Aalto University
School of Electrical
Engineering

Introduction to ROS and Eigen Library

Gökhan Alcan

Dept. of Electrical Engineering and Automation
gokhan.alcan@aalto.fi

ELEC-E8126 - Robotic manipulation

Introduction

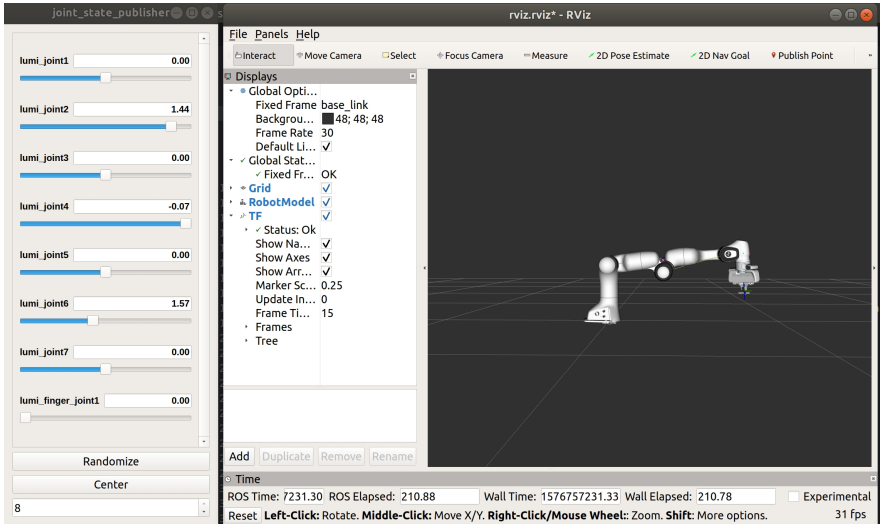
Today

- Introduction to ROS:
  - nodes
  - publisher/subscriber
  - services
  - rosbag
  - params and launch files
- Eigen Library:
  - matrix and vector
  - declarations and initialization
  - basic operations
  - transformations

Robot Operating System

The Robot Operating System (ROS)

- Robotic middleware
- De-facto standard for robotic research
- Main features:
  - Open-source
  - Decentralized architecture
  - Asyncronous communication
  - Visualization and simulation tools
  - Language agnostic



ROS Robots

https://robots.ros.org/

History of ROS Versions

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys **(Recommended)**	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015
ROS Groovy Galapagos	December 31, 2012			July, 2014
ROS Fuerte Turtle	April 23, 2012			--
ROS Electric Emys	August 30, 2011			--
ROS Diamondback	March 2, 2011			--
ROS C Turtle	August 2, 2010			--
ROS Box Turtle	March 2, 2010			--

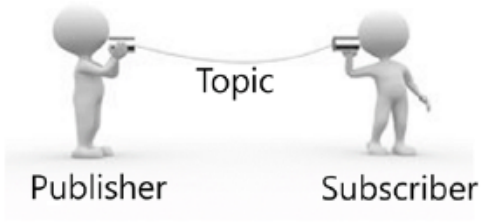
http://wiki.ros.org/Distributions

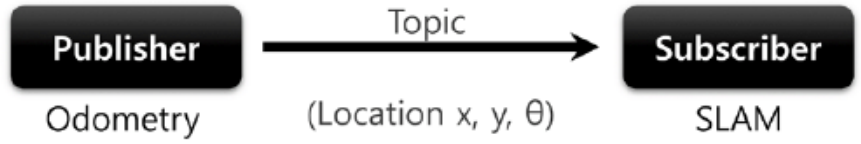
First C++ Node

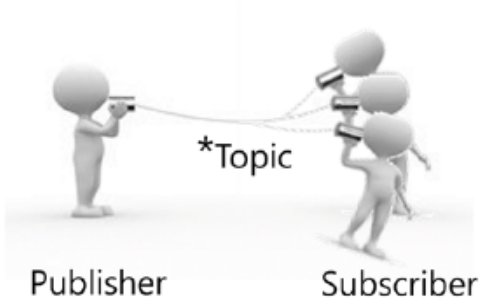
```
1  #include <ros/ros.h>
2
3  int main (int argc, char **argv)
4  {
5          ros::init(argc, argv, "my_first_cpp_node");
6          ros::NodeHandle nh;
7          ROS_INFO("Node has been started");
8
9          ros::Rate rate(10);
10
11         while (ros::ok()) {
12                 ROS_INFO("Hello");
13                 rate.sleep();
14         }
15 }
```

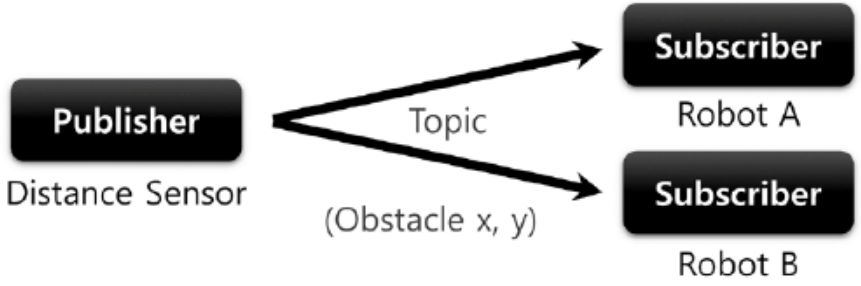
Demo time

ROS Publisher/Subscriber









*Topic not only allows 1:1 Publisher and Subscriber communication, but also supports 1:N, N:1 and N:N depending on the purpose.

Image is from the book of "ROS Robot Programming" by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim.

Publisher

```
1  #include <ros/ros.h>
2  #include <std_msgs/Int64.h>
3
4  int main (int argc, char **argv)
5  {
6          ros::init(argc, argv, "number_publisher", ↩
                ros::init_options::AnonymousName);
7          ros::NodeHandle nh;
8          ros::Publisher pub = nh.advertise<std_msgs::Int64>("/number", 10);
9          ros::Rate rate(1);
10
11         while (ros::ok()) {
12                 std_msgs::Int64 msg;
13                 msg.data = 2;
14                 pub.publish(msg);
15                 rate.sleep();
16         }
17 }
```

Subscriber

```
#include <ros/ros.h>
#include <std_msgs/Int64.h>
int counter = 0;
ros::Publisher pub;
void callback_number(const std_msgs::Int64& msg){
        counter += msg.data;
        std_msgs::Int64 new_msg;
        new_msg.data = counter;
        pub.publish(new_msg);
}
int main (int argc, char **argv)
{
        ros::init(argc, argv, "number_counter");
        ros::NodeHandle nh;
        ros::Subscriber sub = nh.subscribe("/number", 1000, callback_number);
        pub = nh.advertise<std_msgs::Int64>("/number_count", 10);
        ros::spin();
}
```

Demo time

ROS Service

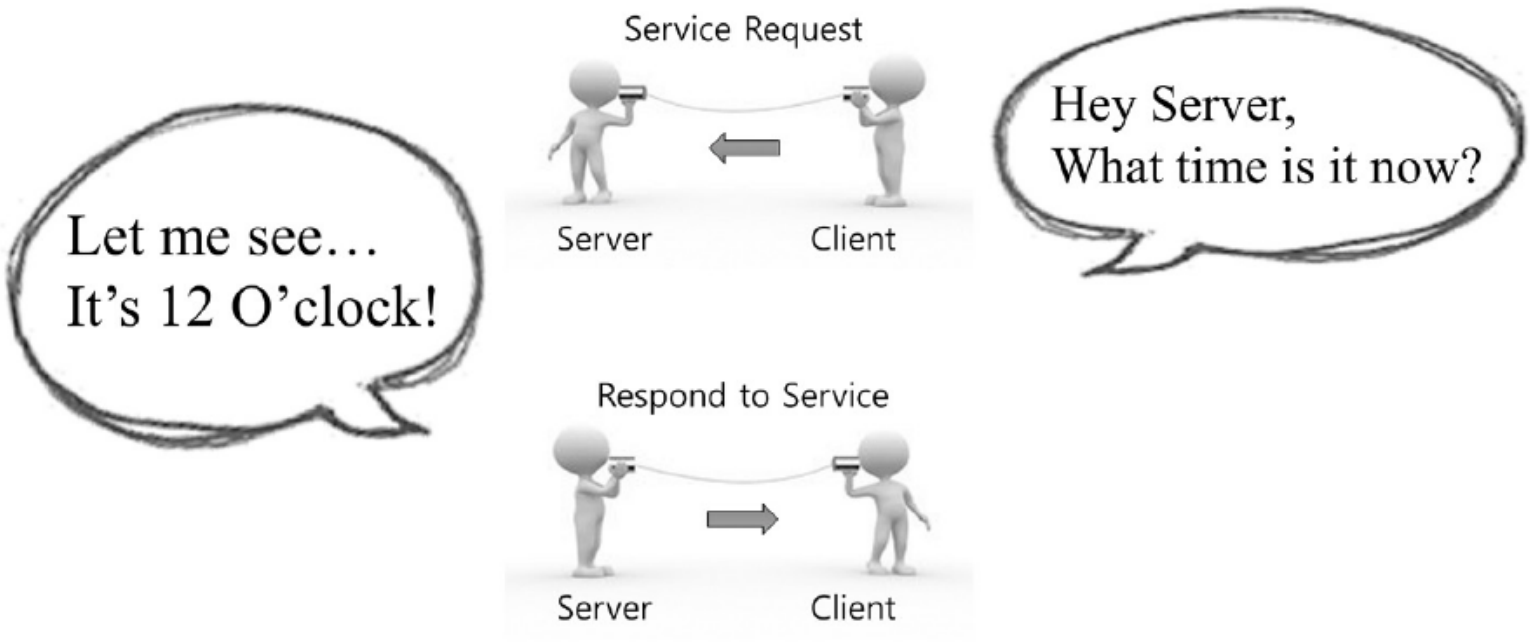


Image is from the book of "ROS Robot Programming" by YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim.

Service Server

```cpp
#include <ros/ros.h>
#include <rospy_tutorials/AddTwoInts.h>
bool handle_add_two_ints(rospy_tutorials::AddTwoInts::Request &req, ↩
     rospy_tutorials::AddTwoInts::Response &res)
{
        int result = req.a + req.b;
        ROS_INFO("%d + %d = %d", (int)req.a, (int)req.b, (int)result);
        res.sum = result;
        return true;
}
int main (int argc, char **argv)
{
        ros::init(argc, argv, "add_two_ints_server");
        ros::NodeHandle nh;
        ros::ServiceServer server = nh.advertiseService("/add_two_ints", ↩
             handle_add_two_ints);
        ros::spin();
}
```

Service Client

```
1  #include <ros/ros.h>
2  #include <rospy_tutorials/AddTwoInts.h>
3  int main (int argc, char **argv)
4  {
5          ros::init(argc, argv, "add_two_ints_client");
6          ros::NodeHandle nh;
7          ros::ServiceClient client = ⟵
               nh.serviceClient<rospy_tutorials::AddTwoInts>("/add_two_ints");
8          rospy_tutorials::AddTwoInts srv;
9          srv.request.a = 12;
10         srv.request.b = 5;
11         if (client.call(srv)) {
12                 ROS_INFO("Returned sum is %d", (int)srv.response.sum);
13         }
14         else {
15                 ROS_WARN("Service call failed");
16         }
17 }
```

Demo time

ROS Bag

- It is a tool for recording from and playing back to ROS topics.
- Recording some rostopics:

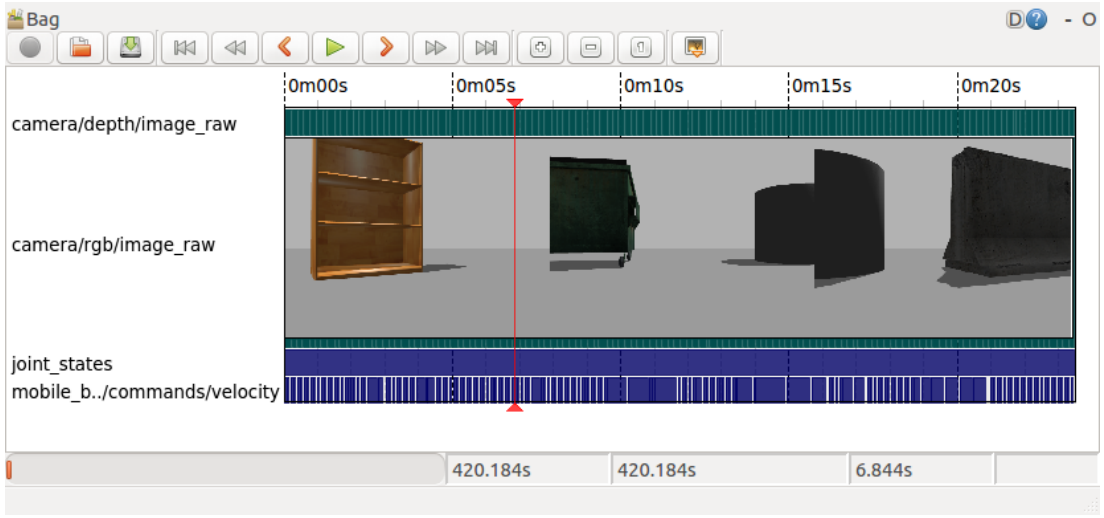
```
rosbag record rostopic1 rostopic2 rostopic3 ...
```

- Playing back the recorded data later:

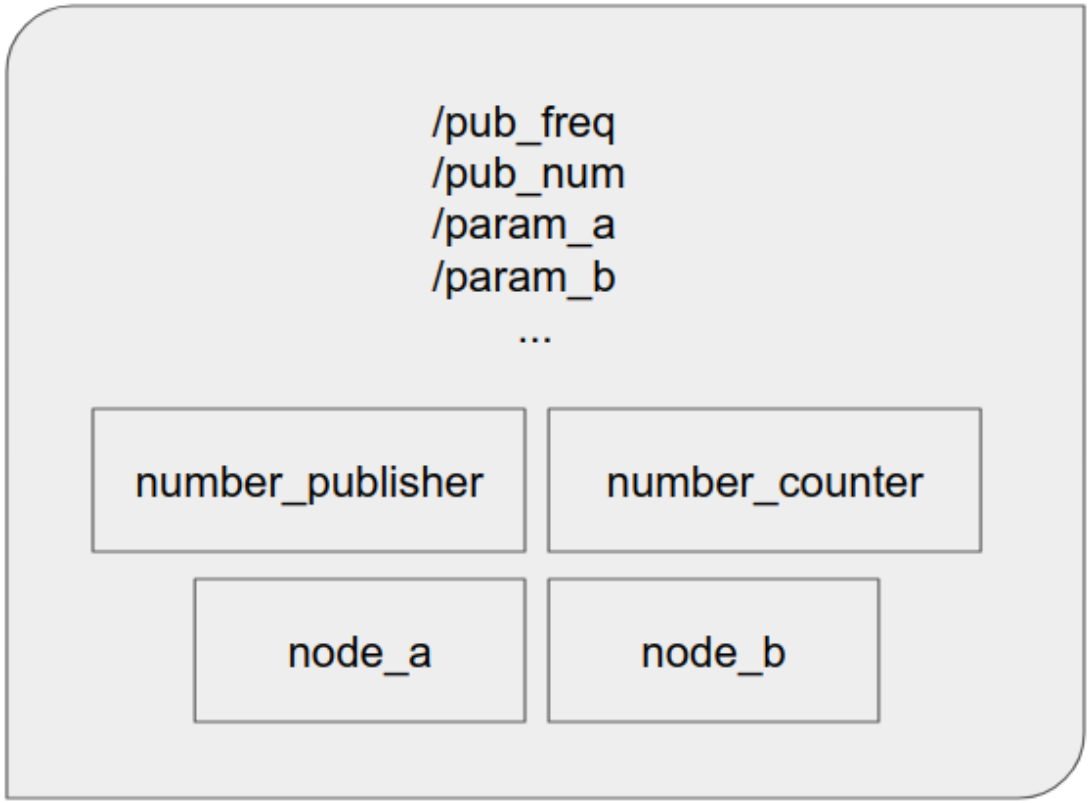
```
rosbag play FILENAME.bag
```

- There is also a nice GUI for rosbags that you can even start recording, playing the existing bags, plot imshow etc.

```
rqt_bag
```



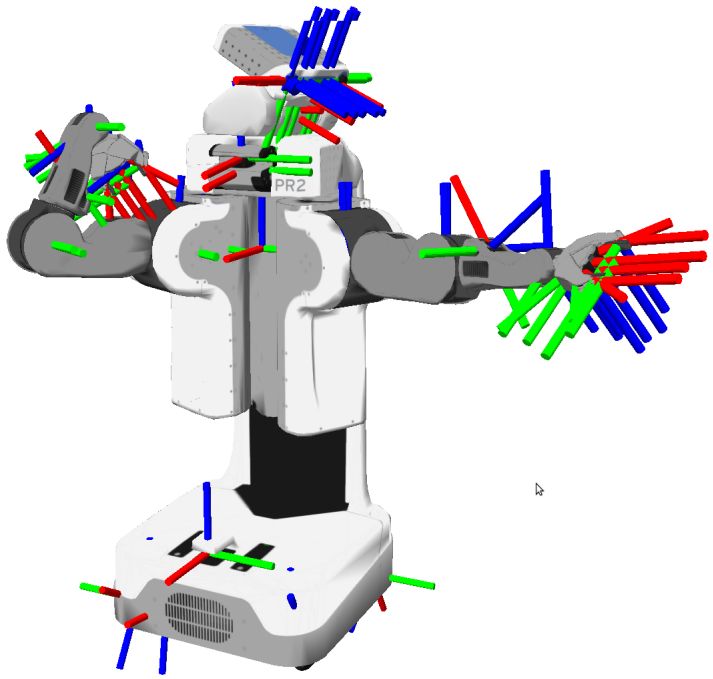
ROS Params and Launch Files



Demo time

Transformation Library

`tf2` is the second generation of the transform library, which lets the user keep track of multiple coordinate frames over time.



http://wiki.ros.org/tf2

http://wiki.ros.org/tf2/Tutorials

C++

1. Writing a tf2 static broadcaster (C++)
   This tutorial teaches you how to broadcast static coordinate frames to tf2
2. Writing a tf2 broadcaster (C++)
   This tutorial teaches you how to broadcast coordinate frames of a robot to tf2.
3. Writing a tf2 listener (C++)
   This tutorial teaches you how to use tf2 to get access to frame transformations.
4. Adding a frame (C++)
   This tutorial teaches you how to add an extra fixed frame to tf2.
5. Learning about tf2 and time (C++)
   This tutorial teaches you to wait for a transform to be available on the tf2 tree when using the lookupTransform() function.
6. Time travel with tf2 (C++)
   This tutorial teaches you about advanced time travel features of tf2

Eigen C++ Library

Eigen C++ Library

Eigen is an efficient high level C++ library for linear algebra, matrix and vector operations, geometrical transformations, numerical solvers and related algorithms. Eigen library:

- is open-source,
- is well-documented,
- is fast and reliable,
- performs its own loop unrolling and vectorization,
- is supported by major compilers,
- does not have any dependencies other than the C++ standard library.

https://eigen.tuxfamily.org/

Matrix and Vector Definitions

Explicit `Eigen::Matrix <data type, rows, cols>`

3×4 matrix with float entries:

```
Eigen::Matrix <float, 3, 4> matA;
```

Dynamic matrix with double entries:

```
Eigen::Matrix <double, Eigen::Dynamic, Eigen::Dynamic> matA;
```

Typedef `Eigen::Matrix`**DIM***type*

4×4 matrix with float entries:

```
Eigen::Matrix4f matA;
```

Dynamic matrix with double entries:

```
Eigen::MatrixXd matA;
```

Typedef `Eigen::Vector`**DIM***type*

3×1 vector with float entries:

```
Eigen::Vector3f vecb;
```

Dynamic vector with double entries:

```
Eigen::VectorXd vecb;
```

Matrix Initializations

Functions

```
matA.setZero();
matA.setOnes();
matA.setIdentity();
matA.setConstant(value);
matA.setRandom();
```

Fill-in the entries

```
Eigen::Matrix2f matA;
matA << 1.3, 4.2, 7.5, 9.7;
```

$$\text{matA} = \begin{bmatrix} 1.3 & 4.2 \\ 7.5 & 9.7 \end{bmatrix}$$

Accessing the Values

- Accessing a single entry

  `matA(1,2)` *output: 6.1*

- Accessing a matrix block

  `matA.block(1, 0, 2, 3)`

  output : $\begin{bmatrix} 7.5 & 9.7 & 6.1 \\ 0.6 & 1.2 & 8.8 \end{bmatrix}$

- Accessing columns and rows of a matrix

  `matA.row(2)`

  output : $\begin{bmatrix} 0.6 & 1.2 & 8.8 & 9.3 \end{bmatrix}$

  `matA.col(0)`

  output : $\begin{bmatrix} 1.3 \\ 7.5 \\ 0.6 \\ 5.9 \end{bmatrix}$

$$\text{matA} = \begin{bmatrix} 1.3 & 4.2 & 3.3 & 5.2 \\ 7.5 & 9.7 & 6.1 & 2.0 \\ 0.6 & 1.2 & 8.8 & 9.3 \\ 5.9 & 2.7 & 0.2 & 1.1 \end{bmatrix}$$

- Accessing the dimensions

  `matA.rows()` *output: 4*

  `matA.cols()` *output: 4*

  `matA.size()` *output: 16*

Some Basic Operations

- Matrix addition

```
matC = matA + matB;
```

- Matrix multiplication

```
matC = matA * matB;
```

- Multiplication with a scalar

```
matC = s1 * matB;
matC = matB * s1;
```

- Transpose operator

```
matC = matA.transpose() + matB;
matA.transposeInPlace();
```

- Inverse operator

```
matC = matA.inverse();
JtpinvL = (J * J.transpose()).inverse() * J;
```

https://eigen.tuxfamily.org/dox/group__TutorialMatrixArithmetic.html

Coordinate frames and transforms

The Lie group $SE(3)$

- Three-dimensional Special Euclidean group:

$$SE(3) = \left\{ \mathbf{A} \mid \mathbf{A} = \left[\begin{array}{c|c} \mathbf{R} & \mathbf{r} \\ \hline \mathbf{0}^{1\times 3} & 1 \end{array}\right], \mathbf{R} \in SO(3), \mathbf{r} \in \mathbb{R}^3 \right\}$$

- **R** represents rotation/orientation
- **r** represents translation

Orientation representations

$SO(3)$ Three-dimensional Special Orthogonal group:

$$SO(3) = \left\{ \mathbf{R} \mid \mathbf{R} \in \mathbb{R}^{3\times 3}, \mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}, \det(\mathbf{R}) = 1 \right\}$$

Euler angles Vector representanting rotation angle in each direction

Axis-angle An orientation vector $\vec{u} = (u_x, u_y, u_z)$ and an angle value θ

Quaternions 4-dimensional complex number $w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ embedding a 3D orientation $\mathbf{q}$:

$$\mathbf{q} = \exp^{\frac{\theta}{2}(u_x\mathbf{i}+u_y\mathbf{j}+u_z\mathbf{k})} = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}$$

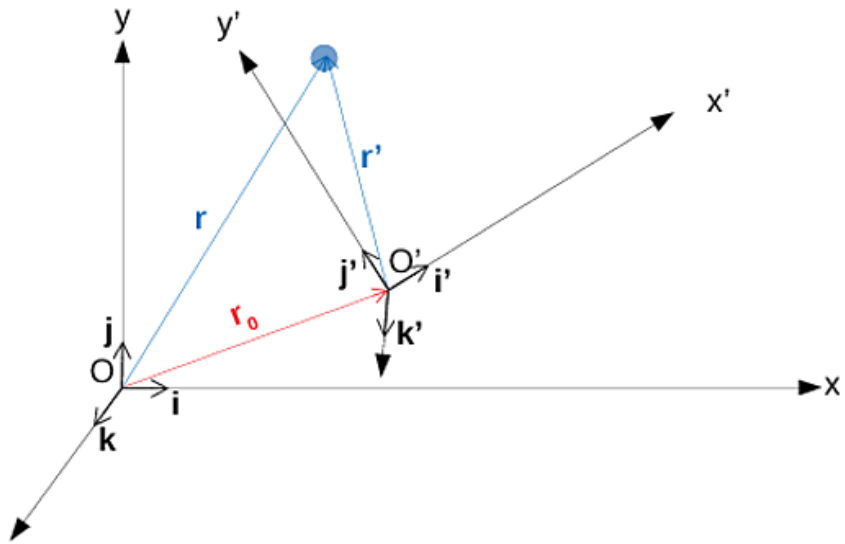
Coordinate transformation

Transformation from O' to O

$$\begin{bmatrix} \mathbf{r} \\ \hline 1 \end{bmatrix} = \left[\begin{array}{c|c} {}^{O}\mathbf{R}_{O'} & {}^{O}\mathbf{r}^{O'} \\ \hline \mathbf{0}^{1\times 3} & 1 \end{array}\right] \begin{bmatrix} \mathbf{r}' \\ \hline 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{r} = {}^{O}\mathbf{R}_{O'}\mathbf{r}' + {}^{O}\mathbf{r}^{O'}$$

Homogeneous transformation

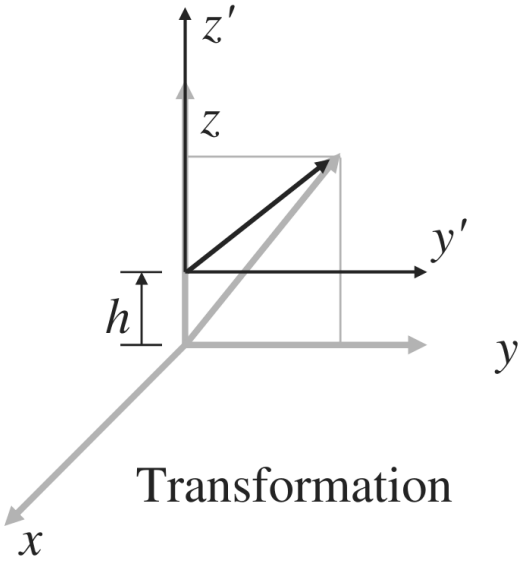
$${}^{A}\mathbf{A}_{B} = \left[\begin{array}{c|c} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{r}^{B} \\ \hline \mathbf{0}^{1\times 3} & 1 \end{array}\right]$$



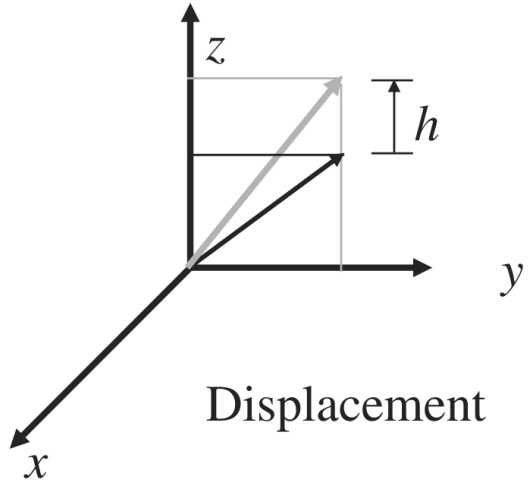
Translation

Translation along the z-axis through h

$$Trans(z,h) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & h \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & h \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}$$



Transformation

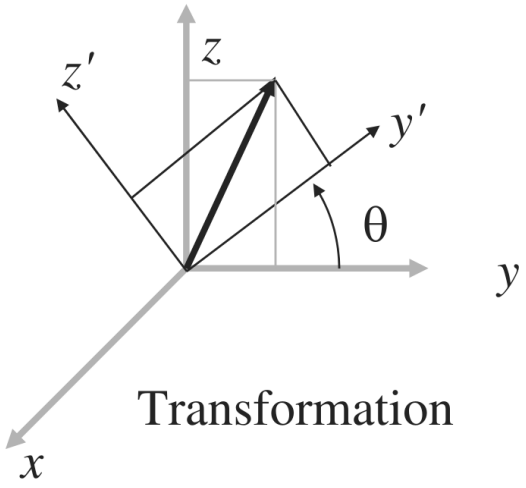


Displacement

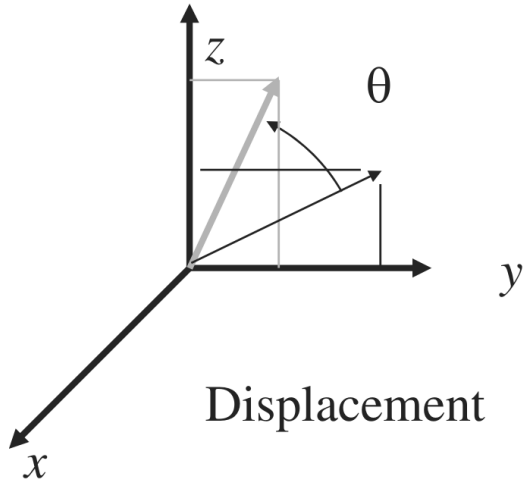
Rotation

Rotation along the x-axis through θ

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



Transformation



Displacement

Rotation (cont.)

Rotation along the y-axis through θ

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

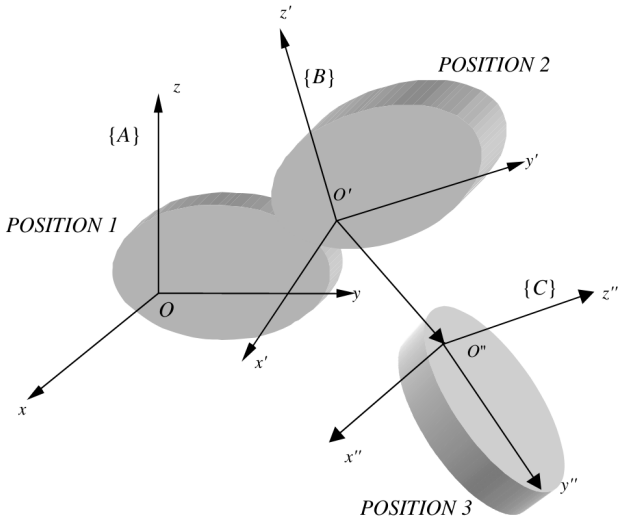
Rotation along the z-axis through θ

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

Composition

Displacement from $\{A\}$ to $\{C\}$

$$
\begin{aligned}
{}^{A}\mathbf{A}_{C} &= {}^{A}\mathbf{A}_{B}\,{}^{B}\mathbf{A}_{C} \\
&= \left[\begin{array}{c|c} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{r}^{B} \\ \hline \mathbf{0}^{1\times 3} & 1 \end{array}\right] \times \left[\begin{array}{c|c} {}^{B}\mathbf{R}_{C} & {}^{B}\mathbf{r}^{C} \\ \hline \mathbf{0}^{1\times 3} & 1 \end{array}\right] \\
&= \left[\begin{array}{c|c} {}^{A}\mathbf{R}_{B} \times {}^{B}\mathbf{R}_{C} & {}^{A}\mathbf{R}_{B} \times {}^{B}\mathbf{r}^{C} + {}^{A}\mathbf{r}^{B} \\ \hline \mathbf{0}^{1\times 3} & 1 \end{array}\right]
\end{aligned}
$$



Transformations in Eigen Library

- Isometric transform

```
Eigen::Isometry3d T;
```

- Affine transform

```
Eigen::Affine3d T;
```

  - `T.linear()`
  - `T.translation()`
  - `T.matrix()`

$$T = \begin{bmatrix} \textit{linear} & \textit{translation} \\ 0 \ldots 0 & 1 \end{bmatrix}$$

- Translation

```
Eigen::Translation3d tranA;
tranA = Eigen::Translation3d(0.0, 0.0, 0.1);
```

- Rotation

```
Eigen::AngleAxisd rotA;
rotA = Eigen::AngleAxisd(M_PI, Eigen::Vector3d::UnitX());
```

Demo time

Today’s takeaways

Questions?