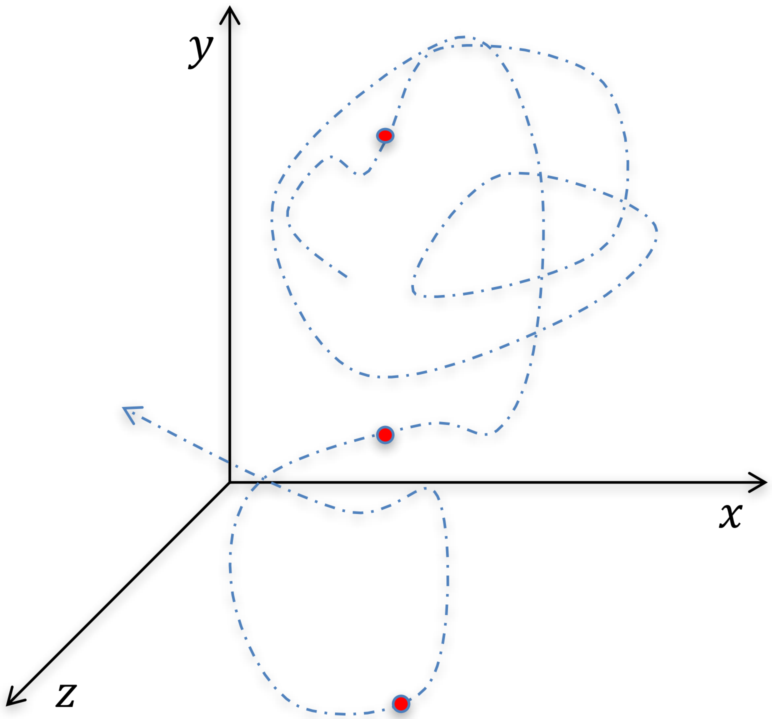
Rigid Bodies

Simulation and Modeling (CSCI 3010U)

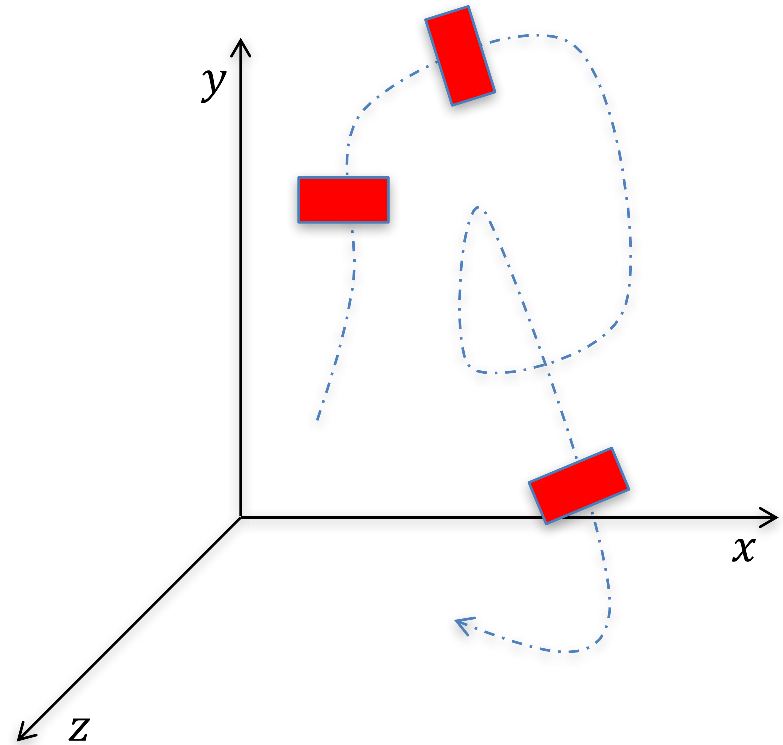
Faisal Z. Qureshi

http://vclab.science.ontariotechu.ca

Rigid Bodies



Particle

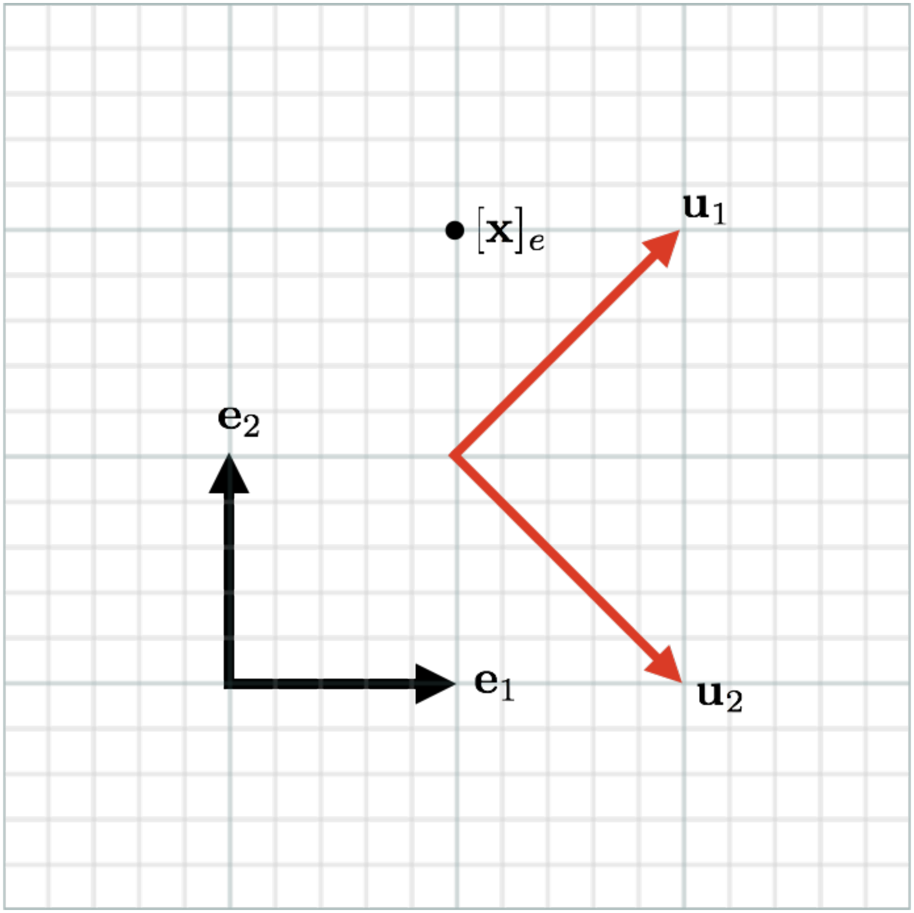


Rigid Body

Particle vs. Rigid Body Dynamics

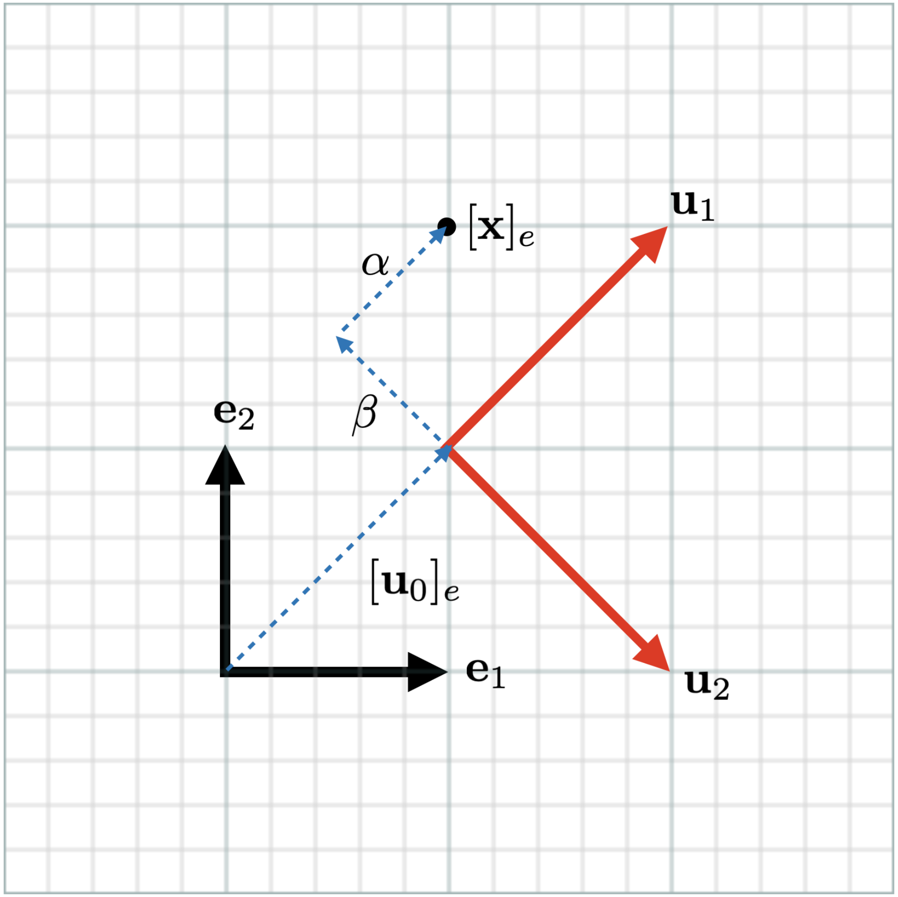
- State of a particle
  - Position $\mathbf{p}$
  - Velocity $\mathbf{v}$
- State of a rigid body
  - Position $\mathbf{p}$
  - Velocity $\mathbf{v}$
  - Orientation θ
  - Angular velocity ω

Coordinate frames



- $[\mathbf{x}]_e$ is $(1, 2)$ in coordinate frame described by $\mathbf{e}_1$ and $\mathbf{e}_2$
- What is $[\mathbf{x}]_u$, i.e., $[\mathbf{x}]_u$ expressed in $\mathbf{u}_1$ and $\mathbf{u}_2$?

Coordinate frames



- $\mathbf{x} = \mathbf{e}_1 + 2\mathbf{e}_2$
- $\mathbf{x} = [\mathbf{u}_0]_e + \alpha[\mathbf{u}_1]_e + \beta[\mathbf{u}_2]_e$
- Note that $[\mathbf{x}]_u = (\alpha, \beta)$, so we are interested in finding values of α and β.

Coordinate frames

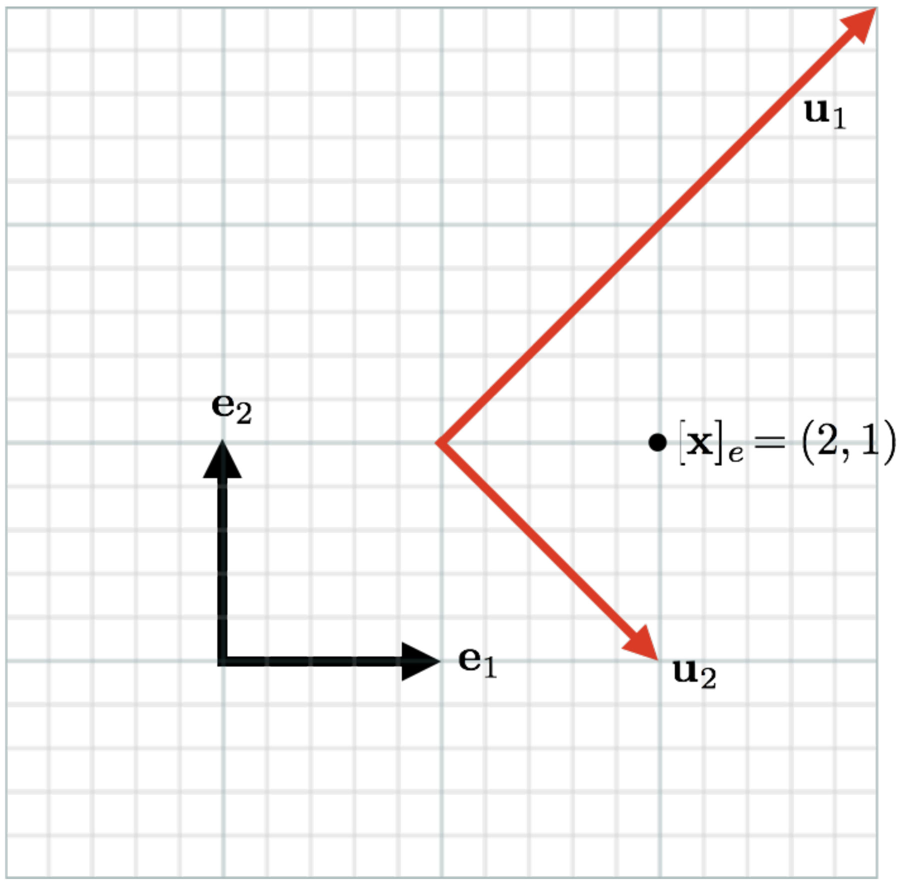
$$
\begin{aligned}
[\mathbf{u}_0]_e + \alpha\,[\mathbf{u}_1]_e + \beta\,[\mathbf{u}_2]_e &= [\mathbf{x}]_e \\
\alpha\,[\mathbf{u}_1]_e + \beta\,[\mathbf{u}_2]_e &= [\mathbf{x}]_e - [\mathbf{u}_0]_e \\
\left[\begin{array}{cc} [\mathbf{u}_1]_e & [\mathbf{u}_2]_e \end{array}\right]\left[\begin{array}{c} \alpha \\ \beta \end{array}\right] &= \\
\left[\begin{array}{c} \alpha \\ \beta \end{array}\right] &= \left[\begin{array}{cc} [\mathbf{u}_1]_e & [\mathbf{u}_2]_e \end{array}\right]^{-1}\left([\mathbf{x}]_e - [\mathbf{u}_0]_e\right)
\end{aligned}
$$

Change of basis

$$
[\mathbf{x}]_u = \left[\begin{array}{cc} [\mathbf{u}_1]_e & [\mathbf{u}_2]_e \end{array}\right]^{-1}\left([\mathbf{x}]_e - [\mathbf{u}_0]_e\right)
$$

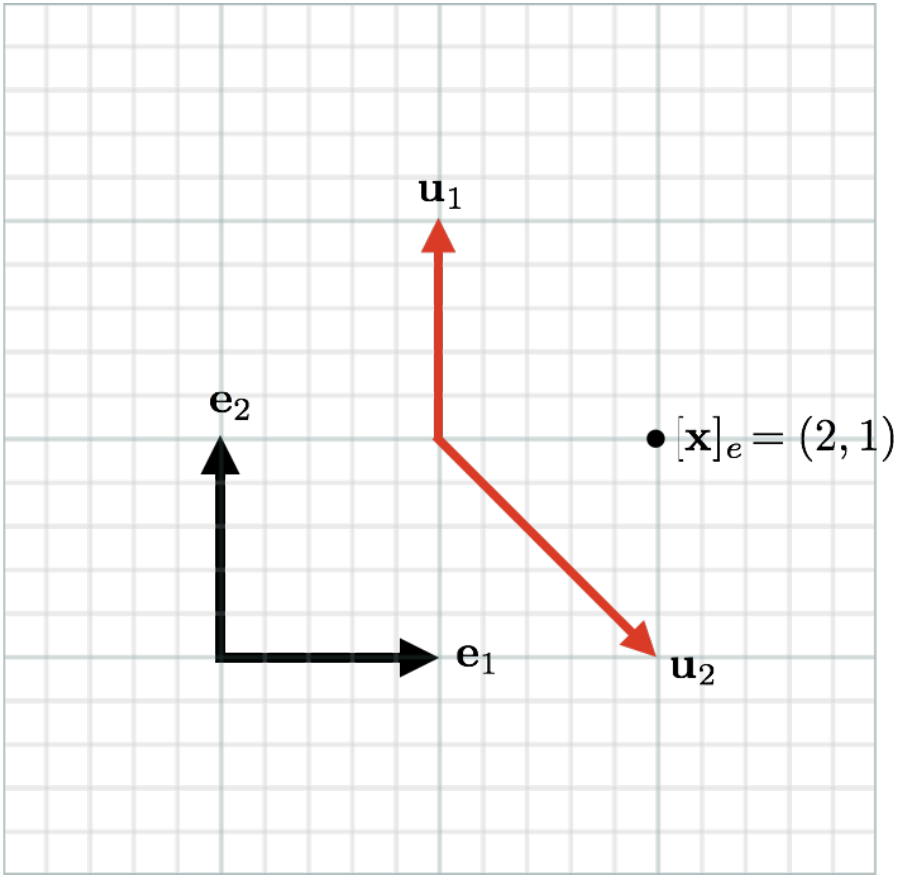
Coordinate Frames Exercise - Part 1

What is $[\mathbf{x}]_u$?

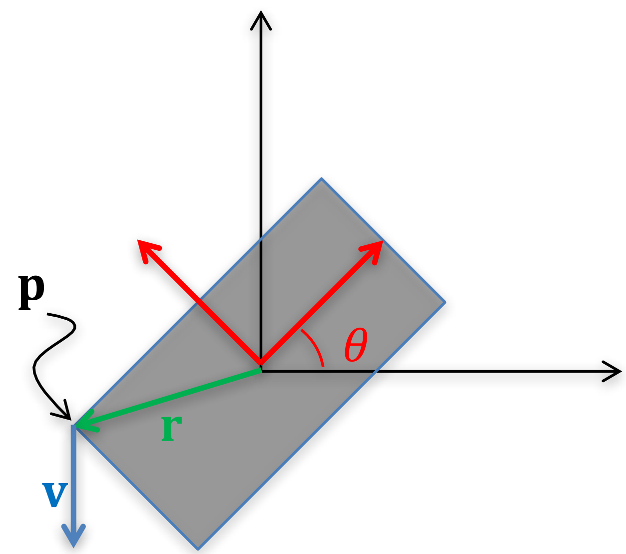


Coordinate Frames Exercise - Part 2

What is $[\mathbf{x}]_u$?



Rigid Bodies in 2D



Angular Velocity

- $\omega = \frac{d\theta}{dt}$
- Units are radians per second
- Radian is the angle subtended by an arc whose length is equal to its radious: $\theta = \frac{l}{r}$

Linear Velocity at a Point on the Body

- $\mathbf{v} = \mathbf{r}\omega$

Rigid Bodies in 3D

- Unlike 2D, orientation in 3D cannot be described using any angle.
- There are many schemes for describing rotations in 3D.
- We will use a 3×3 rotation matrix $\mathbf{R}$ to describe the rotation of rigit body.
- $\mathbf{R}$ is an orthognal matrix
  - Its columns are orthonormal, i.e., $\mathbf{r}_i^T \mathbf{r}_j = 0$ if $i \neq j$, 1 otherwise, where $\mathbf{r}_i$ and $\mathbf{r}_j$ denotes i^{th} and j^{th} columns, respectively
- $\mathbf{R}^T\mathbf{R} = \mathbf{I}$ and $\mathbf{R}\mathbf{R}^T = \mathbf{I}$

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

$$(r_{11}, r_{21}, r_{31}) \cdot (r_{11}, r_{21}, r_{31}) = 1$$

$$r_{11}^2 + r_{21}^2 + r_{31}^2 = 1$$

$$\downarrow \cdot \downarrow = 0$$

Rotations in 3D

Matrices for rotations about the x, y, and z axes

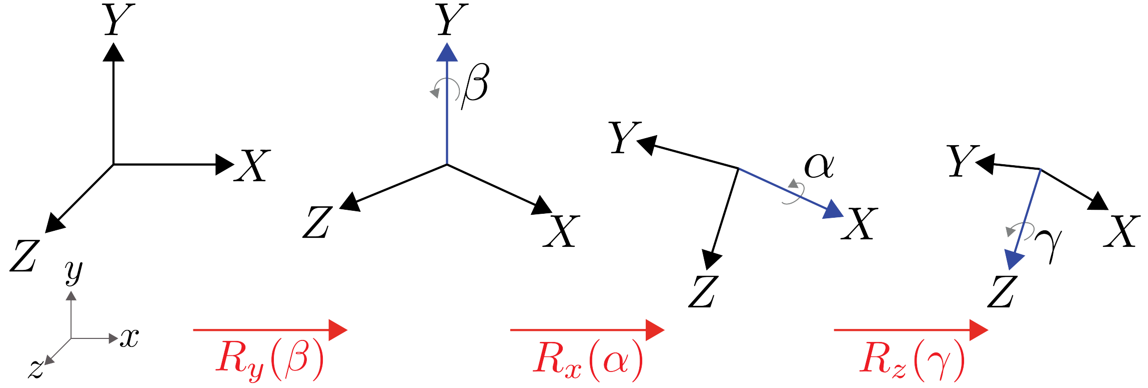
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

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

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

Rotations in 3D

It is possible to describe a rotation as a sequence of three rotations around XYZ coordinate frame axes attached to the moving body

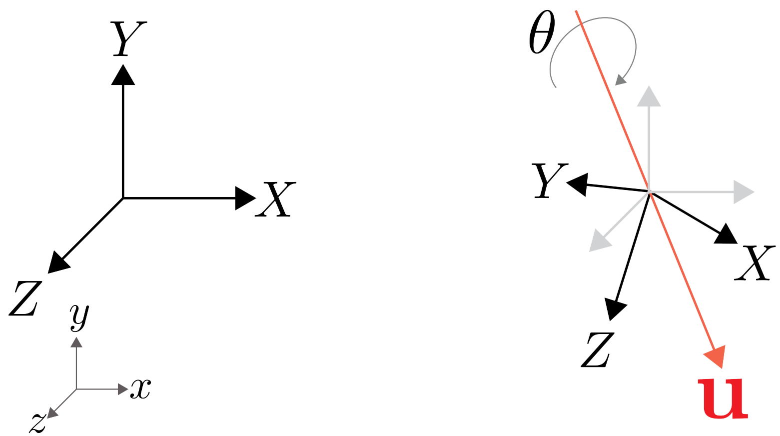


Then rotation matrix $\mathbf{R} = R_z(\gamma)R_x(\theta)R_y(\beta)$

Here, XYZ coordinate system that is attached to the body moves, while the xyz system is fixed. The rotations are with respect to the XYZ coordinate system. It is also possible to define these (elemental) rotations about the axes of a fixed coordinate system xyz.

Rotations in 3D

Another way to represent a rotation in 3D is to use the axis-angle convention.



The matrix of a proper rotation R by angle θ around the axis $\mathbf{u} = (u_x, u_y, u_z)$

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

Rotations in 3D

- We will use a 3x3 rotation matrix $\mathbf{R}$
- We need to find the relationship between angular velocity and rotation matrix.

We will return to this later.

Equations of Motions for Rigid Bodies

Force acting on a Rigid Body

- Net force acting on an object is the rate of change of its linear momentum.

$$\frac{d\mathbf{P}}{dt} = \mathbf{F} \quad \longleftrightarrow \quad F = ma$$

- **Linear momentum**: $\mathbf{P} = m\mathbf{v}$, where m is the mass of the object and $\mathbf{v}$ is its linear velocity

Equations of Motions for Rigid Bodies

Torque acting on a rigid body

- Net torque acting on an object (about point $\mathbf{o}$) is the rate of change of its angular momentum .

$$\frac{d\mathbf{L}}{dt} = \underline{\underline{\mathbf{N}}}$$

torque

- **Angular momentum**: $\underline{\mathbf{L} = \mathbf{I}\omega}$, where $\mathbf{I}$ is the *inertia tensor* and ω is its angular velocity (about point $\mathbf{o}$)

Torque

- Torque (in this example, clockwise or counter-clockwise):

$$\mathbf{T} = \underline{\mathbf{d} \times \mathbf{F}}$$

vector cross-product

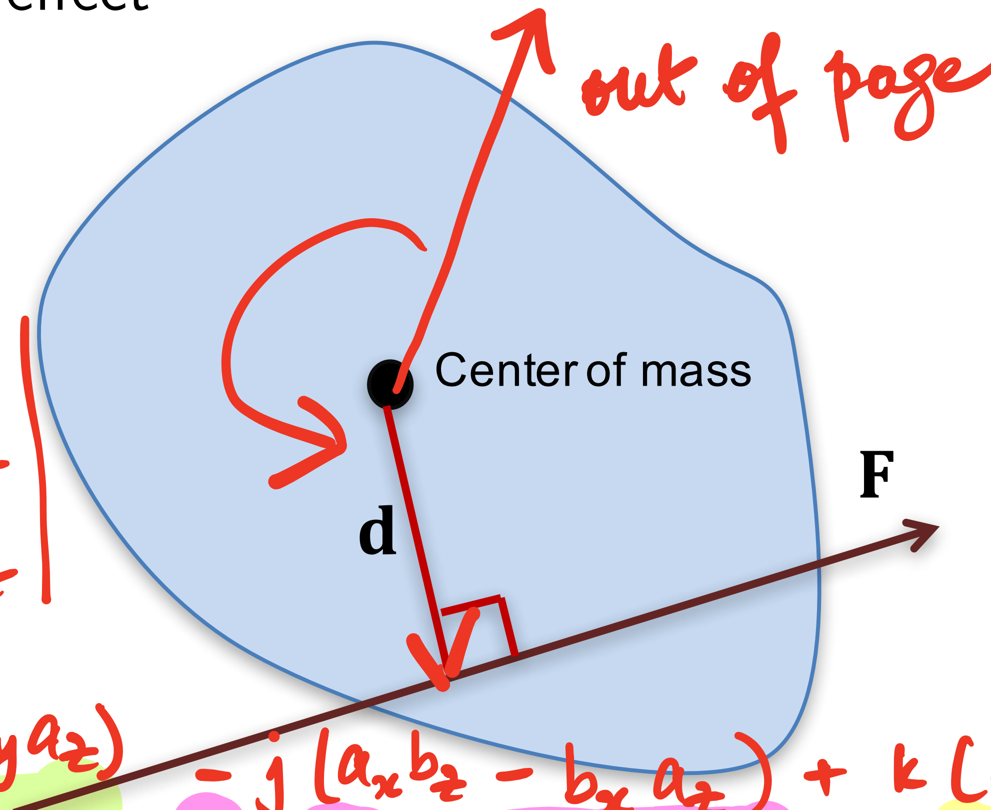
- When force passes through the center of mass, the associated $\mathbf{d}$ vector is zero; therefore, this force produces no torque or rotational effect

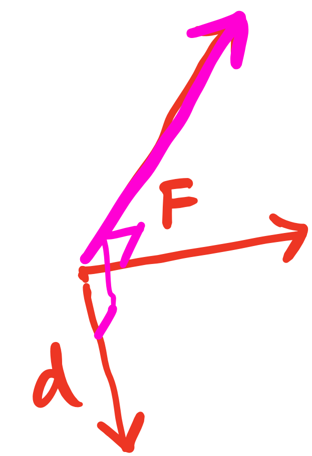
$$\vec{a} = (a_x, a_y, a_z)$$

$$\vec{b} = (b_x, b_y, b_z)$$

$$\vec{a} \times \vec{b} = \begin{vmatrix} i & j & k \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

$$= i(a_y b_z - b_y a_z) - j(a_x b_z - b_x a_z) + k(a_x b_y - b_x a_y)$$



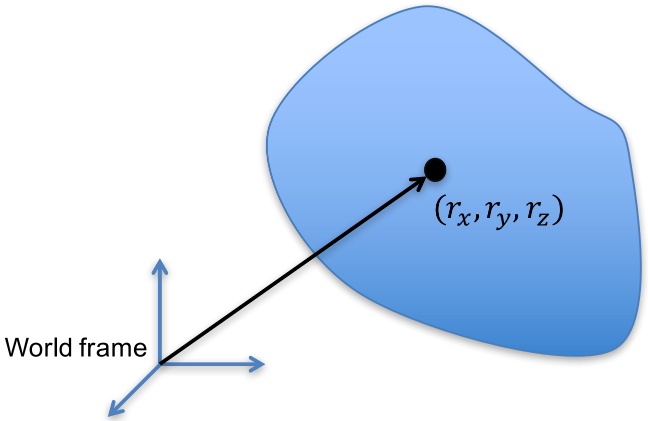


Center of Mass (COM)

- The center of mass is the mean location of all the mass of the body.
- The center of mass $\mathbf{r}$ is defined as

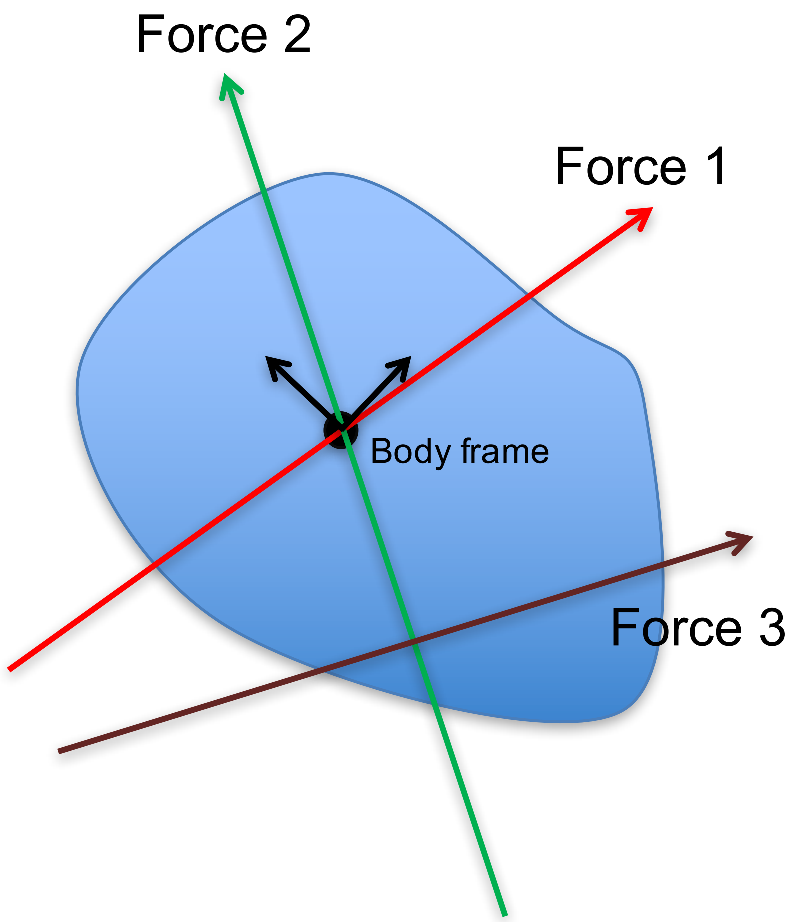
$$r_x = \frac{1}{M}\int \rho(x,y,z)x dV$$
$$r_y = \frac{1}{M}\int \rho(x,y,z)y dV$$
$$r_z = \frac{1}{M}\int \rho(x,y,z)z dV$$

where $\rho(x,y,z)$ is the density at point (x,y,z). M is the total mass of the object. Also, density $= \frac{\text{mass}}{\text{volume}}$.



COM as the origin of the body coordinate frame

- Selecting COM as the origin of the body coordinate frame greatly simpifies the equation of motions
- Any force applied to (or passing through) the COM doesn't induce rotation.



- Force 1 (translation only)
- Force 2 (translation only)
- Force 3 (both translation & rotation)

COM - Discretization

Consider a rigid body composed of N point masses m_i located at positions (x_i, y_i, z_i), respectively, in the world coordinate system. Here $i \in [1, N]$.

Then the center of mass of this rigid body in the world coordinate system is

world coordinates

world coordinate

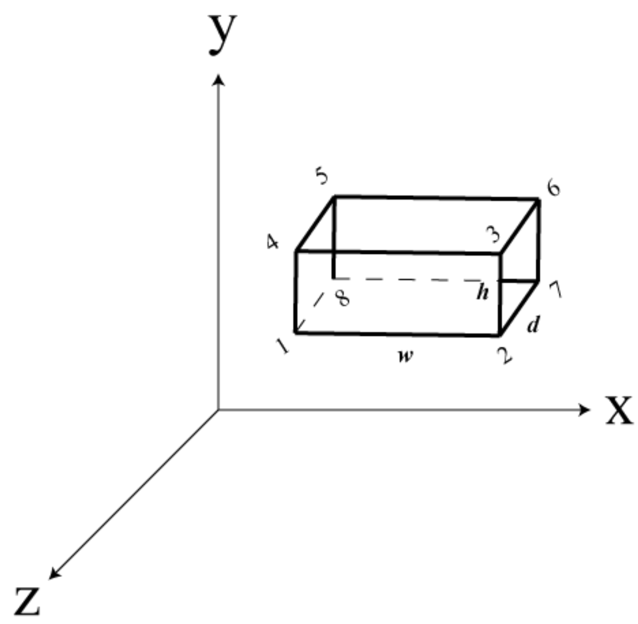
$$r_x = \left(\sum_i m_i \underline{x_i}\right) / \left(\sum_i m_i\right)$$

$$r_y = \left(\sum_i m_i \underline{y_i}\right) / \left(\sum_i m_i\right)$$

$$r_z = \left(\sum_i m_i \underline{z_i}\right) / \left(\sum_i m_i\right)$$

COM - Exercise

Compute the center of mass of a rectangular brick with point masses at its 8 vertices. Assume that vertex 1 is sitting at $(1, 1, 1)$. The values of point masses are m_i, where $i \in [1, 8]$.



Let's assume that $l = 4$, $h = 1$, $d = 2$ and $m_i = 1$ to get things started.

COM - Exercise - Python Code

```python
import numpy as np

#     H ------ G
#    /|       /|
#   / E ---- / F
#  / /      / /
# D ------ C /
# |/       |/
# A ------ B

m = np.ones(8)
r = np.empty((3,8))

l = 4
h = 1
d = 2

r[:,0] = np.array([1,1,1])          # A
r[:,1] = r[:,0] + np.array([l,0,0])  # B
r[:,2] = r[:,0] + np.array([l,h,0])  # C
r[:,3] = r[:,0] + np.array([0,h,0])  # D
r[:,4] = r[:,0] + np.array([0,0,-d]) # E
r[:,5] = r[:,4] + np.array([l,0,0])  # F
r[:,6] = r[:,4] + np.array([l,h,0])  # G
r[:,7] = r[:,4] + np.array([0,h,0])  # H

print('m:\n', m)
print('r:\n', r)
```

location of the point masses in the world

```python
M = np.sum(m) # Total mass
print('M:\n', M)

m_tmp = np.tile(m, (3,1))
print(r * np.tile(m, (3,1)))

center_of_mass =
    np.sum(r * np.tile(m, (3,1)), axis=1) / M
print('center of mass:\n', center_of_mass)
```

COM - Example - Program Output

```
m:
 [1. 1. 1. 1. 1. 1. 1. 1.]
r:
 [[ 1.  5.  5.  1.  1.  5.  5.  1.]
 [ 1.  1.  2.  2.  1.  1.  2.  2.]
 [ 1.  1.  1.  1. -1. -1. -1. -1.]]
M:
 8.0
[[ 1.  5.  5.  1.  1.  5.  5.  1.]
 [ 1.  1.  2.  2.  1.  1.  2.  2.]
 [ 1.  1.  1.  1. -1. -1. -1. -1.]]
center of mass:
 [3.  1.5 0. ]
```

Inertia Tensor

Inertia tensor provides a concise description of the mass distribution around the center of mass. $\rho(x, y, z)$ denotes density at center-of-mass centered point (x, y, z). Recall density $= \frac{\text{mass}}{\text{volume}}$.

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

$$I_{xx} = \int \rho(x, y, z)(y^2 + z^2)dV$$

$$I_{yy} = \int \rho(x, y, z)(z^2 + x^2)dV$$

$$I_{zz} = \int \rho(x, y, z)(x^2 + y^2)dV$$

$$I_{xy} = I_{yx} = \int \rho(x, y, z)xydV$$

$$I_{xz} = I_{zx} = \int \rho(x, y, z)xzdV$$

$$I_{yz} = I_{zy} = \int \rho(x, y, z)yzdV$$

Inertia Tensor - Discretization

Consider a rigid body composed of N point masses m_i located at center-of-mass centered positions (x_i, y_i, z_i), respectively, in the world coordinate system. Here $i \in [1, N]$.

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

$$I_{xx} = \sum_i m_i(\underline{y_i^2 + z_i^2})$$

$$I_{yy} = \sum_i m_i(\underline{z_i^2 + x_i^2})$$

$$I_{zz} = \sum_i m_i(\underline{x_i^2 + y_i^2})$$

$$I_{xy} = I_{yx} = \sum_i m_i \underline{x_i y_i}$$

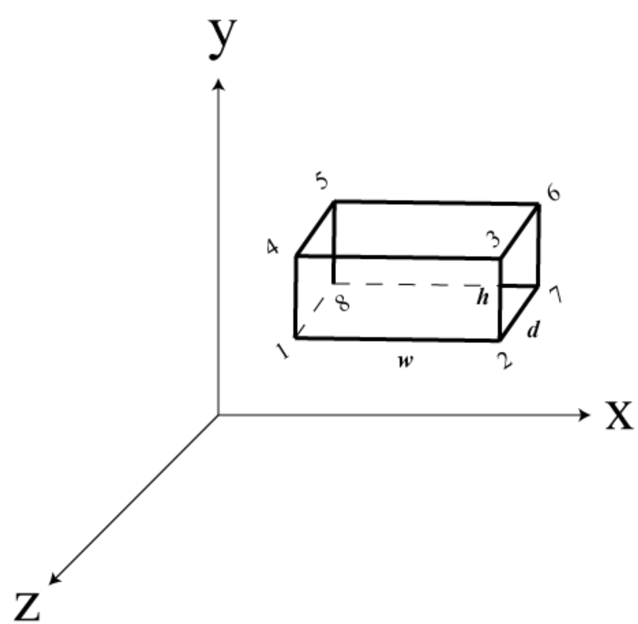
world system

$$I_{xz} = I_{zx} = \sum_i m_i \underline{x_i z_i}$$

$$I_{yz} = I_{zy} = \sum_i m_i \underline{y_i z_i}$$

Inertia Tensor - Exercise

Compute the center of mass of a rectangular brick with point masses at its 8 vertices. Assume that vertex 1 is sitting at $(1, 1, 1)$. The values of point masses are m_i, where $i \in [1, 8]$.



Let's assume that $l = 4$, $h = 1$, $d = 2$ and $m_i = 1$ to get things started.

Inertia Tensor - Exercise - Python Code

com

```
# continued from the previous example

rp = r - np.tile(center_of_mass , (8,1)).T
print('rp:\n', rp)
```

I

```
I = np.empty((3,3))
I[0,0] = np.sum(np.multiply(np.power(rp[1,:],2) + np.power(rp[2,:],2), m))
I[1,1] = np.sum(np.multiply(np.power(rp[2,:],2) + np.power(rp[0,:],2), m))
I[2,2] = np.sum(np.multiply(np.power(rp[0,:],2) + np.power(rp[1,:],2), m))
I[0,1] = I[1,0] = np.sum(np.multiply(np.multiply(rp[0,:], rp[1,:]), m))
I[0,2] = I[2,0] = np.sum(np.multiply(np.multiply(rp[0,:], rp[2,:]), m))
I[1,2] = I[2,1] = np.sum(np.multiply(np.multiply(rp[1,:], rp[2,:]), m))
print('I:\n', I)
```

Intertia Tensor - Exercise - Program Output

```
rp:
 [[-2.   2.   2.  -2.  -2.   2.   2.  -2. ]
 [-0.5 -0.5  0.5  0.5 -0.5 -0.5  0.5  0.5]
 [ 1.   1.   1.   1.  -1.  -1.  -1.  -1. ]]
I:
 [[10.  0.  0.]
 [ 0. 40.  0.]
 [ 0.  0. 34.]]
```

Inertia Tensor in the Body Coordinate Frame

- The inertia tensor $\mathbf{I}$ that we just computed is expressed in the world coordinate frame. Consequently it changes as the orientation of the rigid body changes.
- We can express the inertia tensor in the body coordinate frame.
- We refer to inertia tensor in the body coordinate frame as $\mathbf{I}_{body}$.
- $\mathbf{I}_{body}$ doesn't change as the orientation of the body changes.
- $\mathbf{I}_{body}$ is diagonal, i.e.,

$$\mathbf{I}_{body} = \begin{bmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{bmatrix}$$

A large value I_{11} means rot. around x-axis will be slow compared to other axes for a similar force.

- Inverse of $\mathbf{I}_{body}$:

$$\mathbf{I}_{body}^{-1} = \begin{bmatrix} \frac{1}{I_{11}} & 0 & 0 \\ 0 & \frac{1}{I_{22}} & 0 \\ 0 & 0 & \frac{1}{I_{33}} \end{bmatrix}$$

Computing $\mathbf{I}_{body}$

Option 1

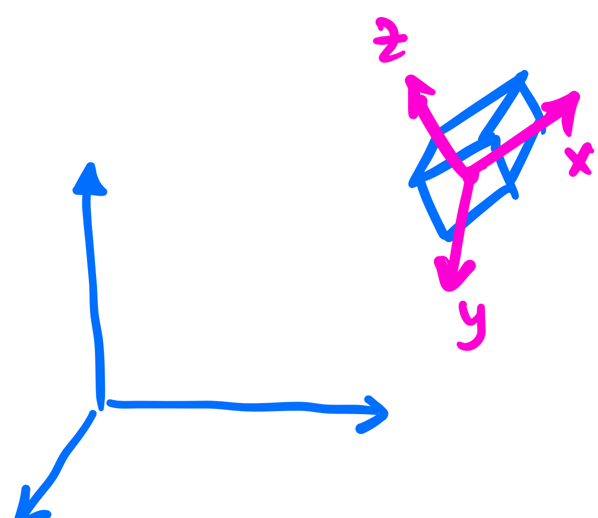
Diagonalize $\mathbf{I}$

- Compute eigenvectors and eigenvalues of $\mathbf{I}$ ✓
- Eigenvalues form the diagonal matrix $\mathbf{I}_{body}$
- Eigenvectors form the 3-by-3 rotation matrix $\mathbf{R}$ that describes the orientation of the rigid body
- This is the preferred approach

Option 2

Use 3-by-3 rotation matrix $\mathbf{R}$ that describes the orientation of the rigid body

- $\mathbf{I}_{body} = \mathbf{R}^T \mathbf{I} \mathbf{R}$



COM → Position

I → mass angular effects

xyz → Rot.

Inertia Tensor - Rotated Body Example - Python Code

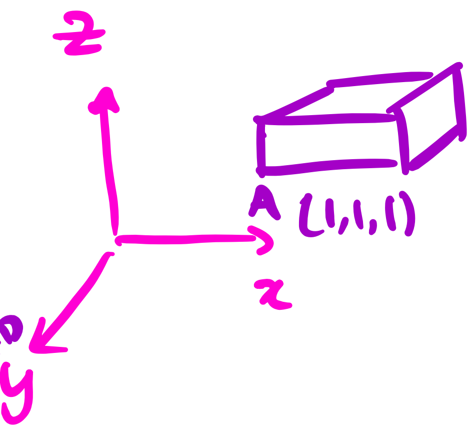
```
# Continued from previous example
from scipy.spatial.transform import Rotation as R

rot_mat = R.from_euler('y',45, degrees=True).as_matrix()
print('rot_mat:\n', rot_mat)

# Note:  1) rp; 2) center of mass; and 3) overwriting r
r = np.dot(rot_mat, rp) + np.tile(center_of_mass, (8,1)).T
print('rotated_r:\n', r)

center_of_mass = np.sum(r * np.tile(m, (3,1)), axis=1) / M
print('center of mass:\n', center_of_mass)
```

COM NEW ROTATED




```
rp = r - np.tile(center_of_mass , (8,1)).T
print('rp:\n', rp)

I = np.empty((3,3))
I[0,0] = np.sum(np.multiply(np.power(rp[1,:],2) + np.power(rp[2,:],2), m))
I[1,1] = np.sum(np.multiply(np.power(rp[2,:],2) + np.power(rp[0,:],2), m))
I[2,2] = np.sum(np.multiply(np.power(rp[0,:],2) + np.power(rp[1,:],2), m))
I[0,1] = I[1,0] = np.sum(np.multiply(np.multiply(rp[0,:], rp[1,:]), m))
I[0,2] = I[2,0] = np.sum(np.multiply(np.multiply(rp[0,:], rp[2,:]), m))
I[1,2] = I[2,1] = np.sum(np.multiply(np.multiply(rp[1,:], rp[2,:]), m))
print('I:\n', I)
```

INERTIA TENSOR ROTATED.

```
# Computing I_body using rotation matrix
I_body = np.dot(np.dot(rot_mat.T, I), rot_mat)
print('I_body:\n', I_body)
```

$R^T I R \to I_{body}$

```
# Computing I_body using eigenvalues and eigenvectors
w, v = np.linalg.eig(I)
print('eigenvalues:\n', w)
print('eigenvectors:\n', v)
```

Inertia Tensor - Rotated Body Example - Program Output

```
rot_mat:
 [[ 0.70710678  0.          0.70710678]
 [ 0.          1.          0.        ]
 [-0.70710678  0.          0.70710678]]
rotated_r:
 [[ 2.29289322  5.12132034  5.12132034  2.29289322  0.87867966  3.70710678
   3.70710678  0.87867966]
 [ 1.          1.          2.          2.          1.          1.
   2.          2.        ]
 [ 2.12132034 -0.70710678 -0.70710678  2.12132034  0.70710678 -2.12132034
  -2.12132034  0.70710678]]
center of mass:
 [3.  1.5 0. ]
rp:
 [[-0.70710678  2.12132034  2.12132034 -0.70710678 -2.12132034  0.70710678
   0.70710678 -2.12132034]
 [-0.5        -0.5         0.5         0.5        -0.5        -0.5
   0.5         0.5       ]
 [ 2.12132034 -0.70710678 -0.70710678  2.12132034  0.70710678 -2.12132034
  -2.12132034  0.70710678]]
I:
 [[ 22.   0. -12.]
 [  0.  40.   0.]
 [-12.   0.  22.]]
I_body:
 [[3.40000000e+01 0.00000000e+00 8.45096405e-15]
 [0.00000000e+00 4.00000000e+01 0.00000000e+00]
 [7.79029649e-15 0.00000000e+00 1.00000000e+01]]
eigenvalues:
 [34. 10. 40.]
eigenvectors:
 [[ 0.70710678  0.70710678  0.        ]
 [ 0.          0.          1.        ]
 [-0.70710678  0.70710678  0.        ]]
```

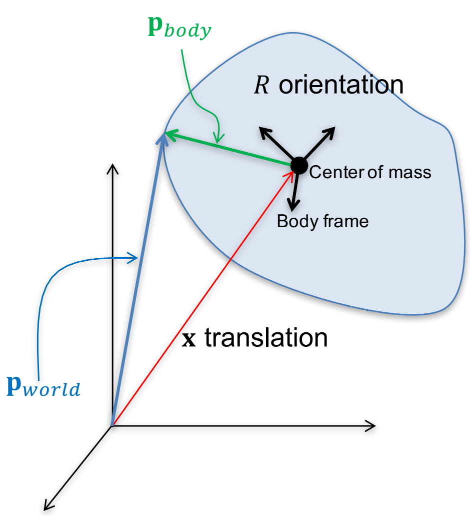
$R_y(45°)$

Inertia Tensor

- Inertia tensors are available for many canonical objects: rectangles, circles, spheres, etc.
- Efficient algorithms exist to compute inertia tensor, center of mass, body coordinate frames a given polygonal model of an object
- Many tools exist to construct polygonal models of 2D/3D rigid objects

Body coordinate frame

Attach a coordinate frame to a rigid body



- **Origin**: center of mass(defined in the world frame)
- **Axes**: defined in the world coordinate frame by a 3-by-3 rotation matrix $\mathbf{R}$. Columns of $\mathbf{R}$ define the x, y and z axes of the body coordinate frame
  - Inertia tensor $\mathbf{I}_{body}$ is constant and diagonal in this frame
- From body coordinate frame to world coordinate frame

$$\mathbf{p}_{world} = \mathbf{R}\mathbf{p}_{body} + \mathbf{x}$$

$$p_{body} = R^T(p_{world} - x)$$

World and Body Coordinate Frames

World coordinate frame

- Collision detection and response
- Display and visualization

Body coordinate frame

- Compute quantitites such as inertia tensor once and store them for later use.

Rigid Body Dynamics

State variables

Position	$\mathbf{x}$	1 by 3 vector
Orientation	R	3 by 3 rotation matrix
Linear Momentum	$\mathbf{P}$	1 by 3 vector
Angular Momentum	$\mathbf{L}$	1 by 3 vector

Constants

Mass	m	scalar
Inertia tensor	I_{body}	3 by 3 matrix (in body frame)

Derived quantities

Linear velocity	v	1 by 3 vector
Angular velocity	ω	1 by 3 vector
Inertia tensor	I^{-1}	3 by 3 matrix (in world frame)

Total force	$\mathbf{F}$	1 by 3 vector
Total torque	$\mathbf{T}$	1 by 3 vector

Rigid Body Dynamics

Linear effects

- $d\mathbf{x}/dt = \mathbf{v}$
- $d\mathbf{P}/dt = \mathbf{F}$
- $\mathbf{v} = \mathbf{P}/m$

Angular effects

- $d\mathbf{R}/dt = \omega^*\mathbf{R}$, where

[3x3] rot. matrix

$$\omega^* = \begin{bmatrix} 0 & -\omega_x & \omega_y \\ \omega_z & 0 & -\omega_x \\ \omega_y & \omega_z & 0 \end{bmatrix}$$

- $d\mathbf{L}/dt = \mathbf{N}$
- $\omega = \mathbf{I}^{-1}\mathbf{L}$
- $\mathbf{I}^{-1} = \mathbf{R}\mathbf{I}_{body}^{-1}\mathbf{R}^T$

Rigid Body Dynamics

18

```
// x - position
state[0] = x[0];
state[1] = x[1];
state[2] = x[2];

// R - orientation
state[3] = R[0][0];
state[4] = R[1][0];
state[5] = R[2][0];
state[6] = R[0][1];
state[7] = R[1][1];
state[8] = R[2][1];
state[9] = R[0][2];
state[10] = R[1][2];
state[11] = R[2][2];

// P - linear momentum
state[12] = P[0];
state[13] = P[1];
state[14] = P[2];

// L - angular momentum
state[15] = L[0];
state[16] = L[1];
state[17] = L[2];

// t - OSP needs it
state[18] = 0.0
```

Init: Flatten state variables into a state vector

```
// dx/dt = v
rate[0] = v[0];
rate[1] = v[1];
rate[2] = v[2];

// dR/dt = w* R
double[][] Rdot =
mult(star(omega), R);
rate[3] = Rdot[0][0];
rate[4] = Rdot[1][0];
rate[5] = Rdot[2][0];
rate[6] = Rdot[0][1];
rate[7] = Rdot[1][1];
rate[8] = Rdot[2][1];
rate[9] = Rdot[0][2];
rate[10] = Rdot[1][2];
rate[11] = Rdot[2][2];

// dP/dt = force
rate[12] = force[0];
rate[13] = force[1];
rate[14] = force[2];

// dL/dt = torque
rate[15] = torque[0];
rate[16] = torque[1];
rate[17] = torque[2];

// dt/dt = 1
rate[18] = 1;
```

Rate[] encodes 1st order ODE for our system

```
odeSolver.step();

// x
x[0] = state[0];
x[1] = state[1];
x[2] = state[2];

// R
R[0][0] = state[3];
R[1][0] = state[4];
R[2][0] = state[5];
R[0][1] = state[6];
R[1][1] = state[7];
R[2][1] = state[8];
R[0][2] = state[9];
R[1][2] = state[10];
R[2][2] = state[11];
R = orthonormalize(R);

// P
P[0] = state[12];
P[1] = state[13];
P[2] = state[14];

// L
L[0] = state[15];
L[1] = state[16];
L[2] = state[17];

Iinv = mult(R, mult(IbodyInv, transpose(R)));
omega = mult(Iinv, L);
```

Let ODE solve the state and then copy the state back to our state variables $\mathbf{x}$, R, $\mathbf{L}$ and $\mathbf{T}$.

Rigid Body Dynamics: Numerical Considerations

- Over time numerical errors accumulate in rotation matrix $\mathbf{R}$
- This effects our computation of $\mathbf{I}$ and ω
- Orthonormalize $\mathbf{R}$ after every timestep

Orthonormalization

1. Normalize $\mathbf{R}_1$
2. $\mathbf{R}_3 = \mathbf{R}_1 \times \mathbf{R}_2$ (normalize $\mathbf{R}_3$)
3. $\mathbf{R}_2 = \mathbf{R}_3 \times \mathbf{R}_1$ (normalize $\mathbf{R}_2$)

Here $\mathbf{R}_i$ represent the i-th row of matrix $\mathbf{R}$
Errors were shifted in the matrix

Representing Rotations

- We chose to represent rotations as 3-by-3 rotation matrices
- Quaternions can be used to represent rotations as well
- Most rigid body dynamics systems use quaterions
- See Ch. 17 of the textbook

Representing Rigid Bodies

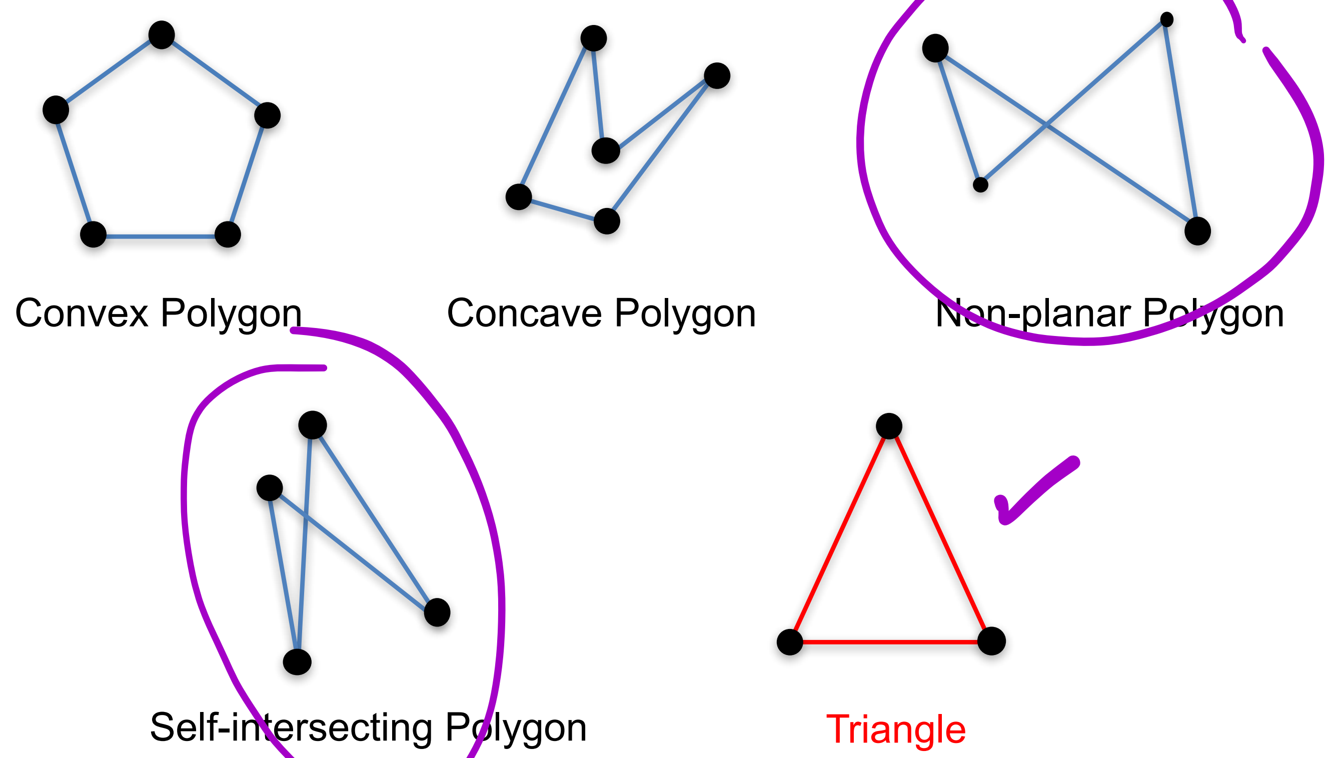
- A rigid body has a shape that does not change over time
- It can translate through space and rotate
- A rigid body occupies a volume of space
- The distribution of its mass over this volume determines its motion or dynamics

Shape representation

- Shape representation is studied extensively in computer graphics and some areas of mechanical engineering and mathematics
- There are many ways of representing shape, each with a different set of advantages and disadvantages
- We will stick to polygons

Shape Representation using Polygons

- The surface of the object is represented by a collection of polygons
- The polygons are connected across their edges to form a continuous surface
- In order to have a well behaved representation we need to constrain our polygons
- First all of the polygons must be convex, note that we can always convert a concave polygon into two or more convex ones



Shape Representation using Triangles

- We will stick to triangles
- Any convex polygon can be converted to a collection of triangles

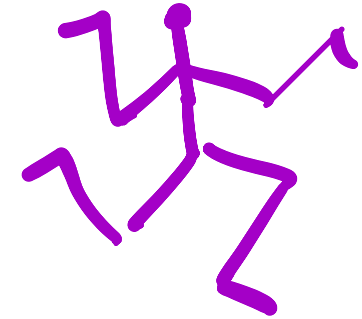
Advantages

- We are only dealing with one type of polygon, a uniform representation
- Triangles are the simplest polygon, makes our algorithms simpler
- Many modeling programs allow us to construct polygonal models
- Easy to display
- Many efficient algorithms exist for manipulating triangles

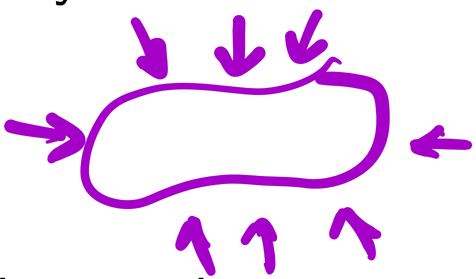
Disadvantages

- Not a compact representation
- Not a good approximation for curved surfaces

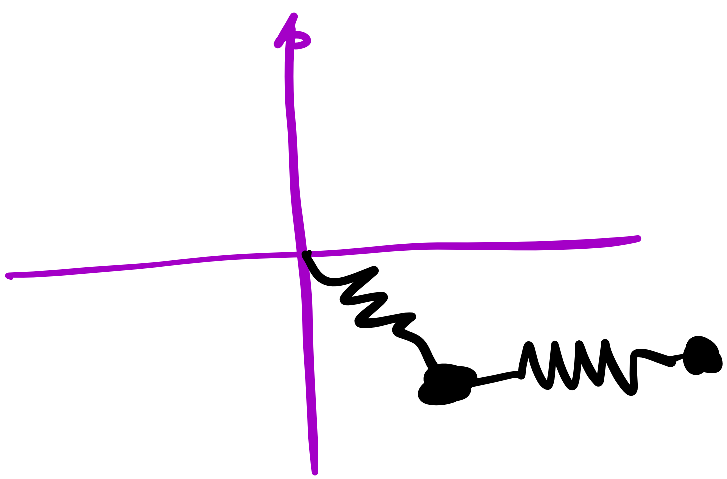
Other Types of Dynamics

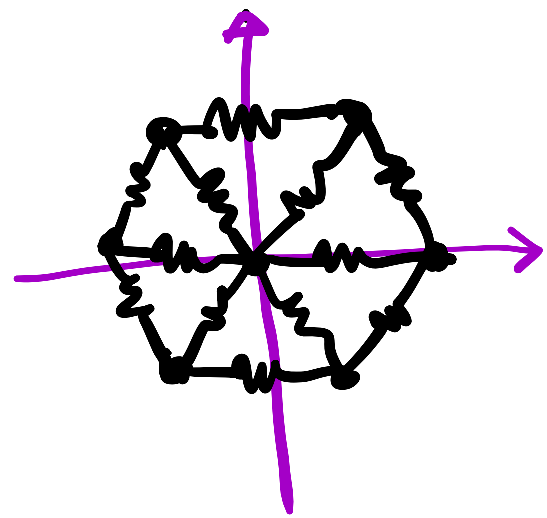


- Articulated figures
  - Rigid bodies connected by joints and hinges
  - Used to model the dynamics of human figures
- Vehicle dynamics used to model the dynamics of various kinds of vehicles
- Deformable objects
  - Cloth, soft toys, etc.



- These are more complicated than what we have seen so far





Readings

- Ch. 17 of the textbook
- *Classical Mechanics (3rd Edition)* by H. Goldstein and C.P. Poole Jr.