

Introduction to Cognitive Robotics

Module 10: Using Turtlesim with CARM

Lecture 1: Pose specification in ROS and Lisp

David Vernon

Carnegie Mellon University Africa

www.vernon.eu

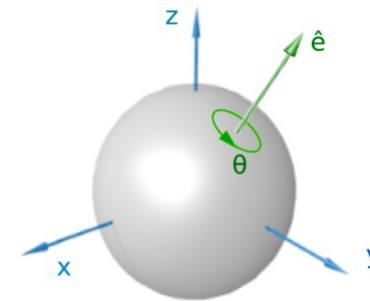
Pose Specification in ROS and Lisp

- Previously, we used homogeneous transformations to specify a frame of reference for end-effector and object **pose**
- We noted that ROS uses a different (but entirely equivalent) approach
 - Specify the origin of the frame as a **3-D vector**
 - Specify the orientation of the frame as a **quaternion**: a single **rotation** about some (appropriate) axis

Pose Specification in ROS and Lisp

- Euler's rotation theorem states that any displacement of a rigid body (in 3D space), such that a point on the rigid body remains fixed,

is equivalent to a **single rotation θ** about **some axis** that runs through the fixed point



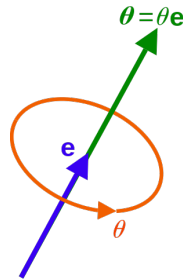
- The axis of rotation is known as an **Euler axis**, typically represented by a unit vector \hat{e}

https://en.wikipedia.org/wiki/Euler%27s_rotation_theorem

See: https://en.wikipedia.org/wiki/Rotation_formalisms_in_three_dimensions

Pose Specification in ROS and Lisp

- The product by $\theta \hat{e}$ is known as an axis-angle



https://en.wikipedia.org/wiki/Axis-angle_representation

- Quaternions** are a simple way to encode this axis-angle representation of a rotation in four numbers

Pose Specification in ROS and Lisp

- Quaternions are hypercomplex numbers

Vector part
(imaginary part)

$$q = w + x \mathbf{i} + y \mathbf{j} + z \mathbf{k}$$

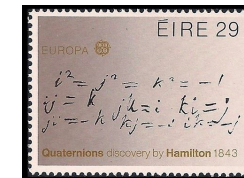
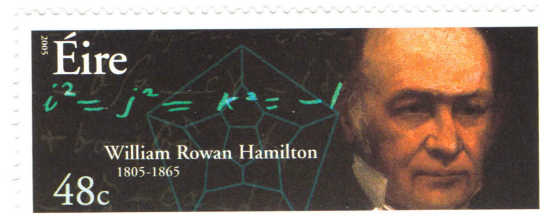
Scalar part
(real part)

$w, x, y,$ and z are real numbers

quaternion units

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

- Discovered by Irish mathematician William Rowan Hamilton in 1843



Pose Specification in ROS and Lisp

- A rotation of θ about the Euler axis $\hat{\mathbf{e}} = e_x \mathbf{i} + e_y \mathbf{j} + e_z \mathbf{k}$ is given by

$$\begin{aligned} q &= w + x \mathbf{i} + y \mathbf{j} + z \mathbf{k} \\ &= \cos(\theta/2) e_x \sin(\theta/2) \mathbf{i} + e_y \sin(\theta/2) \mathbf{j} + e_z \sin(\theta/2) \mathbf{k} \end{aligned}$$

- Equivalently

$$q = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}$$

Pose Specification in ROS and Lisp

- In ROS we write it slightly differently

$$q = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Axis part

Angle part

- In Lisp, we specify a quaternion as follows:

```
CL-TRANSFORMS> (make-quaternion 0 0 0 1)  
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>
```

Axis part

Angle part the cosine of the angle zero divided by two equals one
 $\cos 0/2 = 1$

Note: this and the other example functions are part of the `cl_transforms` library
To use them, you must first load the `cl-transforms` package:

```
CL-USER> (ros-load:load-system "cl_transforms" :cl-transforms)
```

```
CL-USER> (in-package cl-transforms)
```

Note: underscore, not hyphen

Also see Slide 25

Pose Specification in ROS and Lisp

In ROS we specify a 3D **point** in Lisp as follows:

```
CL-TRANSFORMS> (make-3d-vector 1 2 3)  
#<3D-VECTOR (1.0d0 2.0d0 3.0d0)>
```

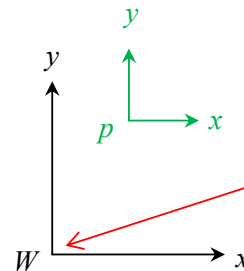

Pose Specification in ROS and Lisp

Putting these together, we can specify a **pose** as follows:

```
CL-TRANSFORMS> (setf p (make-pose  
                        (make-3d-vector 1 2 0)  
                        (make-quaternion 0 0 0 1)))
```

Frame p is a translation (1, 2, 0) w.r.t.
the world frame of reference

```
#<POSE  
#<3D-VECTOR (1.0d0 2.0d0 0.0d0)>  
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
```



The z axis is directed upwards to make a right-hand system.
The ROS documentation states that
“Coordinate systems in ROS are always in 3D,
and are right-handed, with X forward, Y left, and Z up.”
<http://wiki.ros.org/tf/Overview/Transformations>

Pose Specification in ROS and Lisp

We can extract information about a pose:

```
CL-TRANSFORMS> (origin p)  
#<3D-VECTOR (1.0d0 2.0d0 0.0d0)>
```

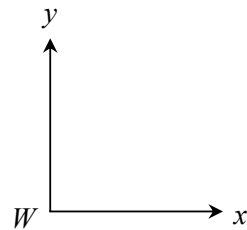
```
CL-TRANSFORMS> (orientation p)  
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>
```

Pose Specification in ROS and Lisp

Some more pose operations:

```
CL-TRANSFORMS> (setf W (make-identity-pose))  
#<POSE  
#<3D-VECTOR (0.0d0 0.0d0 0.0d0)>  
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
```

World frame of reference W

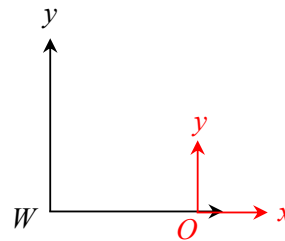


Pose Specification in ROS and Lisp

Some more pose operations:

```
CL-TRANSFORMS> (setf O (make-pose translation (2, 0, 0) w.r.t. to W
                    (make-3d-vector 2 0 0)
                    (make-quaternion 0 0 0 1)))

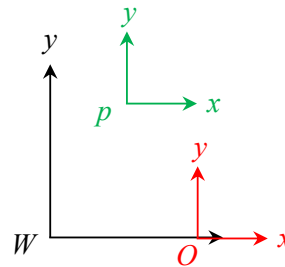
#<POSE
#<3D-VECTOR (2.0d0 0.0d0 0.0d0)>
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
```



Pose Specification in ROS and Lisp

How would we determine the pose of p w.r.t. O ?

$$\begin{aligned} {}^O p &= {}^O W * {}^W p \\ &= ({}^W O)^{-1} * {}^W p \\ &= (\text{Trans}(2, 0, 0))^{-1} * \text{Trans}(1, 2, 0) \\ &= \text{Trans}(-2, 0, 0) * \text{Trans}(1, 2, 0) \\ &= \text{Trans}(-1, 2, 0) \leftarrow \text{w.r.t. } O \end{aligned}$$



Pose Specification in ROS and Lisp

To implement this, we need O to be a **transform** (so that we can form the inverse transform), not a pose

```
CL-TRANSFORMS> (setf O (make-transform
                        (make-3d-vector 2 0 0)
                        (make-quaternion 0 0 0 1)))

#<TRANSFORM
#<3D-VECTOR (2.0d0 0.0d0 0.0d0)>
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
```

Pose Specification in ROS and Lisp

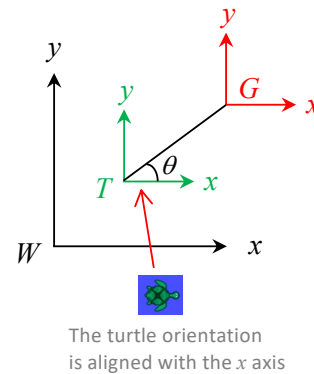
CL-TRANSFORMS> (transform-pose $(^WO)^{-1}$
 (transform-inv 0) $\left. \vphantom{\begin{matrix} (transform-pose \\ (transform-inv 0) \end{matrix}} \right\} (^WO)^{-1} * {}^Wp$
 p)

#<POSE
#<3D-VECTOR (-1.0d0 2.0d0 0.0d0)>
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>

Pose Specification in ROS and Lisp

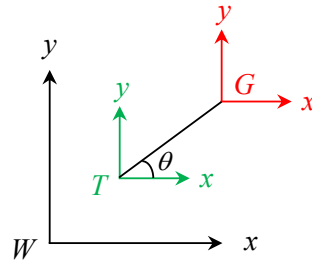
We can use the same approach for determining the pose of the goal with respect to the turtle

$$\begin{aligned} {}^T G &= {}^T W * {}^W G \\ &= ({}^W T)^{-1} * {}^W G \end{aligned}$$



Pose Specification in ROS and Lisp

```
CL-TRANSFORMS> (setf G (make-pose  
                        (make-3d-vector 3 3 0)  
                        (make-quaternion 0 0 0 1)))  
  
#<POSE  
#<3D-VECTOR (3.0d0 3.0d0 0.0d0)>  
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
```

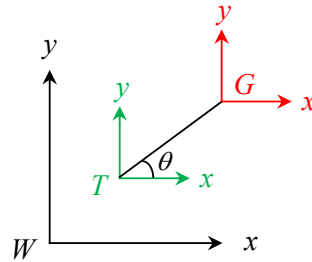


Pose Specification in ROS and Lisp

T is a constant and can't be set, so we use Turtle

```
CL-TRANSFORMS> (setf Turtle (make-transform
                             (make-3d-vector 1 1 0)
                             (make-quaternion 0 0 0 1)))

#<TRANSFORM
#<3D-VECTOR (1.0d0 1.0d0 0.0d0)>
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
```



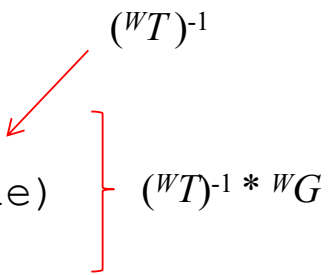
Pose Specification in ROS and Lisp

```
CL-TRANSFORMS> (transform-pose  
                  (transform-inv Turtle)  
                  G)
```

$({}^WT)^{-1}$

$({}^WT)^{-1} * {}^WG$

```
#<POSE  
#<3D-VECTOR (2.0d0 2.0d0 0.0d0)>  
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
```



Pose Specification in ROS and Lisp

What if the **turtle pose** is given by the data published on the **pose** topic, e.g., `pose_msg`?

where, for example, $x = 1$, $y = 1$, and $\theta \neq 0$

Pose Specification in ROS and Lisp

What if the **turtle pose** is given by the data published on the `pose` topic, e.g., `pose_msg`?

```
CL-TRANSFORMS> (transform-pose  
                  (transform-inv (pose->transform pose-msg))  
                  G)
```

$({}^WT)^{-1}$

$({}^WT)^{-1} * {}^WG$

```
#<POSE  
#<3D-VECTOR (2.0d0 2.0d0 0.0d0)>  
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>>
```

This gives us the coordinates
of the goal with respect to the
turtle's frame of reference

The direction to the goal is
given by `atan2(y, x)`



This function converts a pose
message to a transform.
We will implement it in the
next lecture

Pose Specification in ROS and Lisp

What if the **goal** is given, not by a pose, but by a **vector** with the coordinates of a location?

```
CL-TRANSFORMS> (setf goal (make-3d-vector 3 3 0))  
#<3D-VECTOR (3.0d0 3.0d0 0.0d0)>
```

Pose Specification in ROS and Lisp

We use the `transform` function to apply a transform to a vector

```
CL-TRANSFORMS> (transform  
                  (transform-inv (pose->transform pose-msg))  
                  goal)
```

$({}^WT)^{-1}$

$({}^WT)^{-1} * {}^Wgoal$

#<3D-VECTOR (2.0d0 2.0d0 0.0d0)>

This is a vector

This gives us the coordinates
of the goal with respect to the
turtle's frame of reference

The direction to the goal is
given by $\text{atan}(y, x)$



Pose Specification in ROS and Lisp

We will use this version in the
next lecture



```
CL-TRANSFORMS> (transform-point  
                  (transform-inv (pose->transform pose-msg))  
                  goal)  
#<3D-VECTOR (2.0d0 2.0d0 0.0d0)>
```

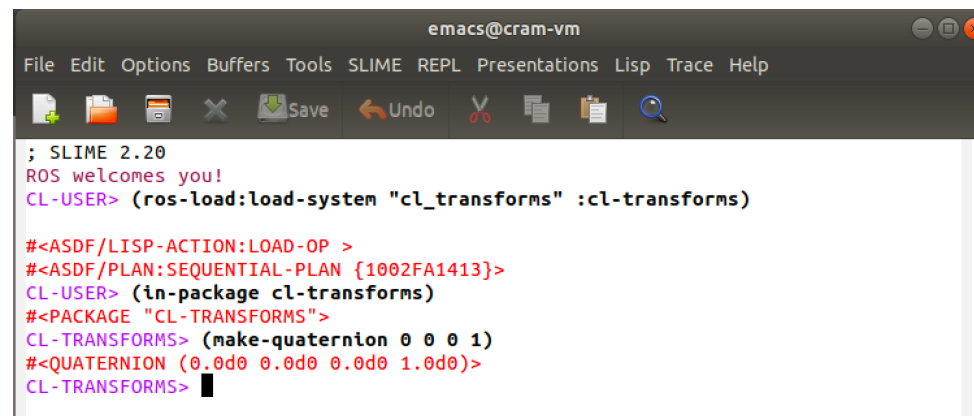

Pose Specification in ROS and Lisp

Note:

If you try these examples in `roslisp_repl` you will need to use the `cl_transforms` library by first loading the `cl_transforms` package, as follows

```
CL-USER> (ros-load:load-system "cl_transforms" :cl-transforms)
CL-USER> (in-package cl-transforms)
CL-TRANSFORMS>
```

Note: underscore, not hyphen



```
; SLIME 2.20
ROS welcomes you!
CL-USER> (ros-load:load-system "cl_transforms" :cl-transforms)

#<ASDF/LISP-ACTION:LOAD-OP >
#<ASDF/PLAN:SEQUENTIAL-PLAN {1002FA1413}>
CL-USER> (in-package cl-transforms)
#<PACKAGE "CL-TRANSFORMS">
CL-TRANSFORMS> (make-quaternion 0 0 0 1)
#<QUATERNION (0.0d0 0.0d0 0.0d0 1.0d0)>
CL-TRANSFORMS>
```