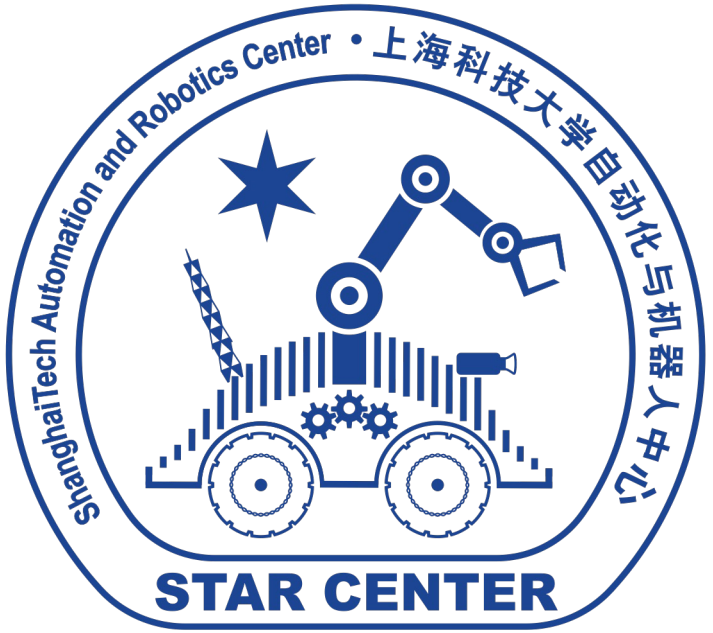
上海科技大学

ShanghaiTech University

CS289: **Mobile Manipulation Fall 2026**

Sören Schwertfeger

ShanghaiTech University



Outline

- Projects
- 3D Geometry
- Kinematics

ADMINISTRIVIA

Project

- 1 credit point!
- Work in groups, min 2 students, max 3 students – maybe 4 students.

- Some topics will be proposed later…
  - You can also do your own topic, but only after approval of Prof. Schwertfeger
    - Prepare a short, written proposal till next Tuesday (Sep 22nd)! Outline what you want to do. With which hardware. How you want to evaluate the results. Who wants to participate. Email to Prof. Schwertfeger, subject “MoMa 26 Project Idea”
  - Choose/ suggest a topic that is in line with your graduate research!
  - We are flexible w.r.t. the topics – as long as they are involved with manipulation or complex mobile robotics.

- One graduate student from my group will co-supervise your project
- Weekly project meetings!

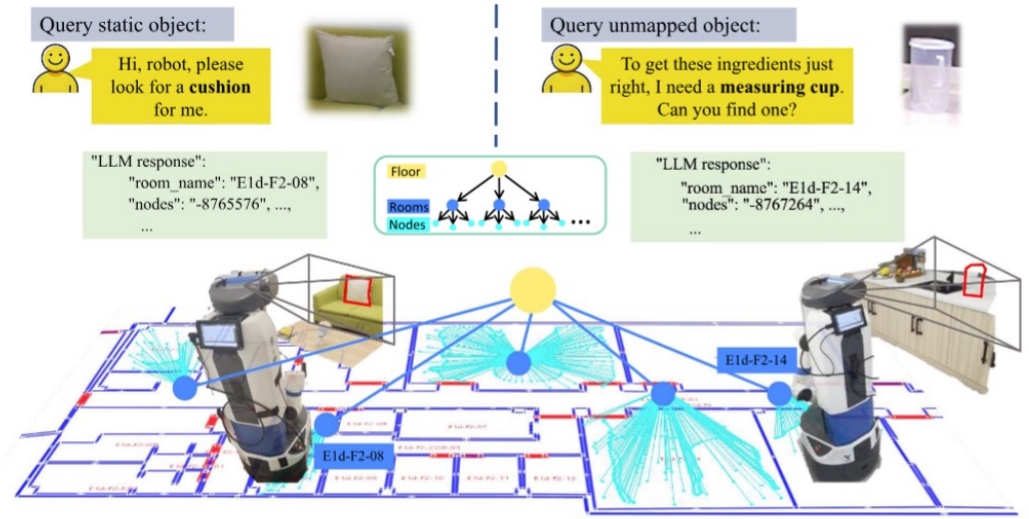
- Oral ”exams” to evaluate the contributions of each member
- No work on project => bad grade of fail

Project Suggestion: Fetch

- Use fetch to do a cool mobile manipulation project…

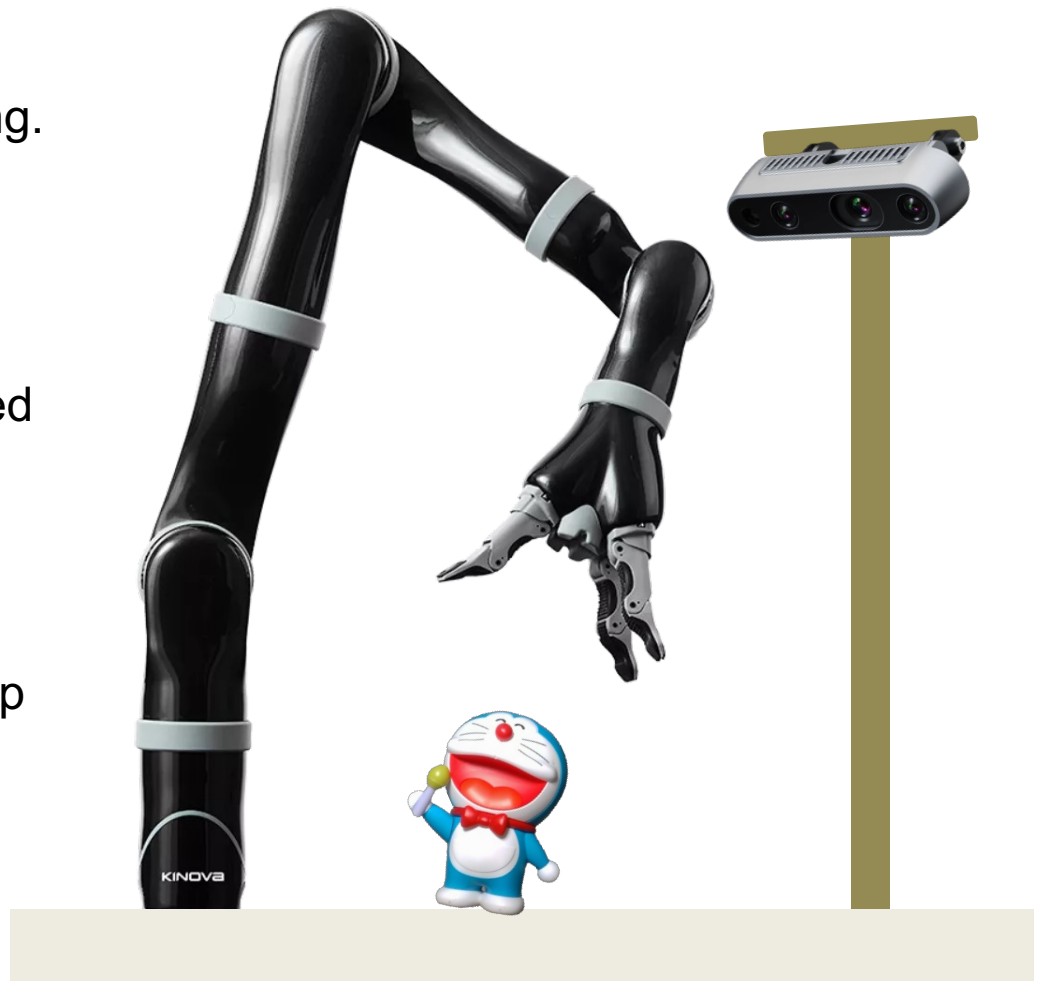
osmAG-Moma

- This project is based on osmAG-LLM. With object extracted and stored in osmAG format, we can navigate to every viewpoint where the object is observed. Structure: Floor - Room - viewpoints - objects. Therefore, we can have the LLM indentify which viewpoint to go.
- The project should be performed in our lab, because you can reuse the osmag map (for semantic) and grid map (for localization). **NOTE:** You probably need to modify osmag map(object part), because a lot of objects changed position.
- Task: **Decide by LLM, navigate to the viewpoint, find the target object**, approach to it, and let Fetch Grasp it. (Bold means it almost done from open-source code - https://github.com/xiexiexiaoxiexie/osmAG-LLM). The challenge lies: determining how close to get and how to execute the grasp.
- Yuxuan Feng (fengyx12025@shanghaitech.edu.cn)
- Recommanded reading: osmAG-LLM, HOV-SG, OK-Robot



Develop a CAD-free active in-hand 3D reconstruction system that combines model-free object tracking, incremental reconstruction, next-best-view planning, and robotic object reorientation.

- Use a fixed RGB-D camera and a robotic arm for in-hand scanning.
- Grasp and scan a previously unseen object without requiring its CAD model.
- Estimate the 6D object pose and incrementally reconstruct its 3D geometry.
- Maintain the current partial object reconstruction from accumulated observations.
- Predict the next best view based on the current reconstruction.
- Convert the next best view into a target in-hand object pose.
- Control the robotic arm to reorient the object to the target pose.
- Repeat the perception–reconstruction–planning–manipulation loop until sufficient surface coverage is achieved.



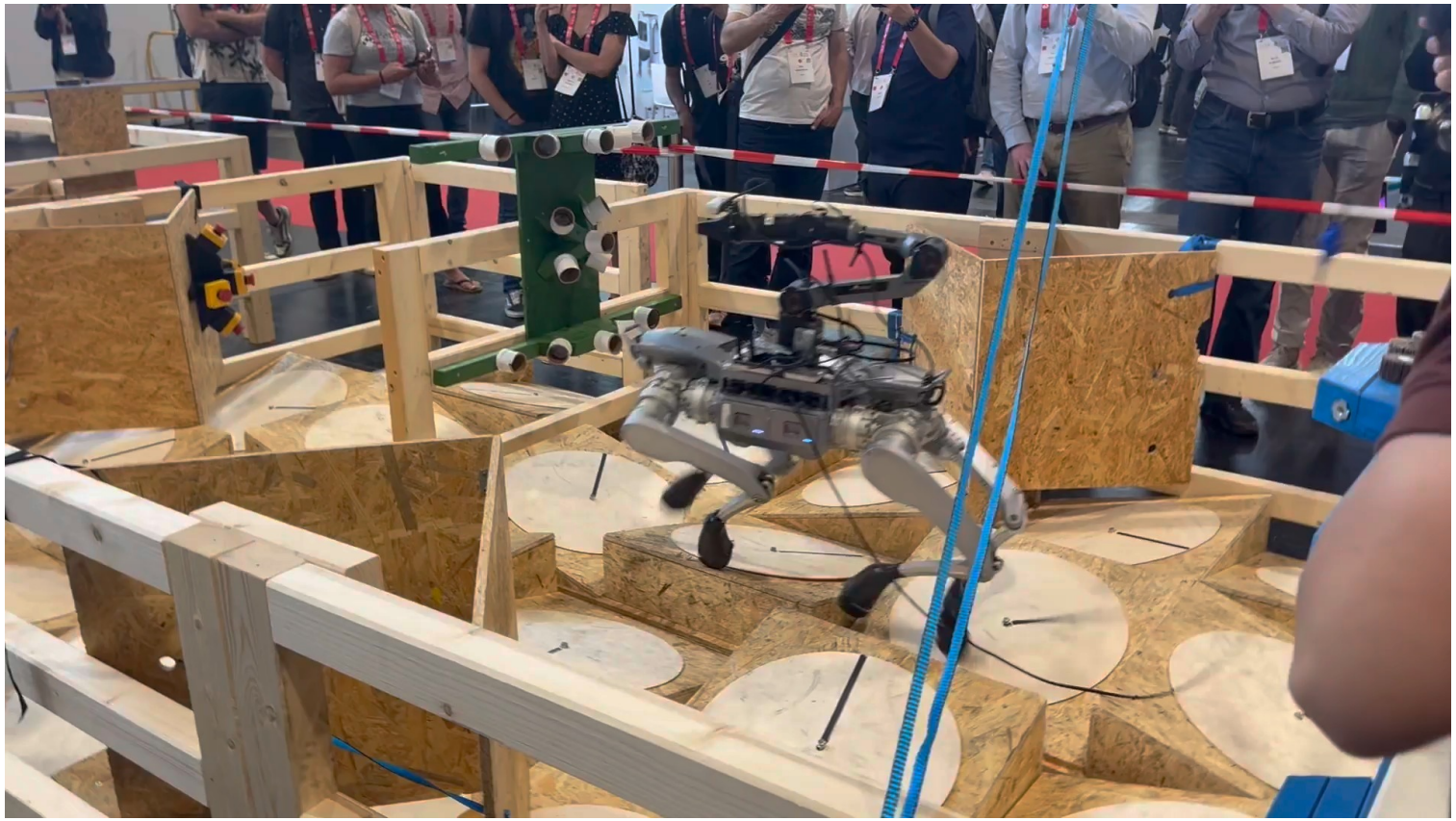
Related Papers:
BundleSDF: Neural 6-DoF Tracking and 3D Reconstruction of Unknown Objects
PIT-NBV: Poisson-Informed Transformer for 6-DOF Next Best View Planning in 3-D Object Reconstruction With Narrow Field of View
Active Implicit Object Reconstruction using Uncertainty-guided Next-Best-View Optimization

Do something cool with this robot…

Husky Mobile Platform with
AUBO Robot Arm

Quadruped Robot Challenges & RoboCup

- Do some work regarding those competitions!

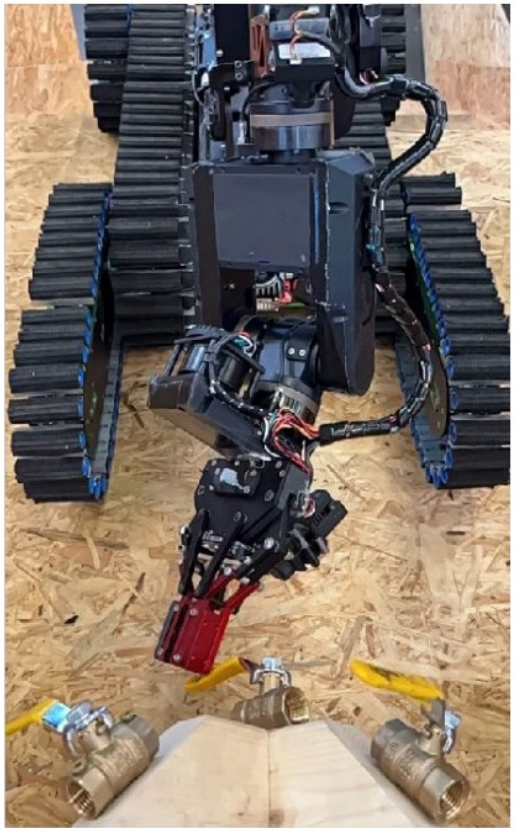


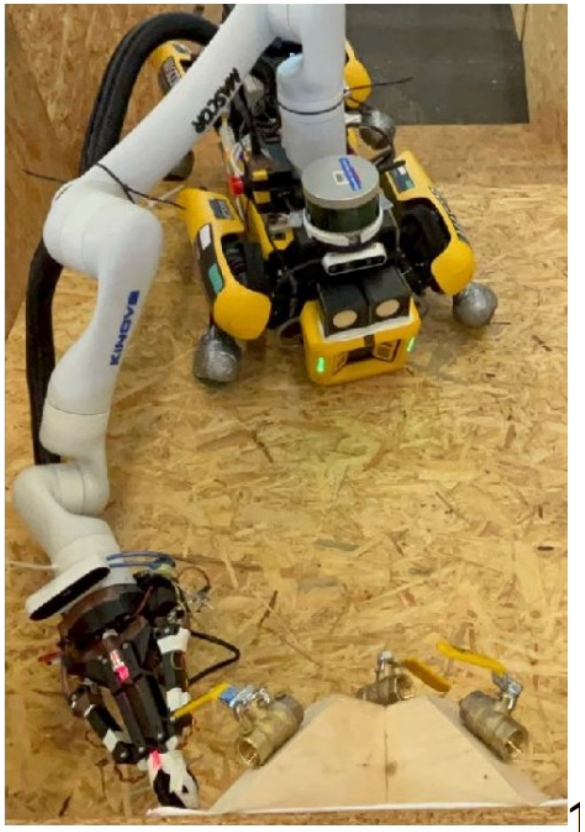
A2-W with Piper X arm

Currently only Go2-W is available

RoboCup Valve Operation with the Arm

- Autonomously trun vavle using RGB-D caerma, Piper X arm
- Later: mounted on mobile robot…





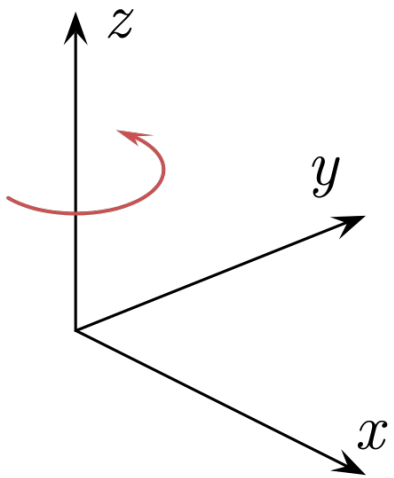
Small Humanoid Robot Crawling

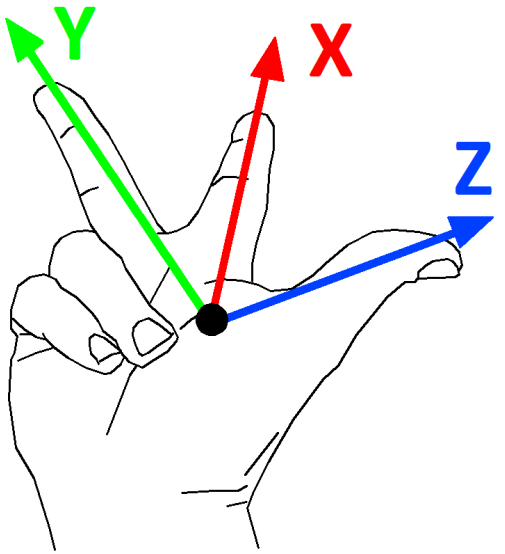
- Have a small humanoid (Hi Torque Mini Pi) climb/ crawl over difficult (RoboCup Rescue) terrain …

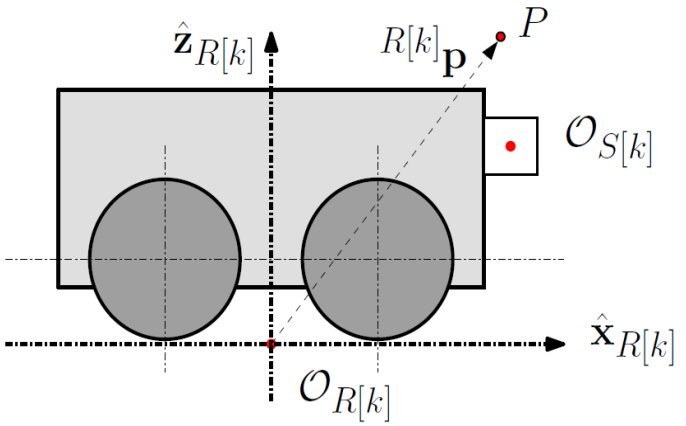
COORDINATE SYSTEM

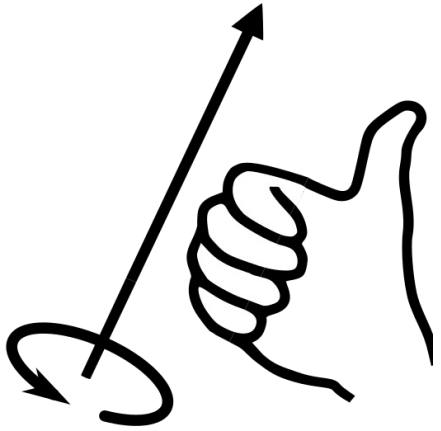
Right Hand Coordinate System

- Standard in Robotics
- Positive rotation around X is anti-clockwise
- Right-hand rule mnemonic:
  - Thumb: z-axis
  - Index finger: x-axis
  - Second finger: y-axis
  - Rotation: Thumb = rotation axis, positive rotation in finger direction
- Robot Coordinate System:
  - X front
  - Z up (Underwater: Z down)
  - Y ???



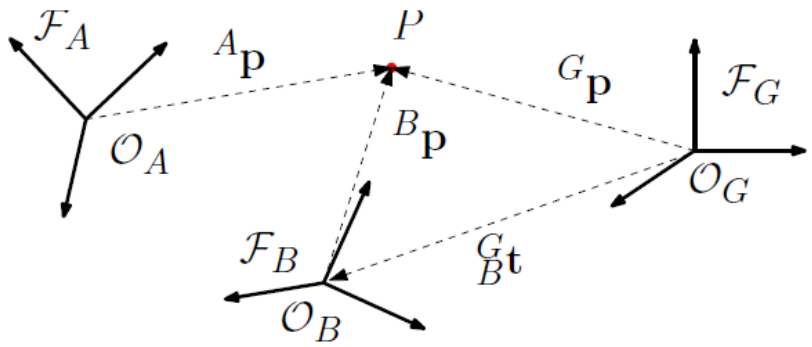






Right Hand Rule http://en.wikipedia.org/wiki/Right-hand_rule

Transform



Notation	Meaning
$\mathcal{F}_{\mathrm{R}[k]}$	Coordinate frame attached to object 'R' (usually the robot) at sample time-instant k.
$\mathcal{O}_{\mathrm{R}[k]}$	Origin of $\mathcal{F}_{\mathrm{R}[k]}$.
${}^{\mathrm{R}[k]}\mathbf{p}$	For any general point P, the position vector $\overrightarrow{\mathcal{O}_{\mathrm{R}[k]}P}$ resolved in $\mathcal{F}_{\mathrm{R}[k]}$.
${}^{\mathrm{H}}\hat{\mathbf{x}}_{\mathrm{R}}$	The x-axis direction of $\mathcal{F}_{\mathrm{R}}$ resolved in $\mathcal{F}_{\mathrm{H}}$. Similarly, ${}^{\mathrm{H}}\hat{\mathbf{y}}_{\mathrm{R}}$, ${}^{\mathrm{H}}\hat{\mathbf{z}}_{\mathrm{R}}$ can be defined. Obviously, ${}^{\mathrm{R}}\hat{\mathbf{x}}_{\mathrm{R}} = \hat{\mathbf{e}}_1$. Time indices can be added to the frames, if necessary.
${}^{\mathrm{R}[k]}_{\mathrm{S}[k']}\mathbf{R}$	The rotation-matrix of $\mathcal{F}_{\mathrm{S}[k']}$ with respect to $\mathcal{F}_{\mathrm{R}[k]}$.
${}^{\mathrm{R}}_{\mathrm{S}}\mathbf{t}$	The translation vector $\overrightarrow{\mathcal{O}_{\mathrm{R}}\mathcal{O}_{\mathrm{S}}}$ resolved in $\mathcal{F}_{\mathrm{R}}$.

Transform between two coordinate frames

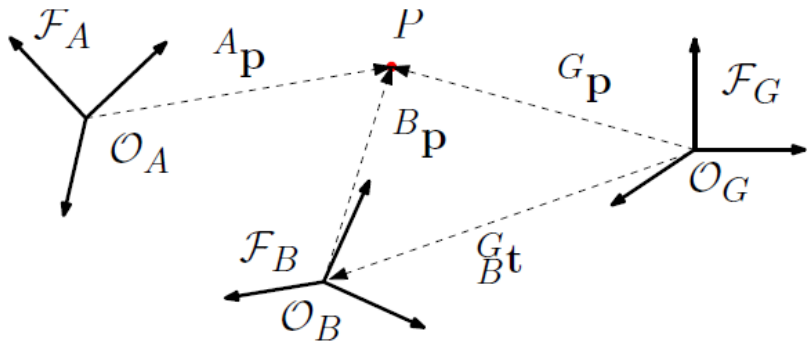
$${}^{\mathrm{G}}_{\mathrm{A}}\mathbf{t} \triangleq \overrightarrow{\mathcal{O}_{\mathrm{G}}\mathcal{O}_{\mathrm{A}}} \text{ resolved in } \mathcal{F}_{\mathrm{G}}$$

$$\begin{pmatrix} {}^{\mathrm{G}}\mathbf{p} \\ 1 \end{pmatrix} \equiv \begin{pmatrix} {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{R} & {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{t} \\ \mathbf{0}_{1\times[2,3]} & 1 \end{pmatrix} \begin{pmatrix} {}^{\mathrm{A}}\mathbf{p} \\ 1 \end{pmatrix} \qquad {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{T} \equiv \left\{ \begin{matrix} {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{t} \\ {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{R} \end{matrix} \right\}$$

$$\begin{aligned} {}^{\mathrm{G}}\mathbf{p} &= {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{R}\ {}^{\mathrm{A}}\mathbf{p} + {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{t} \\ &\triangleq {}^{\mathrm{G}}_{\mathrm{A}}\mathbf{T}({}^{\mathrm{A}}\mathbf{p}). \end{aligned}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & {}^{\mathrm{G}}_{\mathrm{A}}\mathrm{t}_{\mathrm{x}} \\ \sin\theta & \cos\theta & {}^{\mathrm{G}}_{\mathrm{A}}\mathrm{t}_{\mathrm{y}} \\ 0 & 0 & 1 \end{bmatrix}$$

Transform: Operations



Transform between two coordinate frames (chaining, compounding):

$$ {}^G_B\mathbf{T} = {}^G_A\mathbf{T}\,{}^A_B\mathbf{T} \equiv \begin{Bmatrix} {}^G_A\mathbf{R}\,{}^A_B\mathbf{t} + {}^G_A\mathbf{t} \\ {}^G_A\mathbf{R}\,{}^A_B\mathbf{R} \end{Bmatrix} $$

Inverse of a Transform :

$$ {}^B_A\mathbf{T} = {}^A_B\mathbf{T}^{-1} \equiv \begin{Bmatrix} -{}^A_B\mathbf{R}^\mathsf{T}\,{}^A_B\mathbf{t} \\ {}^A_B\mathbf{R}^\mathsf{T} \end{Bmatrix} $$

Relative (Difference) Transform :

$$ {}^B_A\mathbf{T} = {}^G_B\mathbf{T}^{-1}\,{}^G_A\mathbf{T} $$

See: **Quick Reference to Geometric Transforms in Robotics** by Kaustubh Pathak on the webpage!

Chaining :

$$ {}^{G}_{R[X+1]}\mathbf{T} = {}^{G}_{R[X]}\mathbf{T}\ {}^{R[X]}_{R[X+1]}\mathbf{T} \equiv \begin{Bmatrix} {}^{G}_{R[X]}\mathrm{R} & {}^{R[X]}_{R[X+1]}t + {}^{G}_{R[X]}t \\ {}^{G}_{R[X]}\mathrm{R} & {}^{R[X]}_{R[X+1]}\mathrm{R} \end{Bmatrix} = \begin{Bmatrix} {}^{G}_{R[X+1]}t \\ {}^{G}_{R[X+1]}\mathrm{R} \end{Bmatrix} $$

In 2D Translation:

$$ \begin{bmatrix} {}^{G}_{R[X+1]}t_x \\ {}^{G}_{R[X+1]}t_y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos {}^{G}_{R[X]}\theta & -\sin {}^{G}_{R[X]}\theta & {}^{G}_{R[X]}\mathrm{t_x} \\ \sin {}^{G}_{R[X]}\theta & \cos {}^{G}_{R[X]}\theta & {}^{G}_{R[X]}\mathrm{t_y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{R[X]}_{R[X+1]}\mathrm{t_x} \\ {}^{R[X]}_{R[X+1]}\mathrm{t}_y \\ 1 \end{bmatrix} $$

In 2D Rotation:

$$ {}^{G}_{R[X+1]}\mathrm{R} = \begin{bmatrix} \cos {}^{G}_{R[X+1]}\theta & -\sin {}^{G}_{R[X+1]}\theta \\ \sin {}^{G}_{R[X+1]}\theta & \cos {}^{G}_{R[X+1]}\theta \end{bmatrix} = \begin{bmatrix} \cos {}^{G}_{R[X]}\theta & -\sin {}^{G}_{R[X]}\theta \\ \sin {}^{G}_{R[X]}\theta & \cos {}^{G}_{R[X]}\theta \end{bmatrix} \begin{bmatrix} \cos {}^{R[X]}_{R[X+1]}\theta & -\sin {}^{R[X]}_{R[X+1]}\theta \\ \sin {}^{R[X]}_{R[X+1]}\theta & \cos {}^{R[X]}_{R[X+1]}\theta \end{bmatrix} $$

In 2D Rotation (simple): $ {}^{G}_{R[X+1]}\theta = {}^{G}_{R[X]}\theta + {}^{R[X]}_{R[X+1]}\theta $

In ROS: nav_2d_msgs/Pose2DStamped

- First Message at time 97 : G
- Message at time 103 : X
- Next Message at time 107 : X+1

$$ {}^{G}_{R[X+1]}\mathbf{T} = {}^{G}_{R[X]}\mathbf{T}\ {}^{R[X]}_{R[X+1]}\mathbf{T} $$

$$ {}^{R[X]}_{R[X+1]}t_x $$

$$ {}^{R[X]}_{R[X+1]}t_y $$

$$ {}^{R[X]}_{R[X+1]}\Theta $$

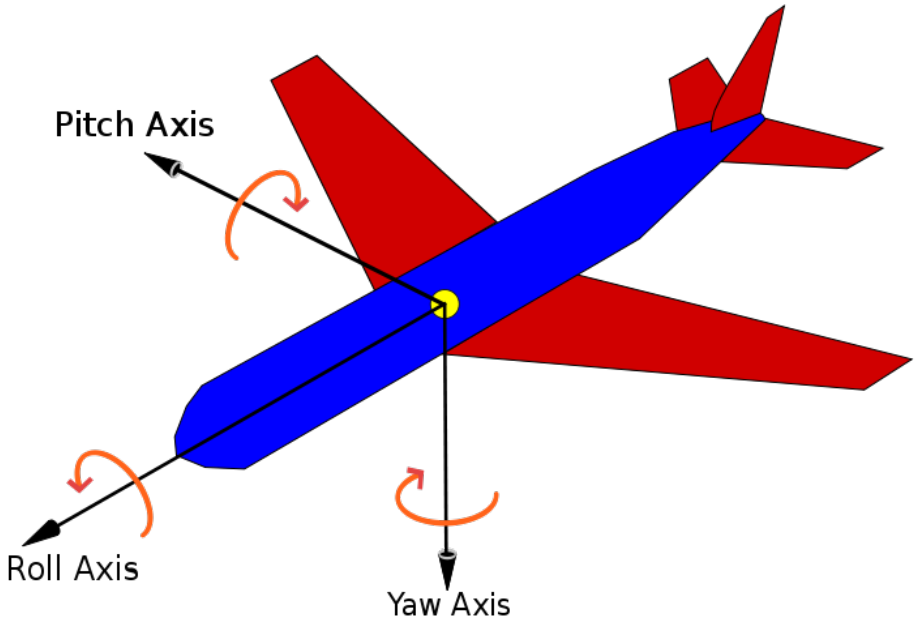
```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose2D pose2D
  float64 x
  float64 y
  float64 theta
```

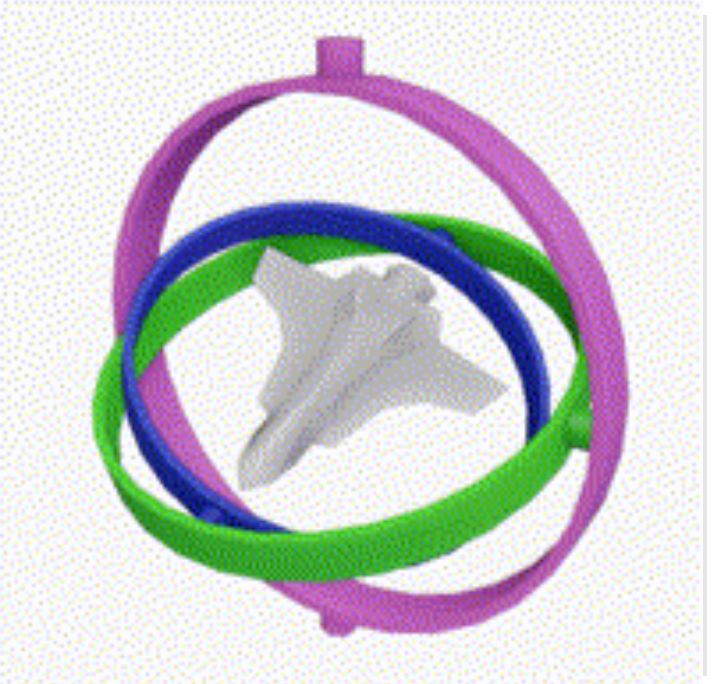
3D Rotation

- Many 3D rotation representations:

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

- Euler angles:
  - Roll: rotation around x-axis
  - Pitch: rotation around y-axis
  - Yaw: rotation around z-axis
  - Apply rotations one after the other…
    - => Order important! E.g.:
      - x-z-x; x-y-z; z-y-x; …
  - ☹ Singularities
  - Gimbal lock in Engineering
    - "a condition caused by the collinear alignment of two or more robot axes resulting in unpredictable robot motion and velocities"





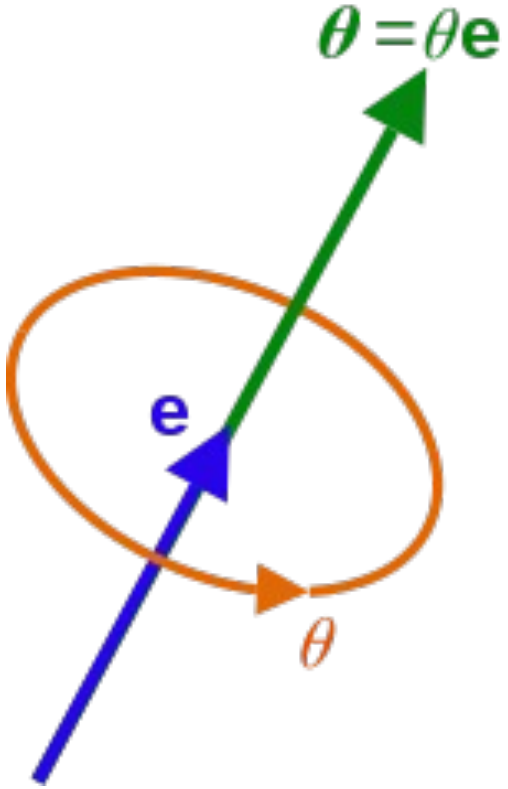
3D Rotation

- Axis Angle
  - Angle θ and
  - Axis unit vector **e** (3D vector with length 1)
    - Can be represented with 2 numbers (e.g. elevation and azimuth angles)

- Euler Angles: sequence of 3 rotations around coordinate axes

  equivalent to:

- Axis Angle: pure rotation around a single fixed axis



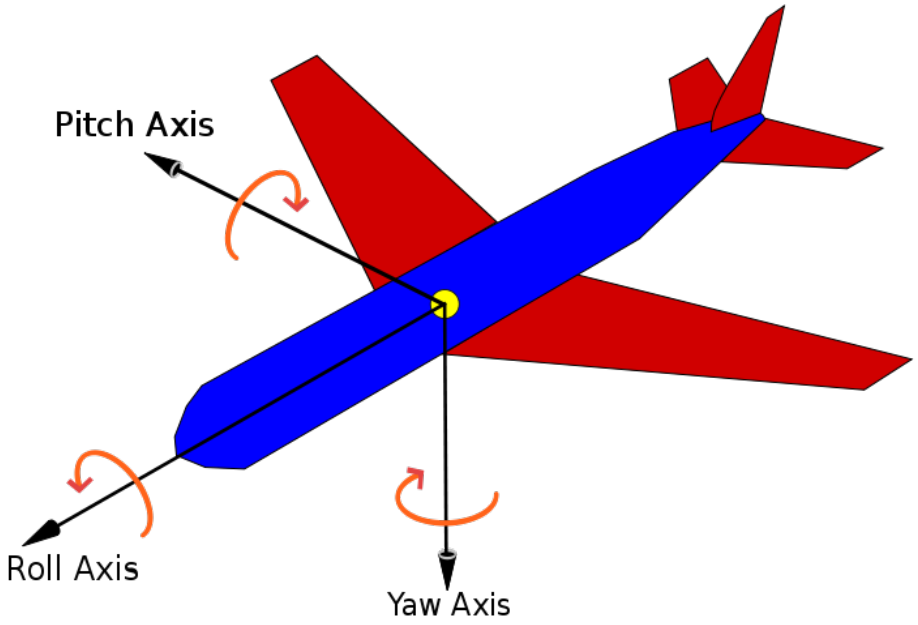
3D Rotation

- Quaternions:
  - Concatenating rotations is computationally faster and numerically more stable
  - Extracting the angle and axis of rotation is simpler
  - Interpolation is more straightforward
  - Unit Quaternion: norm = 1
    - Versor: https://en.wikipedia.org/wiki/Versor

https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation

  - Scalar (real) part: q_0 , sometimes q_w
  - Vector (imaginary) part: q
  - Over determined: 4 variables for 3 DoF (but: unit!)

- Check out: https://eater.net/quaternions ! Excellent interactive video…



$$\check{\mathbf{p}} \equiv p_0 + p_x\mathbf{i} + p_y\mathbf{j} + p_z\mathbf{k}$$

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

$$\check{\mathbf{q}} = \begin{pmatrix} q_0 & q_x & q_y & q_z \end{pmatrix}^\mathsf{T} \equiv \begin{pmatrix} q_0 \\ \mathbf{q} \end{pmatrix}$$

Transform in 3D

Matrix Euler Quaternion

$$ {}^{G}_{A}\mathbf{T} = \begin{bmatrix} {}^{G}_{A}\mathrm{R} & {}^{G}_{A}\mathrm{t} \\ 0_{1x3} & 1 \end{bmatrix} = \begin{pmatrix} {}^{G}_{A}\mathrm{t} \\ {}^{G}_{A}\Theta \end{pmatrix} = \begin{pmatrix} {}^{G}_{A}\mathrm{t} \\ {}^{G}_{A}\check{\mathrm{q}} \end{pmatrix} $$

$$ {}^{G}_{A}\Theta \triangleq (\theta_r, \theta_p, \theta_y)^T $$

In ROS: Quaternions! (w, x, y, z)
Uses Eigen library for Transforms

Rotation Matrix 3x3

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

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

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

$$ R = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma) $$

yaw = α, pitch = β, roll = γ

Eigen

- Don’t have to deal with the details of transforms too much ☺

- Conversions between ROS and Eigen:

http://docs.ros.org/noetic/api/eigen_conversions/html/namespacetf.html

```
Matrix3f m;
m = AngleAxisf(angle1, Vector3f::UnitZ())
    * AngleAxisf(angle2, Vector3f::UnitY())
    * AngleAxisf(angle3, Vector3f::UnitZ());
```

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

class	**Eigen::AlignedBox** An axis aligned box. More...
class	**Eigen::AngleAxis** Represents a 3D rotation as a rotation angle around an arbitrary 3D axis. More...
class	**Eigen::Homogeneous** Expression of one (or a set of) homogeneous vector(s) More...
class	**Eigen::Hyperplane** A hyperplane. More...
class	**Eigen::Map< const Quaternion< _Scalar >, _Options >** **Quaternion** expression mapping a constant memory buffer. More...
class	**Eigen::Map< Quaternion< _Scalar >, _Options >** Expression of a quaternion from a memory buffer. More...
class	**Eigen::ParametrizedLine** A parametrized line. More...
class	**Eigen::Quaternion** The quaternion class used to represent 3D orientations and rotations. More...
class	**Eigen::QuaternionBase** Base class for quaternion expressions. More...
class	**Eigen::Rotation2D** Represents a rotation/orientation in a 2 dimensional space. More...
class	**Scaling** Represents a generic uniform scaling transformation. More...
class	**Eigen::Transform** Represents an homogeneous transformation in a N dimensional space. More...
class	**Eigen::Translation** Represents a translation transformation. More...

Examples of Transforms

- Transform between global coordinate frame and robot frame at time X
- Transform between robot frame at time X and robot frame at time X+1

- Transform between robot camera frame and robot base frame (mounted fixed – not dependend on time! => static transform)
- Transform between map origin and door pose in map (not time dependend)

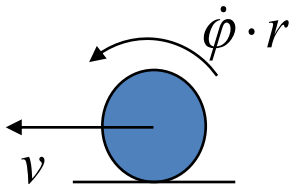
- Transform between robot camera frame and fingers (end-effector) of a robot arm at time X
- Transform between robot camera frame and map frame at time X
- Transform between robot 1 camera at time X and robot 2 camera at time X+n

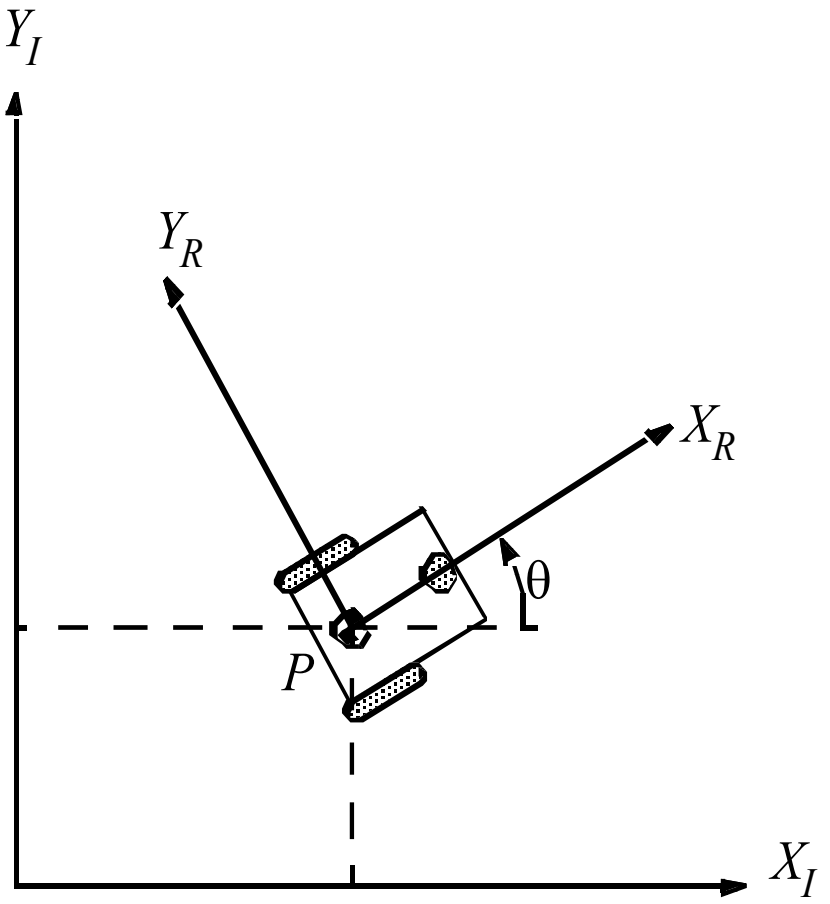
ROS Standards:

- Standard Units of Measure and Coordinate Conventions
  - http://www.ros.org/reps/rep-0103.html
- Coordinate Frames for Mobile Platforms:
  - http://www.ros.org/reps/rep-0105.html

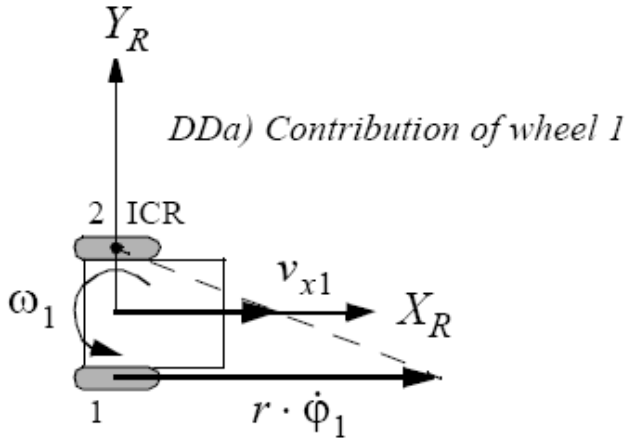
Wheel Kinematic Constraints: Assumptions

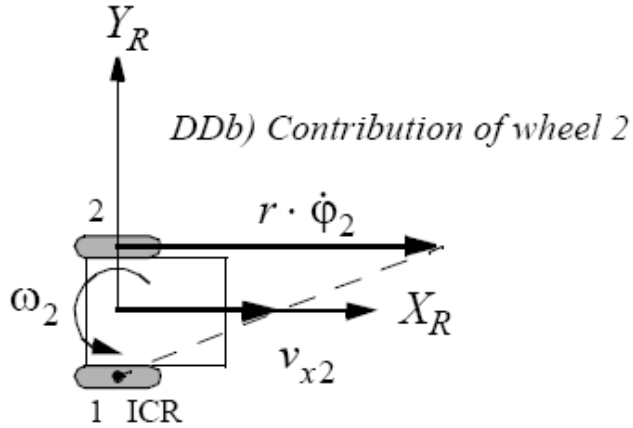
- Movement on a horizontal plane
- Point contact of the wheels
- Wheels not deformable
- Pure rolling
  - $v_c = 0$ at contact point
- No slipping, skidding or sliding
- No friction for rotation around contact point
- Steering axes orthogonal to the surface
- Wheels connected by rigid frame (chassis)





Forward Kinematic Model: Geometric Approach





Differential-Drive:

DDa) $v_{x1} = \frac{1}{2} r\dot{\varphi}_1 \;;\; v_{y1} = 0 \;;\; \omega_1 = \frac{1}{2l} r\dot{\varphi}_1$

DDb) $v_{x2} = \frac{1}{2} r\dot{\varphi}_2 \;;\; v_{y2} = 0 \;;\; \omega_2 = -\frac{1}{2l} r\dot{\varphi}_2$

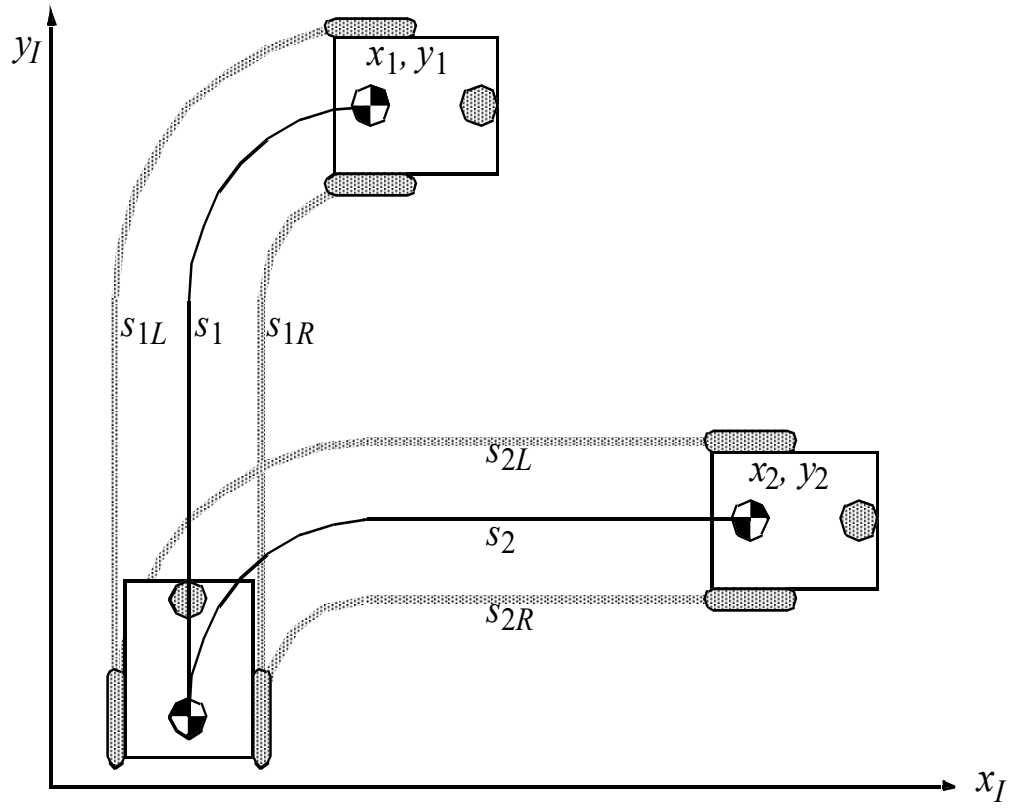
$$\rightarrow \dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_I = R(\theta)^{-1} \begin{bmatrix} v_{x1} + v_{x2} \\ v_{y1} + v_{y2} \\ \omega_1 + \omega_2 \end{bmatrix} = R(\theta)^{-1} \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ \frac{r}{2l} & -\frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \end{bmatrix}$$

Inverse of R => Active and Passive Transform:
http://en.wikipedia.org/wiki/Active_and_passive_transformation

Mobile Robot Kinematics: Non-Holonomic Systems

$s_1 = s_2$; $s_{1R} = s_{2R}$; $s_{1L} = s_{2L}$

but: $x_1 \neq x_2$; $y_1 \neq y_2$



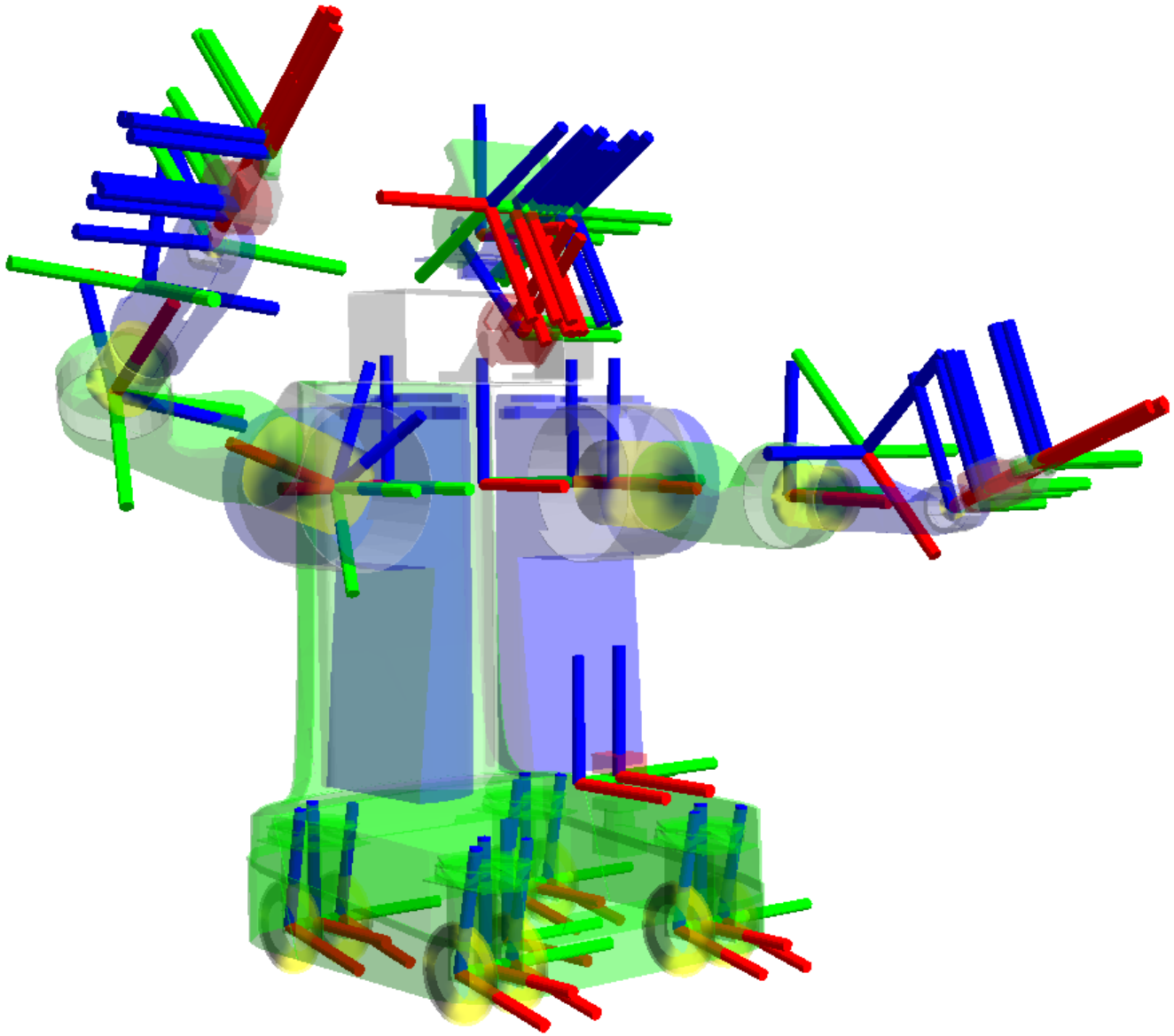
- Non-holonomic systems
  - differential equations are not integrable to the final pose.
  - the measure of the traveled distance of each wheel is not sufficient to calculate the final position of the robot. One has also to know how this movement was executed as a function of time.

Holonomic examples

Uranus, CMU

ROS: 3D Transforms : TF

- http://wiki.ros.org/tf
- http://wiki.ros.org/tf/Tutorials



ROS geometry_msgs/TransformStamped

- $\mathbf{T}^{\text{header.frame\_id[header.stamp]}}_{\text{child\_frame\_id[header.stamp]}}$

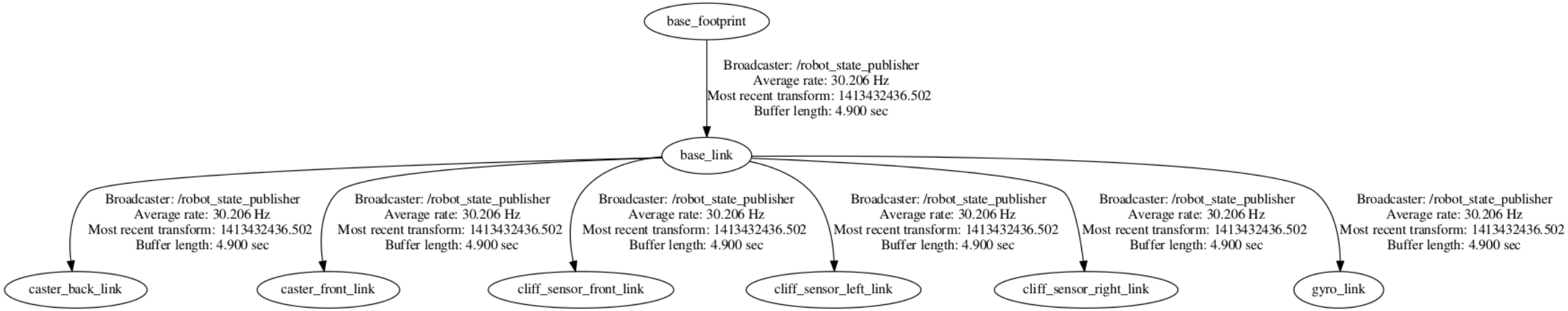
- Transform between header (time and reference frame) and child_frame
- 3D Transform representation:
  - geometry_msgs/Transform:
    - Vector3 for translation (position)
    - Quaternion for rotation (orientation)

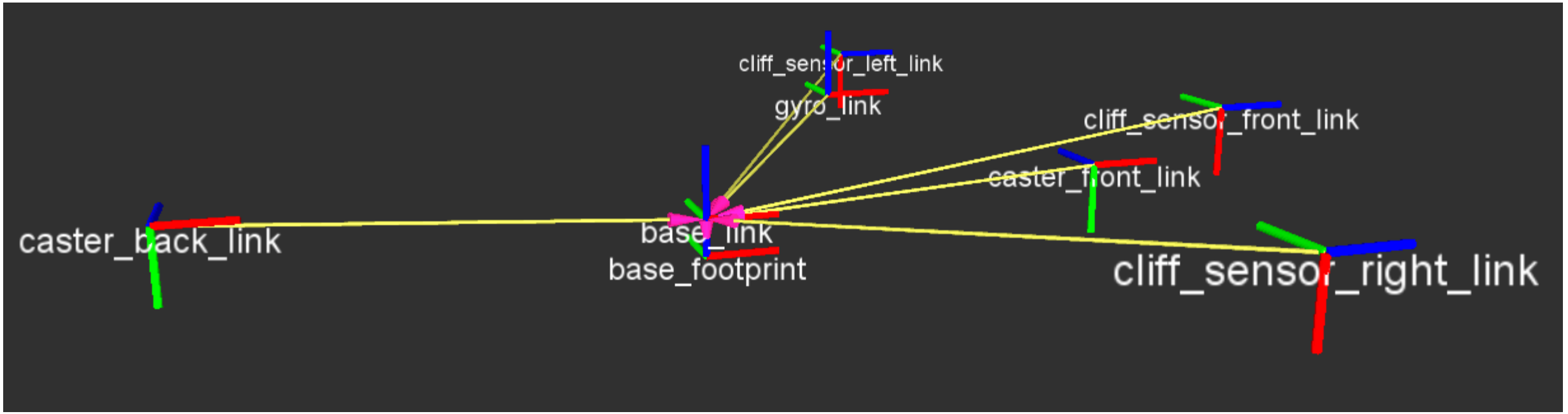
```
rosmsg show geometry_msgs/TransformStamped

std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
string child_frame_id
geometry_msgs/Transform transform
  geometry_msgs/Vector3 translation
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion rotation
    float64 x
    float64 y
    float64 z
    float64 w
```

ROS tf2_msgs/TFMessage

- An array of TransformStamped
- Transforms form a tree
- Transform listener: traverse the tree
  - tf::TransformListener listener;
- Get transform:
  - tf::StampedTransform transform;
  - listener.lookupTransform("/base_link", "/camera1", ros::Time(0), transform);
  - ros::Time(0): get the latest transform
  - Will calculate transform by chaining intermediate transforms, if needed

```
rosmsg show tf2_msgs/TFMessage

geometry_msgs/TransformStamped[] transforms
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  string child_frame_id
  geometry_msgs/Transform transform
    geometry_msgs/Vector3 translation
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion rotation
      float64 x
      float64 y
      float64 z
      float64 w
```


base_footprint
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
base_link
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
caster_back_link
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
caster_front_link
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
cliff_sensor_front_link
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
cliff_sensor_left_link
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
cliff_sensor_right_link
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
gyro_link


cliff_sensor_left_link
gyro_link
cliff_sensor_front_link
caster_front_link
caster_back_link
base_link
base_footprint
cliff_sensor_right_link

Transforms in ROS

- Imagine: Object recognition took 3 seconds – it found an object with:
  - tf::Transform object_transform_camera; // ${}^{\mathrm{Cam}[X]}_{Obj}\mathbf{T}$ (has tf::Vector3 and tf::Quaternion)
  - and header with: ros::Time stamp; // Timestamp of the camera image (== X)
    - and std::string frame_id; // Name of the frame (“Cam”)
- Where is the object in the global frame (= odom frame) “odom” ${}^{\mathrm{G}}_{Obj}\mathbf{T}$?
  - tf::StampedTransform object_transform_global; // the resulting frame
  - listener.lookupTransform(child_frame_id, “/odom", header.stamp, object_transform_global);
- tf::TransformListener keeps a history of transforms – by default 10 seconds