上海科技大学
ShanghaiTech University

CS283: **Robotics Spring 2025: Sensors**

Sören Schwertfeger

ShanghaiTech University

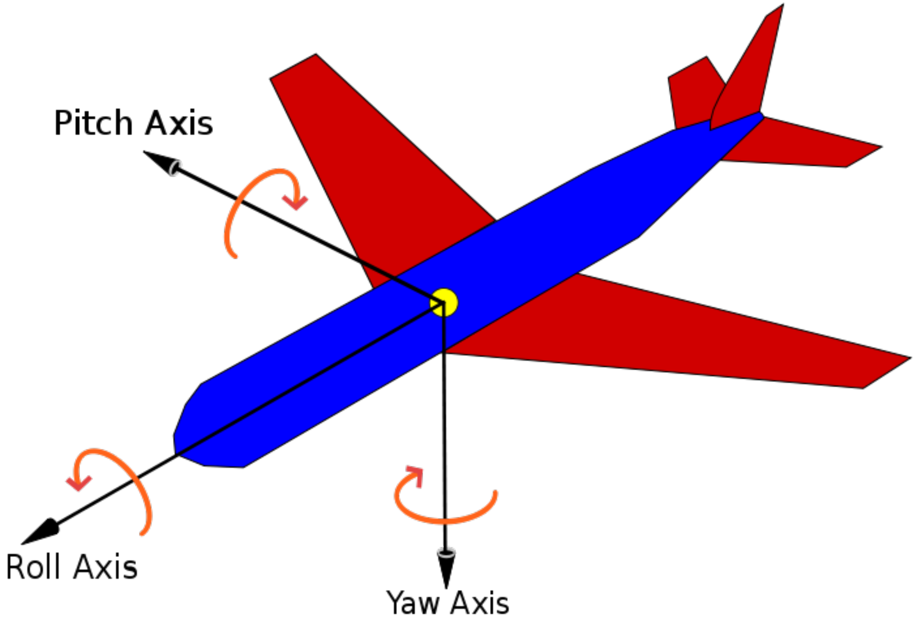
KINEMATICS CONTINUED

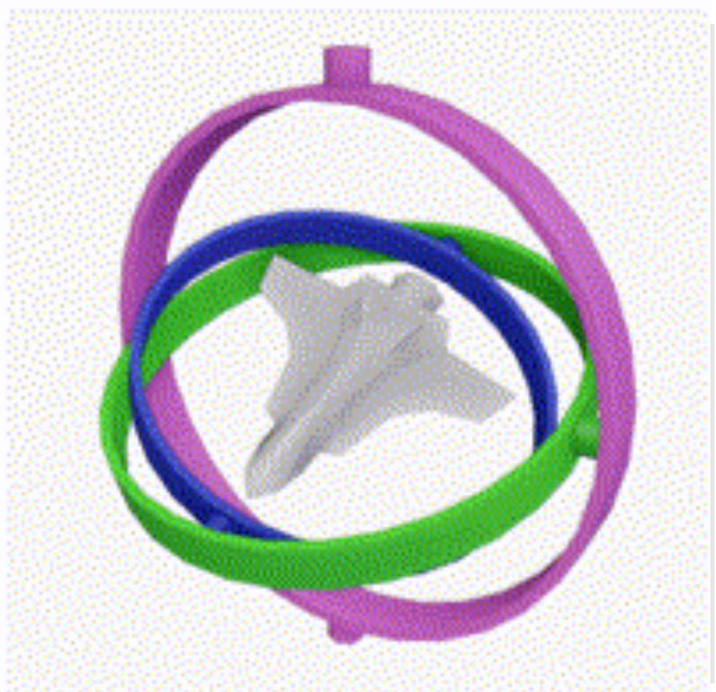
3D Rotation

- 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"





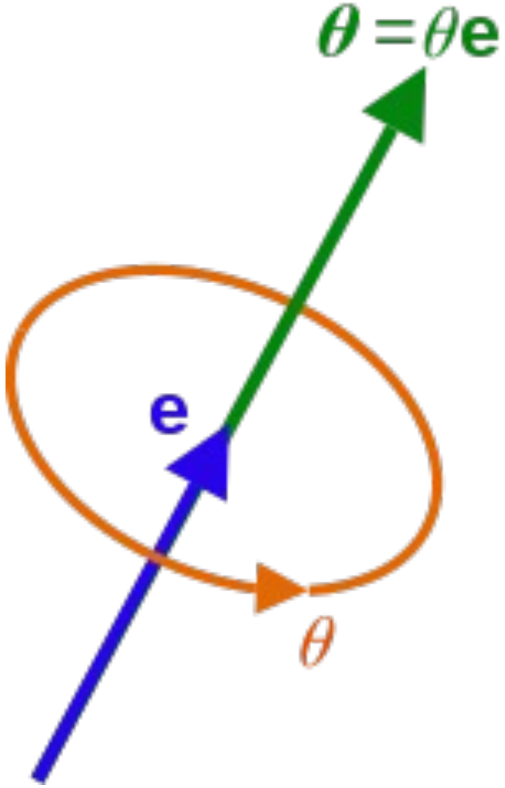
3D Rotation

- 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

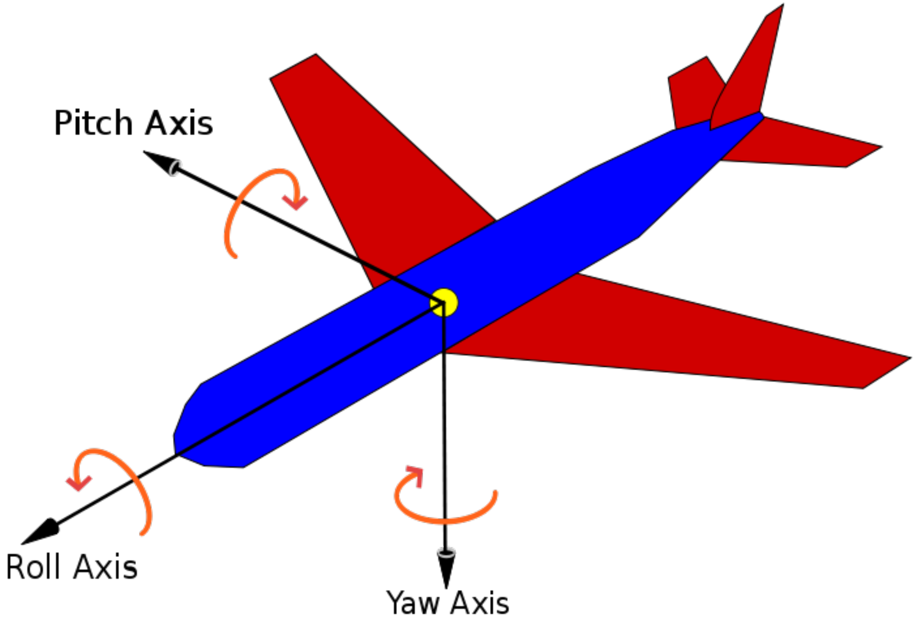


3D Rotation

- 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!)



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

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

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

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

Transform in 3D

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

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

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

Rotation Matrix 3x3

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

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

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

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

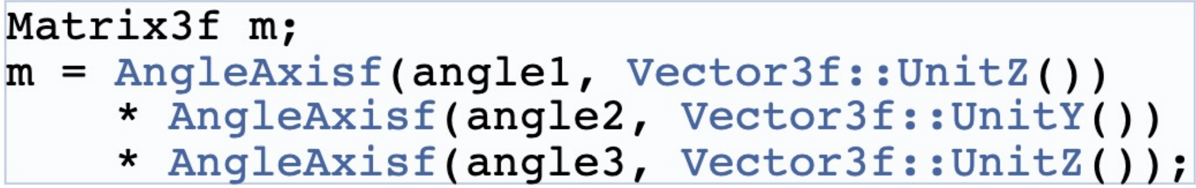
yaw = α, pitch = β, roll = γ

Eigen

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

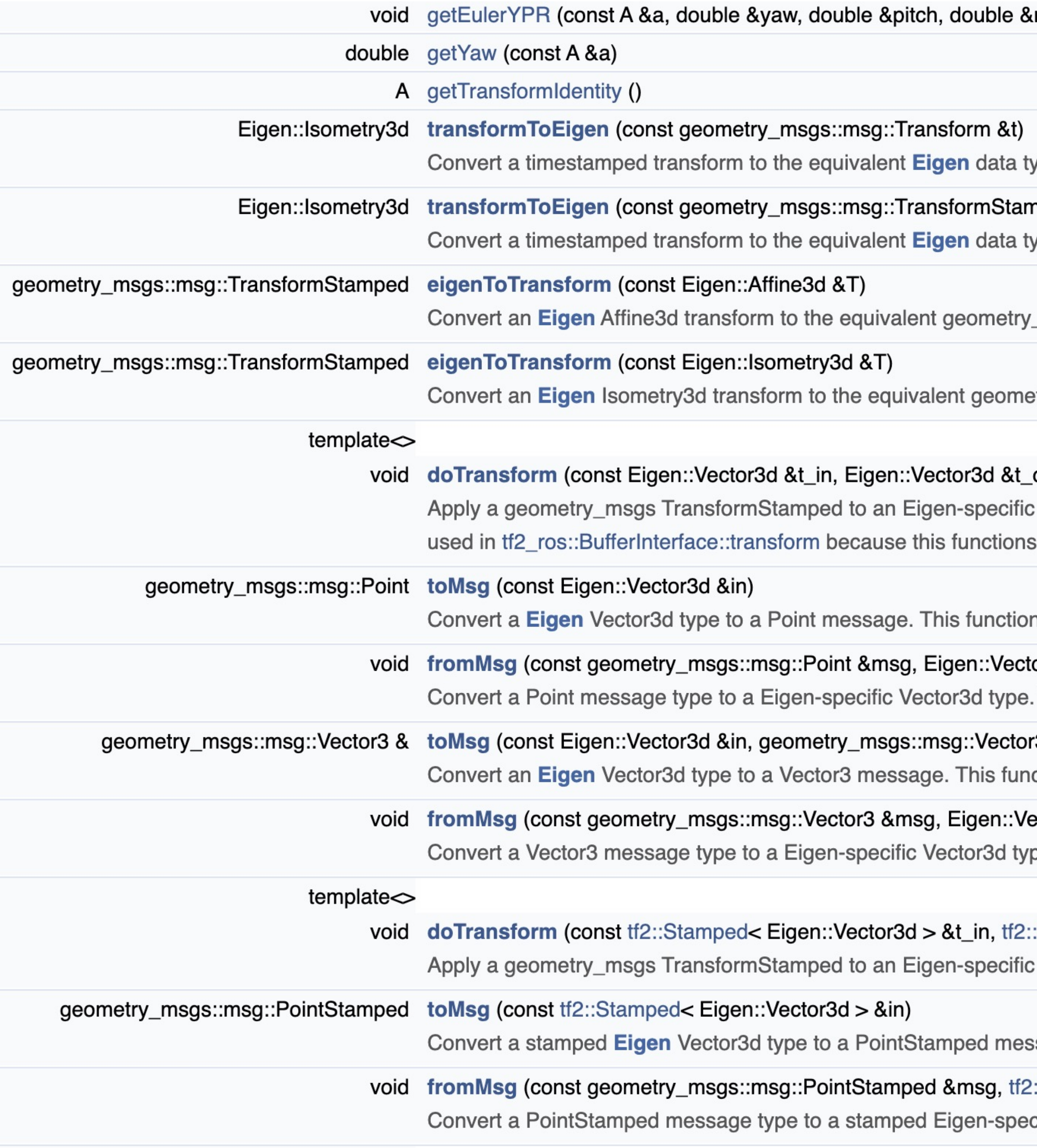
- Conversions between ROS and Eigen:

https://docs.ros2.org/foxy/api/tf2_eigen/namespaces.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



void	getEulerYPR (const A &a, double &yaw, double &pitch, double &
double	getYaw (const A &a)
A	getTransformIdentity ()
Eigen::Isometry3d	**transformToEigen** (const geometry_msgs::msg::Transform &t) Convert a timestamped transform to the equivalent **Eigen** data ty
Eigen::Isometry3d	**transformToEigen** (const geometry_msgs::msg::TransformStam Convert a timestamped transform to the equivalent **Eigen** data ty
geometry_msgs::msg::TransformStamped	**eigenToTransform** (const Eigen::Affine3d &T) Convert an **Eigen** Affine3d transform to the equivalent geometry_
geometry_msgs::msg::TransformStamped	**eigenToTransform** (const Eigen::Isometry3d &T) Convert an **Eigen** Isometry3d transform to the equivalent geome
template<> void	**doTransform** (const Eigen::Vector3d &t_in, Eigen::Vector3d &t_ Apply a geometry_msgs TransformStamped to an Eigen-specific used in tf2_ros::BufferInterface::transform because this functions
geometry_msgs::msg::Point	**toMsg** (const Eigen::Vector3d &in) Convert a **Eigen** Vector3d type to a Point message. This function
void	**fromMsg** (const geometry_msgs::msg::Point &msg, Eigen::Vecto Convert a Point message type to a Eigen-specific Vector3d type.
geometry_msgs::msg::Vector3 &	**toMsg** (const Eigen::Vector3d &in, geometry_msgs::msg::Vector Convert an **Eigen** Vector3d type to a Vector3 message. This func
void	**fromMsg** (const geometry_msgs::msg::Vector3 &msg, Eigen::Ve Convert a Vector3 message type to a Eigen-specific Vector3d typ
template<> void	**doTransform** (const tf2::Stamped< Eigen::Vector3d > &t_in, tf2:: Apply a geometry_msgs TransformStamped to an Eigen-specific
geometry_msgs::msg::PointStamped	**toMsg** (const tf2::Stamped< Eigen::Vector3d > &in) Convert a stamped **Eigen** Vector3d type to a PointStamped mes
void	**fromMsg** (const geometry_msgs::msg::PointStamped &msg, tf2: Convert a PointStamped message type to a stamped Eigen-spec

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

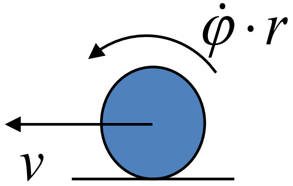
ROS Standards:

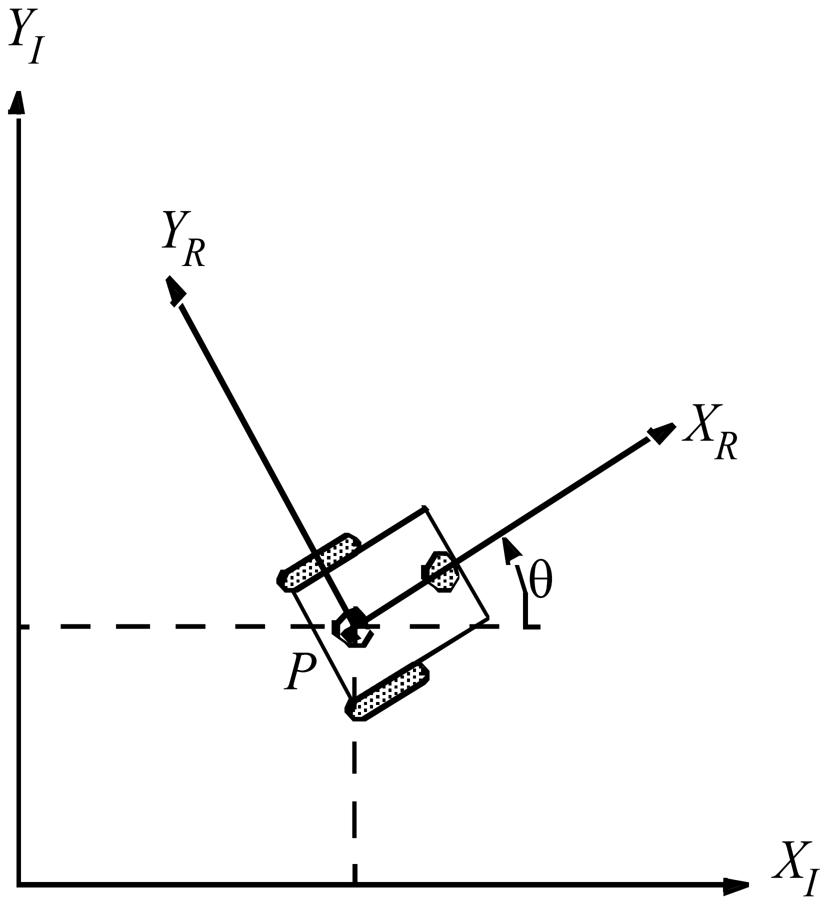
- Standard Units of Measure and Coordinate Conventions
  - http://www.ros.org/reps/rep-0103.html

- Coordinate Frames for Mobile Platforms:
  - http://www.ros.org/reps/rep-0105.html

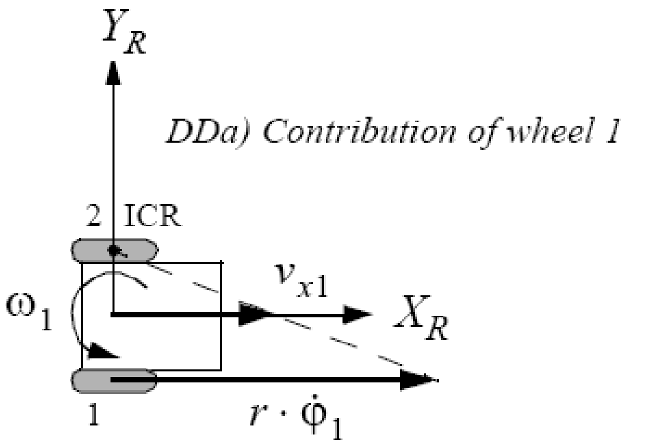
Wheel Kinematic Constraints: Assumptions

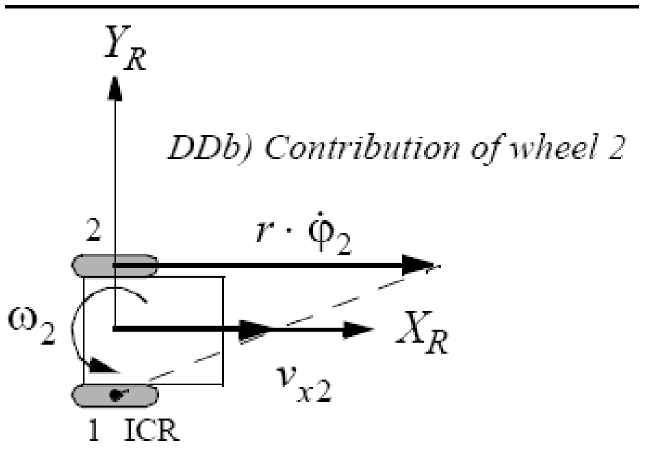
- Movement on a horizontal plane
- Point contact of the wheels
- Wheels not deformable
- Pure rolling
  - $v_c = 0$ at contact point
- No slipping, skidding or sliding
- No friction for rotation around contact point
- Steering axes orthogonal to the surface
- Wheels connected by rigid frame (chassis)





Forward Kinematic Model: Geometric Approach





Differential-Drive:

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

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

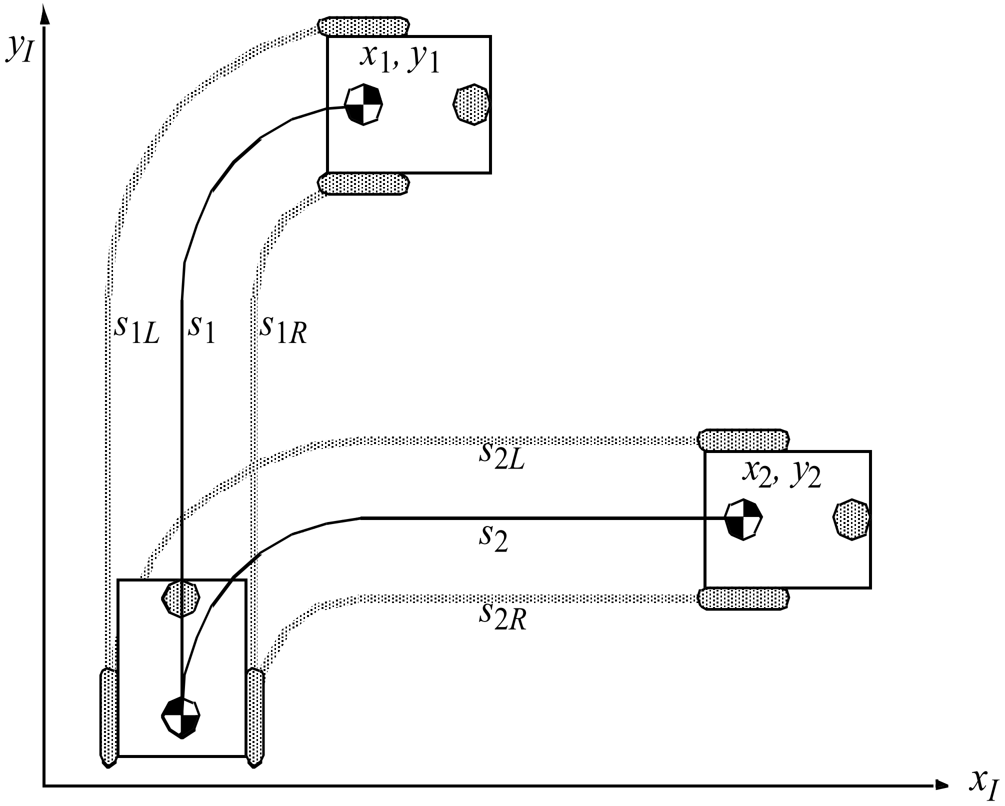
-> $$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_I = R(\theta)^{-1} \begin{bmatrix} v_{x1} + v_{x2} \\ v_{y1} + v_{y2} \\ \omega_1 + \omega_2 \end{bmatrix} = R(\theta)^{-1} \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ \frac{r}{2l} & -\frac{r}{2l} \end{bmatrix}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

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



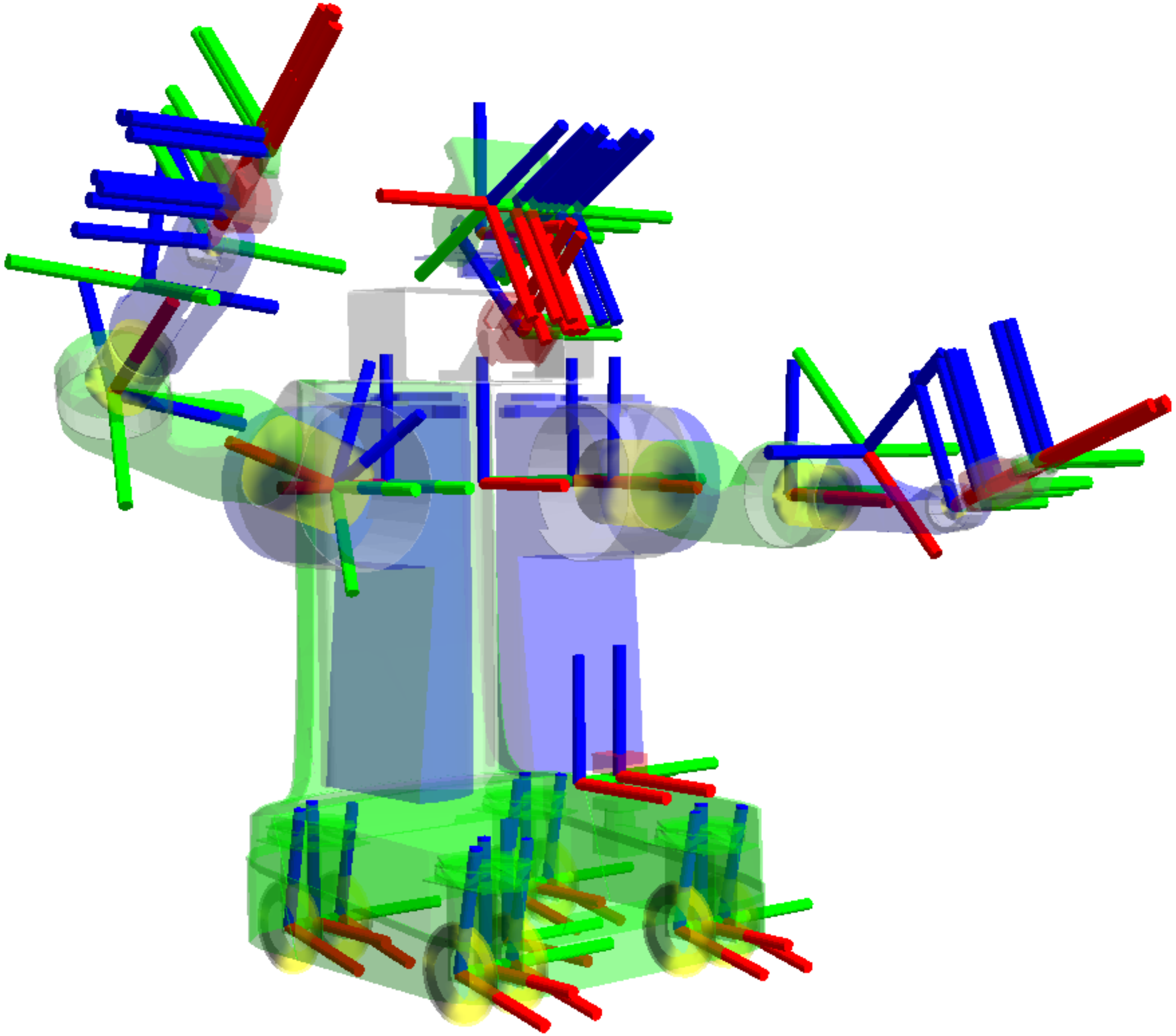
- Non-holonomic systems
  - differential equations are not integrable to the final pose.
  - the measure of the traveled distance of each wheel is not sufficient to calculate the final position of the robot. One has also to know how this movement was executed as a function of time.

Holonomic examples

Uranus, CMU

ROS: 3D Transforms : TF

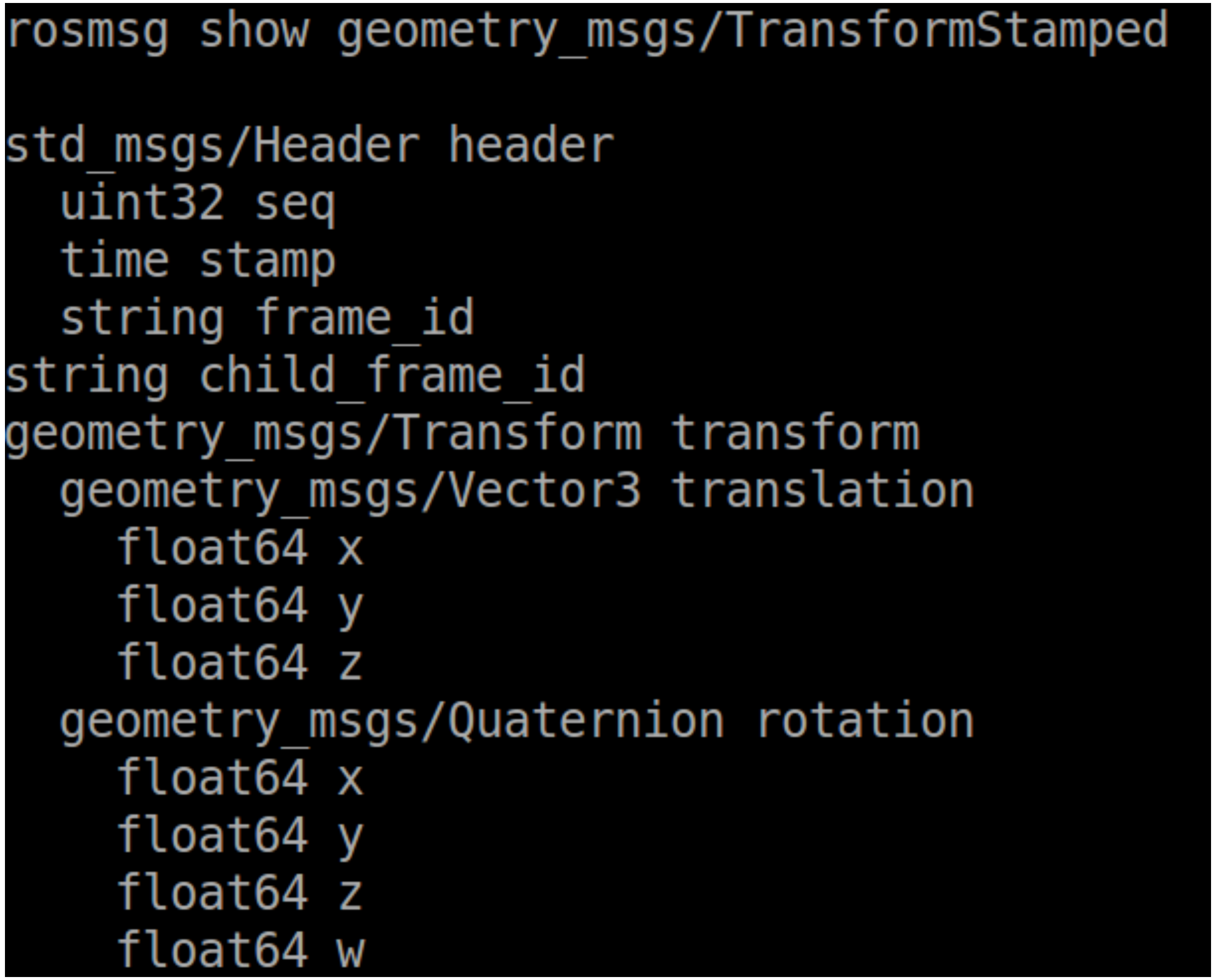
- http://wiki.ros.org/tf
- http://wiki.ros.org/tf/Tutorials



ROS geometry_msgs/TransformStamped

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

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

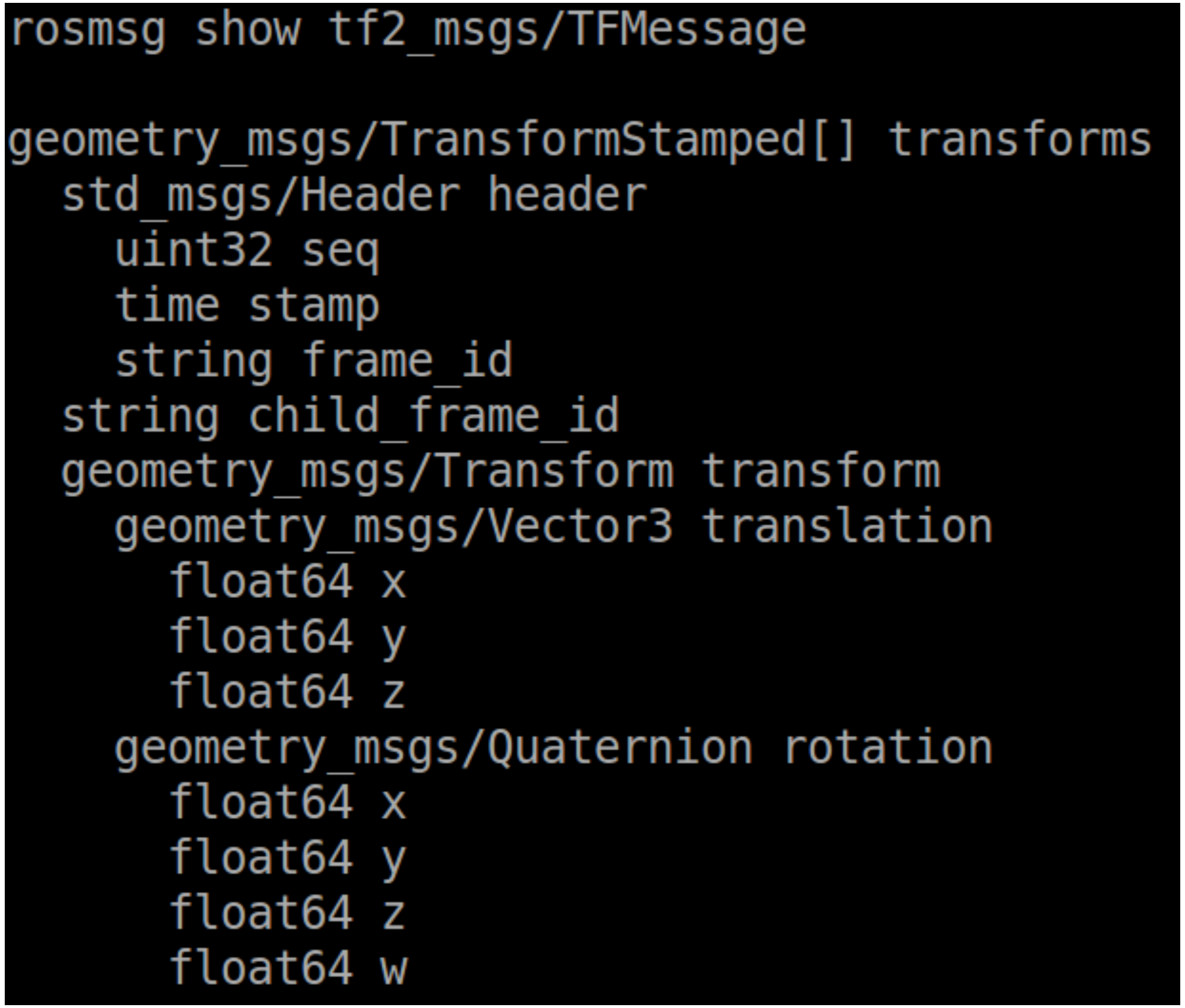


```
rosmsg show geometry_msgs/TransformStamped

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
```

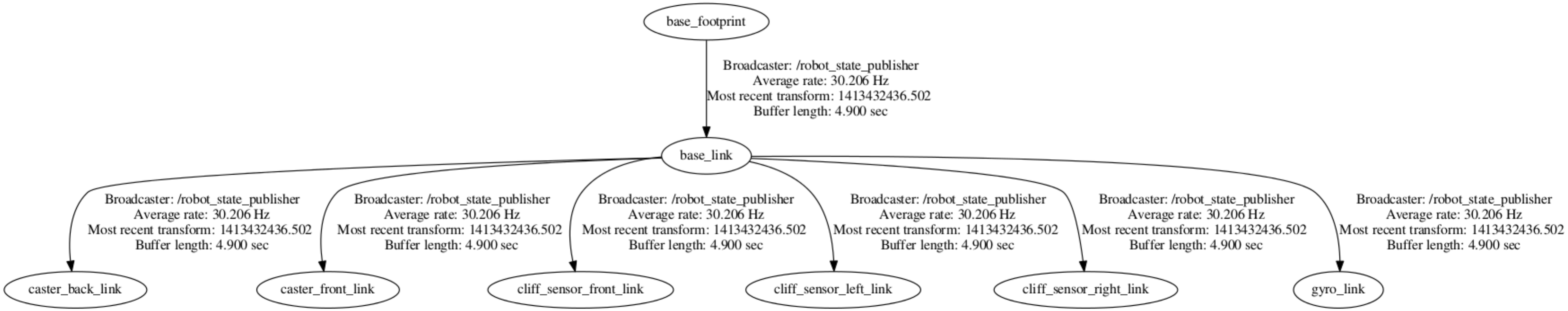
ROS tf2_msgs/TFMessage

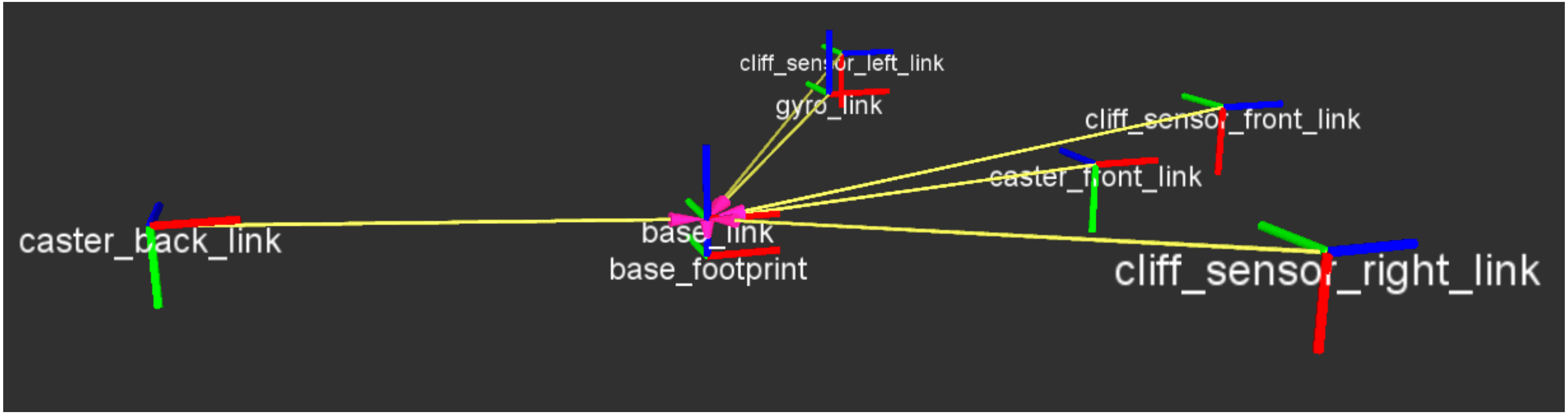
- An array of TransformStamped
- Transforms form a tree
- Transform listener: traverse the tree
  - tf::TransformListener listener;
- Get transform:
  - tf::StampedTransform transform;
  - listener.lookupTransform("/base_link", "/camera1", ros::Time(0), transform);
  - ros::Time(0): get the latest transform
  - 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
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
Broadcaster: /robot_state_publisher
Average rate: 30.206 Hz
Most recent transform: 1413432436.502
Buffer length: 4.900 sec
caster_back_link
caster_front_link
cliff_sensor_front_link
cliff_sensor_left_link
cliff_sensor_right_link
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

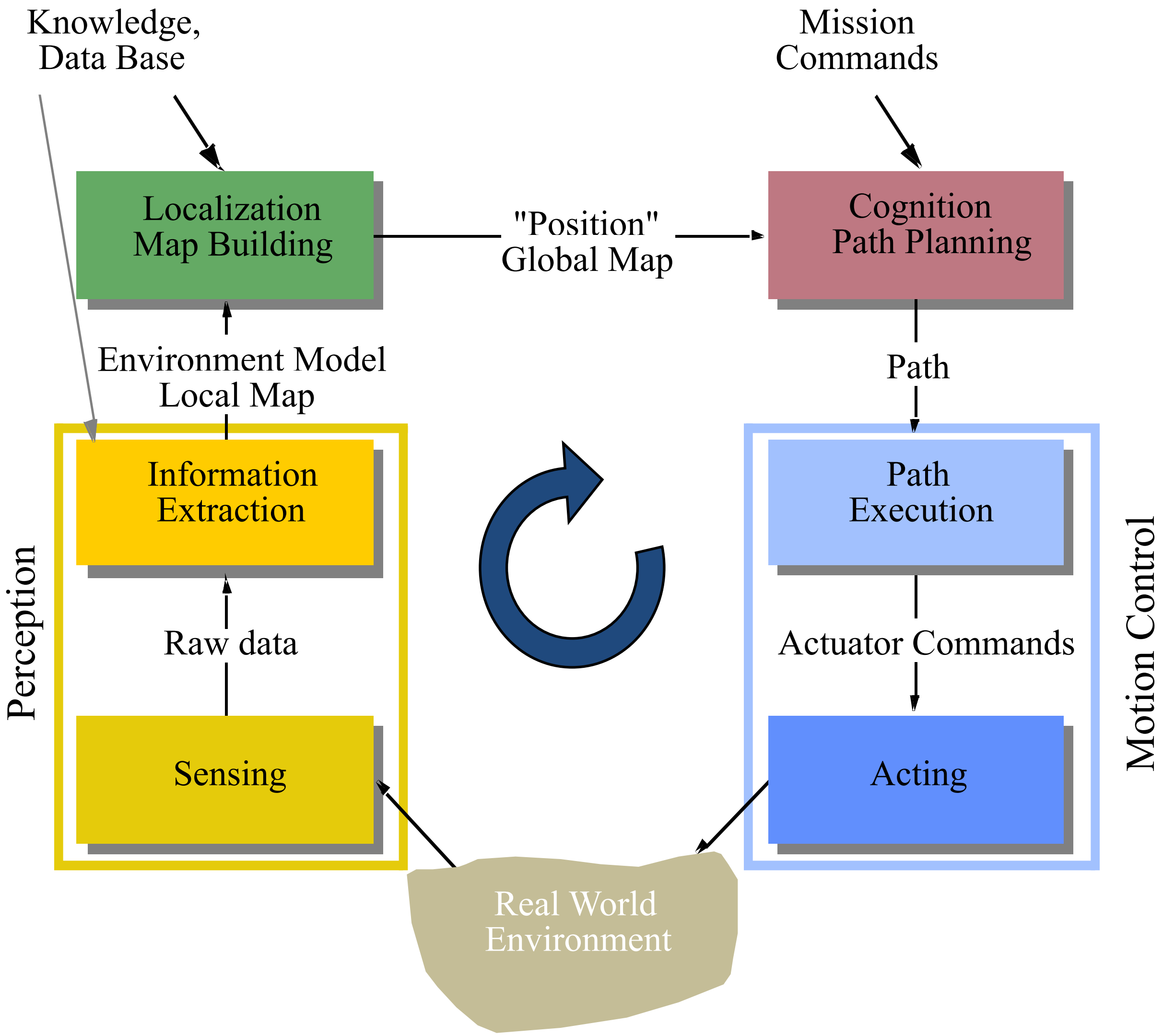
- Imagine: Object recognition took 3 seconds – it found an object with:
  - tf::Transform object_transform_camera; // ${}^{\mathrm{Cam}[X]}_{Obj}\mathbf{T}$ (has tf::Vector3 and tf::Quaternion)
  - and header with: ros::Time stamp; // Timestamp of the camera image (== X)
    - and std::string frame_id; // 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);

- TransformListener keeps a history of transforms – by default 10 seconds

HIGH-LEVEL CONTROL SCHEMES

General Control Scheme for Mobile Robot Systems



SENSORS

Introduction to Autonomous Mobile Robots page 102 ff

Sensors for Mobile Robots

- Why should a robotics engineer know about sensors?
  - Is the **key technology** for perceiving the environment
  - **Understanding the physical principle** enables appropriate use

- Understanding the physical principle behind sensors enables us:
  - To **properly select** the sensors for a given application
  - To **properly model** the sensor system, e.g. resolution, bandwidth, **uncertainties**

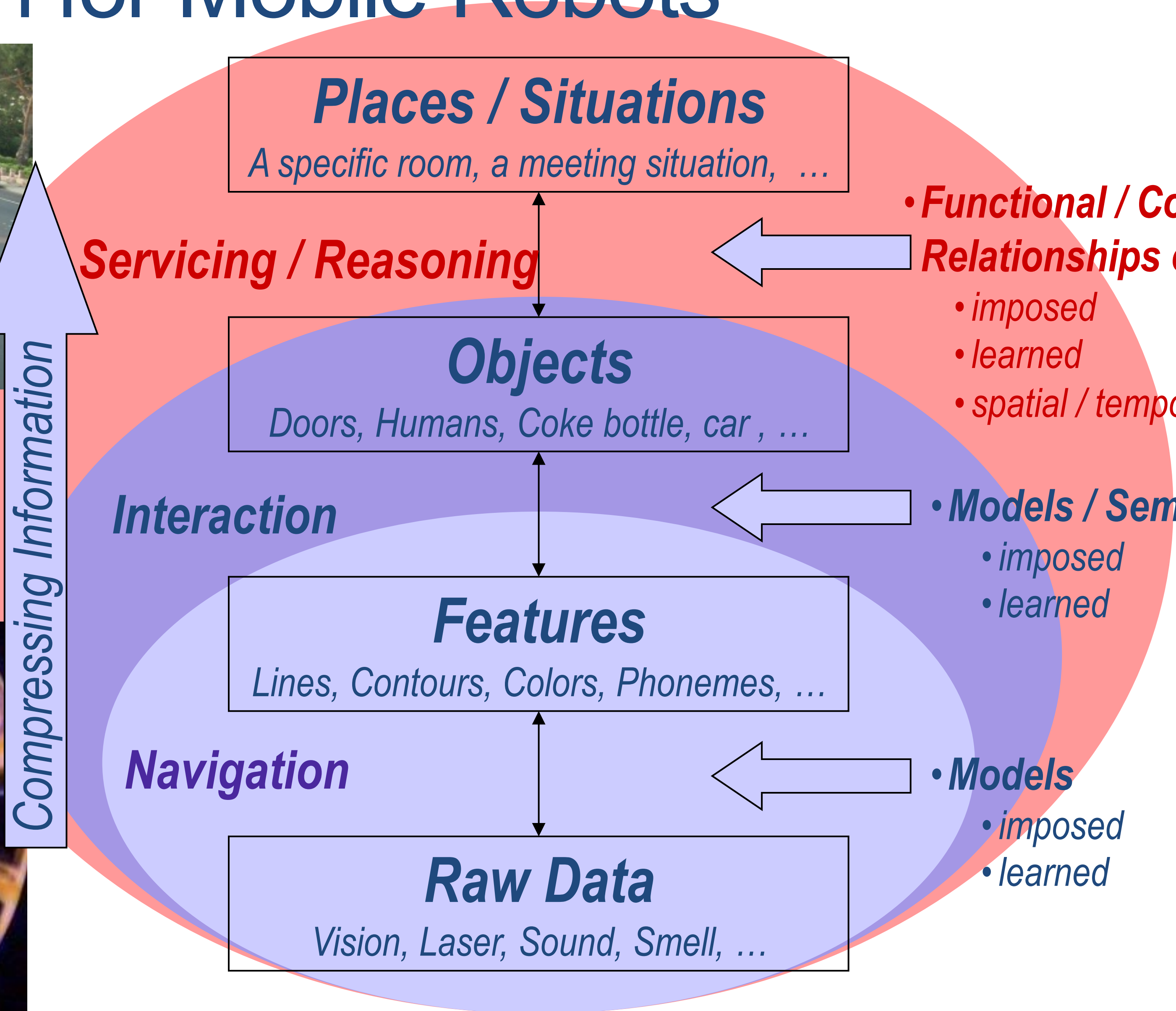
Dealing with Real World Situations

- Reasoning about a situation

- Cognitive systems have to interpret situations based on uncertain and only partially available information
- The need ways to learn functional and contextual information (semantics / understanding)

➡ **Probabilistic Reasoning**

Perception for Mobile Robots



Places / Situations

A specific room, a meeting situation, …

Servicing / Reasoning

Objects

Doors, Humans, Coke bottle, car , …

Interaction

Features

Lines, Contours, Colors, Phonemes, …

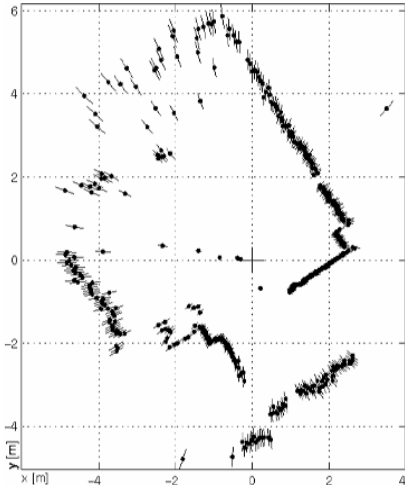
Navigation

Raw Data

Vision, Laser, Sound, Smell, …

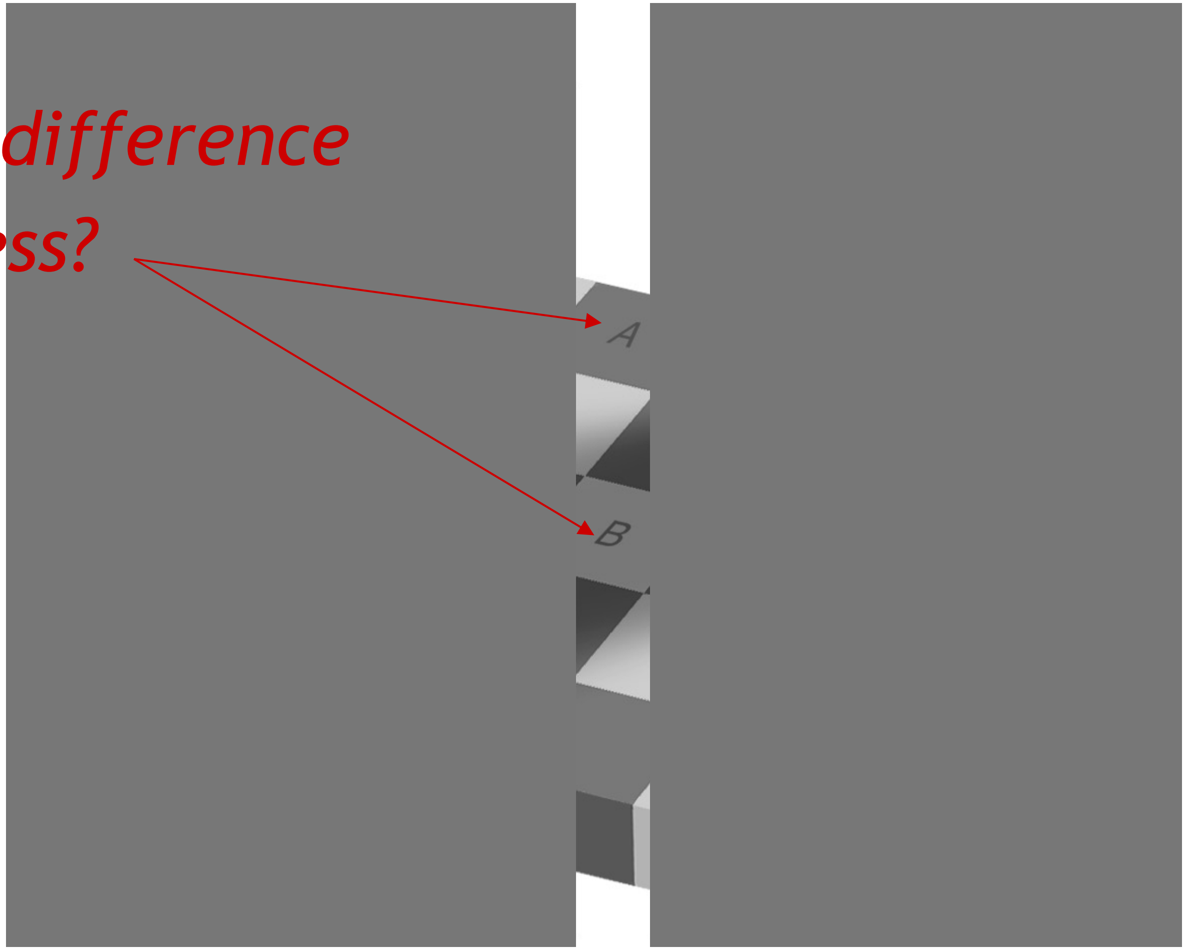
Compressing Information

- ***Functional / Contextual Relationships of Objects***
  - *imposed*
  - *learned*
  - *spatial / temporal/semantic*
- ***Models / Semantics***
  - *imposed*
  - *learned*
- ***Models***
  - *imposed*
  - *learned*



The Challenge

- Perception and models are strongly linked



What is the difference in brightness?

- http://web.mit.edu/persci/people/adelson/checkershadow_downloads.html

Classification of Sensors

- What:
  - Proprioceptive sensors
    - measure values internally to the system (robot),
    - e.g. motor speed, wheel load, heading of the robot, battery status
  - Exteroceptive sensors
    - information from the robots environment
    - distances to objects, intensity of the ambient light, unique features.
- How:
  - Passive sensors
    - Measure energy coming from the environment
  - Active sensors
    - emit their proper energy and measure the reaction
    - better performance, but some influence on environment

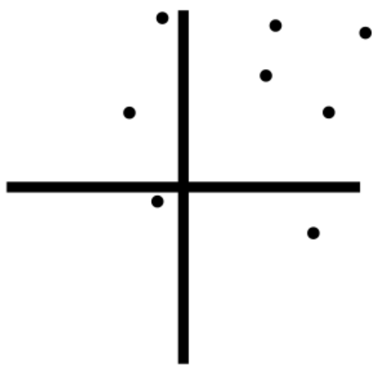
In Situ Sensor Performance

- In Situ: Latin for “in place”

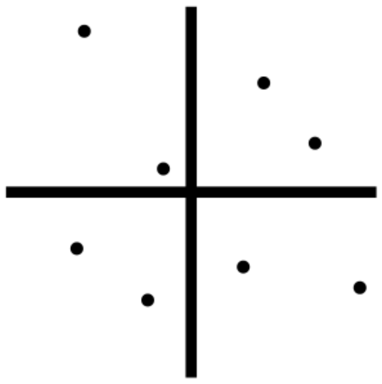
- Error / Accuracy
  - How close to true value
- Precision
  - Reproducibility

$$\left(accuracy = 1 - \frac{\overset{\text{error}}{|m-v|}}{v}\right)$$

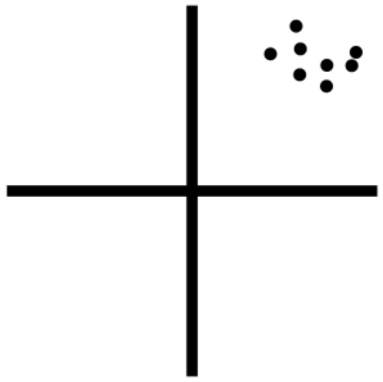
m = measured value
v = true value



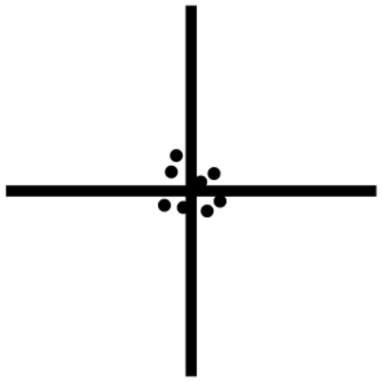
(a) Low precision and low accuracy



(b) Low precision and high accuracy



(c) High precision and low accuracy



(d) High precision and high accuracy

Types of error

- Systematic error -> deterministic errors
  - caused by factors that can (in theory) be modeled -> prediction
  - e.g. calibration of a laser sensor or of the distortion caused by the optic of a camera

- Random error -> non-deterministic
  - no prediction possible
  - however, they can be described probabilistically
  - e.g. Hue instability of camera, black level noise of camera ..

Sensors: outline

- Optical encoders
- Heading sensors
  - Compass
  - Gyroscopes
  - Accelerometer
  - IMU
- GPS
- Range sensors
  - Sonar
  - Laser
  - Structured light
- Vision