Rigid Body Transformations and Eigen

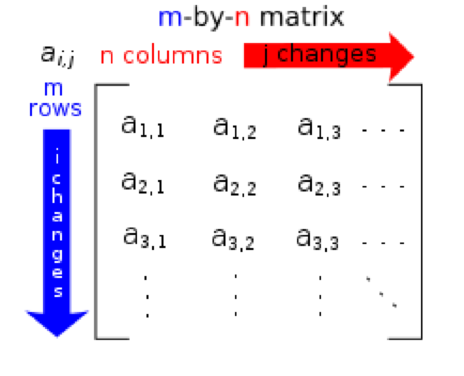
Outline

- Linear algebra review
- Transforms
- Transforms in Eigen

Linear Algebra Review

Matrices and Vectors

- Matrix:



A square matrix has $m = n$

- Vector: an m-by-1 matrix

$$\begin{array}{cc} & \text{1 column} \\ \begin{array}{r}\text{m}\\ \text{rows}\end{array} & \begin{bmatrix} b_1 \\ b_2 \\ \vdots \end{bmatrix} \end{array}$$

Matrix operations

- For matrices with the <u>same dimensions</u>
  - Can add them elementwise, e.g.:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \end{bmatrix}$$

- Can scale them, e.g.:

$$2.4 \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} = \begin{bmatrix} 2.4a_{11} & 2.4a_{12} & 2.4a_{13} \\ 2.4a_{21} & 2.4a_{22} & 2.4a_{23} \end{bmatrix}$$

Matrix Multiplication

- For matrices **A** and **B**, their product is written as **AB**
- Each element in **AB** is *the dot product* of <u>a row</u> of **A** with <u>a column</u> of **B**

Number of columns of **A**

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{n_a} a_{ik} b_{kj}$$

- Example:

$$\begin{array}{ccccc} \mathbf{A} & & \mathbf{B} & & \mathbf{C} \\ \underline{2 \times 3} & & \underline{3 \times 3} & & \underline{2 \times 3} \\ \begin{bmatrix} \cdot & \cdot & \cdot \\ a_{21} & a_{22} & a_{23} \end{bmatrix} & & \begin{bmatrix} \cdot & b_{12} & \cdot \\ \cdot & b_{22} & \cdot \\ \cdot & b_{32} & \cdot \end{bmatrix} & = & \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & c_{22} & \cdot \end{bmatrix} \end{array}$$

$$c_{22} = a_{2:} \cdot b_{:2} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}$$

Matrix Multiplication

- Matrix multiplication is not commutative!

$$\mathbf{AB} \neq \mathbf{BA}$$

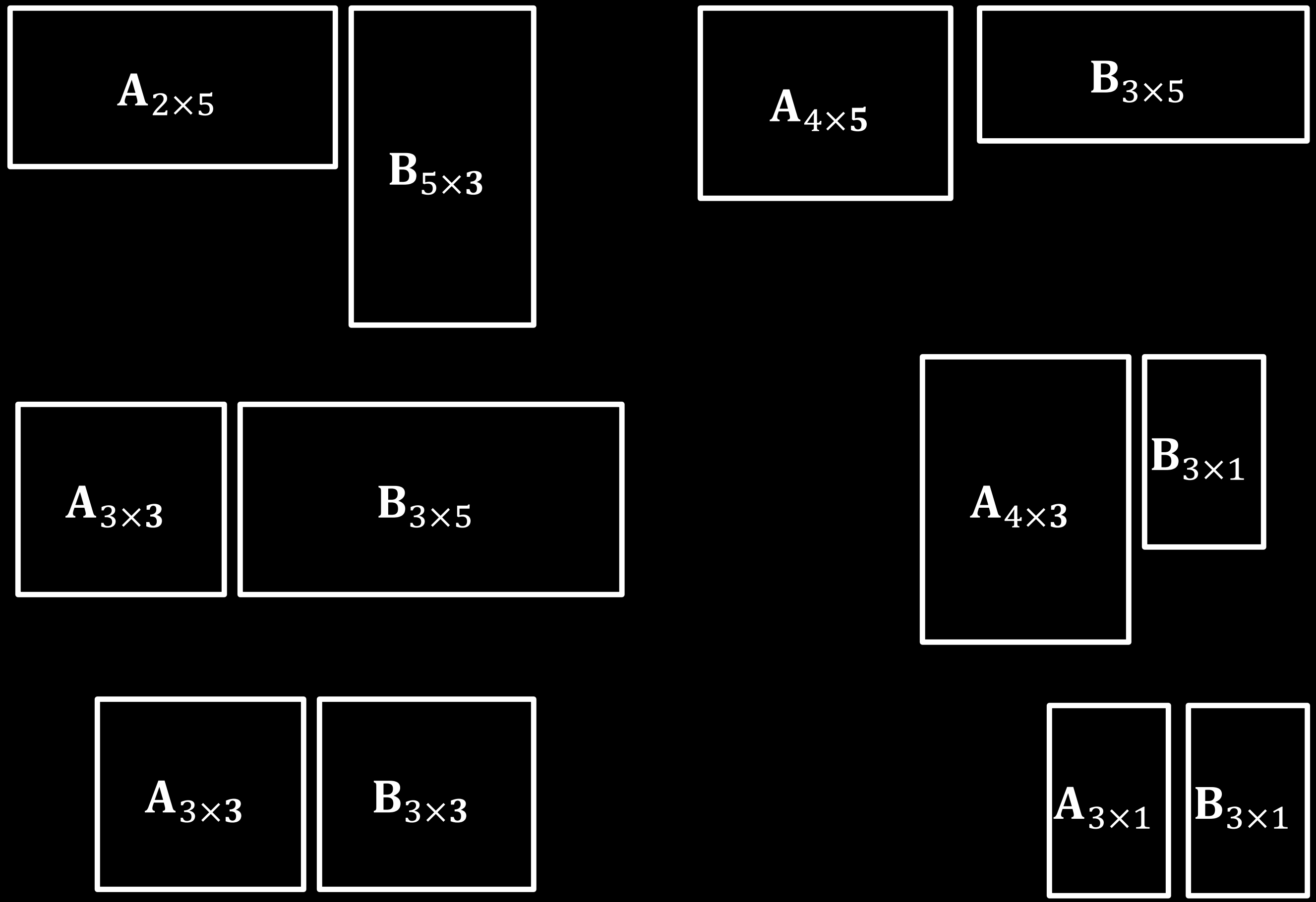
- You can only multiply matrices if they have compatible dimensions

$$\mathbf{A}_{m_a \times n_a} \mathbf{B}_{m_b \times n_b} = \mathbf{C}_{m_a \times n_b}$$

$n_a = m_b$ must be true

Matrix Multiplication

- Which multiplications are valid? What is the dimension of **AB**?



Matrix Transpose

- The transpose of an m-by-n matrix **A** is denoted **A**^{T}
- Transpose is done by flipping the matrix about the diagonal; i.e. swap rows and columns:

$$[\mathbf{A}^T]_{ij} = \mathbf{A}_{ji}$$

- Example:

$$\overset{\underline{2 \text{ x } 3}}{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}^T} = \overset{\underline{3 \text{ x } 2}}{\begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{bmatrix}}$$

- Distributing transpose:

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$$

- $(\mathbf{A}^T)^T = \mathbf{A}$

Identity Matrix

- The identity matrix $\mathbf{I}_n$ is an n x n matrix that does no change when multiplied
  - Diagonal elements ($i = j$) are 1
  - Off-diagonal elements ($i \neq j$) are 0

$$\mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- For an m x n matrix **A**

$$\mathbf{A}\mathbf{I}_n = \mathbf{A}$$

$$\mathbf{I}_m\mathbf{A} = \mathbf{A}$$

Rigid Body Transformations

Rigid Body Transformations

- An understanding of 2D and 3D rigid-body transformations is essential for robotics
- There are many representations, none is the “best”
  - Representations: Transform matrices, quaternions, Euler angles, etc.
  - Each representation is useful in a different way

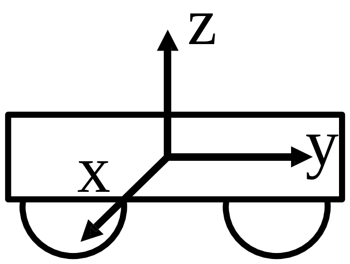
- We’ll only cover Transform Matrices (most common form) today

Homogenous Transforms Outline

- Notational Conventions
- Definitions
- Homogeneous Transforms
- Semantics and Interpretations
- Summary

Objects and Embedded Coordinate Frames

- Objects of interest are real: wheels, sensors, obstacles.
- Abstract them by sets of axes fixed to the body.

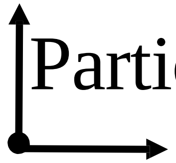


- These axes:
  - Have a state of motion
  - Can be used to express vectors.
- Call them coordinate frames.

Slides by Al Kelly, minor edits by Dmitry Berenson

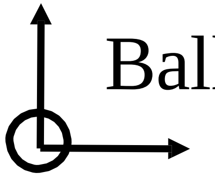
Coordinate Frames

- Points possess position but not orientation:



Particle (orientation undefined)

- Rigid Bodies possess position and orientation:



Ball (can rotate)

Relations

- Mechanics is about relations between objects.
- a is "r-related" to b is written: $\boxed{r_a^b}$
- Example velocity (v) of robot (r) relative to earth (e): $\boxed{v_r^e}$
- Relationship is directional and asymmetric. $\boxed{r_a^b \neq r_b^a}$

Notational Conventions

- Vectors:

$$p = \begin{bmatrix} x & y & z \end{bmatrix}^T$$

- Also <u>sometimes</u> as $\underline{p}$ or as $\vec{p}$ to emphasize it is a vector.
- Matrices:

$$T = \begin{bmatrix} t_{xx} & t_{xy} & t_{xz} \\ t_{yx} & t_{yy} & t_{yz} \\ t_{zx} & t_{zy} & t_{zz} \end{bmatrix}$$

Converting Coordinates

$$\boxed{p^b = T_a^b p^a}$$

- We will see later that T_a^b notation satisfies our conventions where it means the <u>'T' property of 'object' a w.r.t 'object' b</u>.

Homogenous Transforms Outline

- Notational Conventions
- Definitions
- Homogeneous Transforms
- Semantics and Interpretations
- Summary

Slides by Al Kelly, minor edits by Dmitry Berenson

Affine Transformation

- Most general linear transformation

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

- r’s and t’s are the transform constants
- Can be used to effect translation, rotation, scale, reflections, and shear.
- Preserves linearity but not distance (hence, not areas or angles).

Homogeneous Transformation

- Set t1 = t2 = 0:

Homogeneous

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + 0$$

- r's are the transform constants
- Can be used to effect rotation, scale, reflections, and shear (<u>not translation</u>).
- Preserves linearity but not distance (hence, not areas or angles).

THE ROBOTICS INSTITUTE

Orthogonal Transformation

- Looks the same … but:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$\begin{aligned} r_{11} r_{12} + r_{21} r_{22} &= 0 \\ r_{11} r_{11} + r_{21} r_{21} &= 1 \\ r_{12} r_{12} + r_{22} r_{22} &= 1 \end{aligned}$$

- Can be used to effect rotation, reflections.
- Preserves linearity <u>AND distance</u> (hence, areas and angles).

Rotation Matrix

- Looks the same … but:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \underbrace{\begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$r_{11} r_{12} + r_{21} r_{22} = 0$$
$$r_{11} r_{11} + r_{21} r_{21} = 1$$
$$r_{12} r_{12} + r_{22} r_{22} = 1$$

Determinant(**R**)= 1

- Can be used to effect rotation.
- Preserves linearity AND distance (hence, areas and angles).

Definitions

- Heading = angle of path tangent.
- Yaw = rotation about vertical axis
- Pitch = rotation about level sideways axis
- Roll = rotation about “forward” axis.
- Attitude = pitch & roll
- Azimuth = yaw (for a pointing device)
- Elevation = pitch (for a pointing device)

Definitions

- Orientation = attitude & yaw.
- Pose = position & orientation

2D: $\begin{bmatrix} x & y & \psi \end{bmatrix}^T$ 3D: $\begin{bmatrix} x & y & z & \theta & \varphi & \psi \end{bmatrix}^T$

- Posture = Pose plus some configuration

$$\begin{bmatrix} x & y & z & \theta & \varphi & \psi & q \end{bmatrix}^T$$

- Motion = movement of the whole body through space.

THE ROBOTICS INSTITUTE

Homogenous Transforms Outline

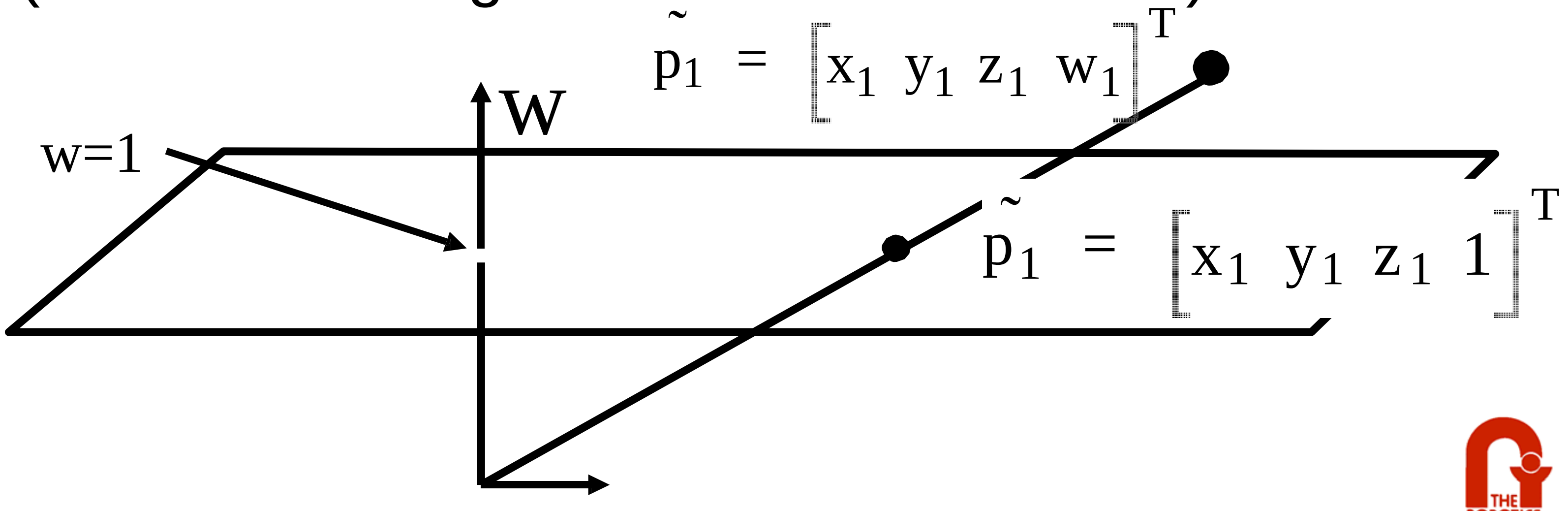
- Notational Conventions
- Definitions
- <u>Homogeneous Transforms</u>
- Semantics and Interpretations
- Summary

Homogeneous Coordinates

- Coordinates which are <u>unique up to a scale factor</u>. i.e

$$\underline{x} = 6\underline{x} = -12\underline{x} = 3.14\underline{x} = \text{same thing}$$

- The numbers in the vectors are not the same but we interpret them to mean the same thing (in fact. the thing whose scale factor is 1).



Pure Directions

- Its also possible to represent pure directions
  - Pure in the sense they "are everywhere" (i.e. have no position and cannot be moved).
- We use a scale factor of zero to get a pure direction:

$$d_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 0 \end{bmatrix}$$

- It will shortly be clear why this works.

Slides by Al Kelly, minor edits by Dmitry Berenson

Why Bother?

- Points in 3D can be rotated, reflected, scaled, and sheared with 3 X 3 matrices….

$$p_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = Tp_1 = \begin{bmatrix} t_{xx} & t_{xy} & t_{xz} \\ t_{yx} & t_{yy} & t_{yz} \\ t_{zx} & t_{zy} & t_{zz} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} t_{xx}x_1 + t_{xy}y_1 + t_{xz}z_1 \\ t_{yx}x_1 + t_{yy}y_1 + t_{yz}z_1 \\ t_{zx}x_1 + t_{zy}y_1 + t_{zz}z_1 \end{bmatrix}$$

- But not translated.

$$p_2 = p_1 + p_k = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} \neq Trans(p_k)p_1$$

THE ROBOTICS INSTITUTE

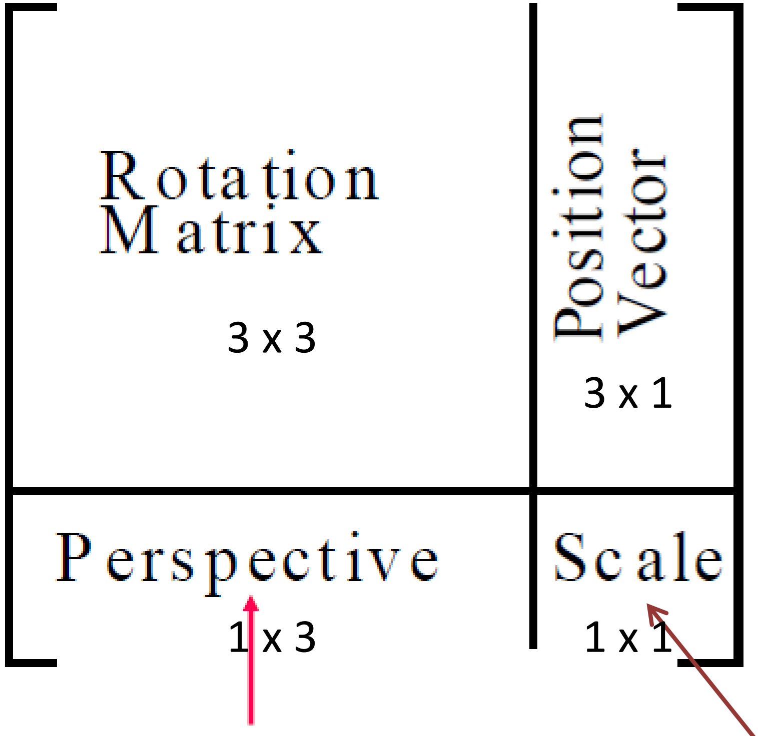
Trick: Move to 4D

$$p_2 = p_1 + p_k = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} + \begin{bmatrix} x_k \\ y_k \\ z_k \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_k \\ 0 & 1 & 0 & y_k \\ 0 & 0 & 1 & z_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \mathrm{Trans}(p_k)p_1$$

$$\boxed{\begin{aligned} x_2 &= 1 \times x_1 + x_k \\ y_2 &= 1 \times y_1 + y_k \\ z_2 &= 1 \times z_1 + z_k \end{aligned}}$$

- The scale factor in the vector is used to add a scaled amount of the 4th matrix column.

Format of Homogeneous Transforms (HTs)



We will not use the perspective part much

Set to 1

Inverse of a HT

$$\left[\begin{array}{ccc|c} & R & & \underline{p} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]^{-1} = \left[\begin{array}{ccc|c} & R^T & & -R^T\underline{p} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

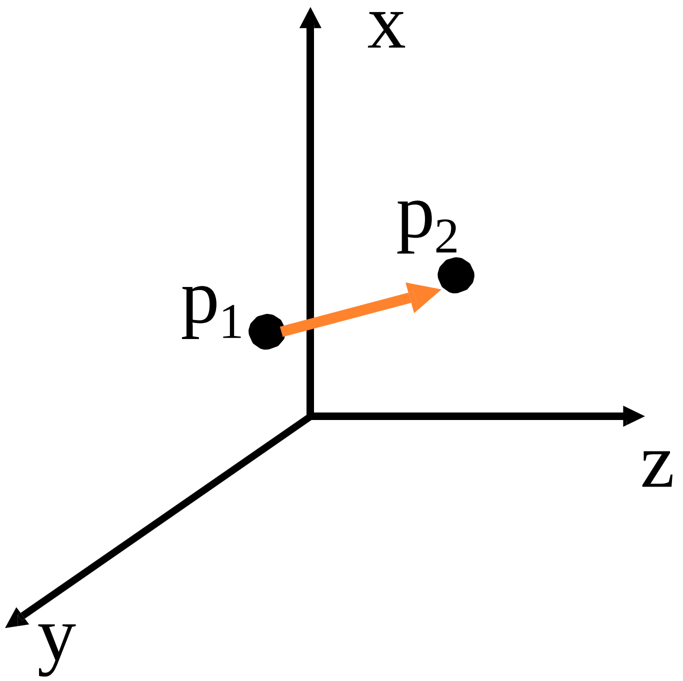
- Of course standard matrix inverse also works, but this is faster

Homogenous Transforms Outline

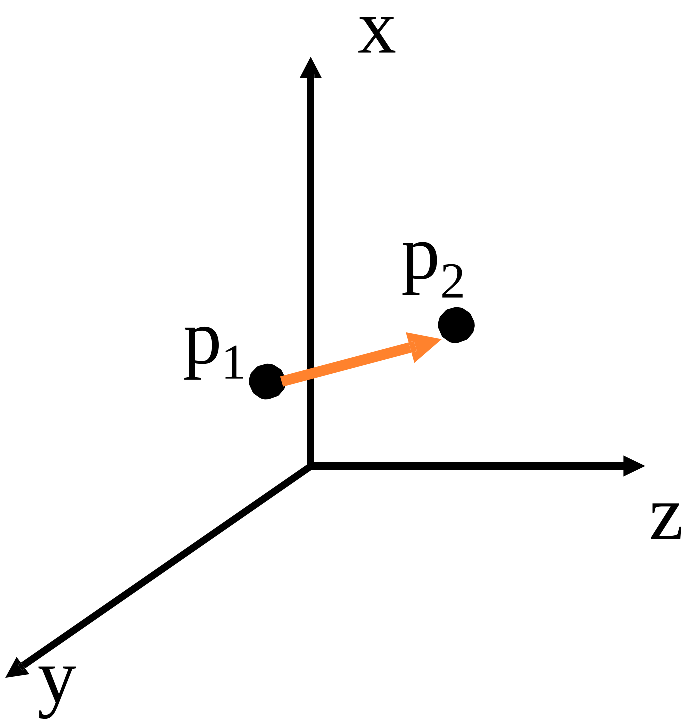
- Notational Conventions
- Definitions
- Homogeneous Transforms
- <u>Semantics and Interpretations</u>
- Summary

Operators

- Mapping:
  - Point1 -> Point2 (both expressed in same coordinates)



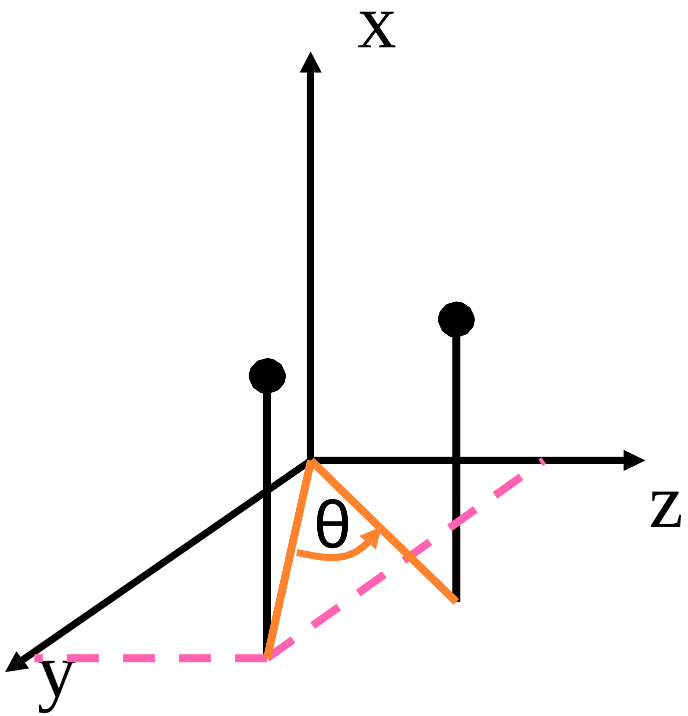
Operators



$$\mathrm{Trans}(u, v, w) = \begin{bmatrix} 1 & 0 & 0 & u \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & w \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Operators

$s\theta = \sin(\theta)$
$c\theta = \cos(\theta)$

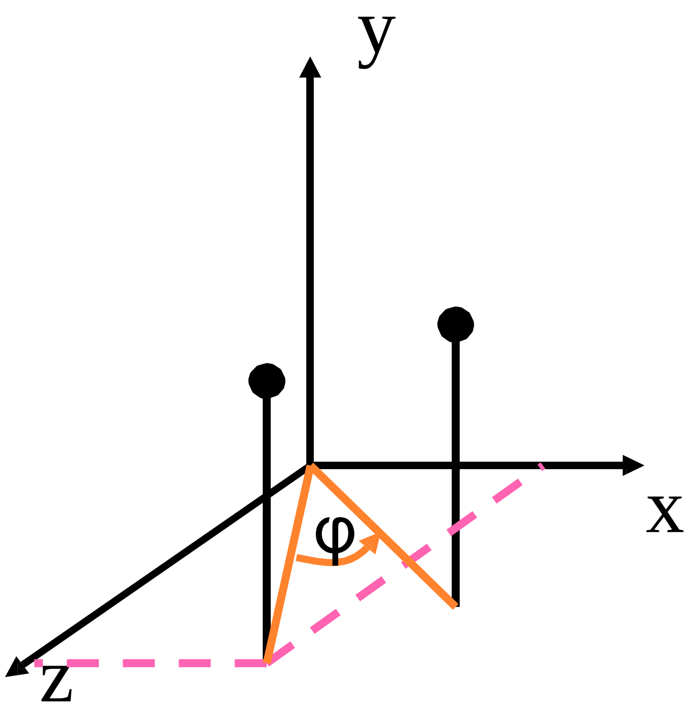


$$Rotx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\theta & -s\theta & 0 \\ 0 & s\theta & c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Slides by Al Kelly, minor edits by Dmitry Berenson

Operators

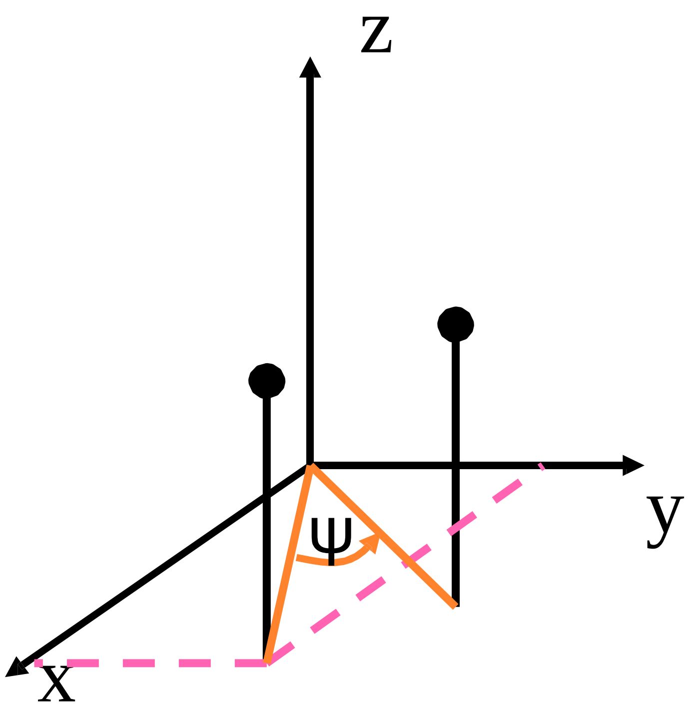
sθ = sin(θ)
cθ = cos(θ)



$$\mathrm{Roty}(\phi) = \begin{bmatrix} c\phi & 0 & s\phi & 0 \\ 0 & 1 & 0 & 0 \\ -s\phi & 0 & c\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Operators

$s\theta = \sin(\theta)$
$c\theta = \cos(\theta)$

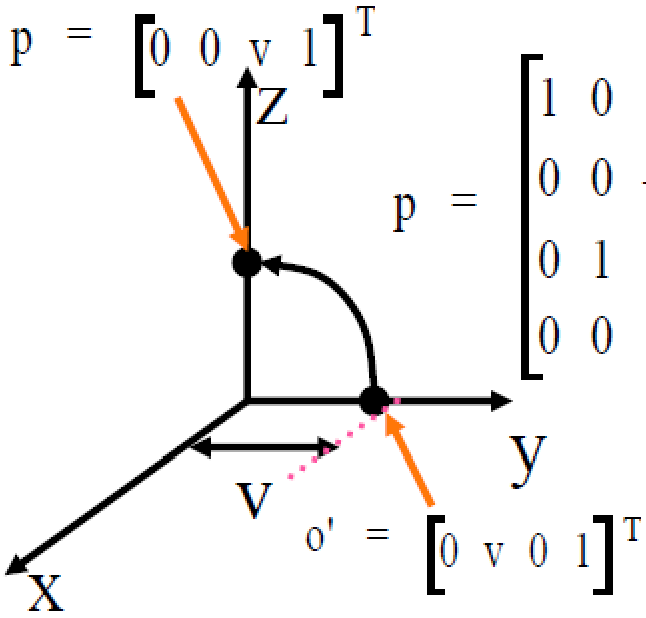


$$Rotz(\psi) = \begin{bmatrix} c\psi & -s\psi & 0 & 0 \\ s\psi & c\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example: Operating on a Point

- A point at the origin is translated along the y axis by ‘v’ units and then the resulting point is rotated by 90 degrees around the x axis.

$$p = Rotx(\pi/2)Trans(0,v,0)o$$



$$p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boxed{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boxed{\begin{bmatrix} 0 \\ v \\ 0 \\ 1 \end{bmatrix}} = \begin{bmatrix} 0 \\ 0 \\ v \\ 1 \end{bmatrix}$$

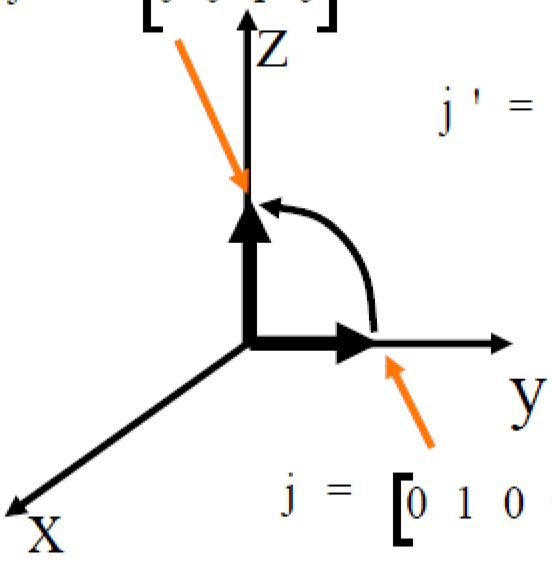
Slides by Al Kelly, minor edits by Dmitry Berenson

Example: Operating on a Direction

- The y axis unit vector is translated along the y axis by v units and then rotated by 90 degrees around the x axis.

$$j' = Rotx(\pi/2)Trans(0,v,0)\ j$$

$$j' = [0 \quad 0 \quad 1 \quad 0]^T$$



$$j = [0 \quad 1 \quad 0 \quad 0]^T$$

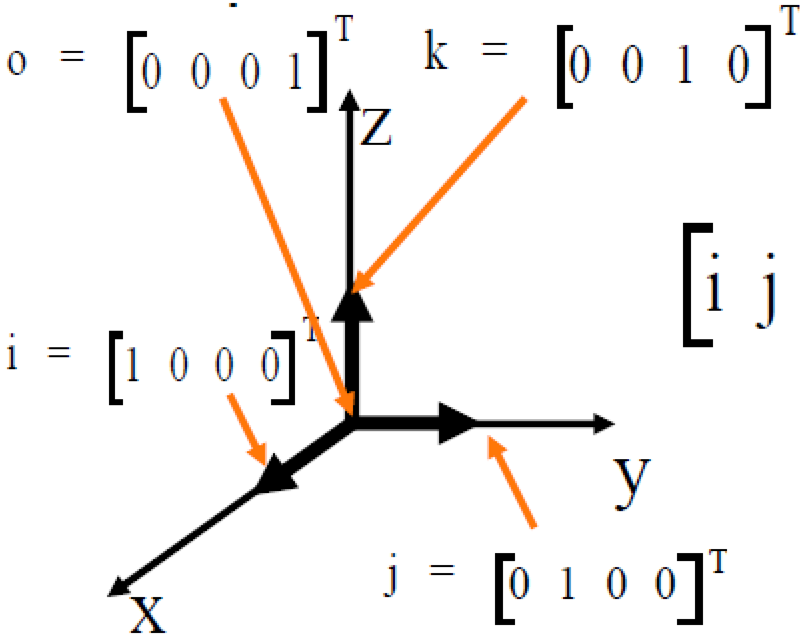
$$j' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

- Having a zero scale factor <u>disables translation</u>.

Slides by Al Kelly, minor edits by Dmitry Berenson

HTs as Coordinate Frames

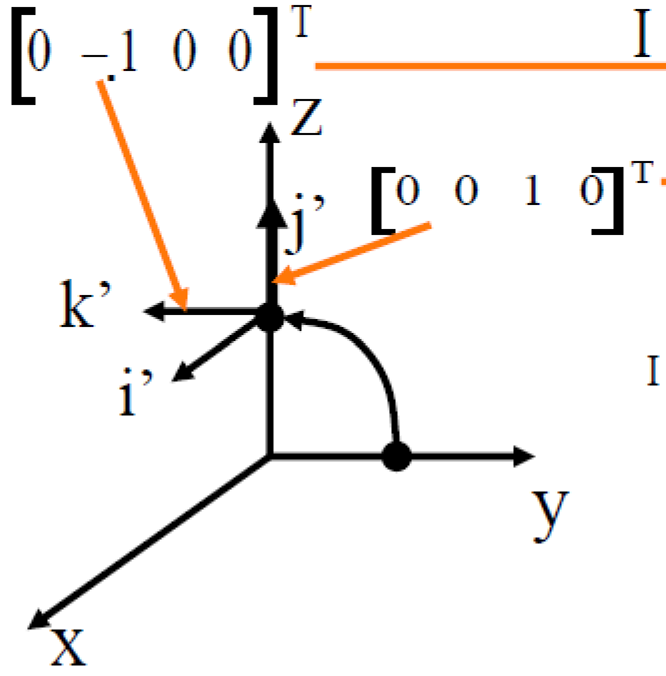
- The columns of the identity HT can be considered to represent 3 directions and a point – the coordinate frame itself.



$$\begin{bmatrix} i & j & k & o \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = I$$

Example: Operating on a Frame

- Each resulting column of this result is the transformation of the corresponding column in the original identity matrix



$$I' = Rotx(\pi/2)Trans(0,v,0)I$$

$$I' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & v \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

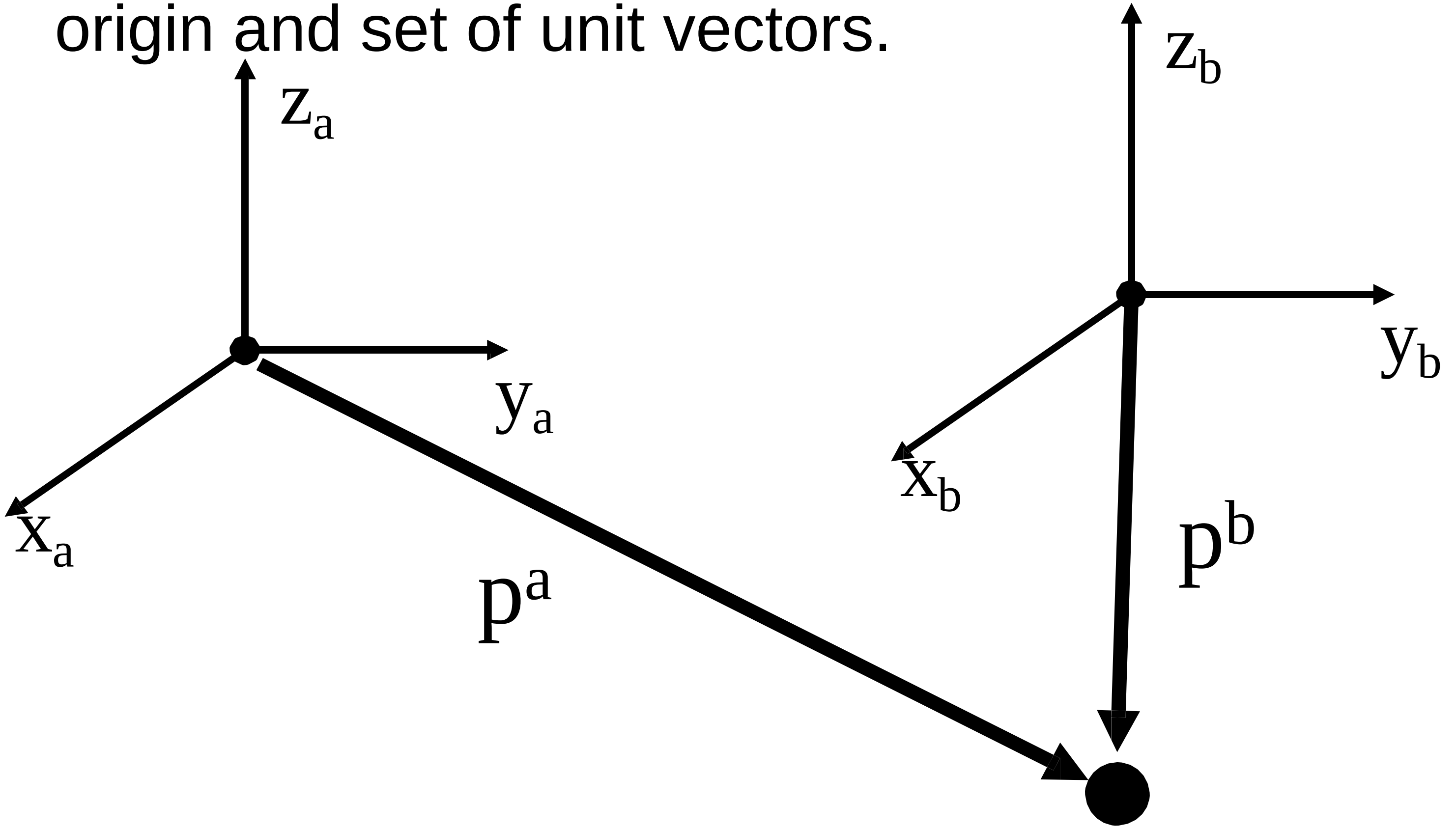
Slides by Al Kelly, minor edits by Dmitry Berenson

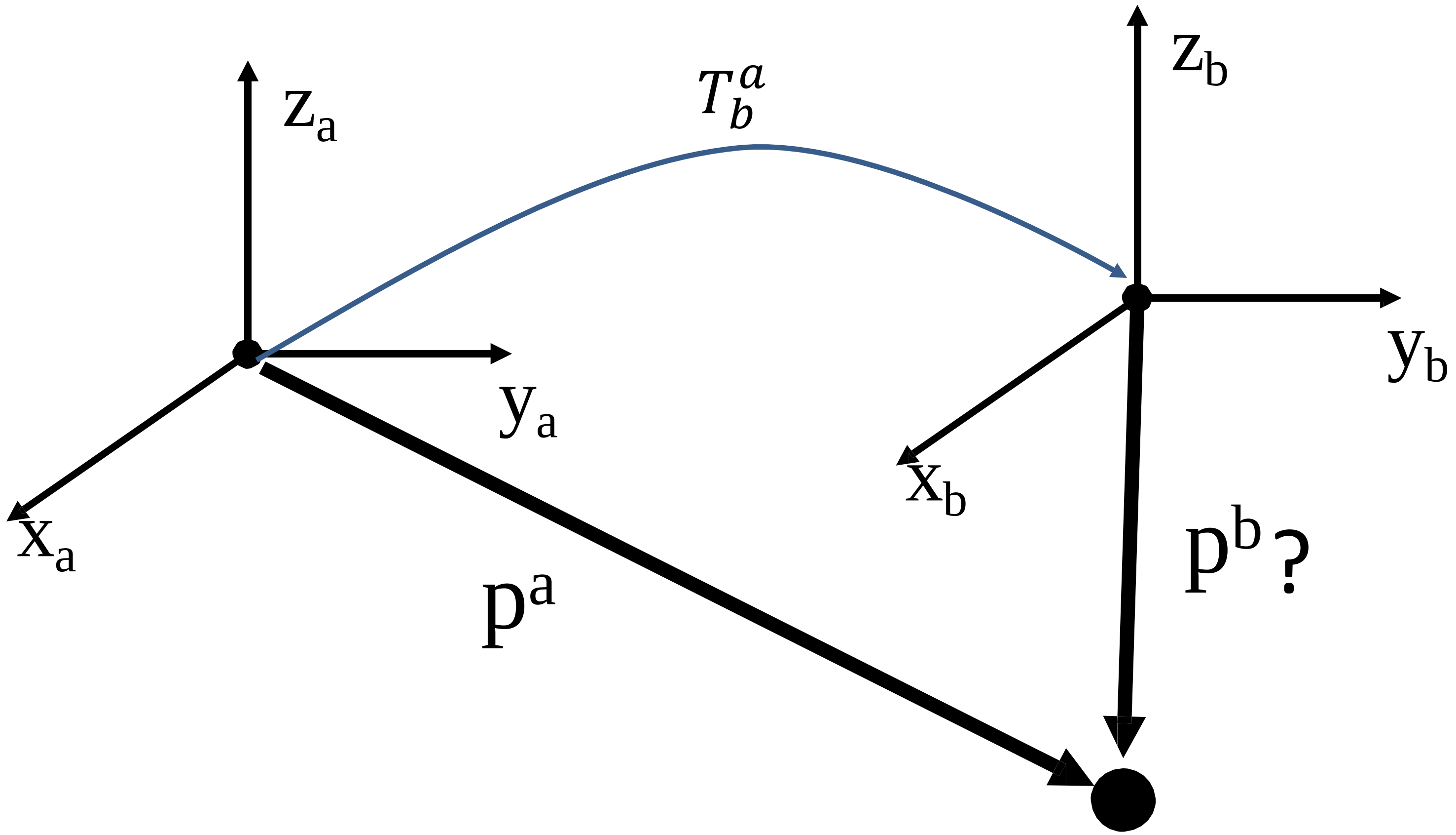
Huh?

- Columns of an input matrix are treated independently in multiplication.
- Every orthonormal matrix can be viewed as one set of axes located with respect to another set.
  - The "locations" can be read right from the matrix – they're just the columns.
- We can use this idea to track the position and orientation of rigid bodies….
  - Imagine embedding frames inside them somewhere and track their motions.

Converting Coordinates

- Converting coordinates is about expressing the same physical point with respect to a new origin and set of unit vectors.





- Given T_b^a and p^a, how to compute p^b?

Similarity Transforms

- Suppose you have a transform A^0 defined relative to frame 0, and you want to know what it is in frame 1. Assume you know T_1^0.

$$B = (T_1^0)^{-1} A^0 T_1^0$$

- B is transform A^0 represented in frame 1

Sequencing Transforms

- Any sequence of transforms can be represented by a single transform (Euler's rotation theorem)

- *How* you sequence transforms depends on if you are transforming w.r.t. the *fixed* or the *current* axes

- Transforming w.r.t. ***current*** axes: multiply *on the **right***

$$T_n^0 = T_1^0 T_2^1 \dots T_n^{n-1}$$

- Transforming w.r.t. ***fixed*** axes of frame 0: multiply *on the **left***

$$T_2^0 = T_1^0[(T_1^0)^{-1} A^0 T_1^0] = A^0 T_1^0$$

Similarity transform

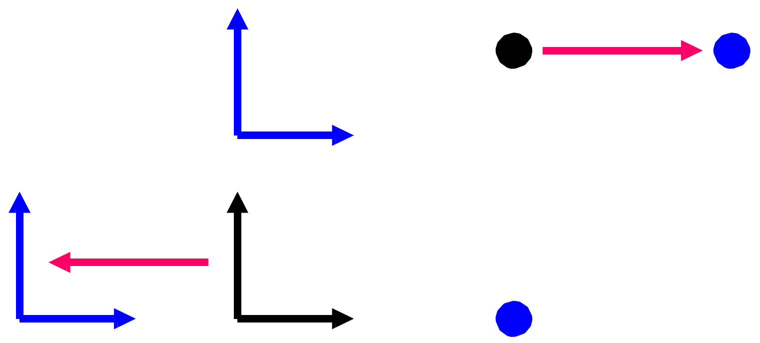
Slides by Al Kelly, minor edits by Dmitry Berenson

Homogenous Transforms Outline

- Notational Conventions
- Definitions
- Homogeneous Transforms
- Semantics and Interpretations
- Summary

Summary

- Everything is relative. There is no way to distinguish moving a point “forward” from moving the coordinate system “backward”.



- In both cases, the resulting (blue) point has the same relationship to the blue frame.

Summary

- Homogeneous Transforms are:
  - Operators
  - Frames
- They can be both the things that <u>operate</u> on other things and the things <u>operated upon</u>.

BREAK

Eigen

What is Eigen?

- Our first external library! 🎉
- A library used for matrices and vectors
  - Used frequently in industry for linear algebra in C++
- Many many functions and classes
  - We'll use just a few of these in this course
- Eigen has it's own geometry classes (e.g. Affine Transform, Translation, etc.)
  - We won't use these in this lecture, just the Matrix and Vector classes

- Very useful quick reference: https://eigen.tuxfamily.org/dox/group__QuickRefPage.html

Eigen “hello world”

```
#include <iostream>
#include <math.h>
#include <eigen3/Eigen/Eigen> //this has already been installed for you on the VM


int main(){
    Eigen::Matrix4d m1; //same as Eigen::MatrixXd m1(4,4);
    m1   << 1, 2, 3, 1,
            1, 1, 0, 2,
            1, 0, 1, 3,
            1, 0, 0, 4;

    m1(0,0) = 3;
    std::cout << "m1: " << std::endl << m1 <<std::endl<<std::endl;
    std::cout << "m1.inverse(): " << std::endl << m1.inverse() <<std::endl<<std::endl;
    std::cout << "m1*m1.inverse(): " << std::endl << m1 * m1.inverse() <<std::endl<<std::endl;
}
```

Output:

```
m1:
3 2 3 1
1 1 0 2
1 0 1 3
1 0 0 4

m1.inverse():
    1    -2    -3     3
 -0.5     2   1.5    -2
-0.25   0.5  1.75  -1.5
-0.25   0.5  0.75  -0.5

m1*m1.inverse():
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
```

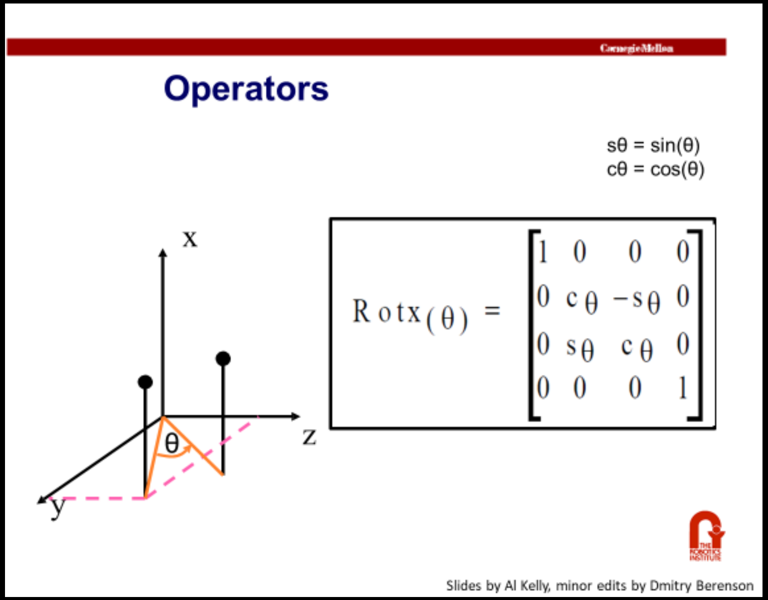
Eigen basics

- Can't create a matrix as easily as matlab or python, need to think about size and datatype, e.g.:
  - `Eigen::Matrix4d` is the type for a 4x4 matrix where each entry is of type `double`
  - `Eigen::Matrix2f` is the type for a 2x2 matrix where each entry is of type `float`
  - We will use matrices that have entries of type `double` in this course

- What if I want to change my matrix size?
  - `Eigen::MatrixXd` is the type for arbitrary size matrices
  - Can use arbitrary size matrices at initialization:
    - `Eigen::MatrixXd x(4,30);`
  - Can initialize to empty and then set size later:
    - ```
      Eigen::MatrixXd x;
      x.resize(7,5); //WARNING: this will delete the data in the matrix
      ```

- Same idea for vectors:
  - E.g. `Eigen::Vector4d` or `Eigen::VectorXd`
      ```

Transforms with Eigen: Rotation

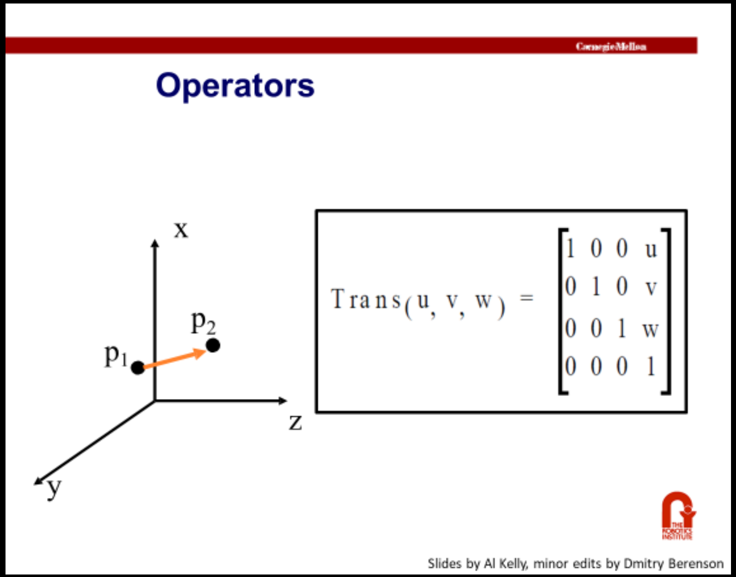



```cpp
#include <iostream>
#include <math.h>
#include <eigen3/Eigen/Eigen>

Eigen::Matrix4d Rotx(double angle)
{
    Eigen::Matrix4d m;
    m    << 1, 0, 0, 0,
            0, cos(angle), -sin(angle), 0,
            0, sin(angle), cos(angle), 0,
            0, 0, 0, 1;
    return m;
}

int main(){

    std::cout << "Rotx(pi/4): " << std::endl
              << Rotx(M_PI/4)<<std::endl;

    return 0;
}
```

Output:

```
Rotx(pi/4):
0  0.707107 -0.707107        0
0  0.707107  0.707107        0
0         0         0        1
```

Transforms with Eigen: Translation




```cpp
#include <iostream>
#include <math.h>
#include <eigen3/Eigen/Eigen>


Eigen::Matrix4d Trans(const Eigen::Vector3d& vec)
{
    Eigen::Matrix4d m;
    m.setIdentity();
    m(0,3) = vec(0);
    m(1,3) = vec(1);
    m(2,3) = vec(2);
    return m;
}

int main(){

    std::cout << "Trans(Vector3d(0.1,0.2,0.3)): "
              << std::endl
              << Trans(Eigen::Vector3d(0.1,0.2,0.3))
              << std::endl;
    return 0;
}
```

Output:

```
Trans(Vector3d(0.1,0.2,0.3)):
  1   0   0 0.1
  0   1   0 0.2
  0   0   1 0.3
  0   0   0   1
```

Transforms with Eigen: Rotate a vector

```cpp
#include <iostream>
#include <math.h>
#include <eigen3/Eigen/Eigen>

Eigen::Matrix4d Rotx(double angle)
{
    Eigen::Matrix4d m;
    m    << 1, 0, 0, 0,
            0, cos(angle), -sin(angle), 0,
            0, sin(angle), cos(angle), 0,
            0, 0, 0, 1;
    return m;
}

int main(){
    //the "1" at the end makes this homogenous coordinates
    Eigen::Vector4d xyzw(0.5,0.7,1.2,1);

    std::cout << "Rotx(pi/4)*xyzw: " << std::endl
              << Rotx(M_PI/4)*xyzw << std::endl;

    return 0;
}
```

Output:

```
Rotx(pi/4)*xyzw:
      0.5
-0.353553
   1.3435
        1
```

Copy contents of an std::vector to an Eigen::Vector

- If you know the std::vector is a certain (small) size, use e.g. `Eigen::Vector3d`
- Otherwise, use `Eigen::VectorXd` , but be careful about how you copy data

```
std::vector<double> std_vec{1.1,2.1,3.1};

//this works because Vector3d is a known size (reads 3 values)
Eigen::Vector3d eigen_vec1(std_vec.data()); //std_vec.data() returns a pointer

//do this if you want to use VectorXd
Eigen::VectorXd eigen_vec2 = Eigen::Map<Eigen::VectorXd>(std_vec.data(), std_vec.size());
```

- WARNING: Make sure the data type of std::vector and Eigen::Vector are the same!

Homework

- Homework 3 due Weds!
- (optional – for review) Boyd Linear Algebra Book: Chapters 1.1-1.4; 3.1-3.2; 5.1-5.3; 6.1-6.4; 10.1; 11
- (not optional – for least squares) Boyd Linear Algebra Book: Chapter 13.1