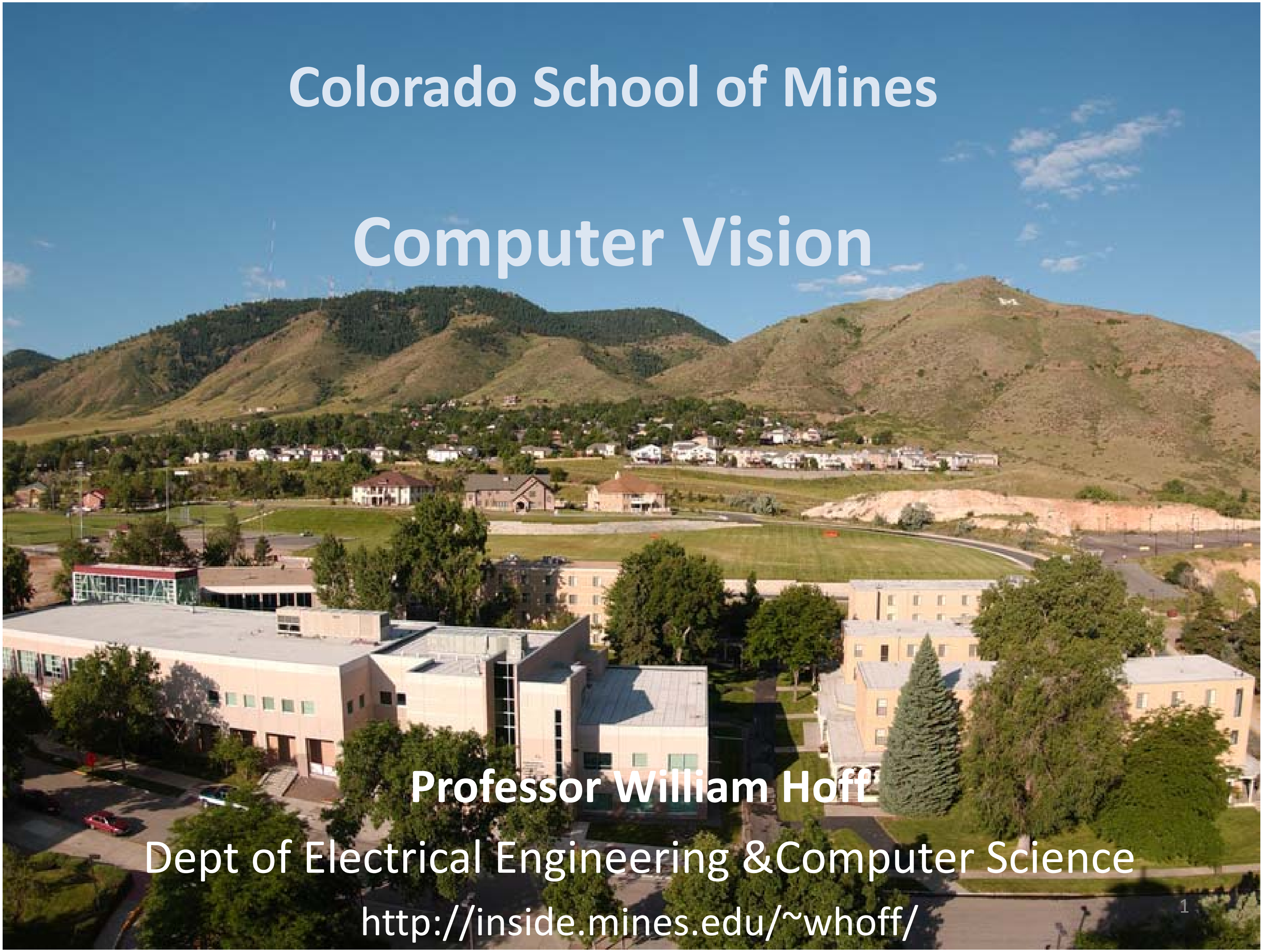


Colorado School of Mines

Computer Vision

Professor William Hoff

Dept of Electrical Engineering &Computer Science

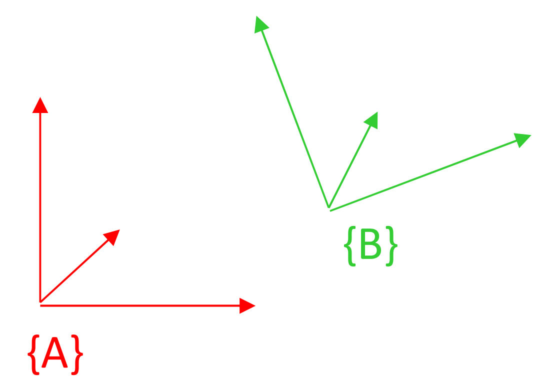
http://inside.mines.edu/~whoff/

3D-3D Coordinate Transforms

An excellent reference is the book "Introduction to Robotics" by John Craig

3D Coordinate Systems

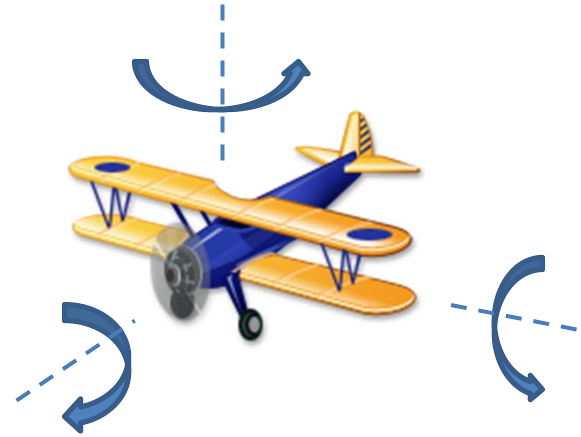
- Coordinate frames
  - Denote as {A}, {B}, etc
  - Examples: camera, world, model
- The pose* of {B} with respect to {A} is described by
  - Translation, or position
  - Rotation, or orientation
- Translation is just a 3D vector; rotation is more complicated



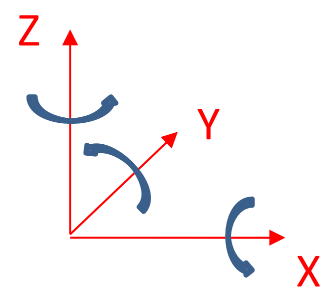
**The term "pose" means position and orientation*

Rotations in 3D

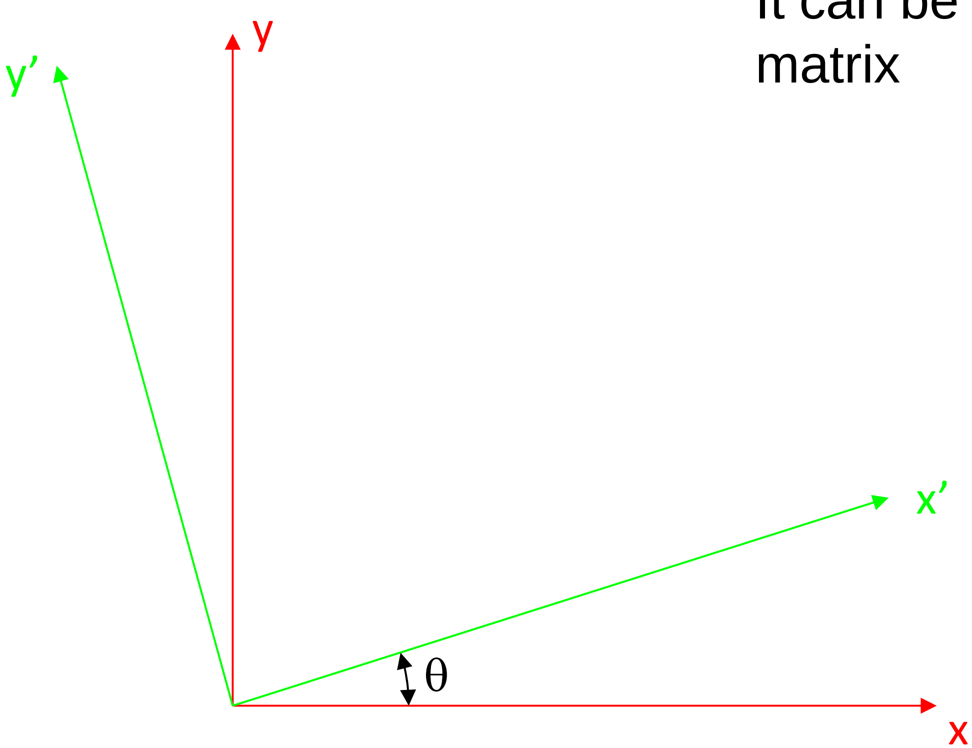
- A 3D rotation has 3 degrees of freedom
  - Namely, it takes 3 numbers to describe the orientation of an object in the world
  - Think of "roll", "pitch", "yaw" for an airplane



- We can represent a 3D rotation by doing successive rotations about the X,Y, and Z axes



You are probably familiar with rotations in 2D



It can be represented by a 2x2 rotation matrix

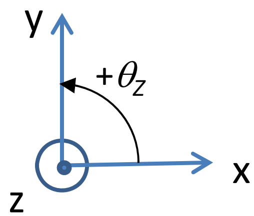
$$\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{R} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

- Note: **R** is orthonormal
  - Rows, columns are orthogonal ($\mathbf{r}_1 \cdot \mathbf{r}_2 = 0$, $\mathbf{c}_1 \cdot \mathbf{c}_2 = 0$)
  - Transpose is the inverse; $\mathbf{R}\mathbf{R}^T = \mathbf{I}$
  - Determinant is $|\mathbf{R}| = 1$

To do 3D rotations, we can create 2D rotation matrices for each axis

- Rotation about the Z axis

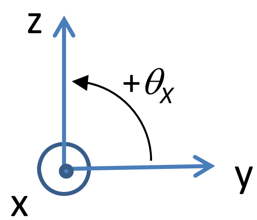


⊙ Points toward me

⊗ Points away from me

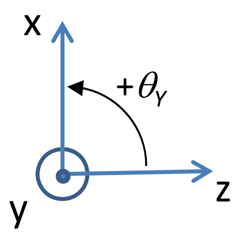
$$\begin{pmatrix} {}^{B}x \\ {}^{B}y \\ {}^{B}z \end{pmatrix} = \begin{pmatrix} \cos\theta_Z & -\sin\theta_Z & 0 \\ \sin\theta_Z & \cos\theta_Z & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} {}^{A}x \\ {}^{A}y \\ {}^{A}z \end{pmatrix}$$

- Rotation about the X axis



$$\begin{pmatrix} {}^{B}x \\ {}^{B}y \\ {}^{B}z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_X & -\sin\theta_X \\ 0 & \sin\theta_X & \cos\theta_X \end{pmatrix} \begin{pmatrix} {}^{A}x \\ {}^{A}y \\ {}^{A}z \end{pmatrix}$$

- Rotation about the Y axis



$$\begin{pmatrix} {}^{B}x \\ {}^{B}y \\ {}^{B}z \end{pmatrix} = \begin{pmatrix} \cos\theta_Y & 0 & \sin\theta_Y \\ 0 & 1 & 0 \\ -\sin\theta_Y & 0 & \cos\theta_Y \end{pmatrix} \begin{pmatrix} {}^{A}x \\ {}^{A}y \\ {}^{A}z \end{pmatrix}$$

Note signs are different than in the other cases

3D Rotation Matrix

- We can concatenate the 3 rotations to yield a single 3x3 rotation matrix; e.g.,

$$
{}^{A}_{B}R_{XYZ}(\theta_X, \theta_Y, \theta_Z) = R_Z(\theta_Z)\ R_Y(\theta_Y)\ R_X(\theta_X)
$$

$$
= \begin{pmatrix} cz & -sz & 0 \\ sz & cz & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cy & 0 & sy \\ 0 & 1 & 0 \\ -sy & 0 & cy \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cx & -sx \\ 0 & sx & cx \end{pmatrix}
$$

where

$$
cx = \cos(\theta_X),\ sy = \sin(\theta_Y),\ \text{etc}
$$

- Note: we use the convention that to rotate a vector, we pre-multiply it; i.e., **v’** = **R v**
  - This means that if **R** = $\mathbf{R}_Z\,\mathbf{R}_Y\,\mathbf{R}_X$, we actually apply the X rotation first, then the Y rotation, then the Z rotation

Python: Creating a Rotation Matrix

```
import numpy as np

ax, ay, az = 0.1, -0.2, 0.3 # radians

sx, sy, sz = np.sin(ax), np.sin(ay), np.sin(az)
cx, cy, cz = np.cos(ax), np.cos(ay), np.cos(az)

Rx = np.array(((1, 0, 0), (0, cx, -sx), (0, sx, cx)))
Ry = np.array(((cy, 0, sy), (0, 1, 0), (-sy, 0, cy)))
Rz = np.array(((cz, -sz, 0), (sz, cz, 0), (0, 0, 1)))

# Apply X rotation first, then Y, then Z
R = Rz @ Ry @ Rx    # Use @ for matrix mult
print(R)


# Apply Z rotation first, then Y, then X
R = Rx @ Ry @ Rz
print(R)
```

Note the strange symbol @ for matrix multiplication!

Python: Creating a Rotation Matrix

```
import numpy as np

ax, ay, az = 0.1, -0.2, 0.3 # radians

sx, sy, sz = np.sin(ax), np.sin(ay), np.sin(az)
cx, cy, cz = np.cos(ax), np.cos(ay), np.cos(az)

Rx = np.array(((1, 0, 0), (0, cx, -sx), (0, sx, cx)))
Ry = np.array(((cy, 0, sy), (0, 1, 0), (-sy, 0, cy)))
Rz = np.array(((cz, -sz, 0), (sz, cz, 0), (0, 0, 1)))

# Apply X rotation first, then Y, then Z
R = Rz @ Ry @ Rx    # Use @ for matrix mult
print(R)
```

```
[[ 0.93629336 -0.31299183 -0.15934508]
 [ 0.28962948  0.94470249 -0.153792  ]
 [ 0.19866933  0.0978434   0.97517033]]
```

```
# Apply Z rotation first, then Y, then X
R = Rx @ Ry @ Rz
print(R)
```

Python: Creating a Rotation Matrix

```
import numpy as np

ax, ay, az = 0.1, -0.2, 0.3 # radians

sx, sy, sz = np.sin(ax), np.sin(ay), np.sin(az)
cx, cy, cz = np.cos(ax), np.cos(ay), np.cos(az)

Rx = np.array(((1, 0, 0), (0, cx, -sx), (0, sx, cx)))
Ry = np.array(((cy, 0, sy), (0, 1, 0), (-sy, 0, cy)))
Rz = np.array(((cz, -sz, 0), (sz, cz, 0), (0, 0, 1)))

# Apply X rotation first, then Y, then Z
R = Rz @ Ry @ Rx    # Use @ for matrix mult
print(R)
```

```
[[ 0.93629336 -0.31299183 -0.15934508]
 [ 0.28962948  0.94470249 -0.153792  ]
 [ 0.19866933  0.0978434   0.97517033]]
```

Different!

```
# Apply Z rotation first, then Y, then X
R = Rx @ Ry @ Rz
print(R)
```

```
[[ 0.93629336 -0.28962948 -0.19866933]
 [ 0.27509585  0.95642509 -0.0978434 ]
 [ 0.21835066  0.03695701  0.97517033]]
```

Problems with XYZ angles

- XYZ angles are intuitive, but they are not good for computation ☹
  - The result depends on the order in which the transforms are applied; i.e., XYZ or ZYX
  - Sometimes one or more angles change dramatically in response to a small change in orientation
  - Some orientations have singularities; i.e., the angles are not well defined

- We'll just use XYZ angles to create a rotation matrix, then work with the rotation matrix
- The rotation matrix is always unique for a given orientation

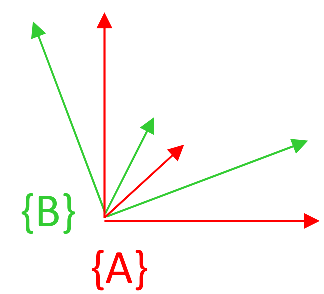
3D Rotation Matrix as a Transformation

- **R** can represent a rotational transformation of one frame to another

$$ {}^{B}_{A}\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} $$

- We can rotate a vector represented in frame *A* to obtain its representation in frame *B*

$$ {}^{B}\mathbf{v} = {}^{B}_{A}\mathbf{R} \; {}^{A}\mathbf{v} $$



- Note: as in 2D, rotation matrices are orthonormal so the inverse of a rotation matrix is just its transpose

$$ \left({}^{B}_{A}\mathbf{R}\right)^{-1} = \left({}^{B}_{A}\mathbf{R}\right)^{T} = {}^{A}_{B}\mathbf{R} $$

Notation

- For vectors, such as ${}^{A}\mathbf{v}$, the leading superscript represents the coordinate frame that the vector is expressed in

$$ {}^{A}\mathbf{v} = \begin{pmatrix} {}^{A}x \\ {}^{A}y \\ {}^{A}z \end{pmatrix} $$

- For transforms, such as ${}^{B}_{A}\mathbf{R}$, this matrix represents a rotational transformation of frame *A* to frame *B*
  - The leading subscript indicates “from”
  - The leading superscript indicates “to”

3D Rotation Matrix

- The elements of **R** are direction cosines (the projections of unit vectors from one frame onto the unit vectors of the other frame)

$$ {}^{B}_{A}\mathbf{R} = \begin{pmatrix} \hat{\mathbf{x}}_A \cdot \hat{\mathbf{x}}_B & \hat{\mathbf{y}}_A \cdot \hat{\mathbf{x}}_B & \hat{\mathbf{z}}_A \cdot \hat{\mathbf{x}}_B \\ \hat{\mathbf{x}}_A \cdot \hat{\mathbf{y}}_B & \hat{\mathbf{y}}_A \cdot \hat{\mathbf{y}}_B & \hat{\mathbf{z}}_A \cdot \hat{\mathbf{y}}_B \\ \hat{\mathbf{x}}_A \cdot \hat{\mathbf{z}}_B & \hat{\mathbf{y}}_A \cdot \hat{\mathbf{z}}_B & \hat{\mathbf{z}}_A \cdot \hat{\mathbf{z}}_B \end{pmatrix} $$

- The columns of **R** are the unit vectors of A, expressed in the B frame

$$ {}^{B}_{A}\mathbf{R} = \begin{pmatrix} \begin{pmatrix} {}^{B}\hat{\mathbf{x}}_A \end{pmatrix} & \begin{pmatrix} {}^{B}\hat{\mathbf{y}}_A \end{pmatrix} & \begin{pmatrix} {}^{B}\hat{\mathbf{z}}_A \end{pmatrix} \end{pmatrix} $$

- The rows of **R** are the unit vectors of {B} expressed in {A}

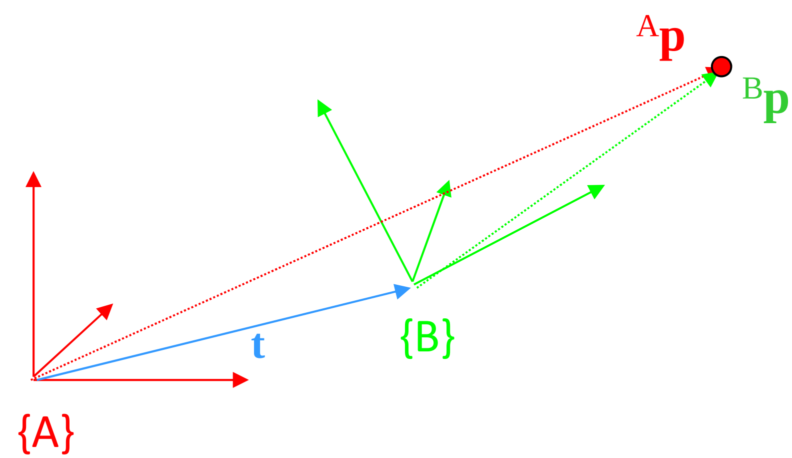
$$ {}^{B}_{A}\mathbf{R} = \begin{pmatrix} \begin{pmatrix} {}^{A}\hat{\mathbf{x}}_B{}^{T} \end{pmatrix} \\ \begin{pmatrix} {}^{A}\hat{\mathbf{y}}_B{}^{T} \end{pmatrix} \\ \begin{pmatrix} {}^{A}\hat{\mathbf{z}}_B{}^{T} \end{pmatrix} \end{pmatrix} $$

$$ {}^{B}_{A}R \, {}^{A}\hat{x}_A = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} r_{11} \\ r_{21} \\ r_{31} \end{pmatrix} = {}^{B}\hat{x}_A $$

Doing a rotation AND a translation

- We can use **R,t** to transform a point from coordinate frame {B} to frame {A}

$$ {}^{A}\mathbf{p} = {}^{A}_{B}\mathbf{R}\ {}^{B}\mathbf{p} + \mathbf{t} $$



- Where
  - ${}^{A}\mathbf{p}$ is the representation of **p** in frame {A}
  - ${}^{B}\mathbf{p}$ is the representation of **p** in frame {B}

- Note

$\mathbf{t}$ is the translation of B's origin in the A frame, ${}^{A}\mathbf{t}_{Borg}$

Homogeneous Coordinates

- We can represent the transformation with a *single* matrix multiplication if we write **p** in *homogeneous* coordinates
  - This simply means to append a 1 as a 4th element
  - If the 4th element ever becomes ≠ 1, we divide through by it

The leading superscript indicates what coordinate frame the point is represented in

$$^{A}\mathbf{p} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} sx \\ sy \\ sz \\ s \end{pmatrix}$$

- Then

$$^{B}\mathbf{p} = \mathbf{H}\ ^{A}\mathbf{p} \quad \text{where} \quad \mathbf{H} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

General Rigid Transformation

- A general rigid transformation can be represented by a single 4x4 homogeneous transformation matrix

$$ {}^{A}_{B}\mathbf{H} = \begin{bmatrix} {}^{A}_{B}\mathbf{R} & {}^{A}\mathbf{t}_{Borg} \\ 0 \;\; 0 \;\; 0 & 1 \end{bmatrix} $$

- A note on notation:

$$ {}^{A}\mathbf{p} = {}^{A}_{B}\mathbf{H} \; {}^{B}\mathbf{p} $$

Cancel leading subscript with trailing superscript

- Can concatenate transformations together
  - Leading subscripts cancel trailing superscripts

$$ {}^{C}_{A}\mathbf{H} = {}^{C}_{B}\mathbf{H} \; {}^{B}_{A}\mathbf{H} \qquad\qquad {}^{D}_{A}\mathbf{H} = {}^{D}_{C}\mathbf{H} \; {}^{C}_{B}\mathbf{H} \; {}^{B}_{A}\mathbf{H}, \quad \text{etc} $$

Example: Transforming a point, using Python

- Assume we have a point **p** = (-1,0,1)^T, in frame A
- Transform it to frame B, if the pose of frame B with respect to frame A is given by ${}^{B}_{A}\mathbf{R}$ and ${}^{B}\mathbf{t}_{Aorg}$

Assume

$$ {}^{B}_{A}\mathbf{R} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} $$

$$ {}^{B}t_{Aorg} = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix} $$

=> We need to do ${}^{B}\mathbf{p} = {}^{B}_{A}\mathbf{H}\ {}^{A}\mathbf{p}$

```
# Rotation matrix of A with respect to B.
R_A_B = np.array(((1,0,0),(0,0,-1),(0,1,0)))    # Get 3x3 matrix

# The translation is the origin of A in B.
tAorg_B = np.array([[1,2,4]]).T   # Get as a 3x1 matrix

# H_A_B means transform A to B.
H_A_B = np.block([[R_A_B, tAorg_B], [0,0,0,1]])  # Get 4x4 matrix

# Define a point in the A frame, as [x,y,z,1].
P_A = np.array([[-1,0,1,1]]).T    # Get as a 4x1 matrix

# Convert point to B frame.
P_B = H_A_B @ P_A
```

Inverse Transformations

- The **matrix inverse** is the inverse transformation

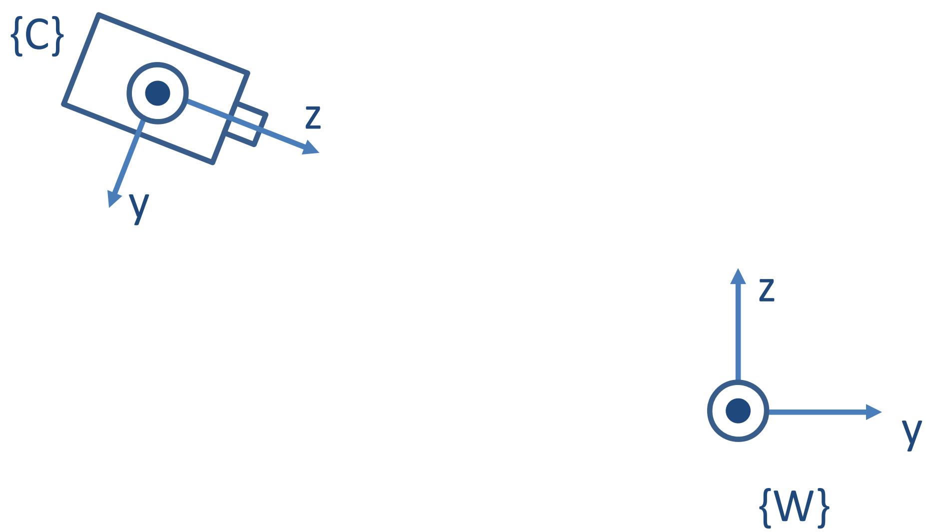
$$ {}_B^A\mathbf{H} = \left({}_A^B\mathbf{H} \right)^{-1} $$

- Note – unlike rotation matrices, the inverse of a full 4x4 homogeneous transformation matrix is **not the transpose**

$$ {}_B^A\mathbf{H} \neq \left({}_A^B\mathbf{H} \right)^{T} $$

Example

- A camera is located at point (0,-5,3) with respect to the world. The camera is tilted down by 30 degrees from the horizontal. Find the transformation from {W} to {C}.



Example (continued)

- The origin of the camera in the world is ${}^{W}\mathbf{t}_{Corg} = (0, -5, 3)^T$
- The rotation matrix is ${}^{W}_{C}\mathbf{R} = \mathbf{R}_x(-120\ deg)$

$$ {}^{W}_{C}\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(-120) & -\sin(-120) \\ 0 & \sin(-120) & \cos(-120) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.5 & +0.86 \\ 0 & -0.86 & -0.5 \end{bmatrix} $$

- To check to see if this makes sense … see if the unit axes of the camera are pointing in the right direction in the world

$$ {}^{W}_{C}\mathbf{R} = \left(\begin{pmatrix} \\ {}^{W}\hat{\mathbf{x}}_C \\ \\ \end{pmatrix} \begin{pmatrix} \\ {}^{W}\hat{\mathbf{y}}_C \\ \\ \end{pmatrix} \begin{pmatrix} \\ {}^{W}\hat{\mathbf{z}}_C \\ \\ \end{pmatrix} \right) \Rightarrow {}^{W}\hat{\mathbf{x}}_C = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad {}^{W}\hat{\mathbf{y}}_C = \begin{pmatrix} 0 \\ -0.5 \\ -0.86 \end{pmatrix}, \quad {}^{W}\hat{\mathbf{z}}_C = \begin{pmatrix} 0 \\ 0.86 \\ -0.5 \end{pmatrix} $$

Example (continued)

- The full 4x4 homogeneous transformation matrix from camera to world is

$$ {}^{W}_{C}\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -0.5 & 0.86 & -5 \\ 0 & -0.86 & -0.5 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

- But, we actually wanted the transformation from world to camera, so take the inverse

$$ {}^{C}_{W}\mathbf{H} = {}^{W}_{C}\mathbf{H}^{-1} $$

```
    1.0000         0         0         0
         0   -0.5000   -0.8660    0.0981
         0    0.8660   -0.5000    5.8301
         0         0         0    1.0000
```

Python code

```
import math
import numpy as np

def main():
    # Construct transformation from camera to world.
    tc_w = np.array([[0, -5.0, 3.0]]).T
    ax = math.radians(-120)     # Convert degrees to radians
    sx = math.sin(ax)
    cx = math.cos(ax)

    Rx = np.array(((1, 0, 0), (0, cx, -sx), (0, sx, cx)))
    R_c_w = Rx      # The only rotation is about x

    H_c_w = np.block([[R_c_w, tc_w], [0,0,0,1]])  # Get as 4x4 matrix
    print("H_c_w:"), print(H_c_w)

    # Get transformation from world to camera.
    H_w_c = np.linalg.inv(H_c_w)
    print("H_w_c:"), print(H_w_c)


if __name__ == "__main__":
    main()
```

Summary

- 3D rigid body transformations (i.e., a rotation and translation) can be represented by a single 4x4 homogeneous transformation matrix
- A 3D rotation is represented uniquely by a 3x3 rotation matrix
- 3D rotations can also be represented by XYZ angles (they are easy to understand, but not computationally stable; also the order matters)