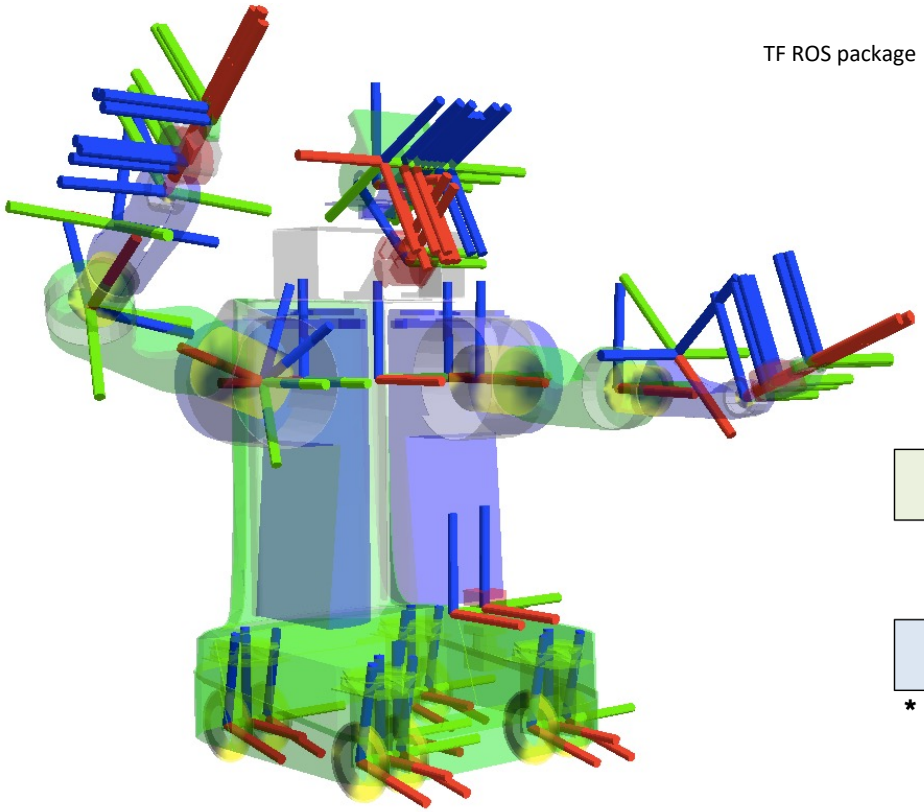


- **map** is a world fixed frame, with its Z-axis pointing upwards. The pose of a mobile platform, relative to the map frame, should not significantly drift over time. The map frame is not continuous, meaning the pose of a mobile platform in the map frame can change in discrete jumps at any time.
- In a typical setup, a localization component constantly re-computes the robot pose in the map frame based on sensor observations, therefore eliminating drift, but causing discrete jumps when new sensor information arrives. The map frame is useful as a long-term global reference, but discrete jumps in position estimators make it a poor reference frame for local sensing and acting.

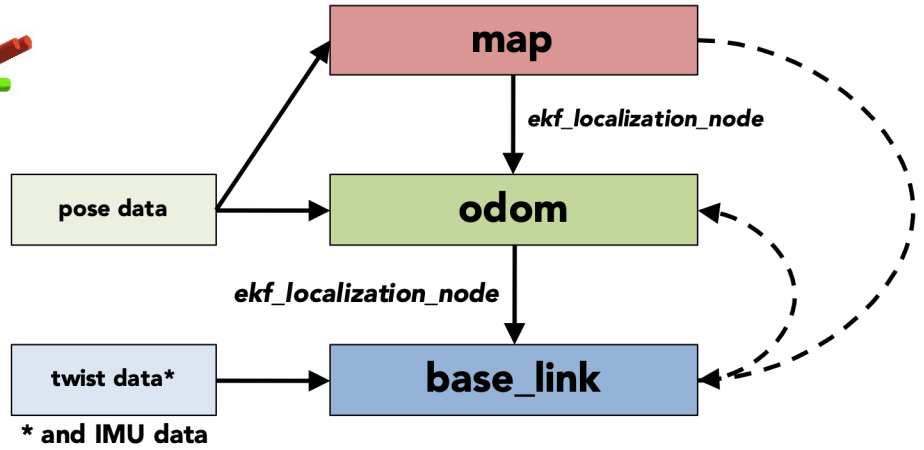
- **earth** is the origin of ECEF. This frame is designed to allow the interaction of multiple robots in different map frames. If the application only needs one map the earth coordinate frame is not expected to be present.

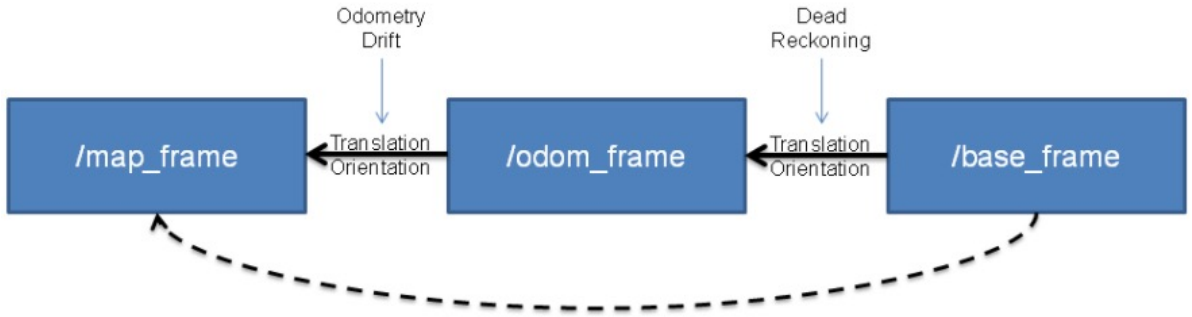
(http://www.ros.org/reps/rep-0105.html)

Transformation between frames



TF ROS package





Odometry msg type in ROS

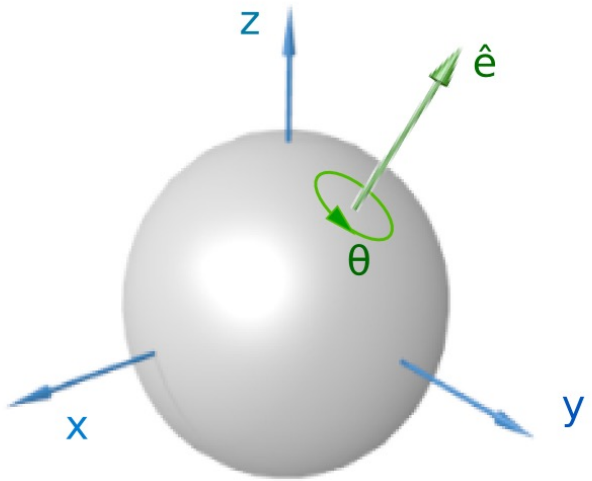
```
$ rosmsg info Odometry

[nav_msgs/Odometry]:
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
geometry_msgs/TwistWithCovariance twist
  geometry_msgs/Twist twist
    geometry_msgs/Vector3 linear
      float64 x
      float64 y
      float64 z
    geometry_msgs/Vector3 angular
      float64 x
      float64 y
      float64 z
  float64[36] covariance
```

```
def callback_odometry(self, msg):
    if self.report_pose == True:
        print "Position: (%5.2f, %5.2f, %5.2f)" % (msg.pose.pose.position.x,
               msg.pose.pose.position.y, msg.pose.pose.position.z)
        self.quaternion_to_euler(msg)
        print "Linear twist: (%5.2f, %5.2f, %5.2f)" %
                              (msg.twist.twist.linear.x,
                               msg.twist.twist.linear.y,
                               msg.twist.twist.linear.z)
        print "Angular twist: (%5.2f, %5.2f, %5.2f)" %
                               (msg.twist.twist.angular.x,
                                msg.twist.twist.angular.y,
                                msg.twist.twist.angular.z)

        print "Ground Truth: ", self.get_ground_truth(self.robot_model,"world")
```

```
def quaternion_to_euler(self, msg):
    quaternion = (msg.pose.pose.orientation.x, msg.pose.pose.orientation.y,
                  msg.pose.pose.orientation.z, msg.pose.pose.orientation.w)
    (roll, pitch, yaw) = tf.transformations.euler_from_quaternion(quaternion)
    print "Roll: %5.2f  Pitch: %5.2f  Yaw: %5.2f" % (roll, pitch, yaw)
    return yaw
```

Quaternions: another (redundant) way to represent an orientation

- In 3D, according to Euler's rotation theorem, any rotation about a fixed point is equivalent to a single rotation by a given angle θ about a fixed axis (called the **Euler axis**) that runs through the fixed point.
- The Euler axis is typically represented by a unit vector $\hat{e}$
- Therefore, any rotation in three dimensions can be represented as a combination of a vector $\hat{e}$ and a scalar θ.
- **Quaternions** give a handy way to encode this axis–angle representation in 4 numbers and can be used to calculate the corresponding rotation to a position vector (x, y, z), representing a point relative to the origin.



➢ **Redundant / implicit representation** of a rotation (only 3 angles would be needed)

➢ A quaternion is a 3-dimensional extension of a complex number, defined by a 4-tuple

$$q_0 + iq_1 + jq_2 + kq_3$$

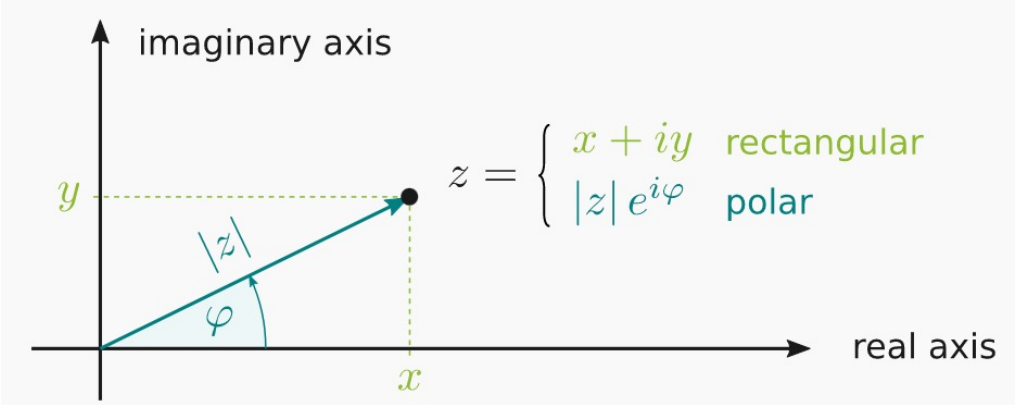
Where q_l are real numbers and the symbols i, j, k *are imaginary units*

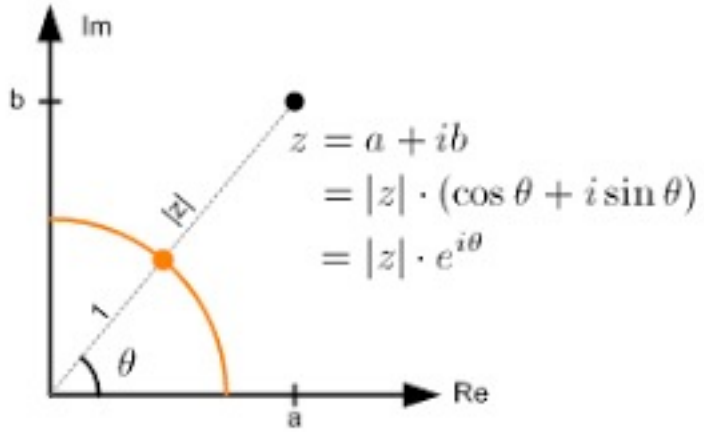
$$i^2 = j^2 = k^2 = -1$$
$$ij = k,\ ji = -k$$
$$jk = i,\ kj = -i$$
$$ki = j,\ ik = -j\,.$$

Complex numbers and rotations

$$x + iy$$

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



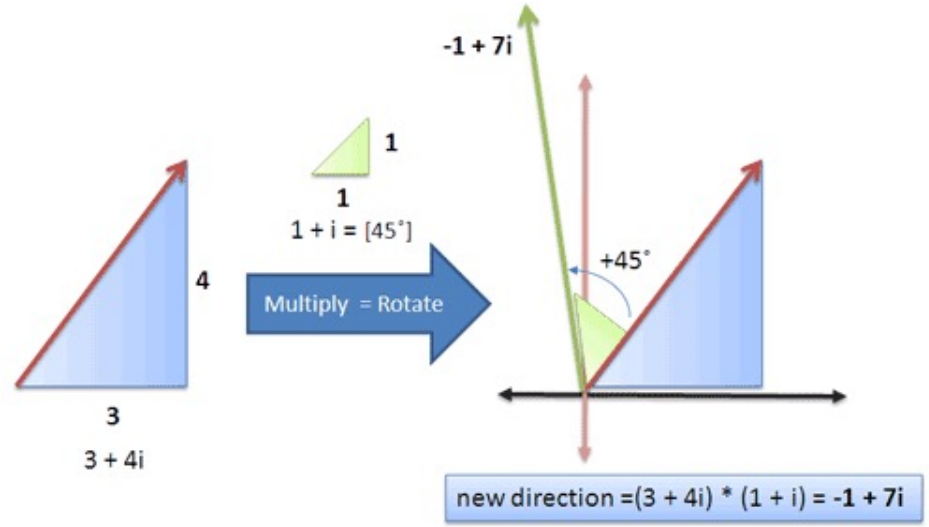


- **Multiplication** between two complex numbers, one in polar and the other in Cartesian coordinates

$$\begin{aligned} e^{i\theta}(x + iy) &= (\cos\theta + i\sin\theta)(x + yi) \\ &= (x\cos\theta - y\sin\theta) + i(x\sin\theta + y\cos\theta) \end{aligned}$$

which can be rewritten as a 2D rotation by θ of vector (x, y)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Quaternions: basic properties

- A point $\boldsymbol{P} = (x, y, z)$, as a **vector in 3D** can be expressed as a quaternion with $q_0 = 0$:

$$q = 0 + xi + yj + zk$$

- A **rotation in 3D** can be expressed as a **unit quaternion,** q_R :

$$|q_R| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$$

- A **transformation of coordinate for a point** $\boldsymbol{P}$, represented by a quaternion q_A, from a frame A to a frame B:

$$q_B = q_R q_A q_R^* \qquad \textit{Conjugate } q^* = q_0 - q_1 i - q_2 j - q_3 k$$

✓ This computation of the new coordinates does not require the computation of trigonometric functions, only the addition and multiplication of real numbers.

Quaternions: Rotations and transformations

- A **transformation of coordinates for a point $\boldsymbol{P} = (x, y, z)$**, represented as a quaternion $q_A = (xi, yj, zk)$, from a frame A to a frame B, where relative frame rotation is represented by $q_R = q_0 + iq_1 + jq_2 + kq_3$

$$q_B = q_R q_A q_R^*$$

$$\begin{aligned} q_R\, q_A\, q_R^* &= (q_0 + q_1 i + q_2 j + q_3 k)(xi + yj + zk)(q_0 - q_1 i - q_2 j - q_3 k) \\ &= (x(q_0^2 + q_1^2 - q_2^2 - q_3^2) + 2y(q_1 q_2 - q_0 q_3) + 2z(q_0 q_2 + q_1 q_3))i + \\ &\quad (2x(q_0 q_3 + q_1 q_2) + y(q_0^2 - q_1^2 + q_2^2 - q_3^2) + 2z(q_2 q_3 - q_0 q_1))j + \\ &\quad (2x(q_1 q_3 - q_0 q_2) + 2y(q_0 q_1 + q_2 q_3) + z(q_0^2 - q_1^2 - q_2^2 + q_3^2))k. \end{aligned}$$

➢ $q_R q_A q_R^*$ is a rotation → Can be **expressed as a rotation matrix $\boldsymbol{R}$**

$$R = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_0 q_2 + q_1 q_3) \\ 2(q_0 q_3 + q_1 q_2) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_0 q_1 + q_2 q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} = 2 \cdot \begin{bmatrix} q_0^2 + q_1^2 - 0.5 & q_1 q_2 - q_0 q_3 & q_0 q_2 + q_1 q_3 \\ q_0 q_3 + q_1 q_2 & q_0^2 + q_2^2 - 0.5 & q_2 q_3 - q_0 q_1 \\ q_1 q_3 - q_0 q_2 & q_0 q_1 + q_2 q_3 & q_0^2 + q_3^2 - 0.5 \end{bmatrix}$$

$$\begin{bmatrix} q_1^B \\ q_2^B \\ q_3^B \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Quaternions: From angles to quaternions

✓ **Given a rotation matrix M** → we can compute the quaternion that represents the same rotation

$$\begin{aligned} Trace(M) &= M_{11} + M_{22} + M_{33} \\ &= 2(3q_0^2 + q_1^2 + q_2^2 + q_3^2 - 1.5) \\ &= 2(3q_0^2 + (1 - q_0^2) - 1.5) \\ &= 2(2q_0^2 - 0.5) \\ &= 4q_0^2 - 1. \end{aligned}$$

$$|q_0| = \sqrt{\frac{Trace(M)+1}{4}}.$$

$$|q_1| = \sqrt{\frac{M_{11}}{2} + \frac{1 - Trace(M)}{4}}$$

$$|q_2| = \sqrt{\frac{M_{22}}{2} + \frac{1 - Trace(M)}{4}}.$$

$$|q_3| = \sqrt{\frac{M_{33}}{2} + \frac{1 - Trace(M)}{4}}.$$

✓ **Given a rotation by an angle** → the corresponding quaternion can be computed

E.g., Rotation of ψ around z

$$M_\psi = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$Trace(M) = 2\cos\psi + 1$:

$$\begin{aligned} |q_0| &= \sqrt{\frac{2\cos\psi + 1 + 1}{4}} = \sqrt{\frac{1+\cos\psi}{2}} = \cos\frac{\psi}{2}, \\ |q_1| = |q_2| &= \sqrt{\frac{\cos\psi}{2} + \frac{1 - (2\cos\psi + 1)}{4}} = 0, \\ |q_3| &= \sqrt{\frac{1}{2} + \frac{1 - (2\cos\psi + 1)}{4}} = \sqrt{\frac{1-\cos\psi}{2}} = \sin\frac{\psi}{2} \end{aligned}$$

$$q_\psi = \cos\frac{\psi}{2} + k\sin\frac{\psi}{2} \qquad (z)$$

$$q_\theta = \cos\frac{\theta}{2} + j\sin\frac{\theta}{2} \qquad (y)$$

$$q_\phi = \cos\frac{\phi}{2} + i\sin\frac{\phi}{2} \qquad (x)$$

Important concepts to take home

- Robot pose (positions + orientations) and its importance / use
- Representation and composition of (relative poses)
- Poses as an additive group
- Representation of relative poses as a graph
- Implicit representation of orientations using rotation matrices
- 2D and 3D representations
- Properties of rotation matrices
- Homogeneous transformations (translations + rotations)
- Euler angles for explicit representation of rotations (singularity issues)
- Quaternions as implicit representations of orientations
- Basic ROS frames
- Use of relative pose transformations in ROS, TF package