
上海科技大学
ShanghaiTech University

# CS283: **Robotics Spring 2026: Sensors**

Sören Schwertfeger

ShanghaiTech University

# ADMIN

# Project

- 2 credit points!
- Work in groups, min 2 students, max 3 students!
- Next lecture: Topics will be proposed…
  - You can also do your own topic, but only after approval of Prof. Schwertfeger
    - Prepare a short, written proposal till next Tuesday!

- Topic selection: Next Thursday!
  - One member writes an email for the whole group to Bichi:  zhangbch2025 (at) shanghaitech.edu.cn ; Put the other group members on CC
  - Subject: [Robotics] Group Selection

- 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

# Campus Autonomy: Crowd Navigation on the Campus

- Use the osmAG navigation stack to navigate the robot
- Detect people with the omni camera
- Predict their motion
- Plan a path, take the predicted motions into account
- Navigate

- Difficulty: medium
- Requite: good demo
- Supervisor: Yongqi Zhang

# Campus Autonomy: Delivery Application

- Build a simple delivery hardware:
  - A “cabinet” that the robot can lock
- Program simple a server and app (website, wechat?)
  - For users to log in and order a delivery service
  - The server dispatches the robot

- Difficulty: medium
- Requite: good demo
- Supervisor: Yongqi Zhang

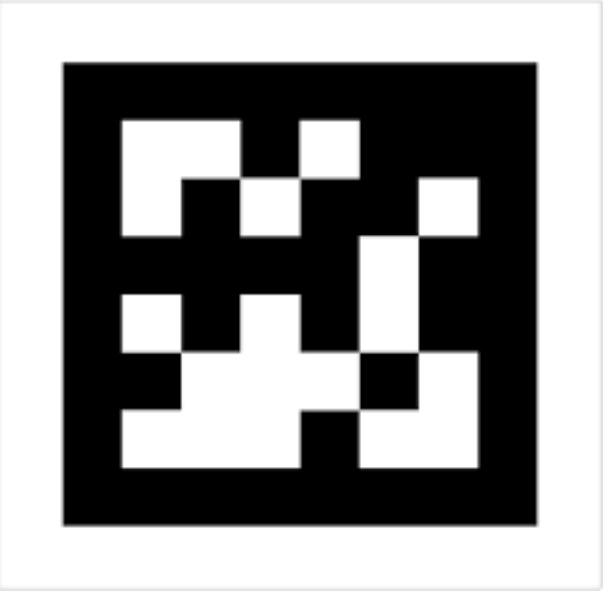
# Competition: LRC & RoboCup

- We have Go2-w robot dogs with wheels
- Teleoperation and AUTONOMY in a difficult terrain
- A team of students already exists – help is welcome

- Various tasks: localization, autonomy, manipulation, object detection, …

- Difficulty: medium/ high
- Requite: good demo
- Supervisor: Bichi Zhang

# Ground Truth Localization via AprilTags



- Continuation of last year's project – done:
  - Print (very big) AprilTags – and distribute in scenario (e.g. underground parking)
  - Use Faro 3D scanner to (semi-) automatically detect and locate AprilTags ->
  - Build ground truth 3D map of AprilTag poses
  - Write a small program to detect AprilTags in the sensor data
- ToDo:
- Optimize map (April tag coordinates)
- Generate ground truth trajectories with this
- Difficulty: Medium
- Graduate Supervisor: Jipeng Kong

# Camera Localization for Dataset GroundTruth

- Work with the Mapping Robot Dataset
- Given a rough pose estimate:
- Use a camera image and the rendered GT for fine localization
- Repeat for several image =>
- Optimize the pose of the robot
- (Potentially: optimize the extrinsic camera calibration)

- Difficulty: medium/ high
- Requite: well working code
- Supervisor: Bichi Zhang

# Fetch Robot

- Some nice project with fetch robot

- Difficulty: Advanced
- Requite: good demo
- Supervisor: Fujing Xie

# Project Suggestion: Draw with Sand

- Build and program such a robot …
- Quite difficult …
- But cool …
- Bigger group (with sub-tasks) allowed

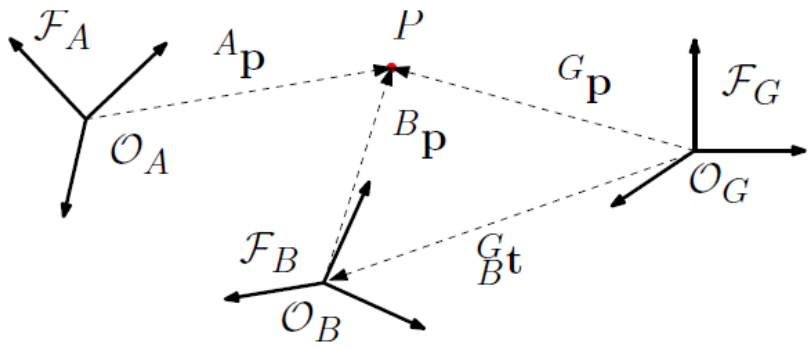
# Robot Introspection for LLMs

- Continue previous projects
- Collect all kinds of robot status data, e.g.:
  - Size, height, weight, capabilities, max speed, urdf, …
  - Current speed, current power consumption, current direction, current mission objective, current battery status, current CPU temp, cpu usage, mem usage
  - ROS status, running nodes, available topics & services, joint values, console log, …
  - All kinds of other, robot intrinsic data
- Feed it into an LLM
- Generate a benchmark to test how well the LLM understands the robot
- Supervisor: Prof. Schwertfeger

- Max one group per topic!
- In case of double selection we will discuss alternatives with both groups
- If no one changes it, it will be “First come - First Serve”

# KINEMATICS CONTINUED

# 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

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

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

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

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

# Chaining :

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

## For 2D Translation:

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

## For 2D Translation using Matrix Math:

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

## For 2D Rotation:

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

## For 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

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

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

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

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

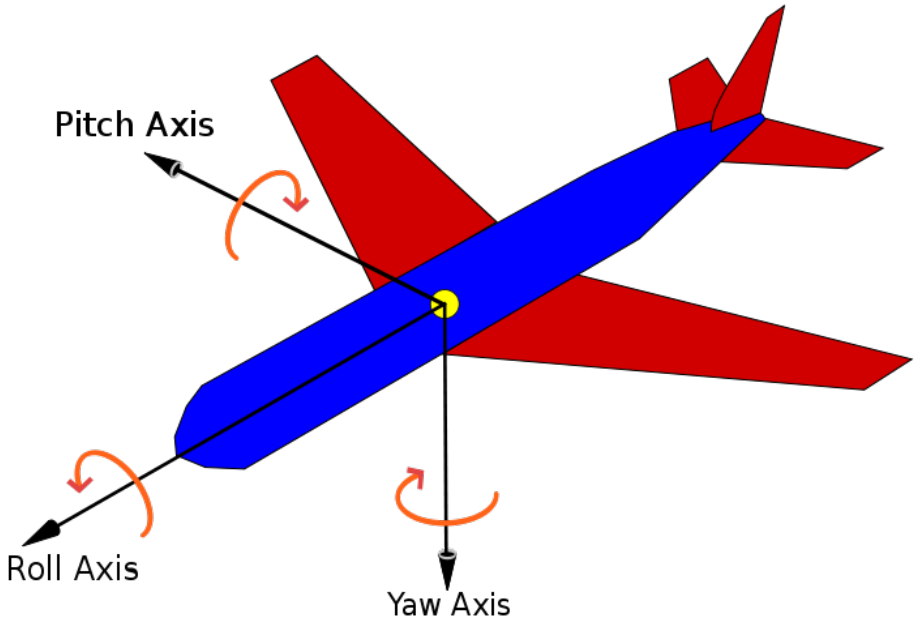
```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose2D pose2D
  float64 x
  float64 y
  float64 theta
```

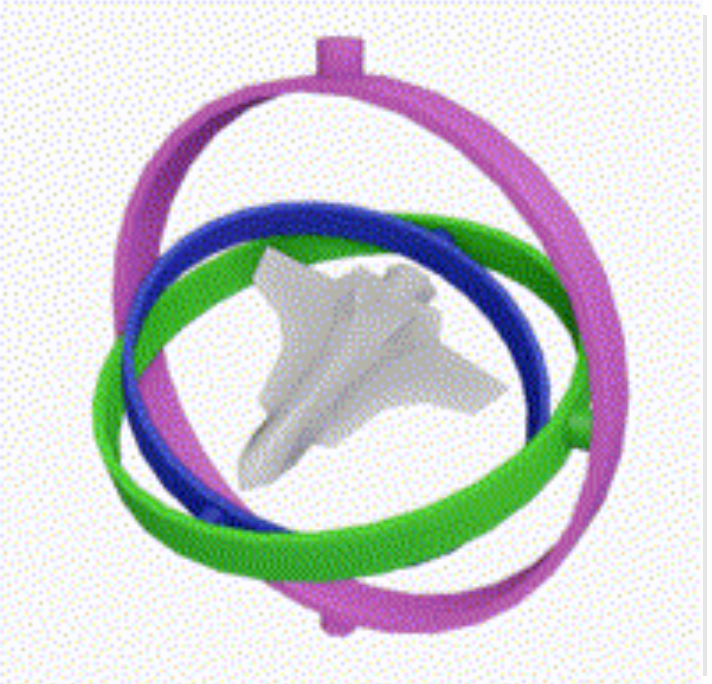
# 3D Rotation

- Many 3D rotation representations:

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

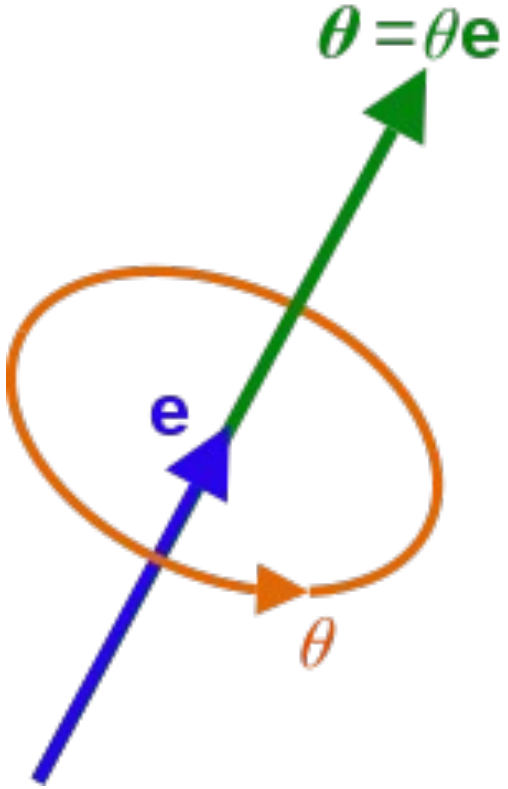
- Euler angles:
  - Roll: rotation around x-axis
  - Pitch: rotation around y-axis
  - Yaw: rotation around z-axis
  - Apply rotations one after the other…
    - => Order important! E.g.:
      - x-z-x; x-y-z; z-y-x; …
  - ☹ Singularities
  - Gimbal lock in Engineering
    - "a condition caused by the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



# 3D Rotation

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

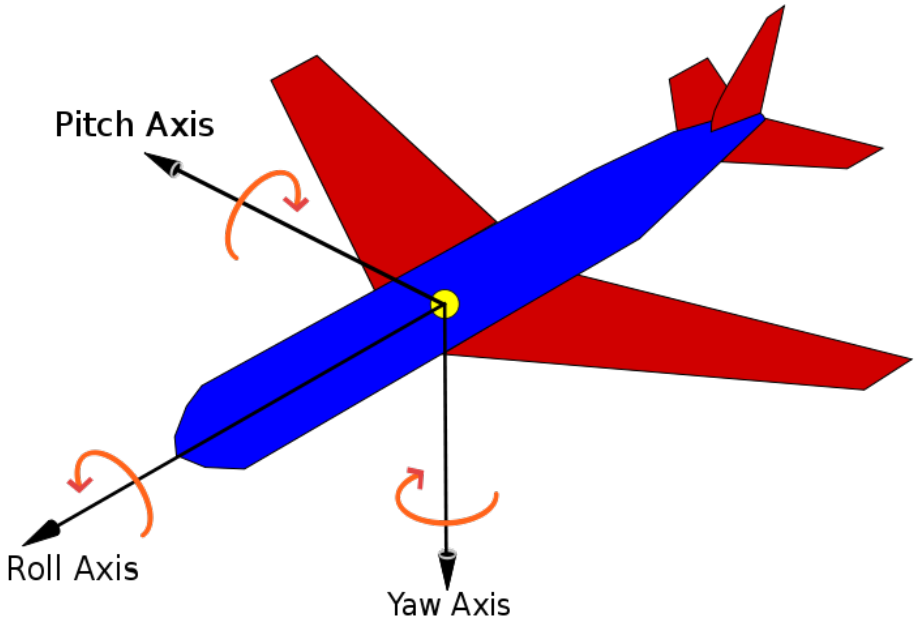



- Euler Angles: sequence of 3 rotations around coordinate axes

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

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

# 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:

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)<br>Convert a timestamped transform to the equivalent **Eigen** data |
| Eigen::Isometry3d | **transformToEigen** (const geometry_msgs::msg::TransformSta<br>Convert a timestamped transform to the equivalent **Eigen** data |
| geometry_msgs::msg::TransformStamped | **eigenToTransform** (const Eigen::Affine3d &T)<br>Convert an **Eigen** Affine3d transform to the equivalent geometry |
| geometry_msgs::msg::TransformStamped | **eigenToTransform** (const Eigen::Isometry3d &T)<br>Convert an **Eigen** Isometry3d transform to the equivalent geom |
| template<> | |
| void | **doTransform** (const Eigen::Vector3d &t_in, Eigen::Vector3d &t<br>Apply a geometry_msgs TransformStamped to an Eigen-specifi<br>used in tf2_ros::BufferInterface::transform because this function |
| geometry_msgs::msg::Point | **toMsg** (const Eigen::Vector3d &in)<br>Convert a **Eigen** Vector3d type to a Point message. This functio |
| void | **fromMsg** (const geometry_msgs::msg::Point &msg, Eigen::Vec<br>Convert a Point message type to a Eigen-specific Vector3d type |
| geometry_msgs::msg::Vector3 & | **toMsg** (const Eigen::Vector3d &in, geometry_msgs::msg::Vecto<br>Convert an **Eigen** Vector3d type to a Vector3 message. This fu |
| void | **fromMsg** (const geometry_msgs::msg::Vector3 &msg, Eigen::V<br>Convert a Vector3 message type to a Eigen-specific Vector3d ty |
| template<> | |
| void | **doTransform** (const tf2::Stamped< Eigen::Vector3d > &t_in, tf2<br>Apply a geometry_msgs TransformStamped to an Eigen-specifi |
| geometry_msgs::msg::PointStamped | **toMsg** (const tf2::Stamped< Eigen::Vector3d > &in)<br>Convert a stamped **Eigen** Vector3d type to a PointStamped me |
| void | **fromMsg** (const geometry_msgs::msg::PointStamped &msg, tf<br>Convert a PointStamped message type to a stamped Eigen-spe |

# 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

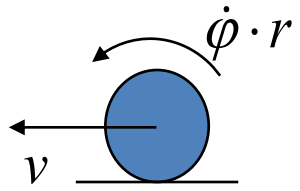
# ROS Standards:

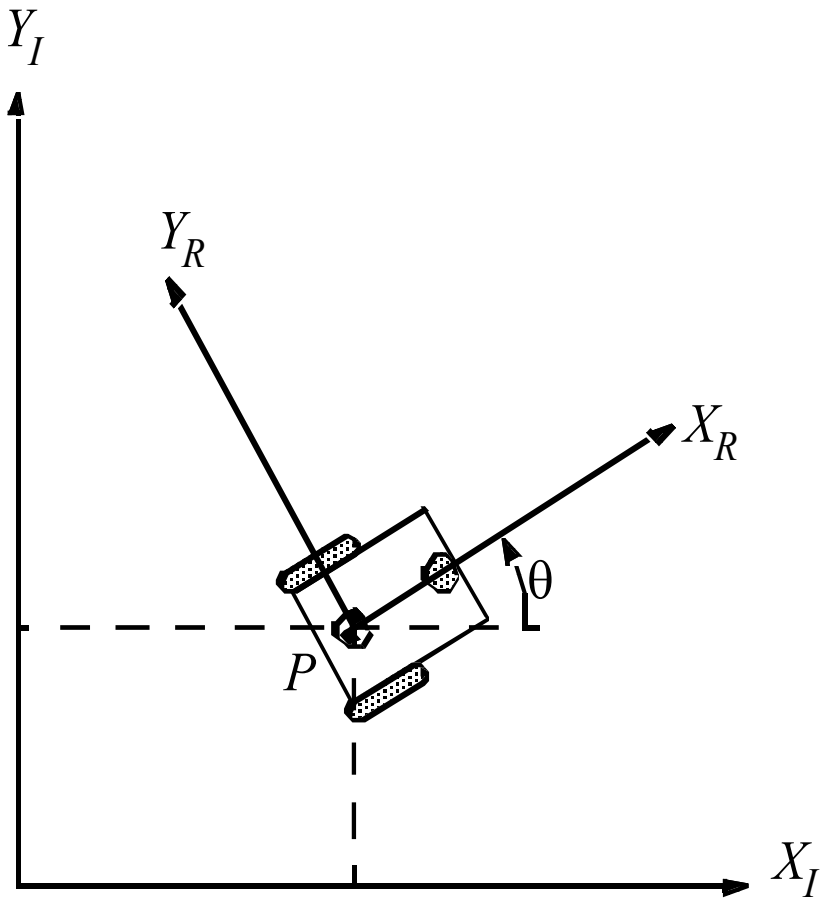
- Standard Units of Measure and Coordinate Conventions
  - http://www.ros.org/reps/rep-0103.html
- Coordinate Frames for Mobile Platforms:
  - http://www.ros.org/reps/rep-0105.html

# ROBOT KINEMATICS: DIFFERENTIAL DRIVE

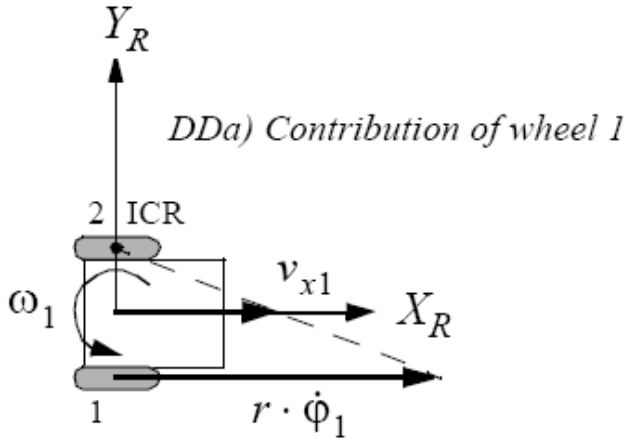
# Wheel Kinematic Constraints: Assumptions

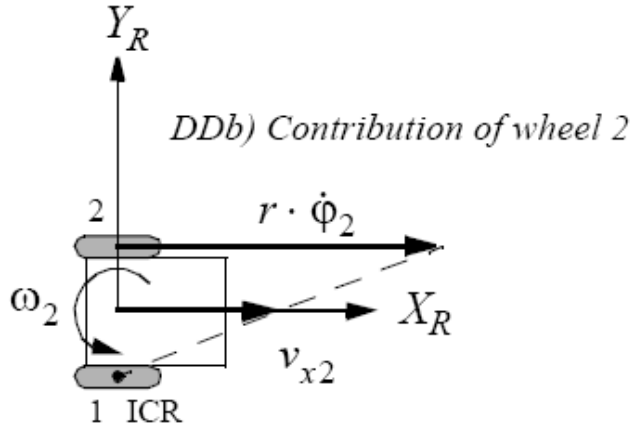
- Movement on a horizontal plane
- Point contact of the wheels
- Wheels not deformable
- Pure rolling
  - $v_c = 0$ at contact point
- No slipping, skidding or sliding
- No friction for rotation around contact point
- Steering axes orthogonal to the surface
- Wheels connected by rigid frame (chassis)



# Forward Kinematic Model: Geometric Approach




***Differential-Drive:***

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

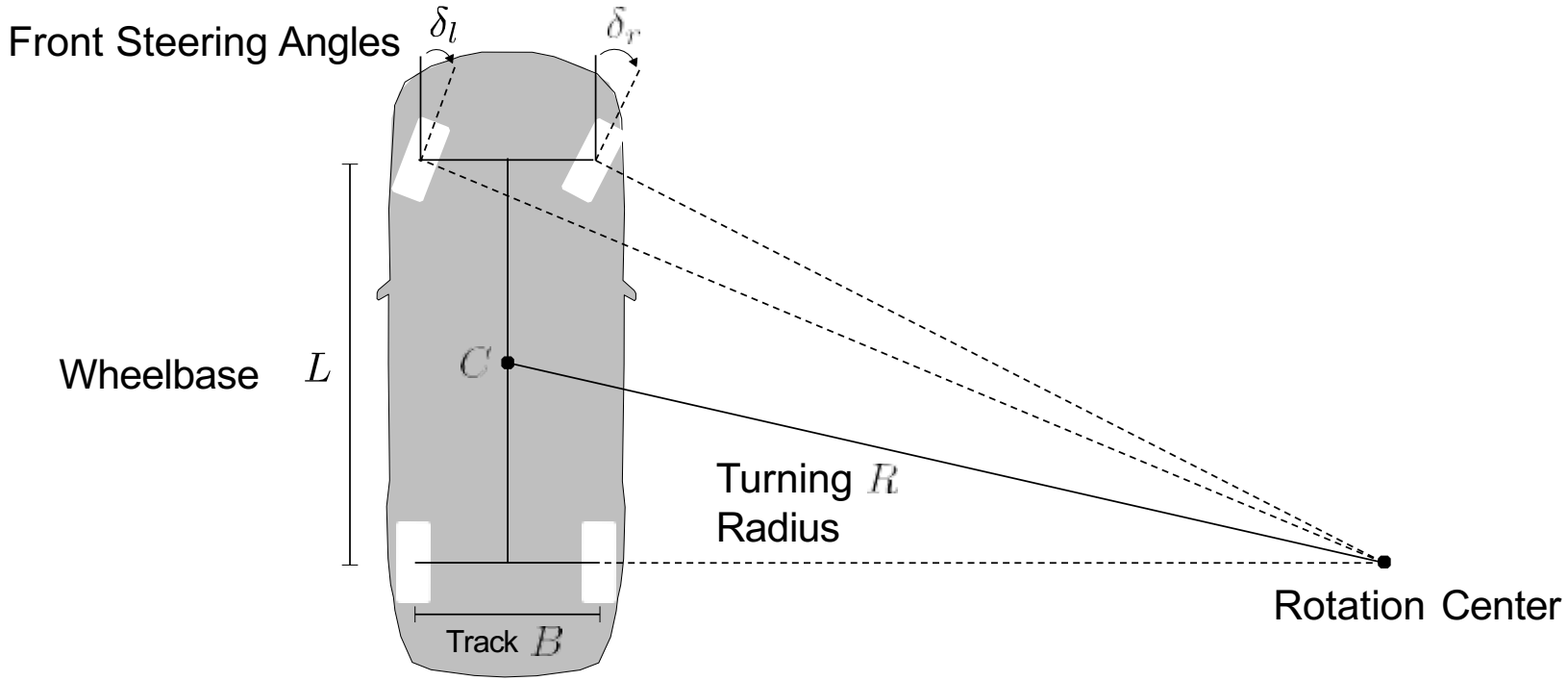
DDb) $v_{x2} = \frac{1}{2} r\dot{\varphi}_2 \quad ; \quad v_{y2} = 0 \quad ; \quad \omega_2 = -\frac{1}{2l} r\dot{\varphi}_2$

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

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

# ROBOT KINEMATICS: ACKERMANN

# Ackermann Steering Geometry

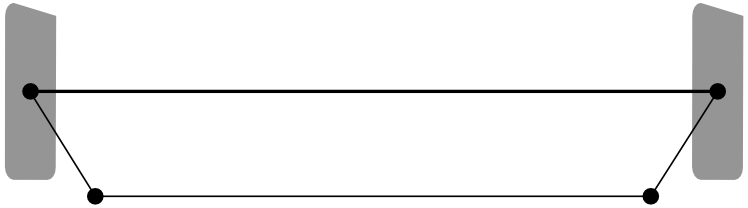



- In practice, the left and right wheel steering angles are not equal if no wheel slip
- Combination of admissible steering angles called Ackerman steering geometry
- If angles are small, the left/right steering wheel angles can be approximated:

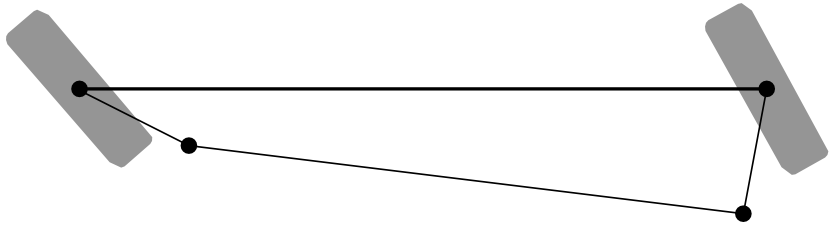
$$\delta_l \approx \tan\left(\frac{L}{R+0.5B}\right) \approx \frac{L}{R+0.5B} \qquad \delta_r \approx \tan\left(\frac{L}{R-0.5B}\right) \approx \frac{L}{R-0.5B}$$

# Ackermann Steering Geometry

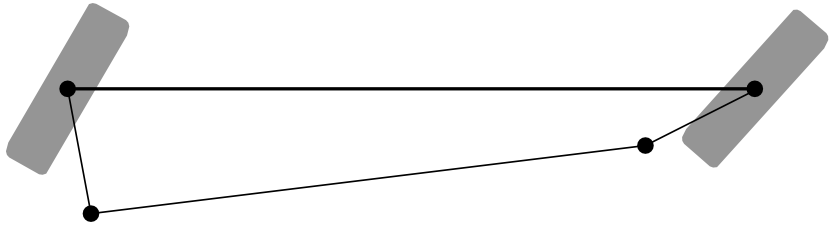
Trapezoidal Geometry



Left Turn



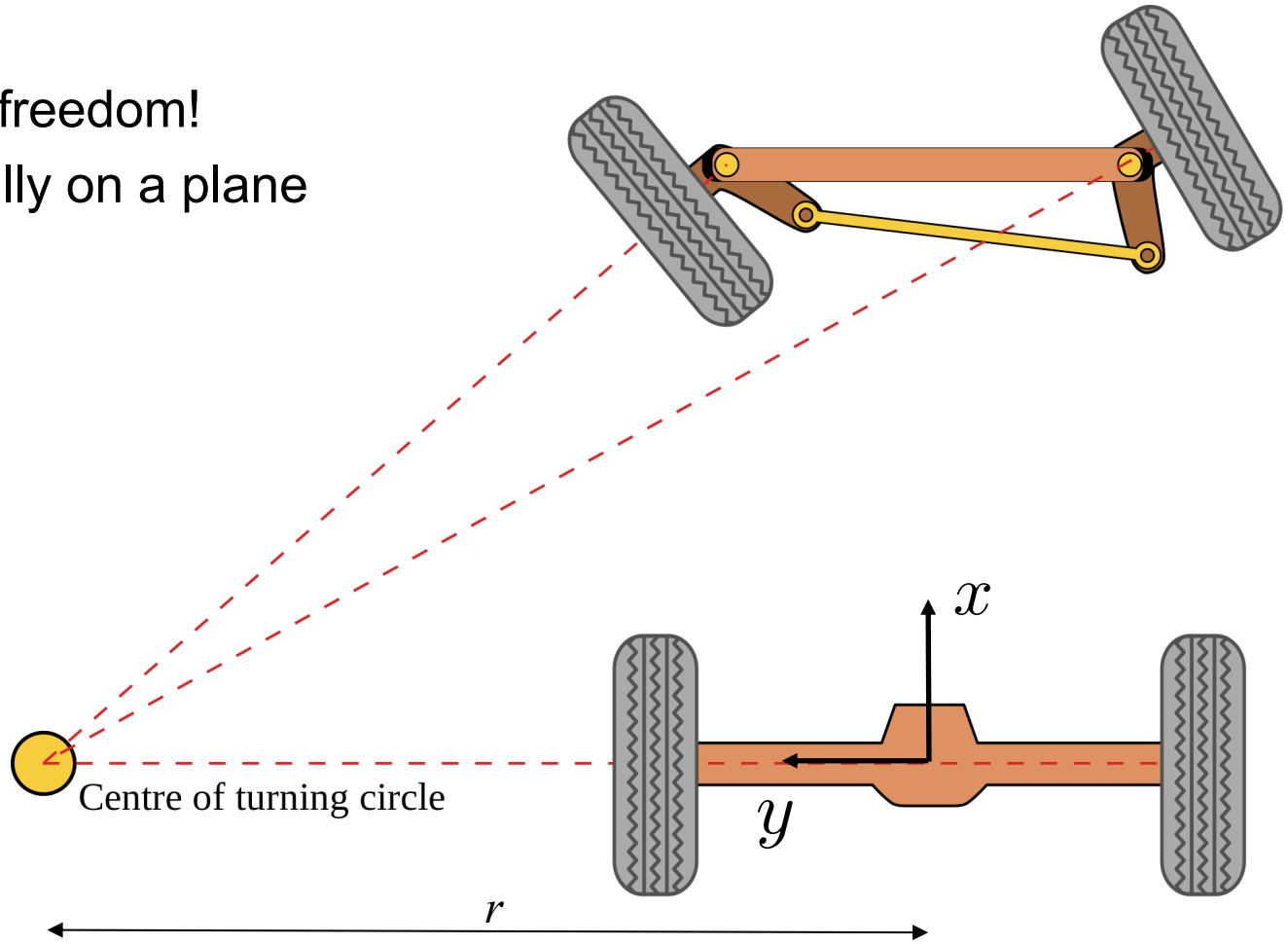
Right Turn



- In practice, this setup can be realized using a trapezoidal tie rod arrangement

# Vehicle kinematic motion model

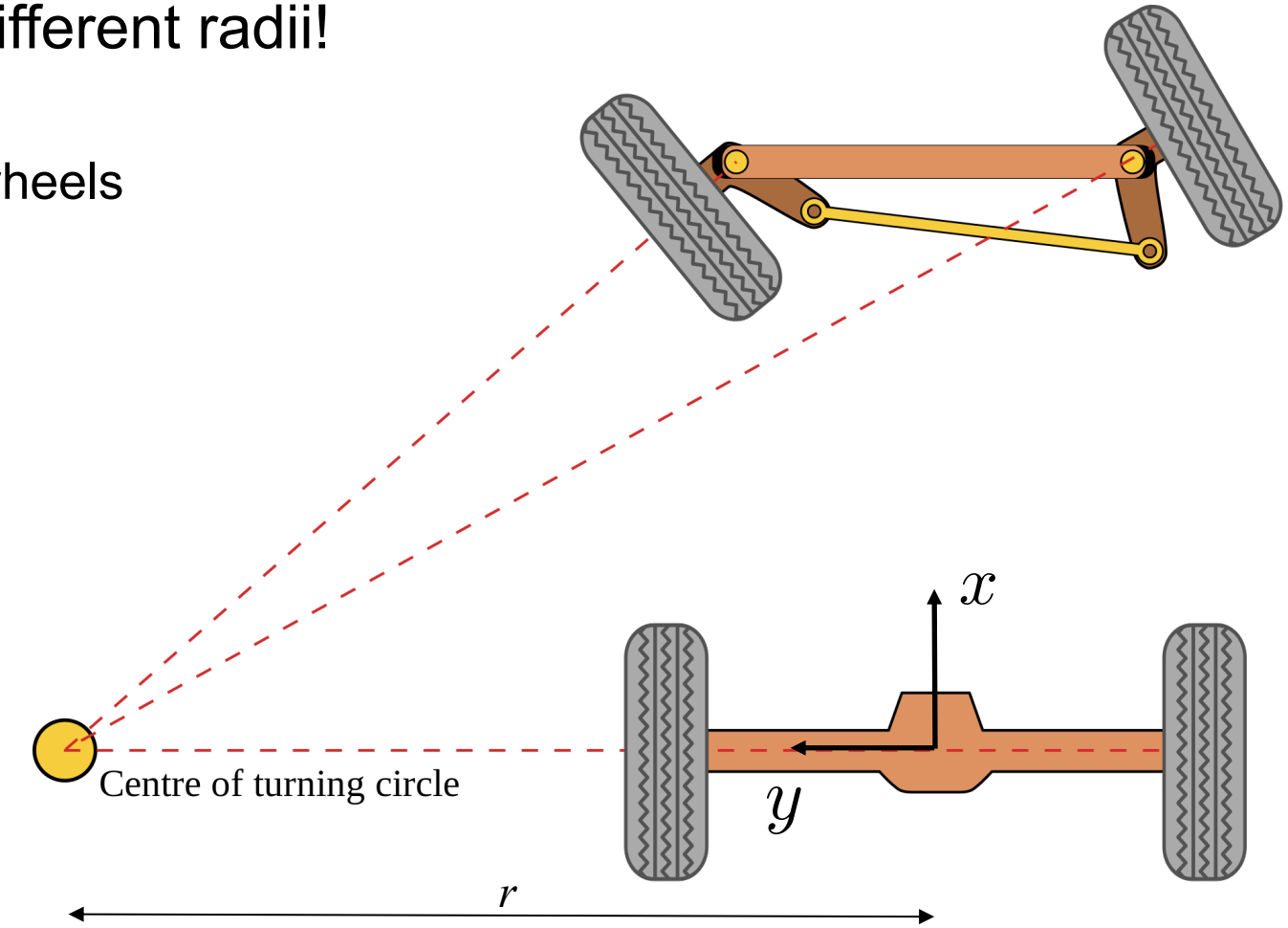
- Important insights:
  - The vehicle has only 2 degrees of freedom!
  - The vehicle's motion happens locally on a plane
    - 2 DoF translation displacement
    - 1 DoF rotation displacement
  - → Vehicle motion is exposed to non-holonomic constraints



# Vehicle kinematic motion model

- Wheels drive on circles with different radii!
  - => wheels have different speeds!
  - Often: vehicle is driven on back wheels
  - => need a way to drive wheels with different speeds!
  - => Differential (gear box)

    != Differential drive robot!

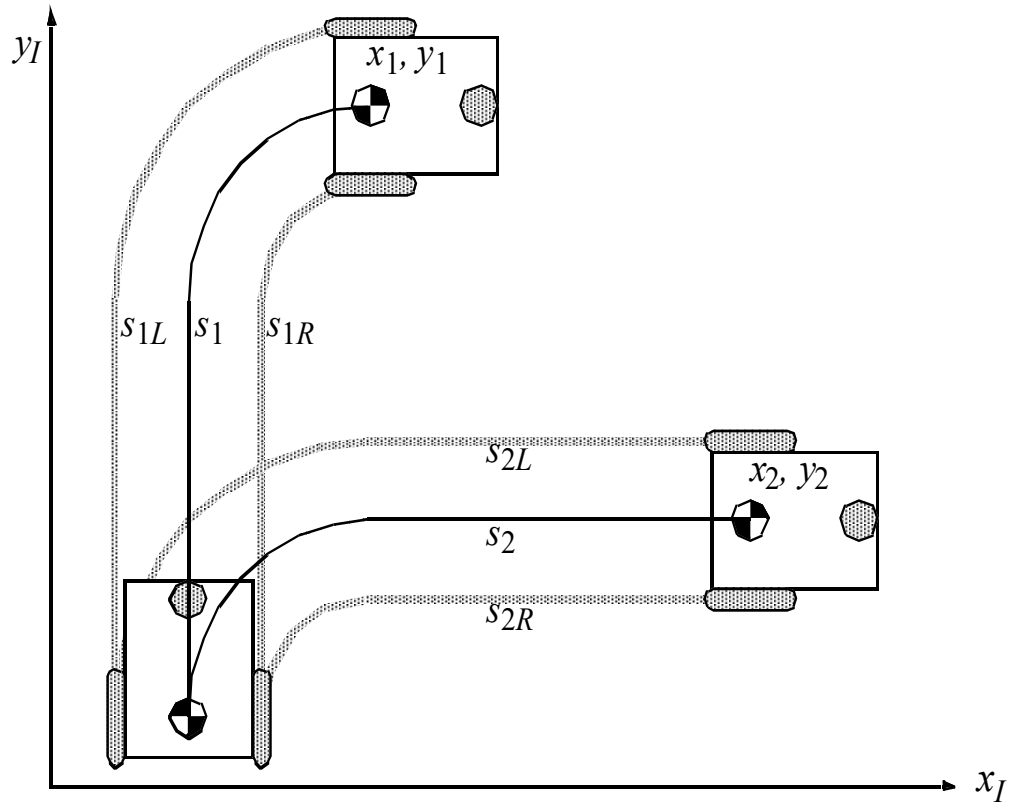



https://www.youtube.com/watch?v=nC6fsNXdcMQ

# Mobile Robot Kinematics: Non-Holonomic Systems

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

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



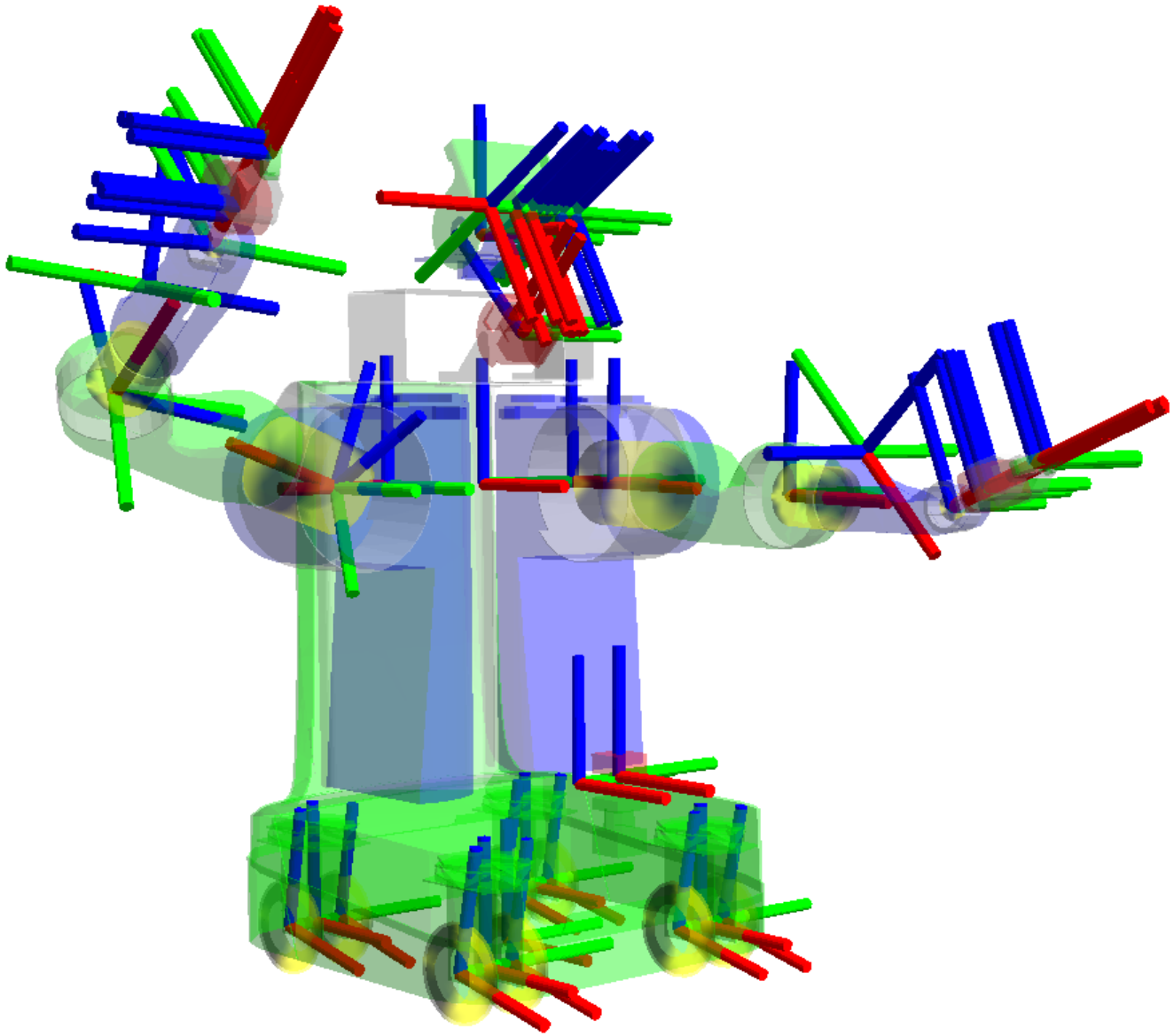

- Non-holonomic systems
  - differential equations are not integrable to the final pose.
  - the measure of the traveled distance of each wheel is not sufficient to calculate the final position of the robot. One has also to know how this movement was executed as a function of time.

# Holonomic examples

Uranus, CMU

# ROS: 3D Transforms : TF

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



# ROS 1 geometry_msgs/TransformStamped

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

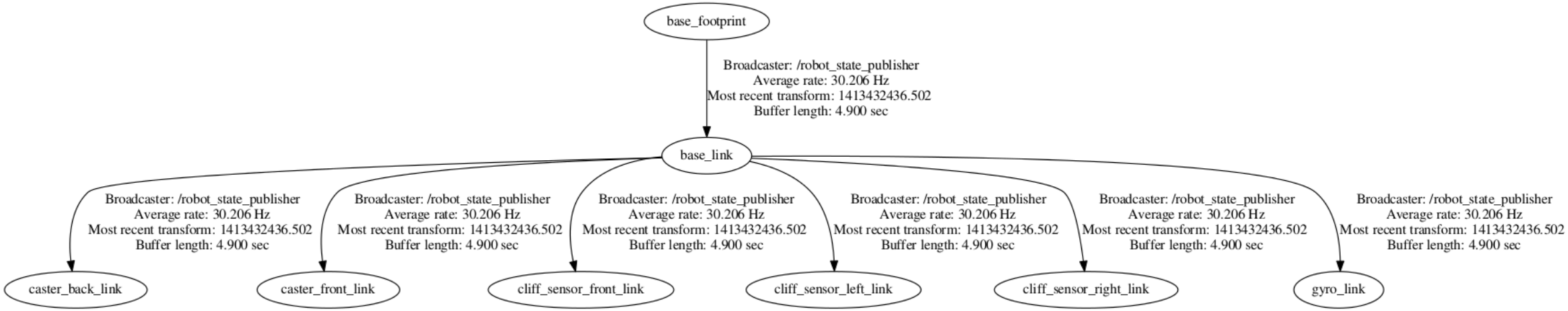
- Transform between header (time and reference frame) and child_frame
- 3D Transform representation:
  - geometry_msgs/Transform:
    - Vector3 for translation (position)
    - Quaternion for rotation (orientation)

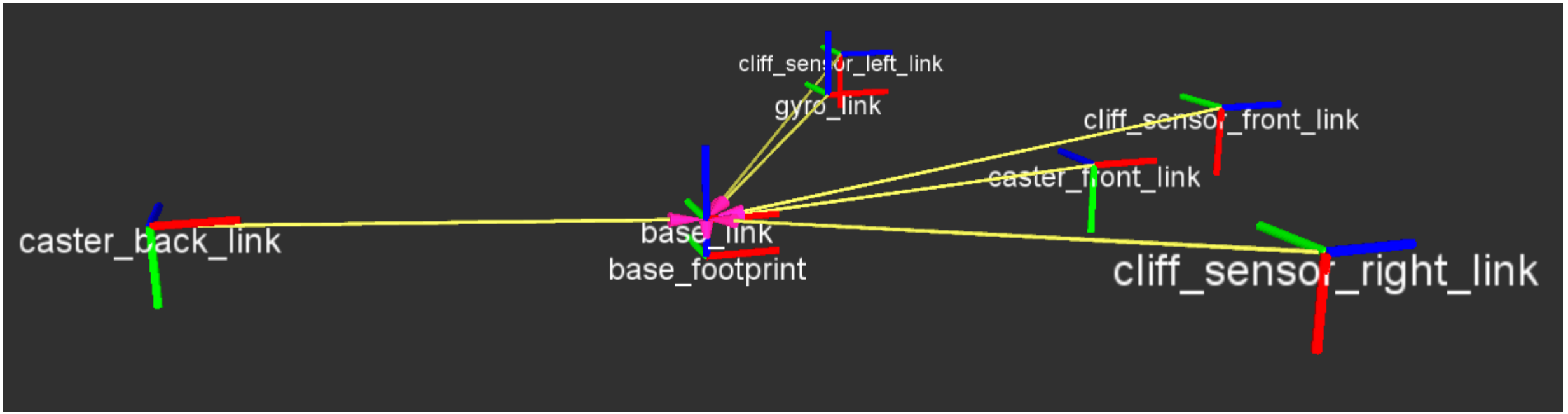
```
rosmsg show geometry_msgs/TransformStamped

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

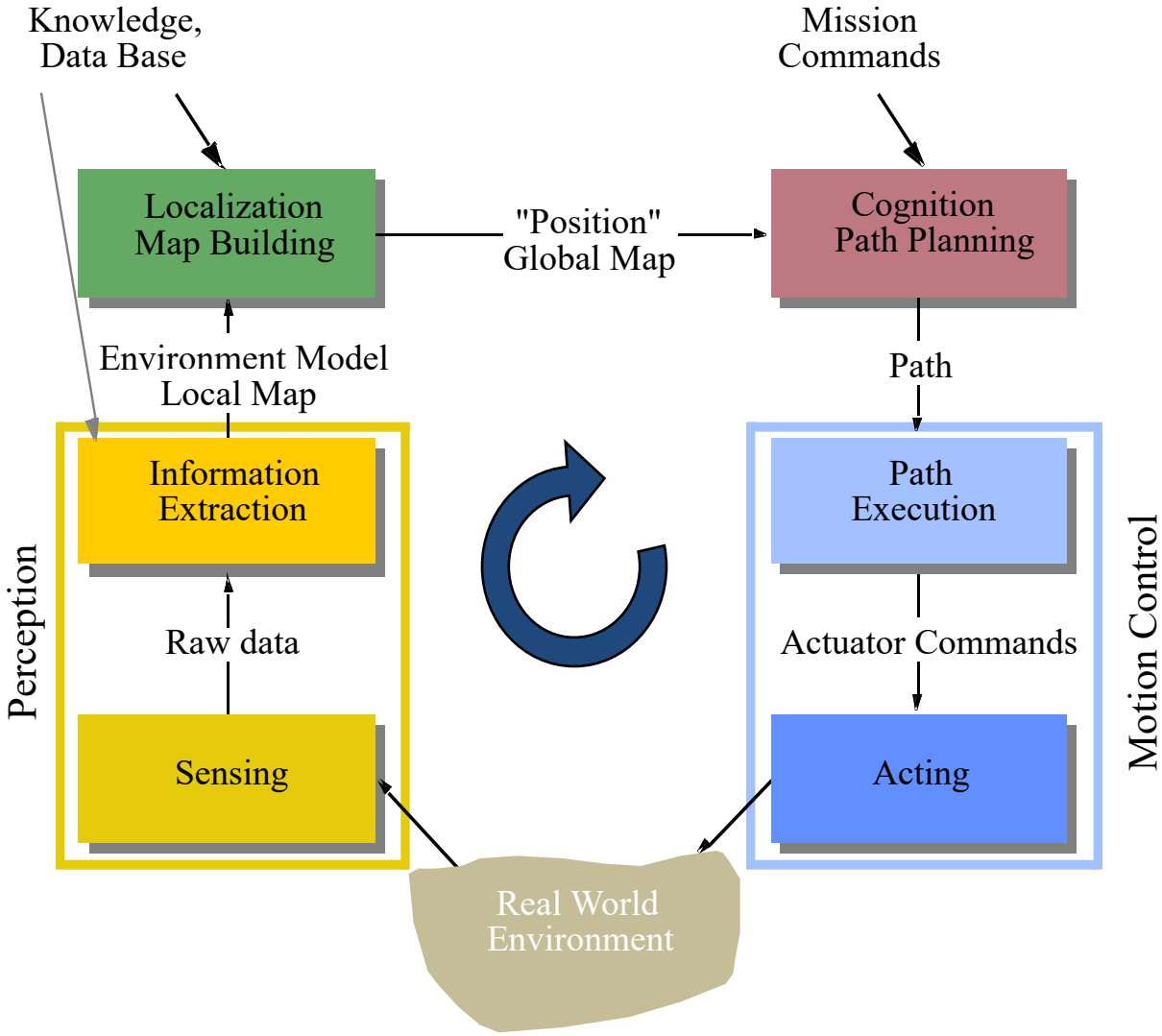

cliff_sensor_left_link
gyro_link
cliff_sensor_front_link
caster_front_link
caster_back_link
base_link
base_footprint
cliff_sensor_right_link

# 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);
- 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**

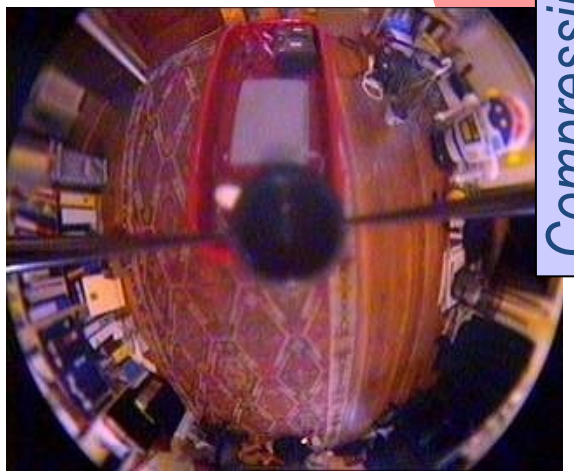
# Dealing with Real World Situations

- 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**

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

**Objects**
*Doors, Humans, Coke bottle, car , …*

**Interaction**

- **Models / Semantics**
  - *imposed*
  - *learned*

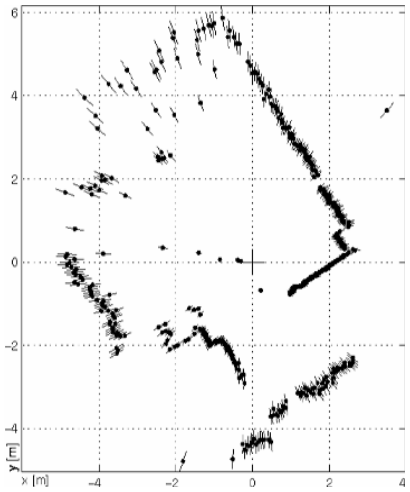
**Features**
*Lines, Contours, Colors, Phonemes, …*

**Navigation**

- **Models**
  - *imposed*
  - *learned*

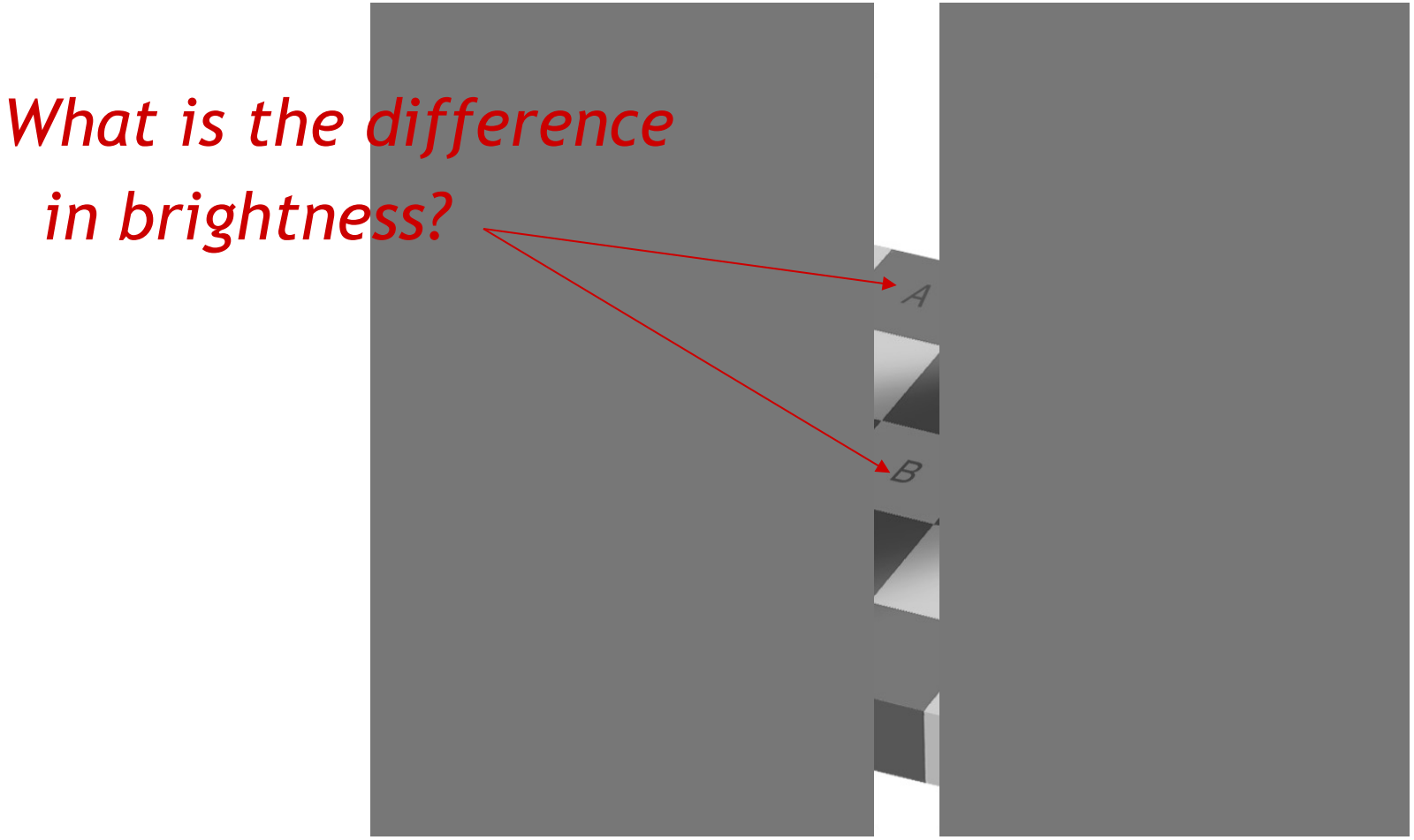
**Raw Data**
*Vision, Laser, Sound, Smell, …*

*Compressing Information*



# The Challenge

- Perception and models are strongly linked




- 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

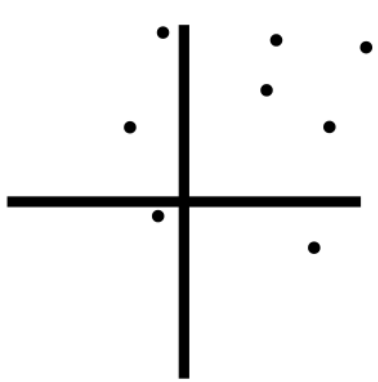
- In Situ: Latin for “in place”

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

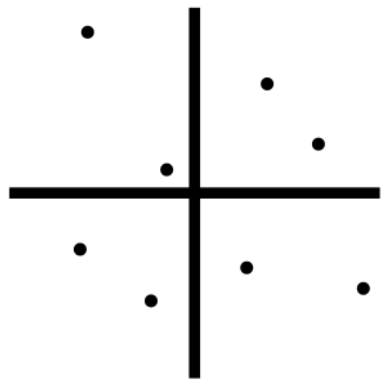
$$\left(accuracy = 1 - \frac{|m - v|}{v}\right)$$

*error*

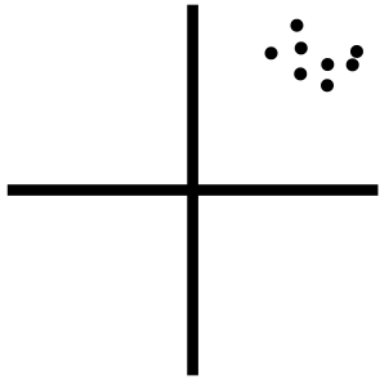
*m = measured value*
*v = true value*



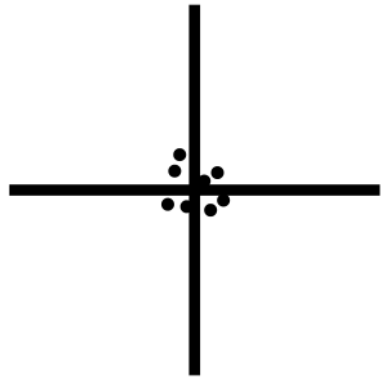
(a) Low precision and low accuracy



(b) Low precision and high accuracy



(c) High precision and low accuracy



(d) High precision and high accuracy

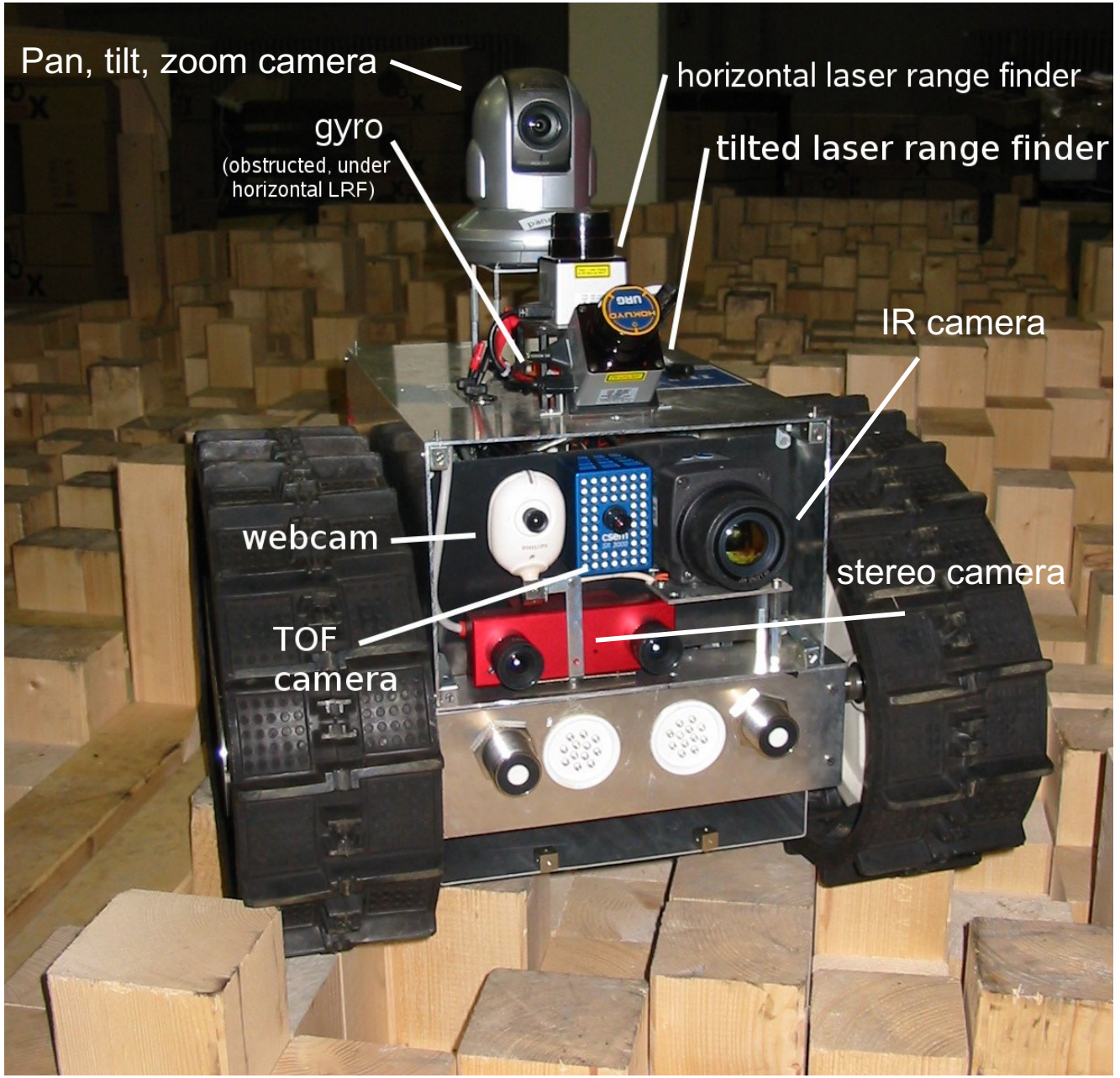
# Types of error

- 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 ..

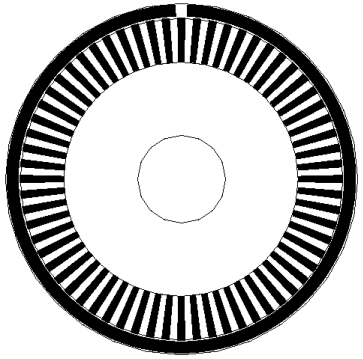
# Sensors: outline

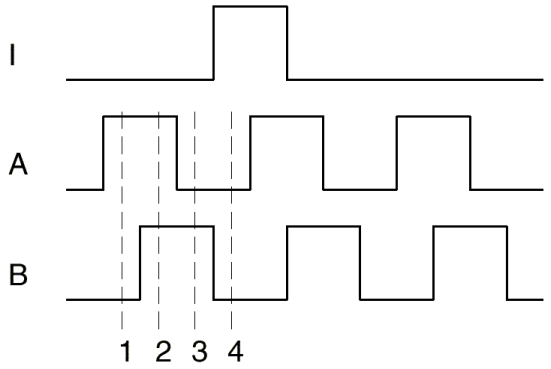
- Optical encoders
- Heading sensors
  - Compass
  - Gyroscopes
  - Accelerometer
  - IMU
- GPS
- Range sensors
  - Sonar
  - Laser
  - Structured light
- Vision



# Wheel / Motor Encoders

- measure position or speed of the wheels or steering
- integrate wheel movements to get an estimate of the position -> odometry
- optical encoders are proprioceptive sensors
- typical resolutions: 64 - 2048 increments per revolution.
  - for high resolution: interpolation
- optical encoders
  - regular: counts the number of transitions but cannot tell the direction of motion
  - quadrature: uses two sensors in quadrature-phase shift. The ordering of which wave produces a rising edge first tells the direction of motion. Additionally, resolution is 4 times bigger
  - a single slot in the outer track generates a reference pulse per revolution

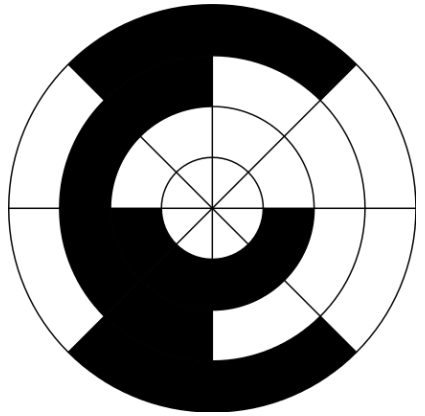





| State | Ch A | Ch B |
|---|---|---|
| $S_1$ | High | Low |
| $S_2$ | High | High |
| $S_3$ | Low | High |
| $S_4$ | Low | Low |

# Gray Encoder

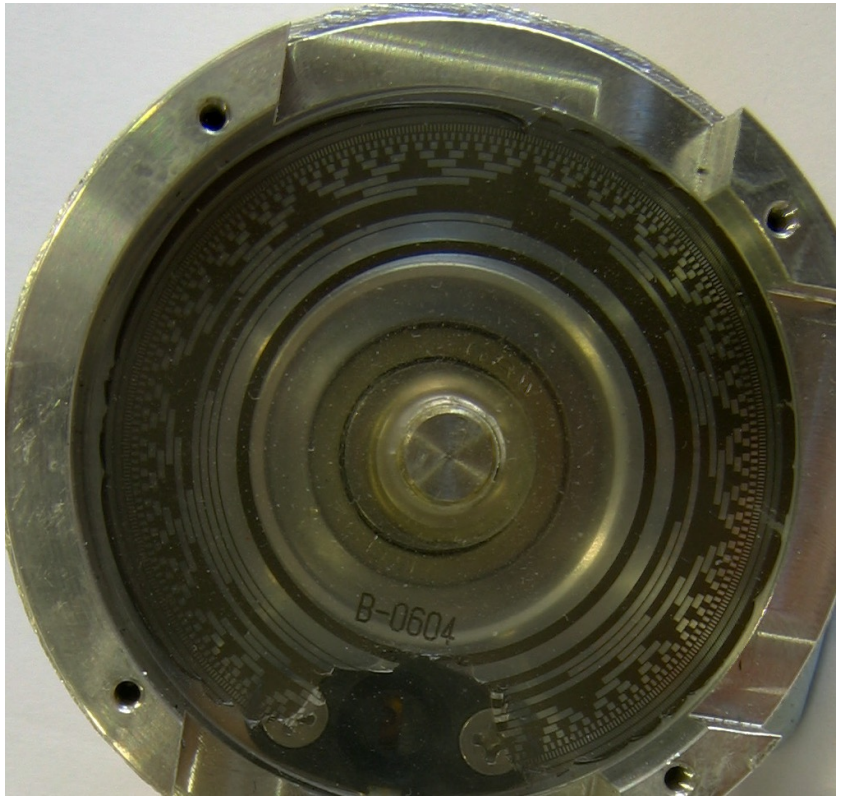
http://en.wikipedia.org/wiki/Gray_code



- Aka: reflected binary code, Gray Code
  - Binary numeral system where two successive values differ in only one bit
  - Also used for error correction in digital communications

- Absolute position encoder
  - Normal binary => change from 011 to 100
  - 2 bits change – NEVER simultaneously =>
  - 011 -> 111 -> 101 -> 100 or
  - 011 -> 010 -> 110 -> 100 ….
  - => wrong encoder positions might be read
  - Gray encoding: only one bit change!

| Dec | Gray | Binary |
|---|---|---|
| 0 | 000 | 000 |
| 1 | 001 | 001 |
| 2 | 011 | 010 |
| 3 | 010 | 011 |
| 4 | 110 | 100 |
| 5 | 111 | 101 |
| 6 | 101 | 110 |
| 7 | 100 | 111 |



# Heading Sensors

- Heading sensors can be proprioceptive (gyroscope, acceleration) or exteroceptive (compass, inclinometer).
- Used to determine the robots orientation and inclination.
- Allow, together with an appropriate velocity information, to integrate the movement to a position estimate.
  - This procedure is called **deduced reckoning** (ship navigation)

# Compass

- Magnetic field on earth
  - absolute measure for orientation
- Large variety of solutions to measure the earth magnetic field
  - mechanical magnetic compass
  - direct measure of the magnetic field (Hall-effect, magneto-resistive sensors)
- Major drawback
  - weakness of the earth field (30 µTesla)
  - easily disturbed by magnetic objects or other sources
  - bandwidth limitations (0.5 Hz) and susceptible to vibrations
  - not feasible for indoor environments for absolute orientation
  - useful indoor (only locally)