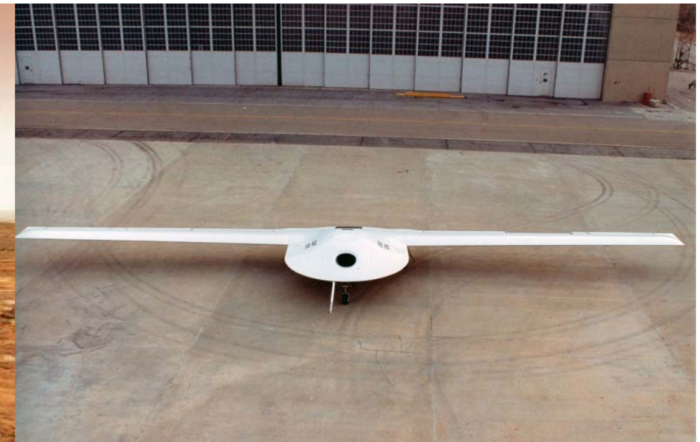





# ME 597: Autonomous Mobile Robotics
# Section 2 – Coordinate Transforms

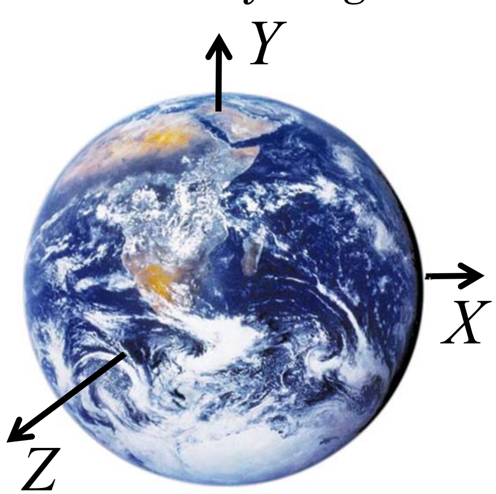
Prof. Steven Waslander

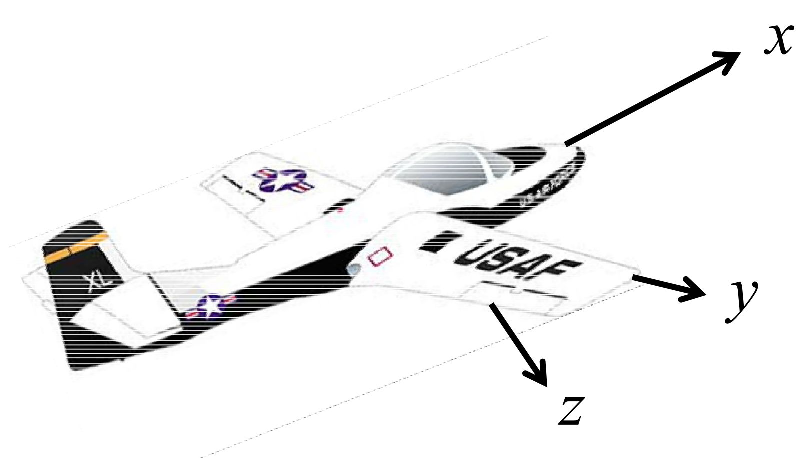
# OUTLINE

- Coordinate Frames and Transforms
  - Rotation Matrices
  - Euler Angles
  - Quaternions
  - Homogeneous Transforms

# COORDINATE FRAMES

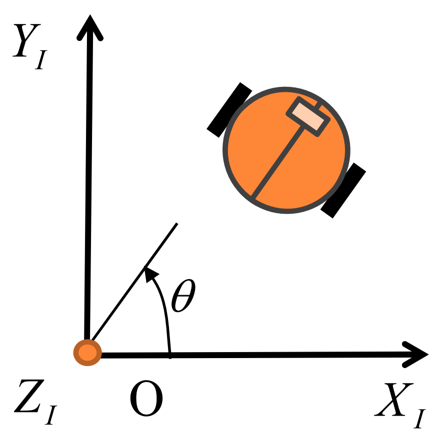
- Used to define environment, vehicle motion
- Right-handed by convention
  - Inertial frame – fixed, usually relative to the earth
    - GPS: Earth Centered Earth Fixed (ECEF), Latitude, Longitude, Altitude (LLA), East North Up (ENU)
    - Aeronautics: North East Down (NED)
  - Body/sensor frame – attached to vehicle/sensor, useful for expressing motion/measurements
    - Body origin at vehicle CG, sensor at optical center



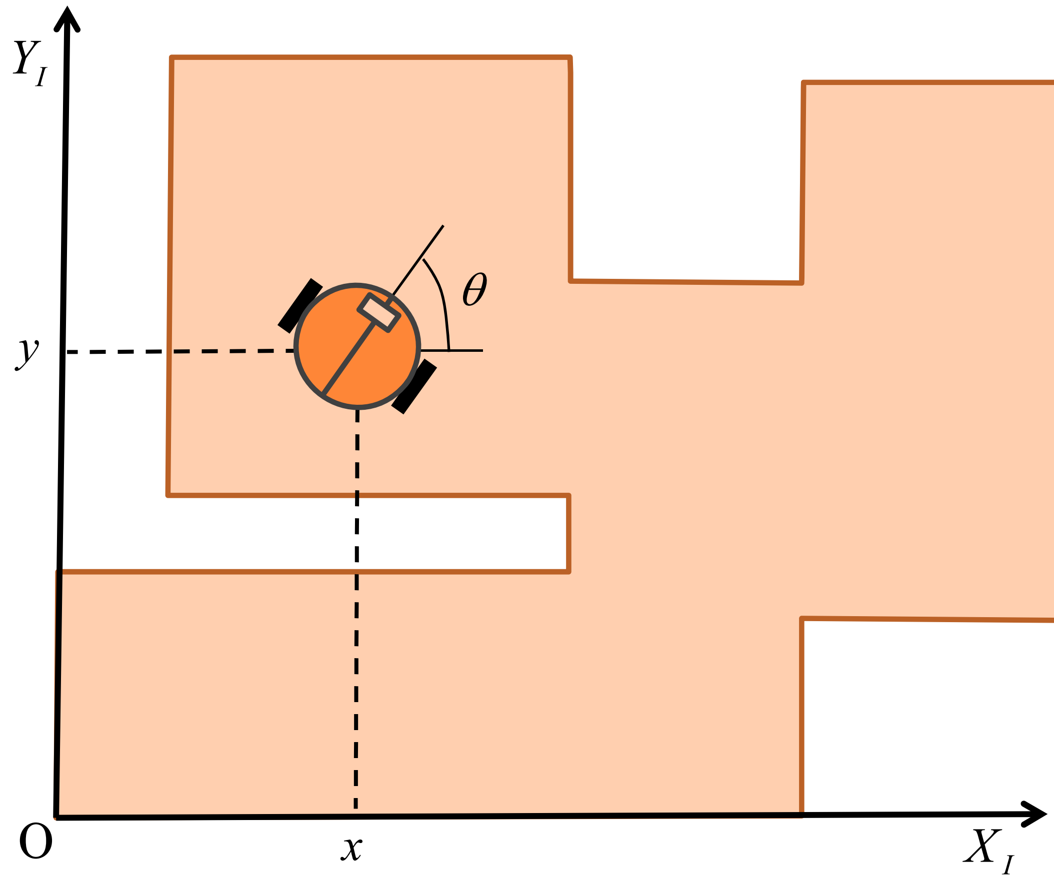
# Coordinate Frames

- 2D Rigid Body Motion
  - For ground robots, two dimensional motion definition often enough
    - If robot has a heading, third axis is implicit
    - Right hand rule defines direction of rotation



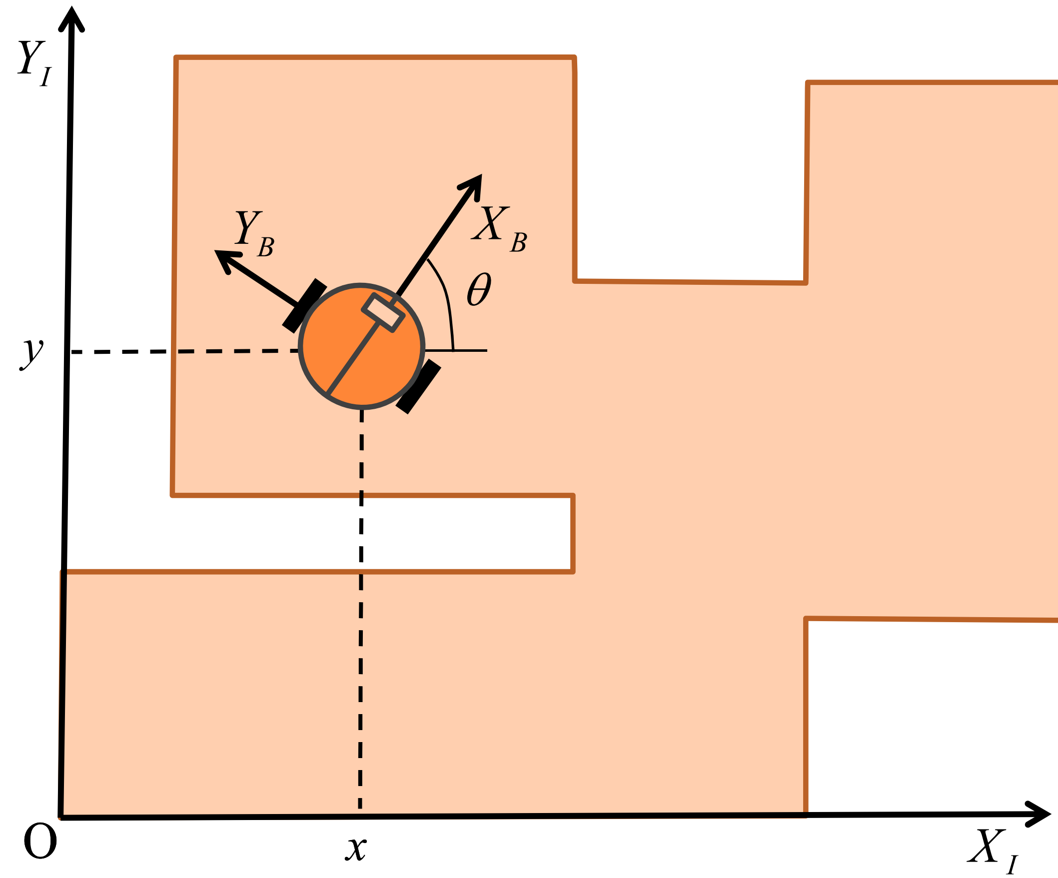
# Coordinate Frames

- Inertial Frame: $X_I, Y_I$
- Vehicle State: $\xi_I = [x, y, \theta]$



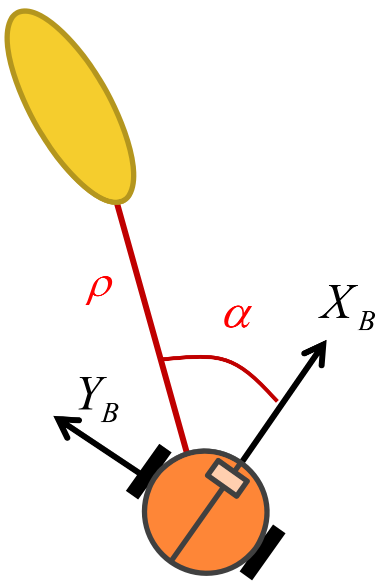
# Coordinate Frames

- Body Frame: $X_B, Y_B$
- Vehicle State: $\xi_B = [0, 0, 0]$



# Coordinate Frames

- Body frame useful for understanding sensor measurements, environment relative to vehicle
  - Bearing and range to an obstacle:




$$x_{object,B} = \rho \cos \alpha$$

$$y_{object,B} = \rho \sin \alpha$$

# Rotation Matrices

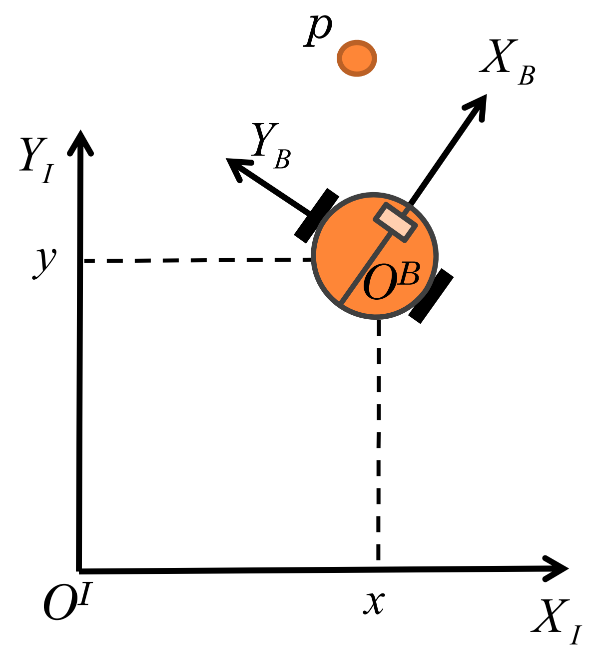
- Conversion between Inertial and Body coordinates is done with a translation vector and a rotation matrix
  - Rotate vectors using 2X2 rotation matrix

$$R_I^B(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

  - Full transformation is translation and then rotation

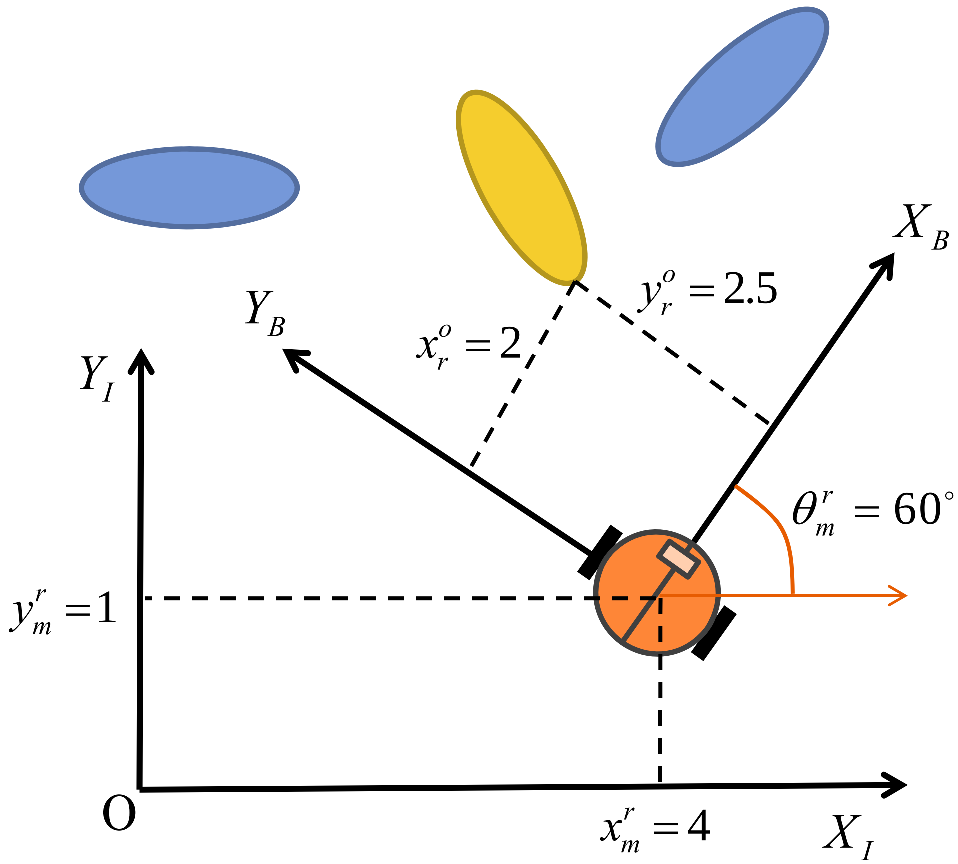
$$p_B = R_I^B(\theta) p_I + O_B^I$$

$$p_I = R_B^I(\theta) p_B + O_I^B$$



# Coordinate Transformation

- To map the location of the obstacle in a local map, need to transform the current measurement into the map reference frame:




$$p_I = R_B^I(\theta) p_B + O_I^B$$

$$\begin{bmatrix} x_I^o \\ y_I^o \end{bmatrix} = \left(R_I^B\right)^{-1} \begin{bmatrix} x_B^o \\ y_B^o \end{bmatrix} + \begin{bmatrix} x_I^B \\ y_I^B \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & -0.866 \\ 0.866 & .5 \end{bmatrix} \begin{bmatrix} 2 \\ 2.5 \end{bmatrix} + \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

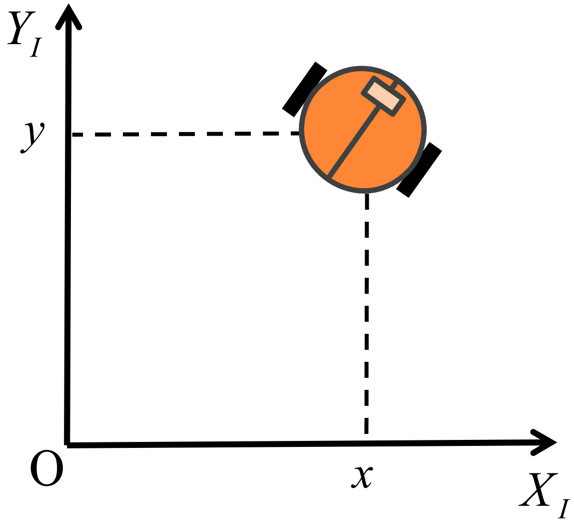
$$= \begin{bmatrix} -1.165 \\ 2.982 \end{bmatrix} + \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2.835 \\ 3.982 \end{bmatrix}$$

# Rotation Matrices

- In fact, the rotation can also be seen as a 3D rotation, about the $Z_I$ axis.

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




  - Can be generalized to arbitrary rotations about any axis (Euler angles)

  - Rotation matrices are orthogonal

$$R^{-1}(\theta) = R^{T}(\theta)$$

# 3D Coordinate Transforms

- Often handy to concisely define a transformation between coordinate frames
  - Define
    - $t$, the translation vector between the origins of the two frames
    - $R$, a 3X3 rotation matrix from one frame to the next
    - $\hat{x}$ , a homogenous form of the state,

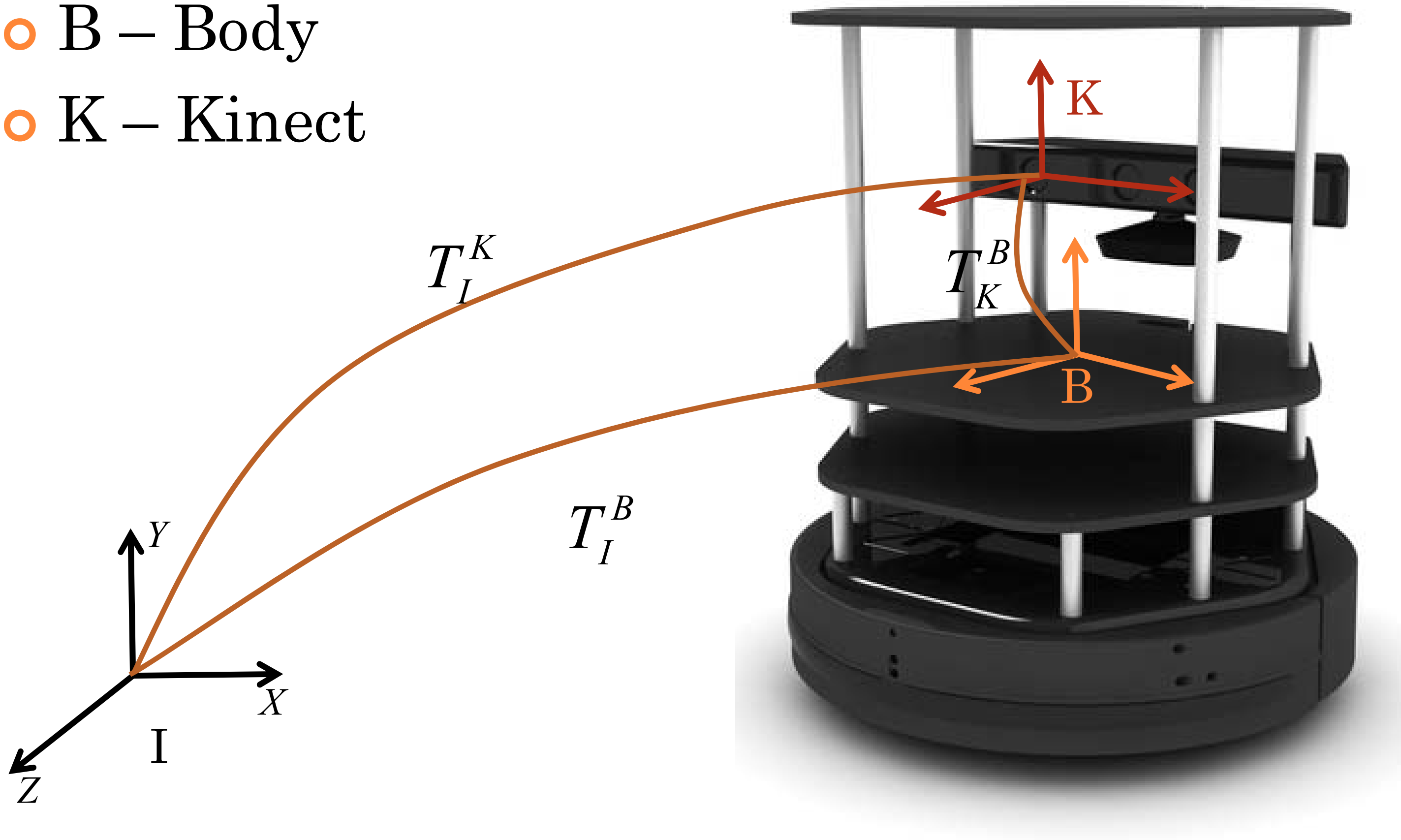
$$\hat{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$$

  - Combine into a homogeneous transform

$$\begin{bmatrix} x^I \\ 1 \end{bmatrix} = T_B^I \hat{x}^B = \begin{bmatrix} R_B^I & t_B^I \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x^B \\ 1 \end{bmatrix}$$

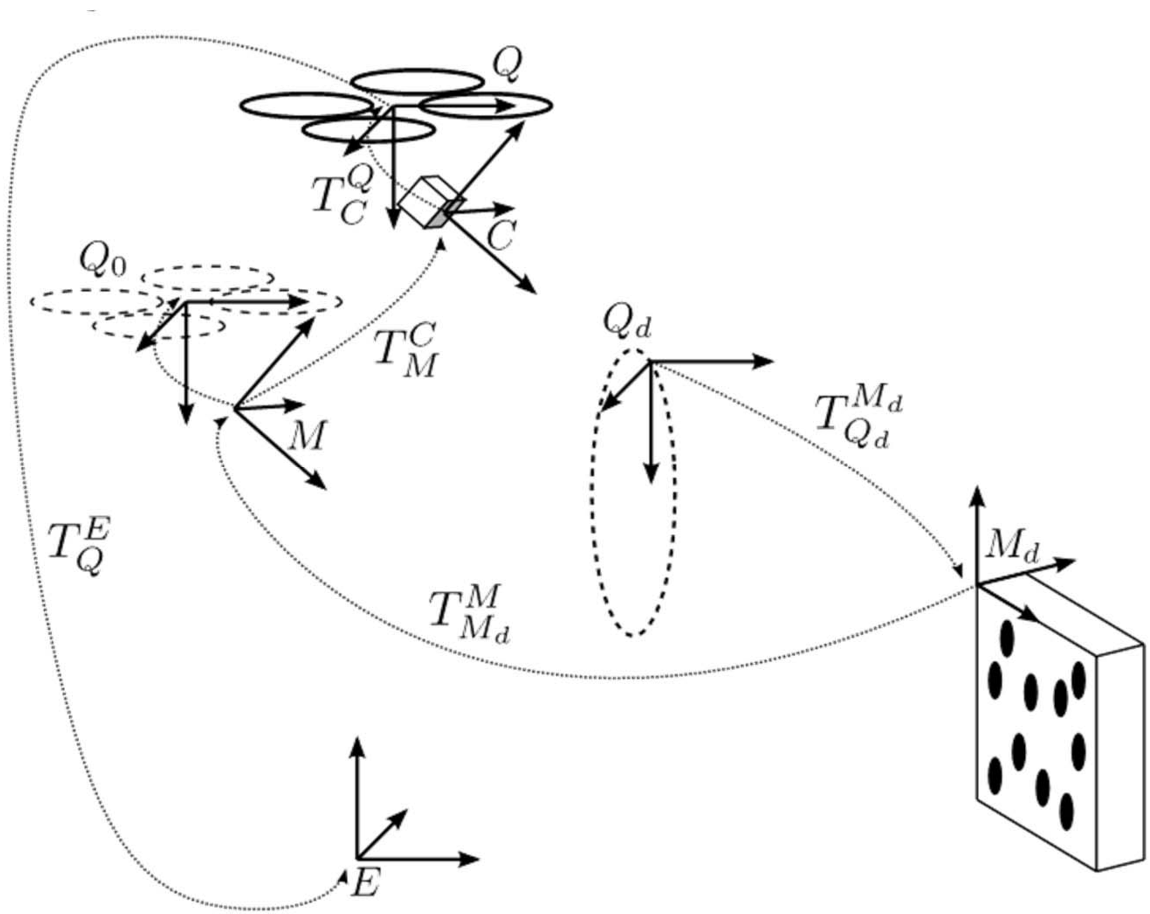
# TurtleBot Transforms

- I – Inertial
- B – Body
- K – Kinect



# Quadrotor Transform – Extreme Case

- $E$ – Earth Fixed
- $Q$ – Current quadrotor pose
- $C$ – Camera frame
- $M$ – Model frame
- $M_d$ – Target fixed frame
- $Q_d$ – Desired quadrotor pose




- Quadrotor inertial pose error equation:

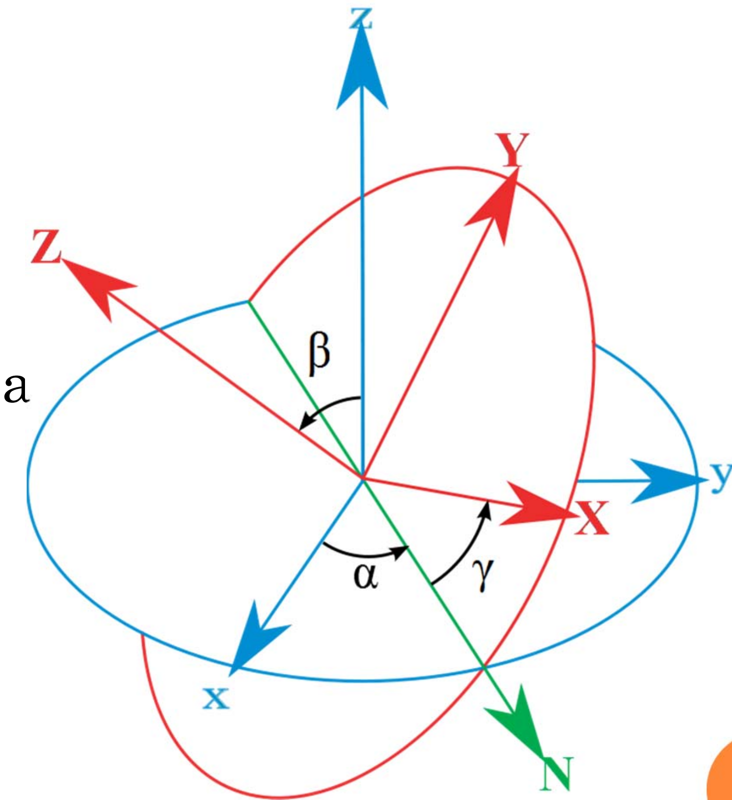
$$T_{Q_d}^{Q}\Big|^{E}(t) = R_Q^E T_C^Q T_M^C T_{M_d}^M T_{Q_d}^{M_d}(t)$$

# Rotations in 3D

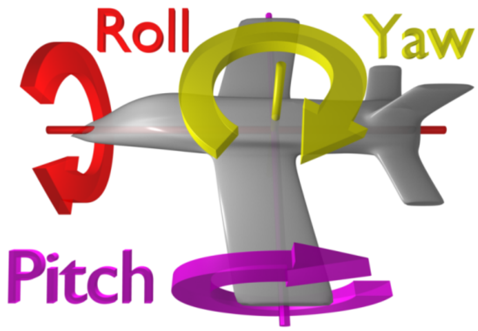
- There are at least 3 ways to represent rotations in 3D
  - Euler angles
    - Intuitive, easy to understand, sequence of rotations
    - Have singularity known as "gimbal lock" where rotation cannot be properly represented
  - Quaternions
    - Represents rotations as a 4 element unit quaternion
    - Easy to update, no singularities
    - Requires unit norm, not intuitive
  - Rotation Matrix
    - Complete, exact, unique, symmetric 3X3 matrix
    - Also easy to update, no singularities
    - Has to have a unit determinant, not intuitive
  - Others include Rodriguez, modified Rodriguez, etc.

# EULER ANGLES

- Given First Axes (xyz), rotate to Second Axes (XYZ) through 3 successive rotations, using 3D rotation matrix.
  - Rotation 1: About z by alpha
  - Rotation 2: About N by beta
  - Rotation 3: About Z by gamma
  - Known as 3-1-3 Euler Angles



# EULER ANGLES




- Aero convention: 3-2-1 Euler Angles
  - Yaw, Pitch, Roll: $\psi, \theta, \phi$
  - Rotation Matrices
    - 3 - Yaw

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

    - 2- Pitch

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

    - 1- Roll

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

# EULER ANGLES

- Rotation Matrix ( often Direction Cosine Matrix (DCM))
  - All three rotations combined

$$R_I^B = R_{\phi,1}R_{\theta,2}R_{\psi,3} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix}$$

  - Rotate from inertial to body coordinates

- To rotate from body to inertial, inverse mapping
  - Recall, inverse = transpose

$$R_B^I = \left(R_I^B\right)^T$$

# Angular Rate Rotations

- Angular rates measured in body frame (*p*,*q*,*r*)
- Euler angles are measured relative to multiple intermediate coordinate frames (3-2-1),
- Euler rates used to update Euler angles in motion
  - Not a rotation matrix
  - Cannot simply transpose for inverse.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix}$$

$$+ \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}$$

# Angular Rate Rotations

- Resulting transformations

$R_e$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

$\overline{R}_e$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \dfrac{\sin\phi}{\cos\theta} & \dfrac{\cos\phi}{\cos\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

# Quaternions

- An alternative way of representing rotations is through quaternions
  - Hamilton (1843) was looking for a field of dimension 4
    - Reals are a field of dimension 1, complex are a field of dimension 2
    - While walking with his wife in Dublin, scribbled the rule of quaternions on a bridge so he would not forget it.

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

    - Everything but commutative multiplication works for quaternions (almost a field)


# QUATERNIONS

- Quaternions are a 4-tuple, divided into a scalar and a 3-vector
  - Let
$$\mathbf{i} = [1 \quad 0 \quad 0]$$
$$\mathbf{j} = [0 \quad 1 \quad 0]$$
$$\mathbf{k} = [0 \quad 0 \quad 1]$$
  - Then a quaternion $q = (q_0, q_1, q_2, q_3) \in \mathbb{R}^4$ can be written as
$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = (q_0, \mathbf{q})$$
  - Addition simply adds the elements
$$q + p = (q_0 + p_0) + (q_1 + p_1)\mathbf{i} + (q_2 + p_2)\mathbf{j} + (q_3 + p_3)\mathbf{k}$$

# QUATERNIONS

- ***Unit quaternions*** can be related to an angle (and a vector), which enables them to represent rotations

$$q_0^2 + \|\mathbf{q}\|^2 = 1 \qquad \cos^2\theta + \sin^2\theta = 1$$

  - Therefore, there must exist an angle $\theta \in (-\pi, \pi]$ defined by a quaternion $q$, such that $\sin\theta = \|\mathbf{q}\|$ and

$$\mathbf{u} = \frac{\mathbf{q}}{\|\mathbf{q}\|} = \frac{\mathbf{q}}{\sin\theta}$$

  - And we can express the unit quaternion and its conjugate as

$$q = \cos\theta + \mathbf{u}\sin\theta$$

$$q^* = \cos\theta - \mathbf{u}\sin\theta$$

# Quaternion update equations

- Similar to the Euler angle update, quaternions can be updated directly from body rotation rates

- If you measure the body rotation rate and form a quaternion version

$$\omega_B = (0, \boldsymbol{\omega}_B)$$

- The quaternion update equation becomes

$$\dot{q} = \frac{1}{2} q \omega_B$$

# Conversions – coded for you

- Matlab code to switch between representations now included in code package
  - Converts between Rotation Matrix, Quaternion, Euler angles and Euler vector, angle representations

```
OUTPUT=SpinCalc(CONVERSION,INPUT,tol,ichk)
```

  - Simple code to create 3D rotation matrices

```
rot.m rotates by an angle about one of 3
principle axes
```

- ROS uses primarily quaternions, but also has built in conversion functions

```
Geometry/RotationMethods
```

# Extra Slides

# QUATERNIONS

- Quaternions are a 4-tuple, divided into a scalar and a 3-vector
  - Multiplication by a constant

$$cq = cq_0 + cq_1\mathbf{i} + cq_2\mathbf{j} + cq_3\mathbf{k}$$

  - The product of two quaternions is defined by Hamilton's rule

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

    - Which implies

$$\mathbf{ij} = \mathbf{k} = -\mathbf{ji}$$
$$\mathbf{jk} = \mathbf{i} = -\mathbf{kj}$$
$$\mathbf{ki} = \mathbf{j} = -\mathbf{ik}$$

# Quaternions

- To get the rule for multiplication, do it out longhand and simplify

$$pq = (p_0 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k})(q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k})$$

  - Let $p = (p_0, \mathbf{p}), q = (q_0, \mathbf{q})$ , then

$$r = pq = \underbrace{p_0 q_0 - \mathbf{p}\cdot\mathbf{q}}_{\text{Scalar part, } r_0} + \underbrace{p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p}\times\mathbf{q}}_{\text{Vector part, } \mathbf{r}}$$

  - In matrix form,

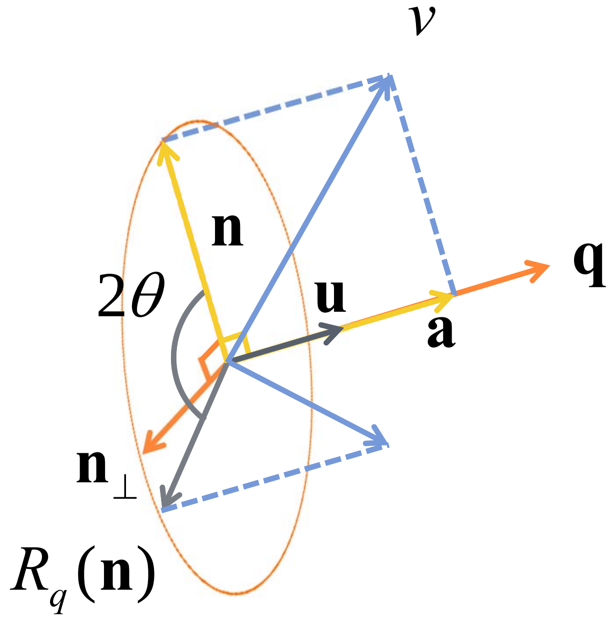
$$r = pq = \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

# QUATERNIONS FOR ROTATIONS

- Theorem: *The quaternion rotation operator*

$$R_q(v) = qvq^*$$

*performs a rotation of vector v by 2θ about axis* **q**.



# Quaternions for rotations

- So now we have a physical interpretation of the quaternion as a combination of a rotation axis **q** and a rotation angle $2\theta$

- We can write the rotation operator in matrix form and extract a conversion to a rotation matrix

$$
\begin{aligned}
\mathbf{v}' &= \begin{bmatrix} 2(q_0^2 + q_1^2) - 1 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & 2(q_0^2 + q_2^2) - 1 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & 2(q_0^2 + q_3^2) - 1 \end{bmatrix} \mathbf{v} \\
&= R\mathbf{v}
\end{aligned}
$$