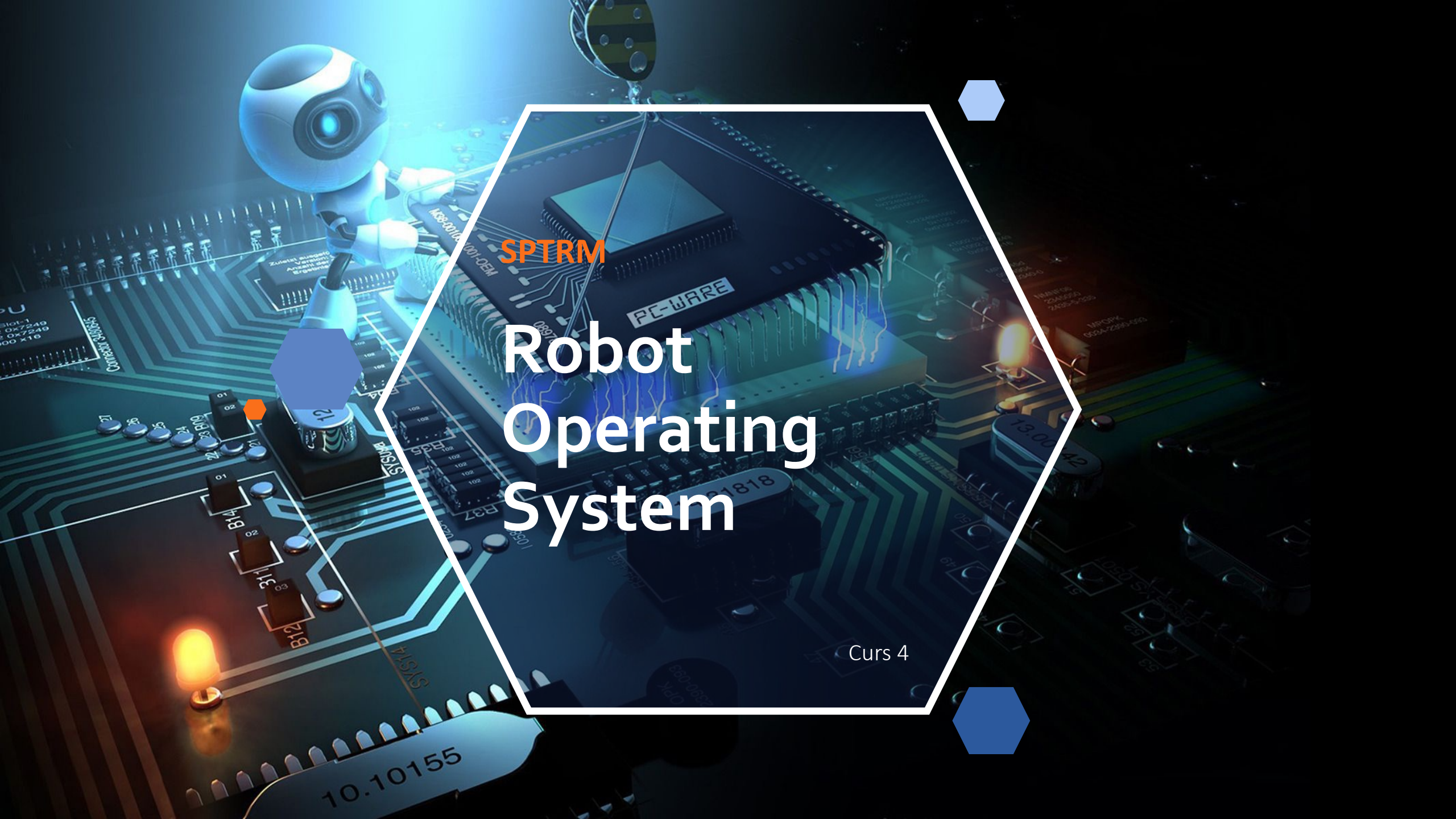
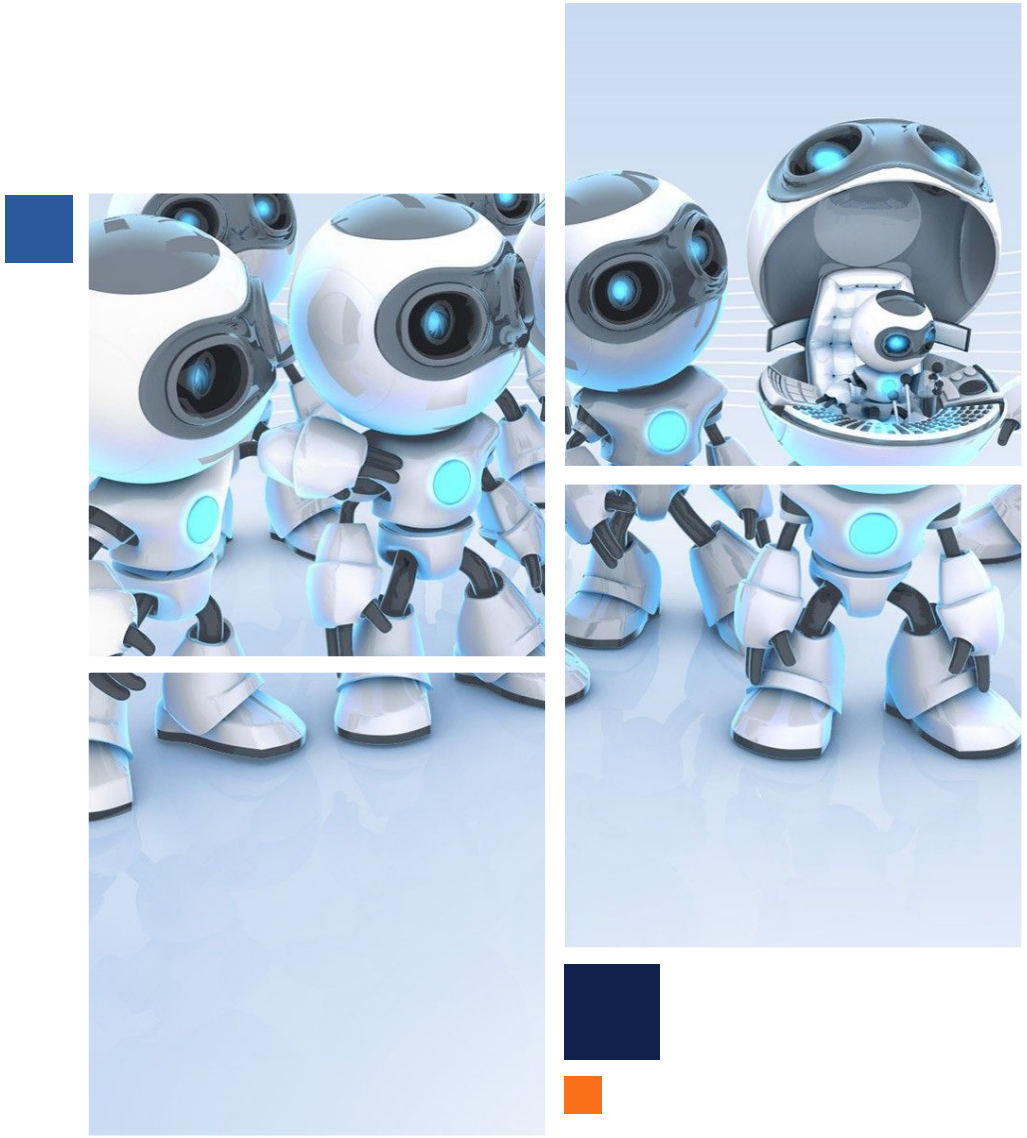
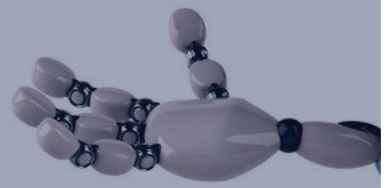

SPTRM
Robot Operating System
Curs 4

Agenda

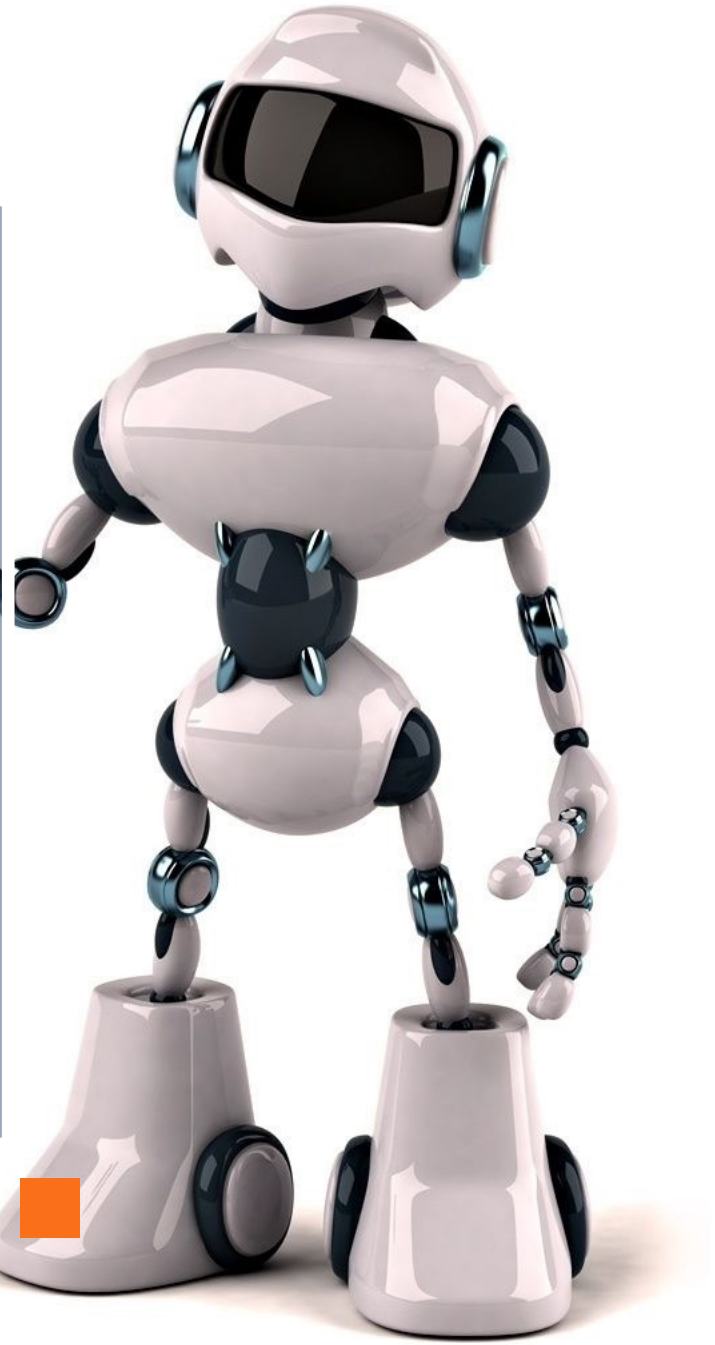
- Recap
- Coordinate Frames
- Transformations
  - Rotation
  - Translation
- Robot Description
  - URDF + SDF
- ROS Transform Frame
  - TF
  - Transform Tree



Recap

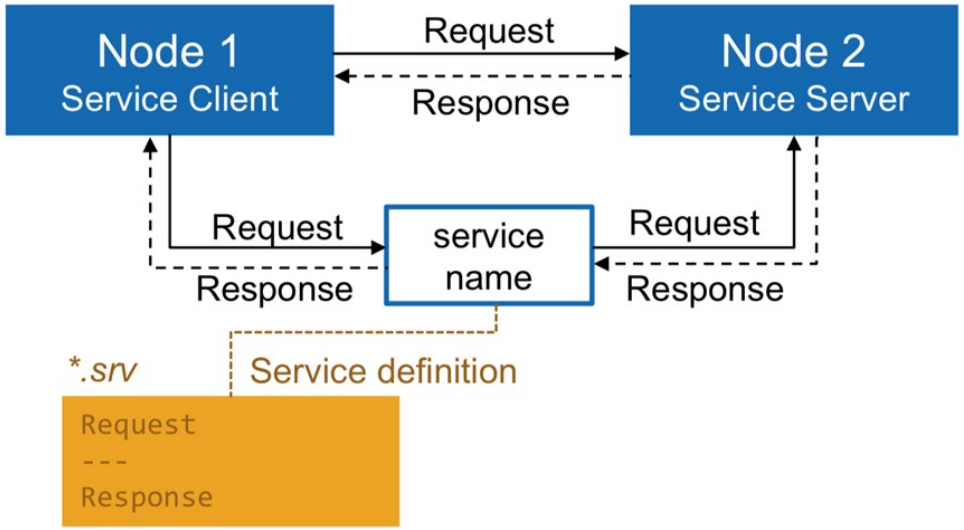


From the last episode…



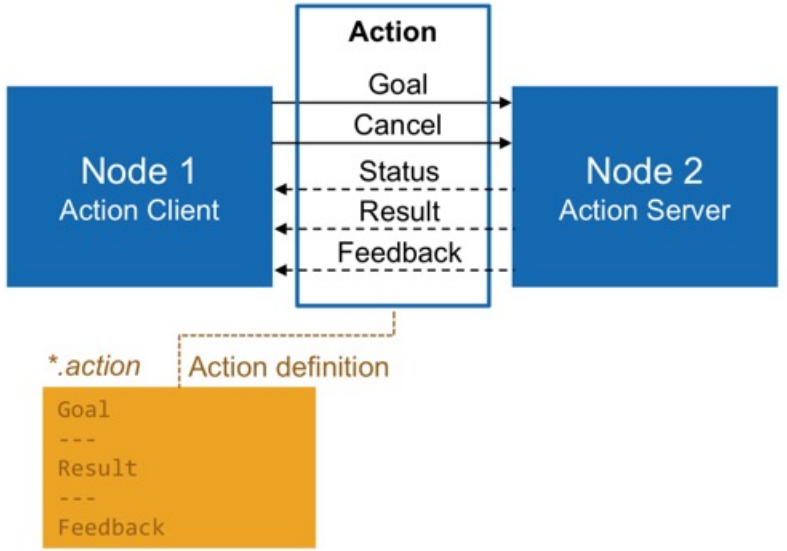
ROS Services

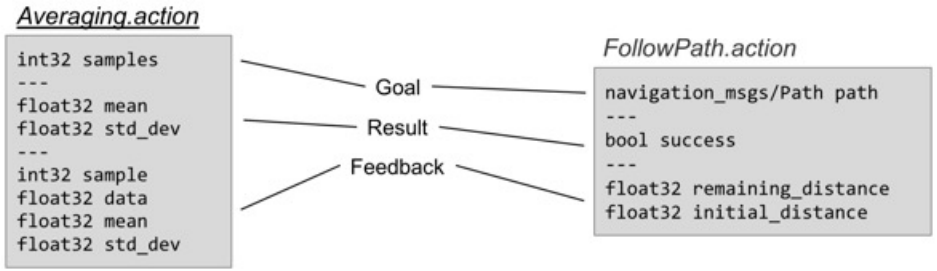
- Services defined in *.srv files
- 2 sections:
  - Request parameters
  - Response parameters
- Similar to ROS messages
- Can contain zero or multiple entries per section (simple date or complex types)



ROS Actions (actionlib)

- Similar to service calls
- Allows task cancelling (preempt) and progress feedback
- Best way to implement interfaces to time-extended, goal-oriented behaviors
- Similar in structure to services, action are defined in *.action files
- Internally, actions are implemented with a set of topics





ROS Time and Duration

- Normally, ROS uses the PC's system clock as time source (**wall time**)
- APIs:

  ros::Time, ros::Duration

  ros::WallTime, ros::WallDuration
- Message definition

```
int32 sec
int32 nsec
```

- Get current time

  ros::Time begin = ros::Time::now();

- Arithmetics

  1 hour + 1 hour = 2 hours (*duration + duration = duration*)
  2 hours - 1 hour = 1 hour (*duration - duration = duration*)
  Today + 1 day = tomorrow (*time + duration = time*)
  Today - tomorrow = -1 day (*time - time = duration*)
  Today + tomorrow = *error* (*time + time is undefined*)

- Conversion to floating point secods

```
double secs =ros::Time::now().toSec();

ros::Duration d(0.5);
secs = d.toSec();
```

Simulated Time in ROS

- For simulations or playback of logged data, it is convenient to work with a simulated time (pause, slow-down etc.)
- Using a simulated clock: rosparam set use_sim_time true
- When using simulated Clock time, now() returns time 0 until first message has been received on /clock, so 0 means essentially that the client does not know clock time yet.
- A value of 0 should therefore be treated differently, such as looping over now() until non-zero is returned.

```
ros::Time a_little_after_the_beginning(0.001);
```

ROS Bags

RECORDING BAG FILES

ROSBAG RECORD -O FILENAME.BAG TOPIC-NAMES

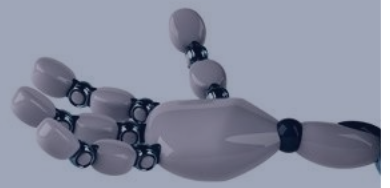
INSPECTING BAG FILES

ROSBAG INFO FILENAME.BAG

REPLAYING BAG FILES

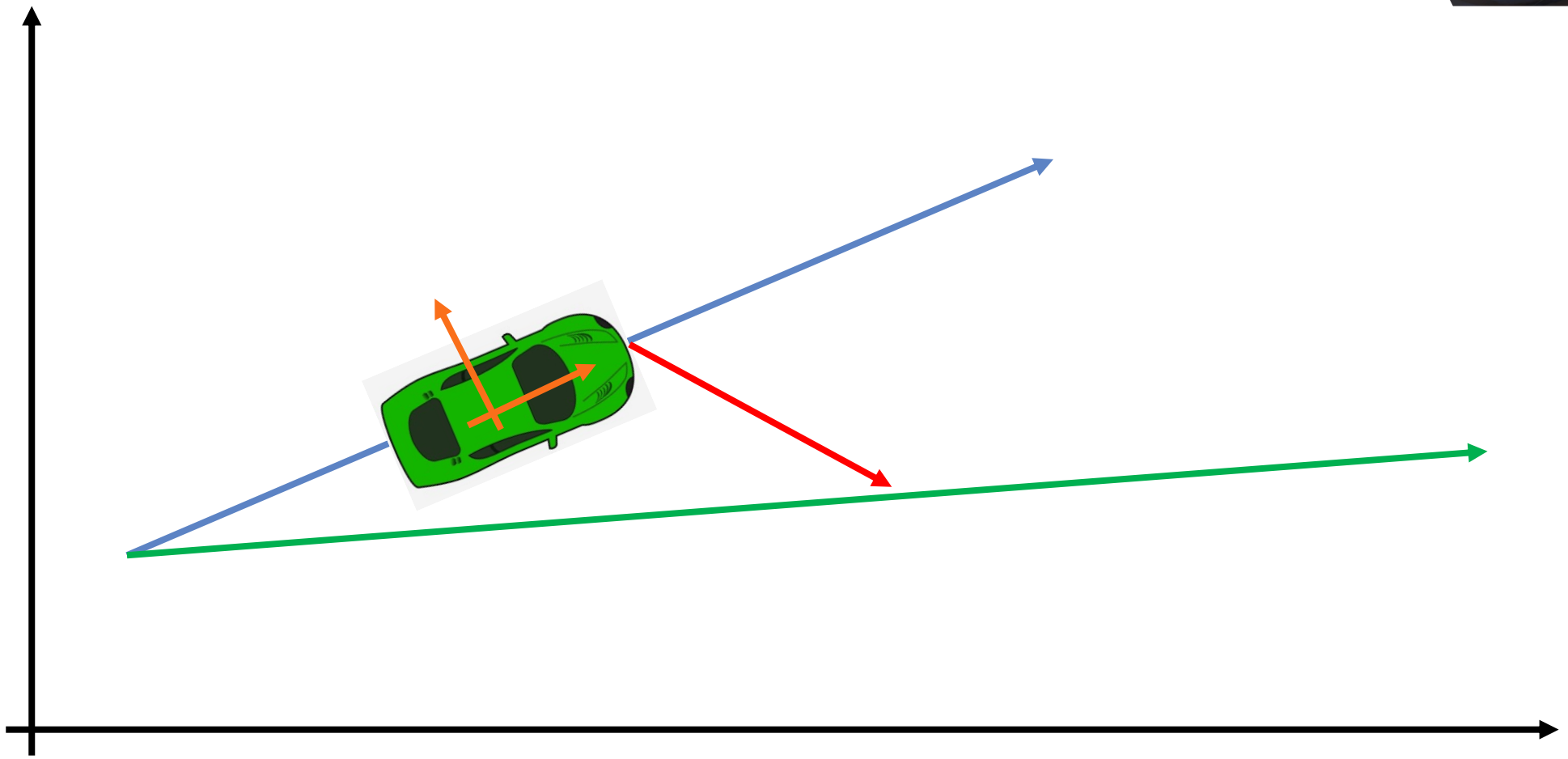
ROSBAG PLAY FILENAME.BAG

Coordinate Frames

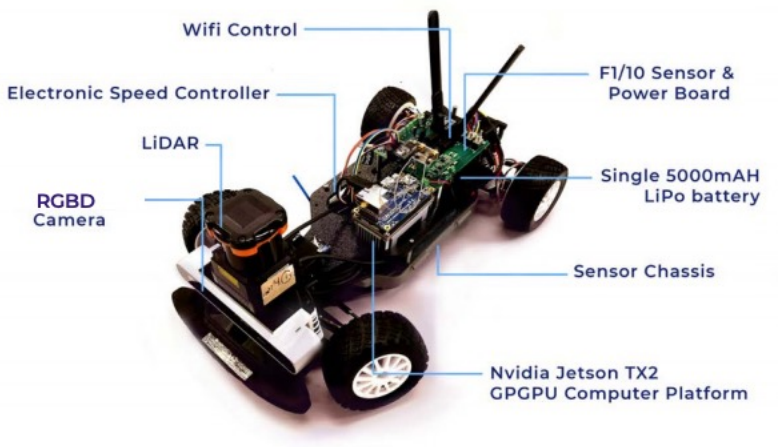


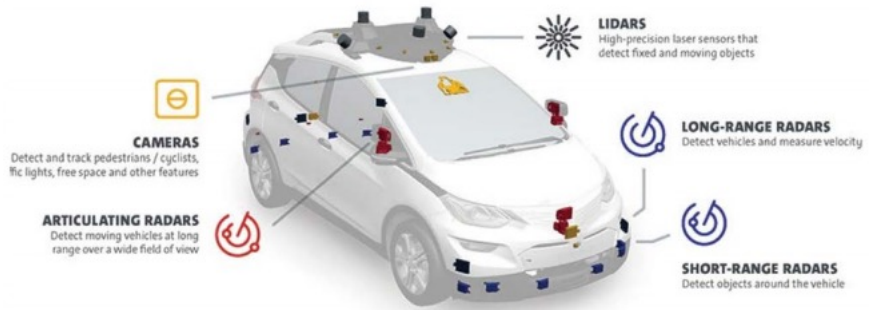
Theory

Example

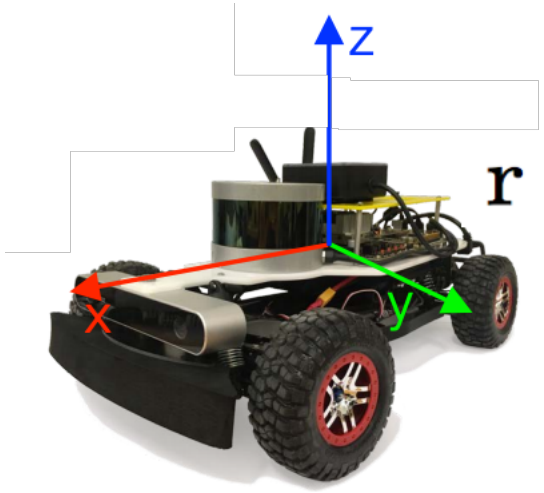


Sensor data


Wifi Control
Electronic Speed Controller
LiDAR
RGBD Camera
F1/10 Sensor & Power Board
Single 5000mAH LiPo battery
Sensor Chassis
Nvidia Jetson TX2 GPGPU Computer Platform

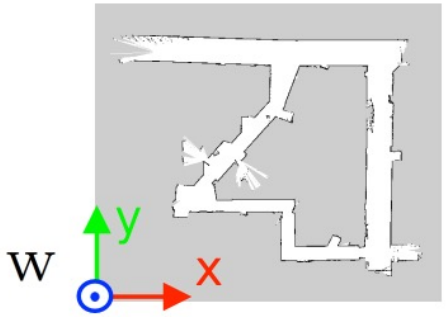

LIDARS
High-precision laser sensors that detect fixed and moving objects
CAMERAS
Detect and track pedestrians / cyclists, ffic lights, free space and other features
LONG-RANGE RADARS
Detect vehicles and measure velocity
ARTICULATING RADARS
Detect moving vehicles at long range over a wide field of view
SHORT-RANGE RADARS
Detect objects around the vehicle

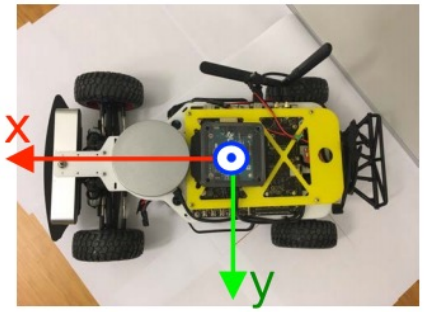
What is a coordinate frame?



- Set of orthogonal axes attached to an object that serves to describe position of points relative to it
- Origin – intersection point of the axes

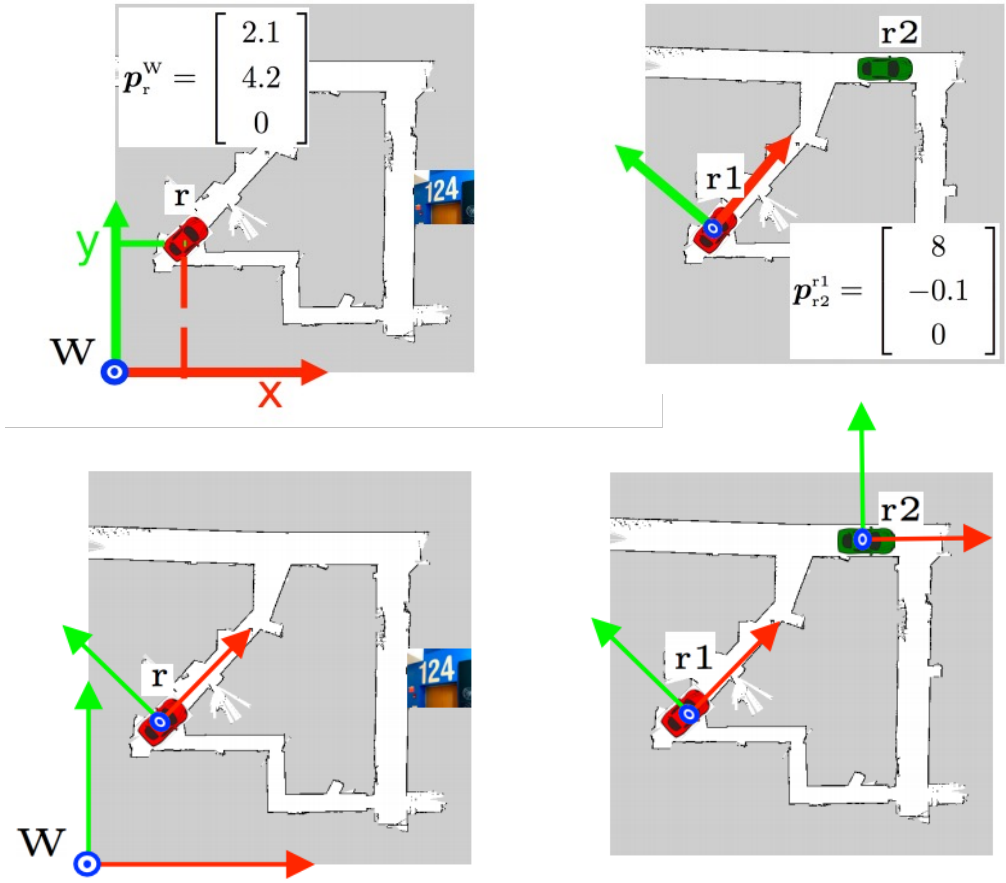
- First step is to assign a coordinate frame to all objects of interest





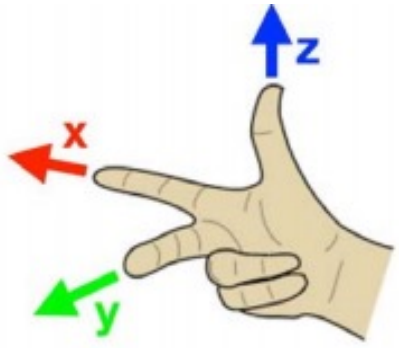
Coordinate frames representation

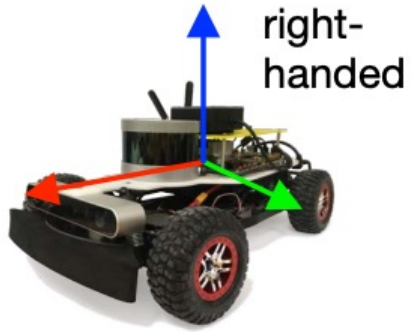
- Compact representation of points and orientations
- Coordinates are always considered in regard to a coordinate frame
- Coordinates have no meaning without specifying coordinate frame
- Pose = position + rotation

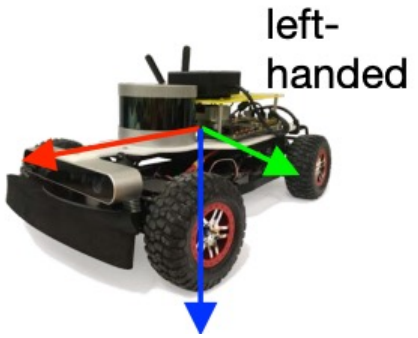


Left vs Right-handed Coordinate Frames

- Direction of axis is important
- Mnemonic:
  - positive x axis points along your index
  - finger, positive y axis points along your
  - middle finger.
  - In so-called "right-handed" coordinate systems, positive z-axis points along the thumb of your right hand.
  - In so-called "left-handed" coordinate systems, positive z-axis points along the thumb of your left hand.
- robotics always uses right-handed coordinate systems
- left-handed systems are sometimes used in graphics



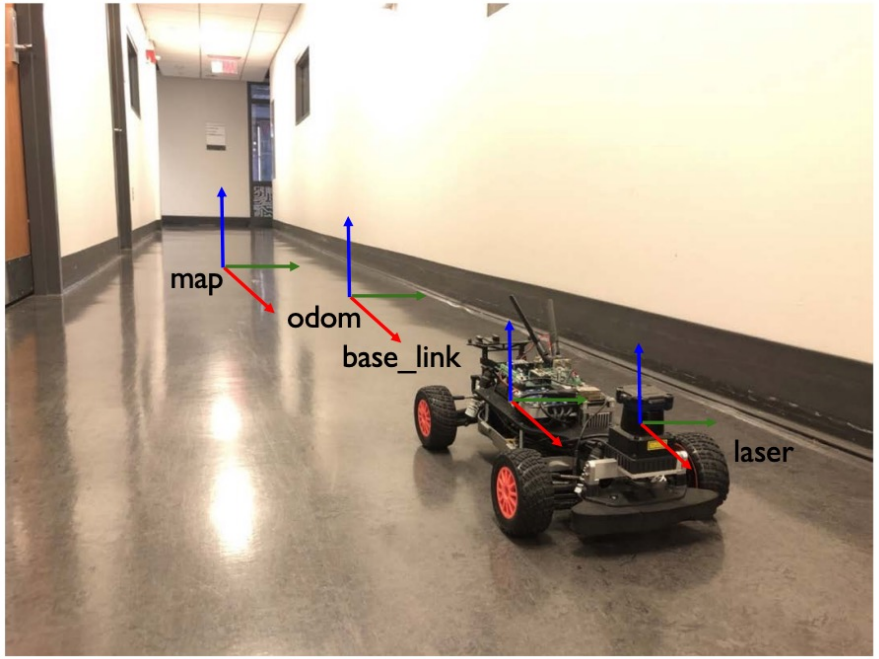




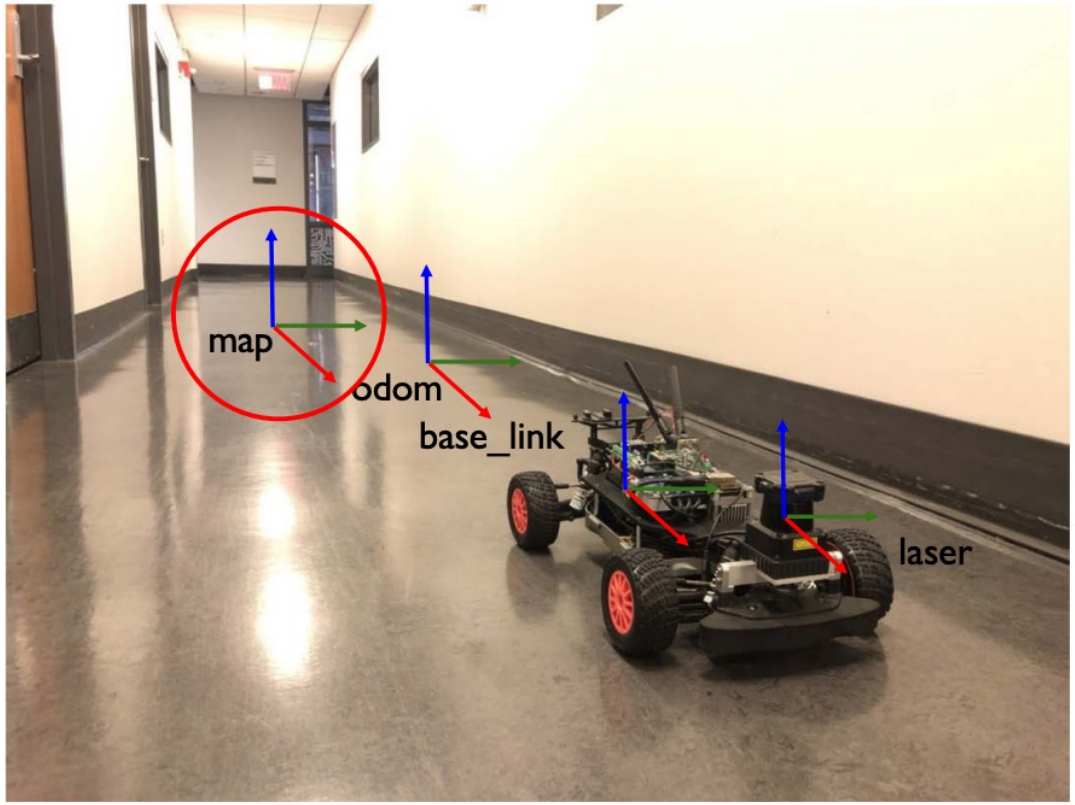
ROS Coordinate Frames

- Usually in ROS one can find 4 coordinate frames
  - Map frame
  - Robot Base (base_link) frame
  - Tool frame
  - Odometry frame
- Odometry = distance measurement technique for vehicles and pedestrians

RGB → XYZ

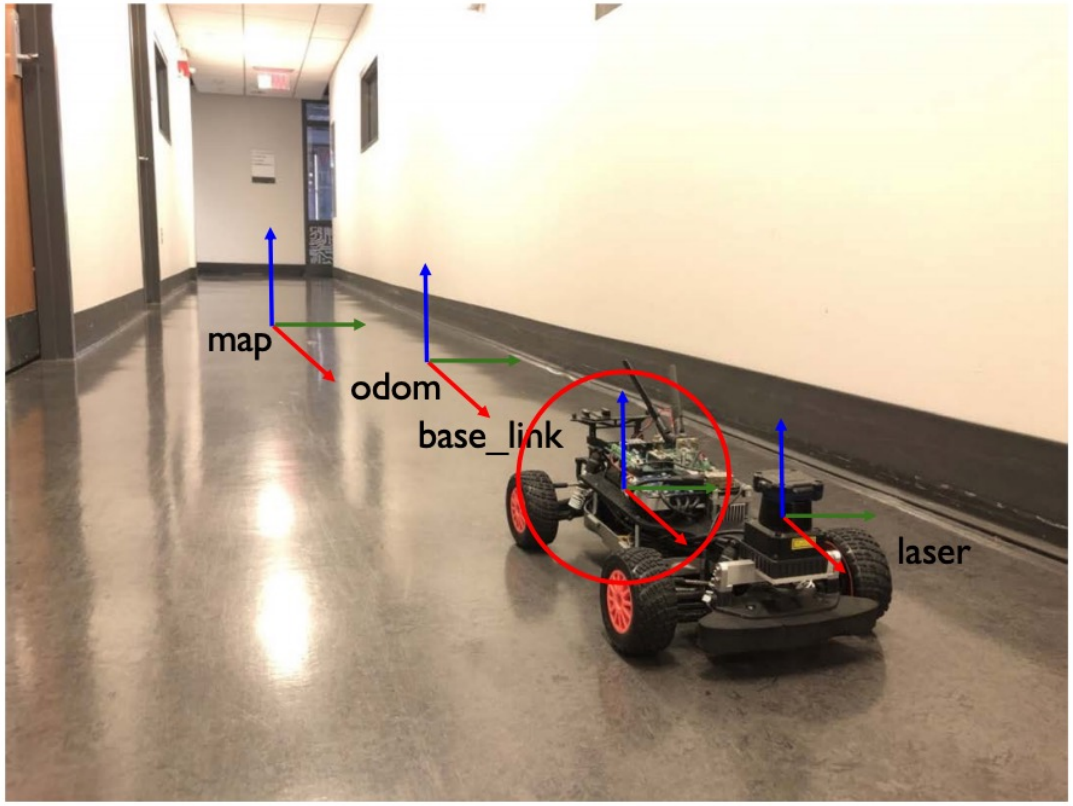


Map frame



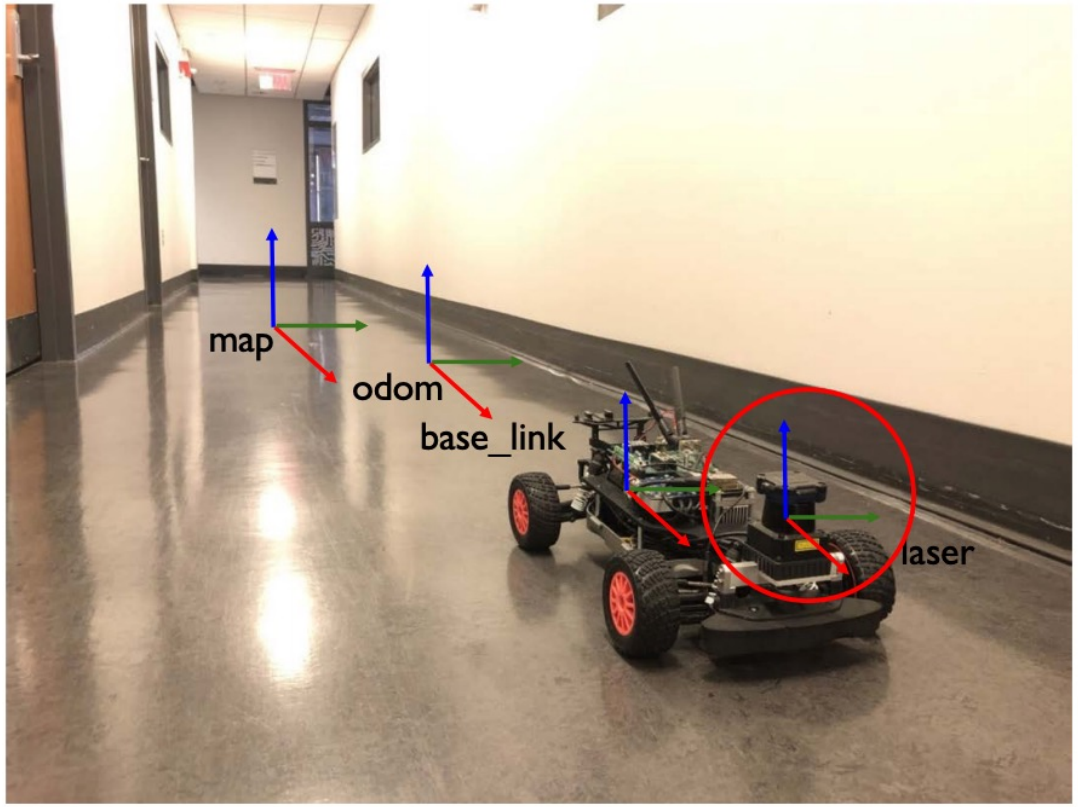
- Can be set arbitrarily
- Represents the environment around the robot
- Static frame
- Also called **global frame** or **environment frame**

Base_link frame



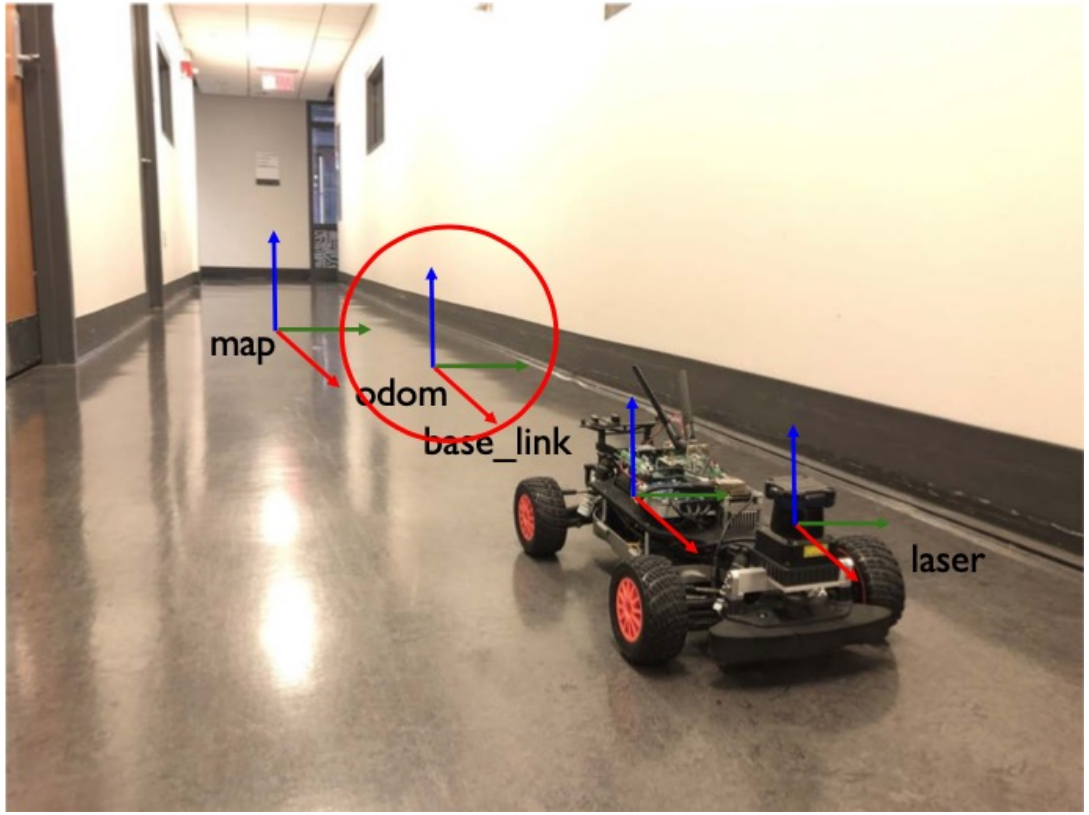
- Defined as the main frame for the robot/object (eg.: center of the rear axle)
- Moves together with the object
- The reference point for all the robot's parts

Tool frame



- The frame in which actions are performed or sensors provides data
- Can have a fixed or variable position relative to the base_link
- Eg. Laser frame

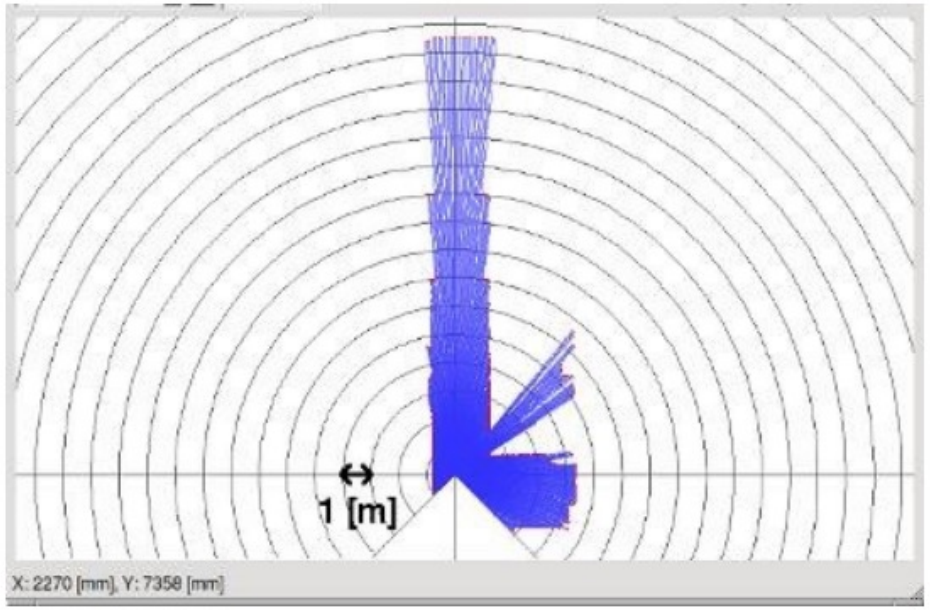
Odometry frame

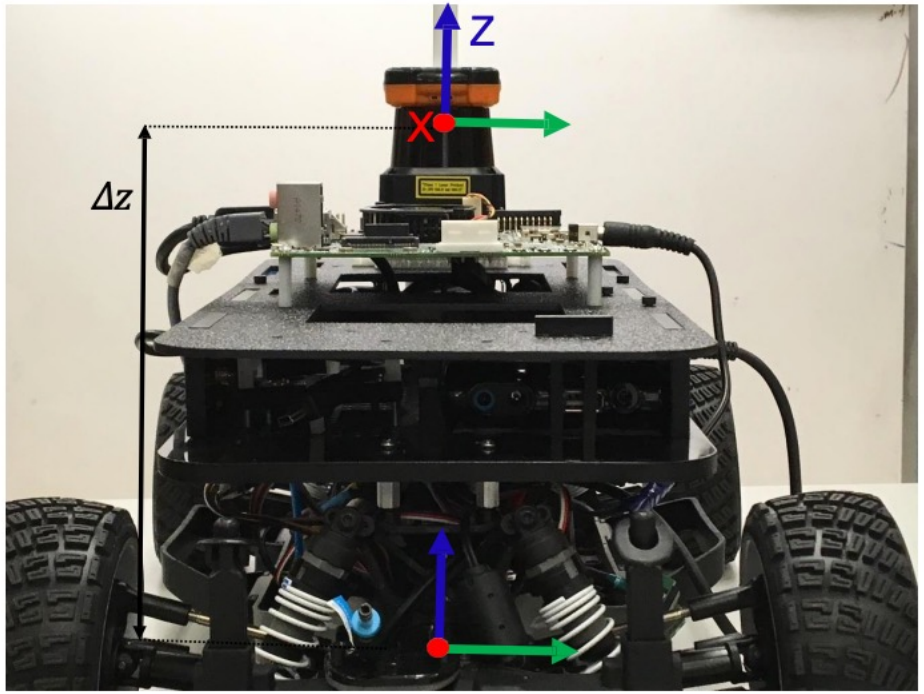


- Frame in which all measurements are taken
- Fixed relative to the map
- E.g: Origin point where the car started
- Not explicitly defined
- Optional frame

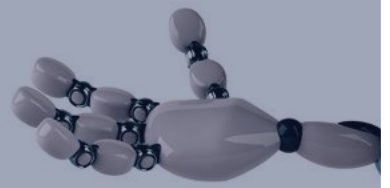
Why do we need coordinate frames

- Distance measurements returned by a LIDAR





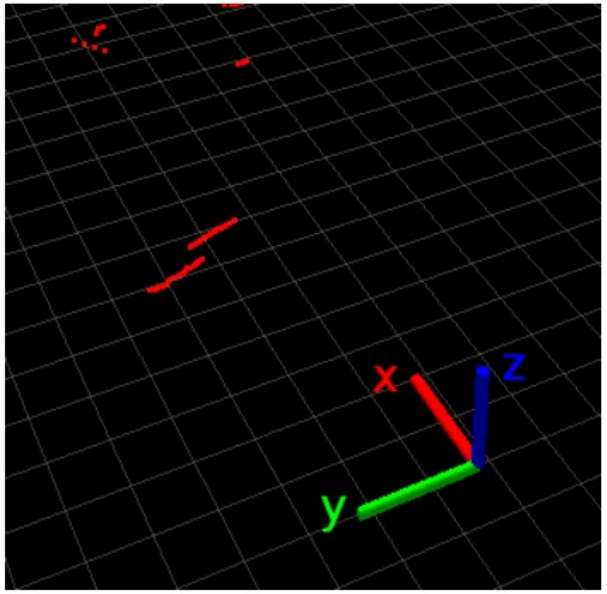
Transformations

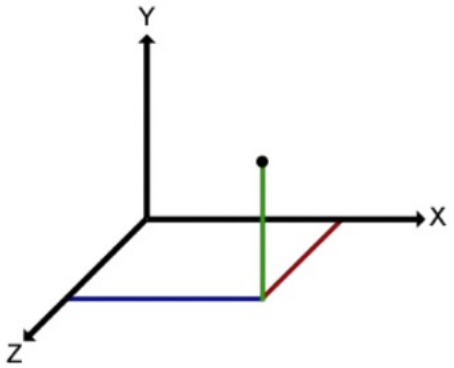


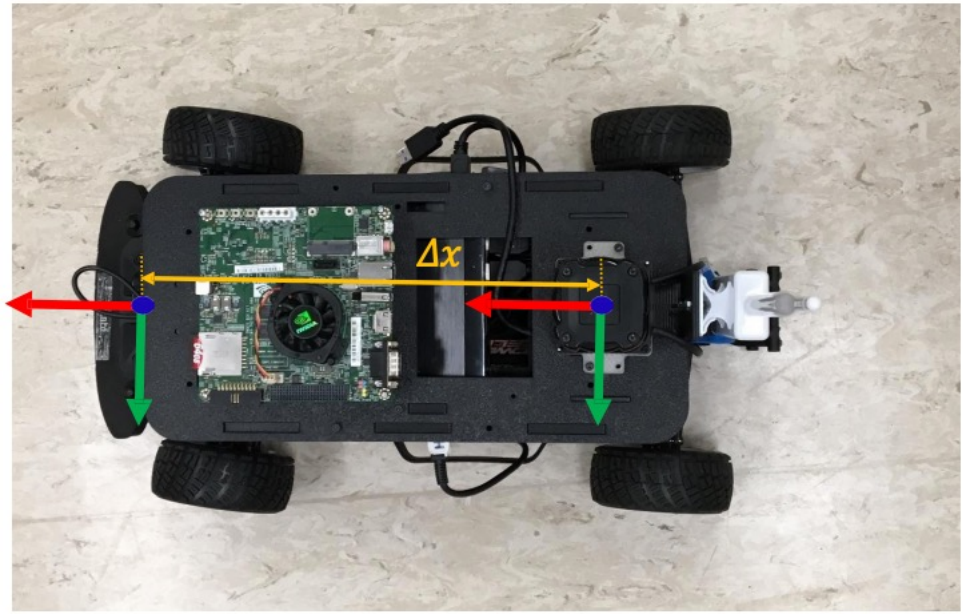
Rotation. Translation

Transformations

- Measurements are always taken in the reference frame
- The LIDAR's scanned values do not tell how far the objects are and offsets need to be calculated
- Transformations => convert coordinates from one frame to another

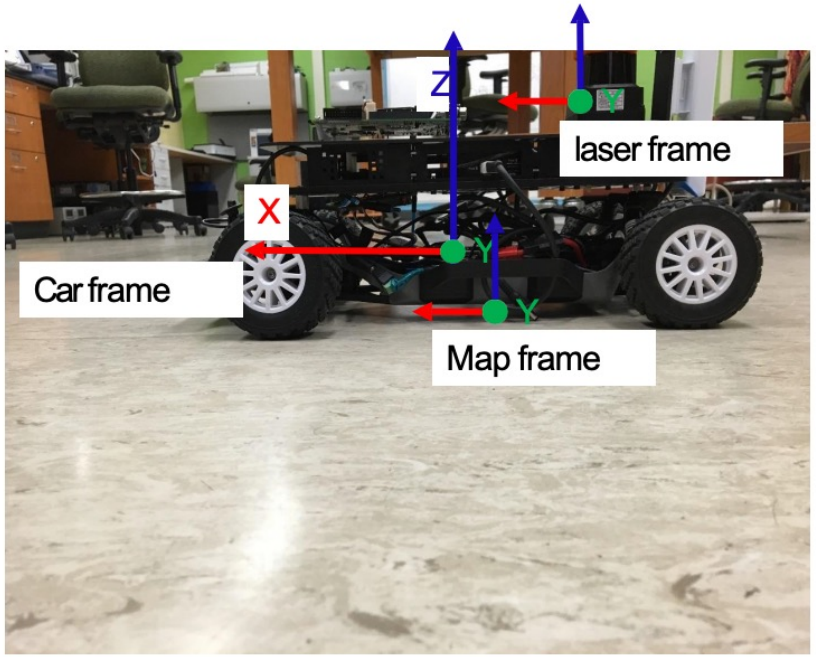


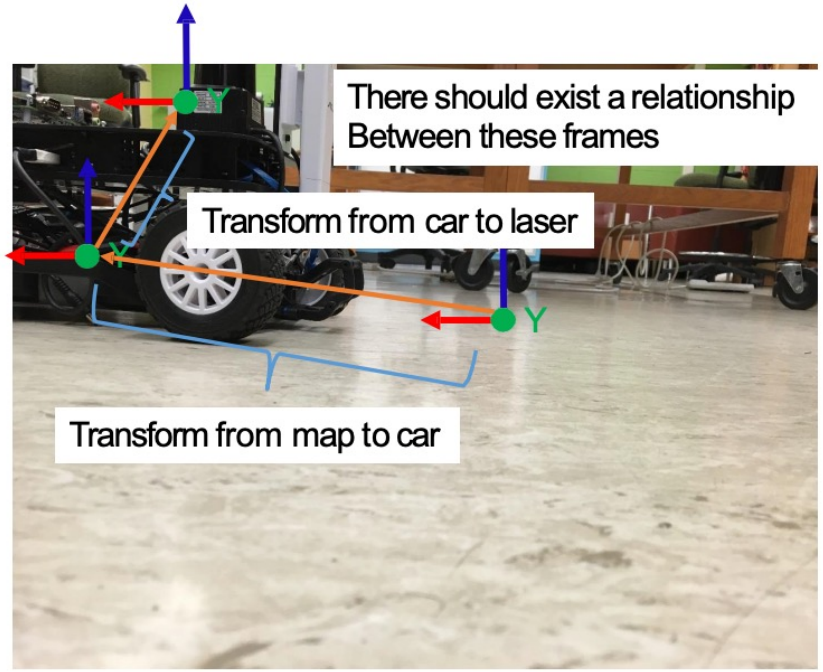




Transformations

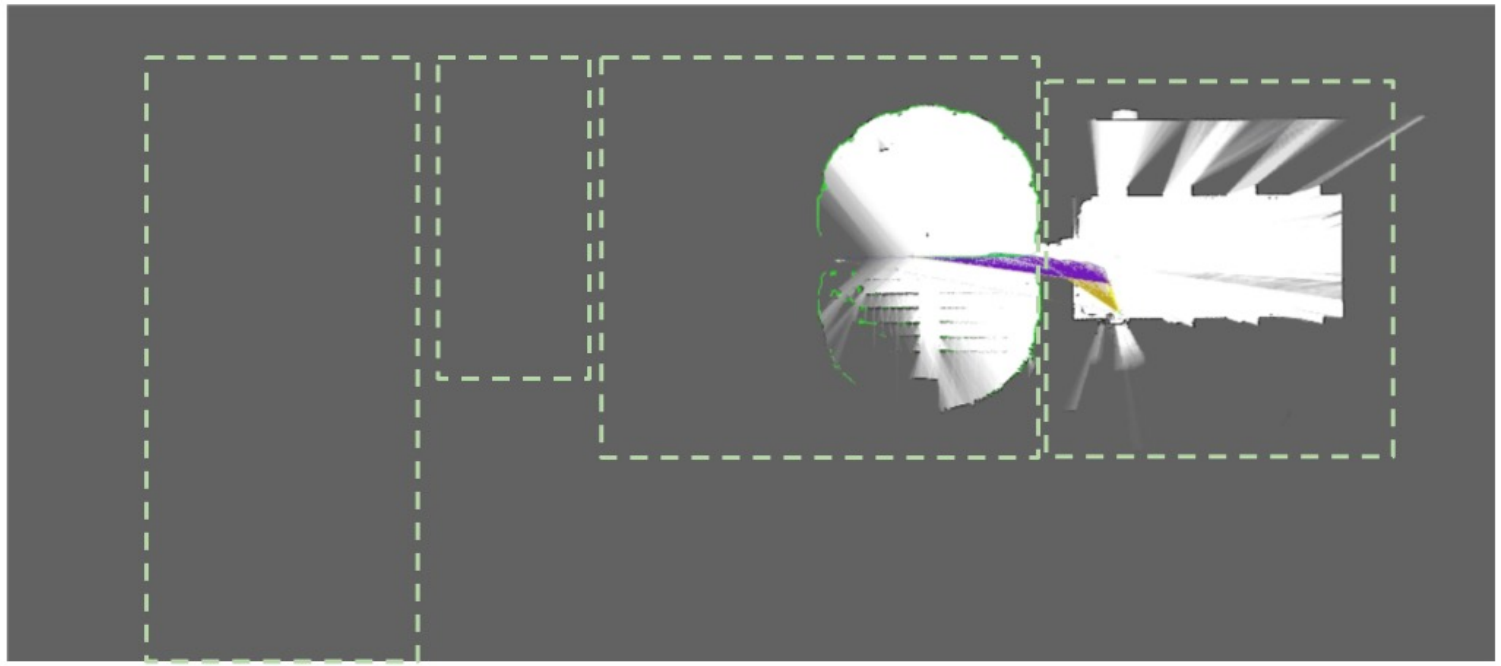
- Define relations between frames
- Convert measurements from one frame to another
- Eg: Relationship between the measurement frame and the actuation frame





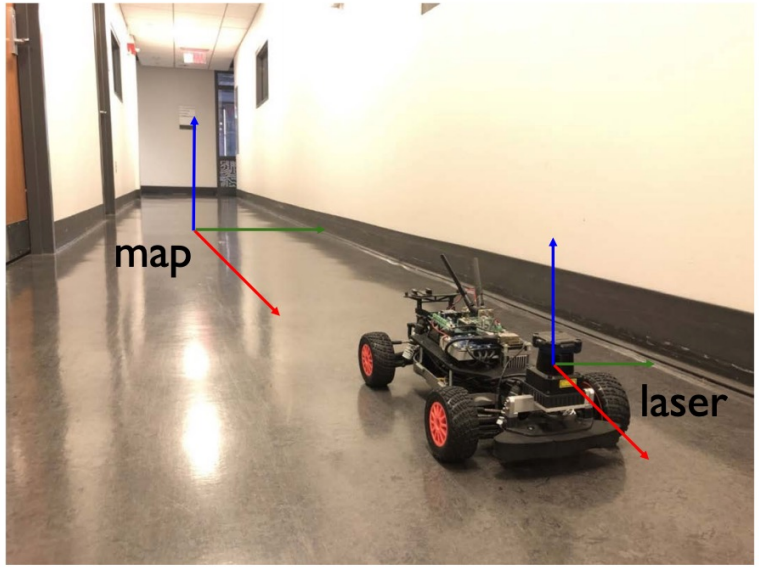
Transformations

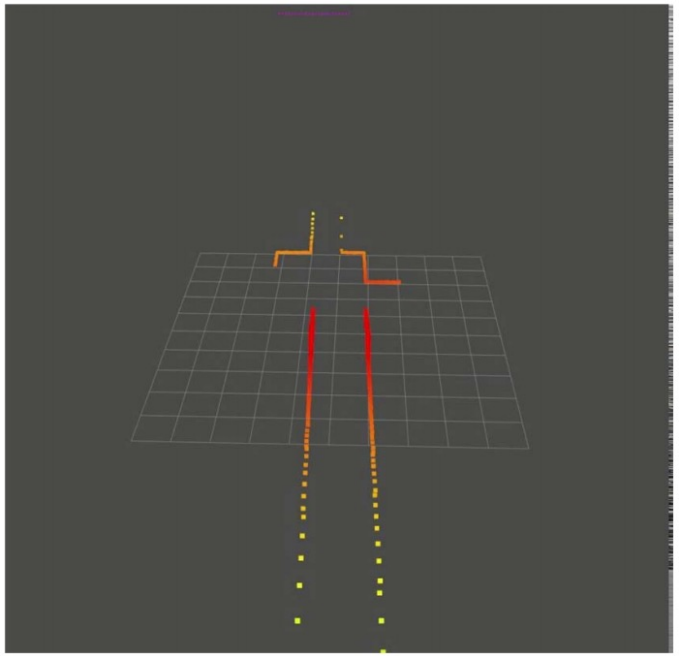
- E.g. Mapping is usually done by tying submaps together

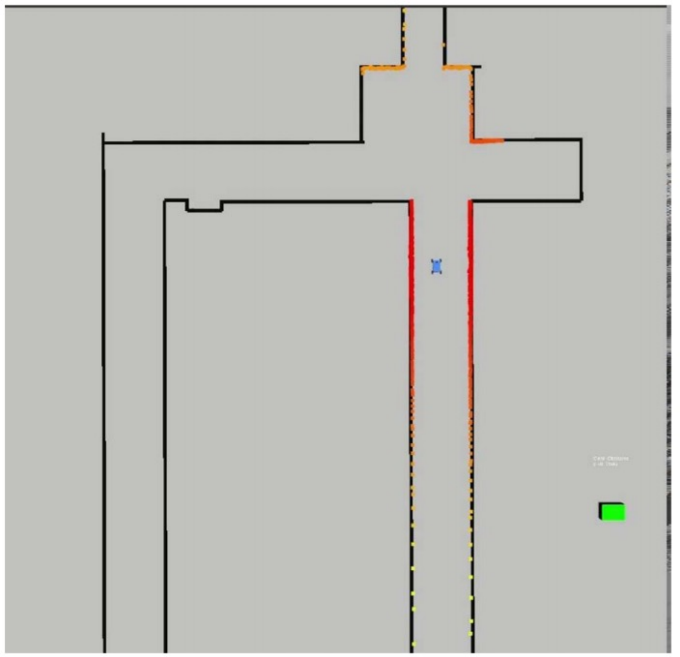


Transformations

- Giving sense to collected data (submaps)
- Left: 2 coordinate frames: map + laser
- Middle: Laser frame
- Right: Map frame

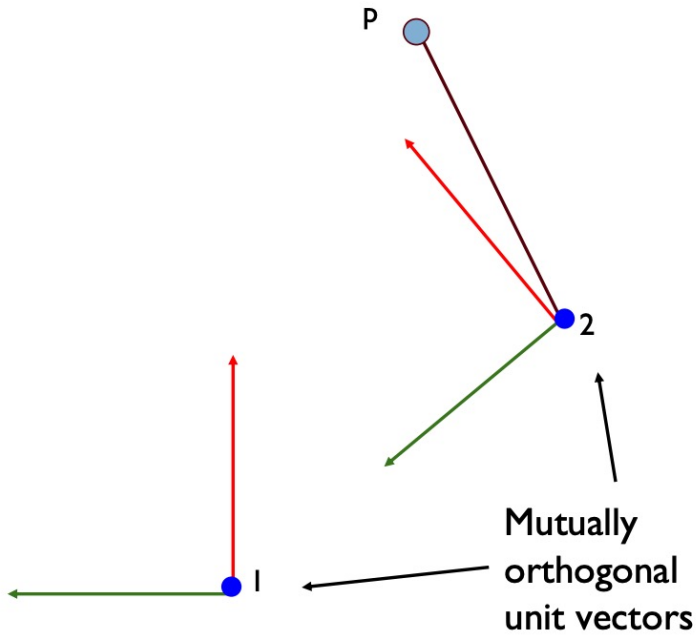






Rigid Body Transforms

- Find the coordinate point of p in frame 2, given the coordinate point p in frame 1

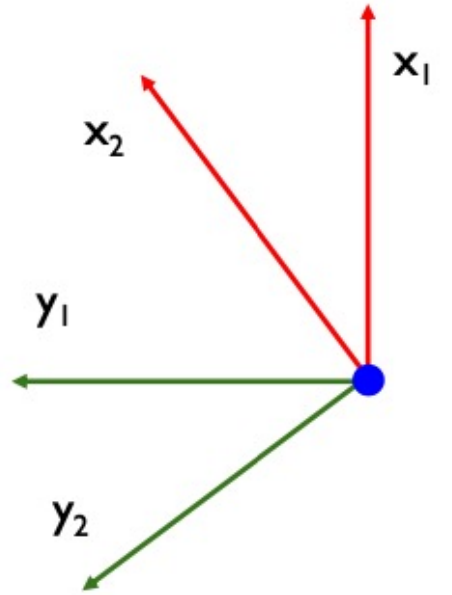


Rigid Body Transforms

- Find the coordinate point of p in frame 2, given the coordinate point p in frame 1
- The mutually orthogonal unit vectors in one frame, can be calculated as linear combinations of the mutually orthogonal unit vectors in the other frame

$$\mathbf{x}_2 = R_{11}\mathbf{x}_1 + R_{21}\mathbf{y}_1$$
$$\mathbf{y}_2 = R_{12}\mathbf{x}_1 + R_{22}\mathbf{y}_1$$

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}$$

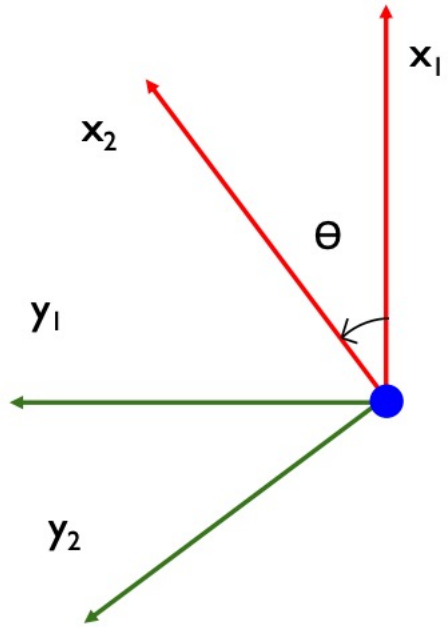


Rotation Matrix

$$\mathbf{x}_2 = cos(\theta)\mathbf{x}_1 + sin(\theta)\mathbf{y}_1$$
$$\mathbf{y}_2 = -sin(\theta)\mathbf{x}_1 + cos(\theta)\mathbf{y}_1$$

$$\mathbf{R} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

x y

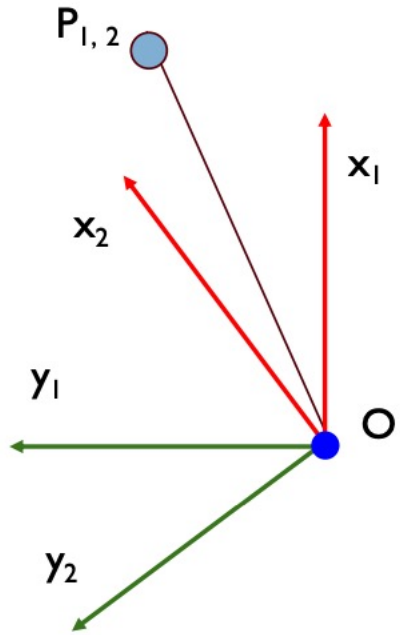


Rotation Matrix

- Same as in case of the axes
- P1 – point representation in O1
- P2 – point representation in O2

$$OP_1 = p_{x1}x_1 + p_{y1}y_1$$

$$OP_2 = p_{x2}x_2 + p_{y2}y_2$$



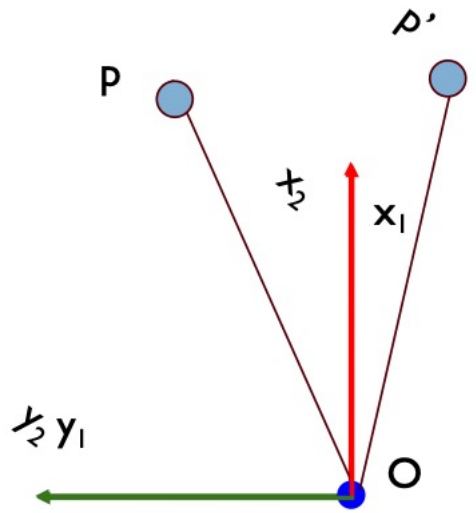
Rotation Matrix

- If we would consider the same coordinate frame, then we would get the following points P and P'

$$OP = p_x x_1 + p_y y_1$$
$$OP' = p'_x x_1 + p'_y y_1$$

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} p'_x \\ p'_y \end{bmatrix}$$

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix} \begin{bmatrix} p'_x \\ p'_y \end{bmatrix}$$



3D Rotation Matrix

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

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

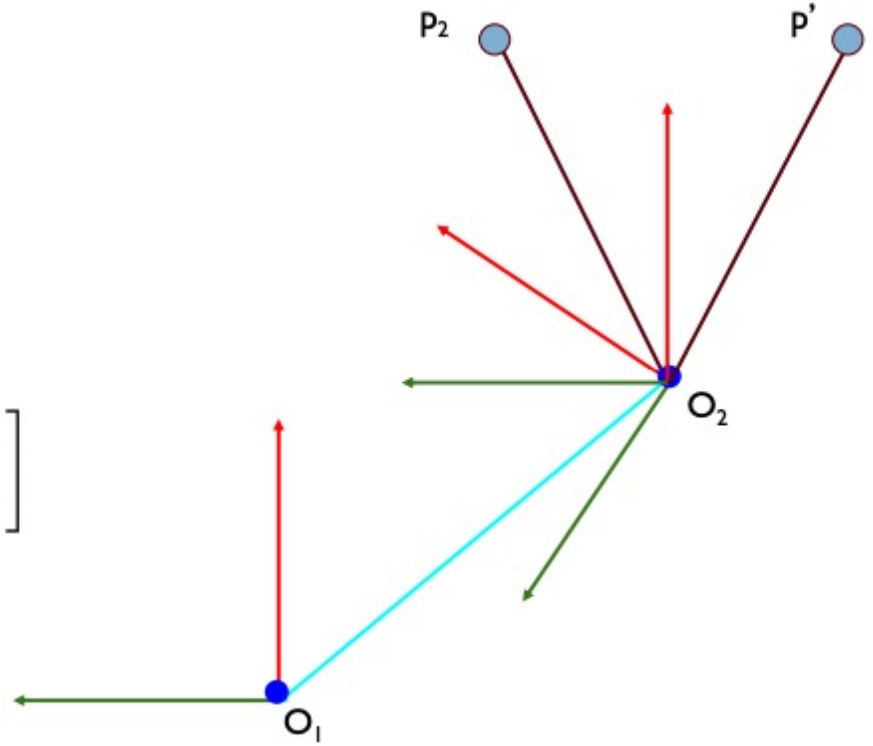
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation and Translation

$$p' = \mathbf{R}_2^1 p_2$$

$$p_1 = \mathbf{R}_2^1 p_2 + O_{21}$$

$$p_1 = \mathbf{H}_2^1 p_2 = \begin{bmatrix} \mathbf{R}_2^1 & O_{21} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} p_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_2^1 p_2 + O_{21} \\ 1 \end{bmatrix} = \begin{bmatrix} p_1 \\ 1 \end{bmatrix}$$



Homogeneous Transformation

- A Homogeneous Transformation is a matrix representation of a rigid body transformation:

$$\mathbf{H} = \begin{bmatrix} \mathbf{R} & \mathbf{d} \\ \mathbf{0} & 1 \end{bmatrix}$$

where **R** is a 2x2 (2D) / 3x3 (3D) rotation matrix and **d** is a 2x1 (2D) / 3x1 (3D) displacement vector

- The Homogeneous Representation of a vector is

$$\mathbf{P} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$$

where p is a 2x1 (2D) / 3x1 (3D) vector

Robot Description

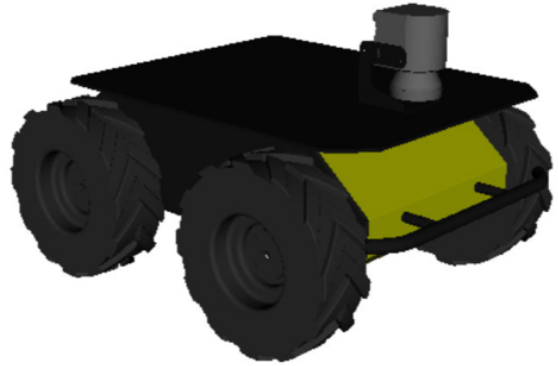
URDF. SDF

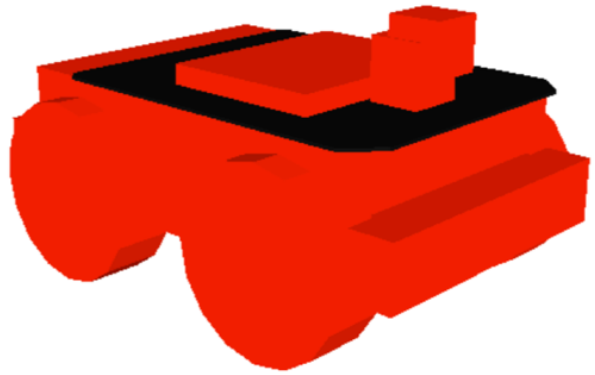
Robot models

- Unified Robot Description Format (URDF)

  http://wiki.ros.org/urdf
- XML file for representing a robot model
- Kinematic and dynamic description
- Visual representation (mesh)
- Collision model (primitives)
- URDF generation can be be scripted with XACRO

  http://wiki.ros.org/xacro

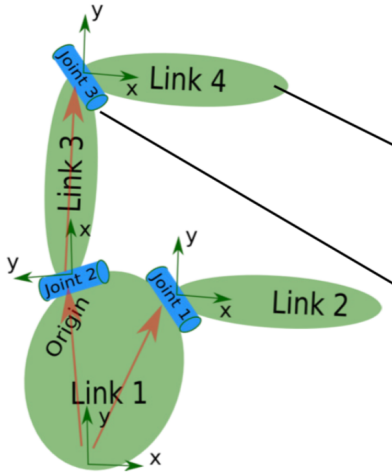




Unified Robot Description Format

Description consists of a set of *link* elements and a set of *joint* elements

Joints connect the links together



robot.urdf

```
<robot name="robot">
  <link> ... </link>
  <link> ... </link>
  <link> ... </link>

  <joint> .... </joint>
  <joint> .... </joint>
  <joint> .... </joint>
</robot>
```

```
<link name="link_name">
  <visual>
    <geometry>
      <mesh filename="mesh.dae"/>
    </geometry>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.6" radius="0.2"/>
    </geometry>
  </collision>
  <inertial>
    <mass value="10"/>
    <inertia ixx="0.4" ixy="0.0" …/>
  </inertial>
</link>
```

```
<joint name="joint_name" type="revolute">
  <axis xyz="0 0 1"/>
  <limit effort="1000.0" upper="0.548" … />
  <origin rpy="0 0 0" xyz="0.2 0.01 0"/>
  <parent link="parent_link_name"/>
  <child link="child_link_name"/>
</joint>
```

Usage

- The robot description (URDF) is stored on the parameter server (typically) under /robot_description
- Robot model visualization in RViz with the RobotModel plugin

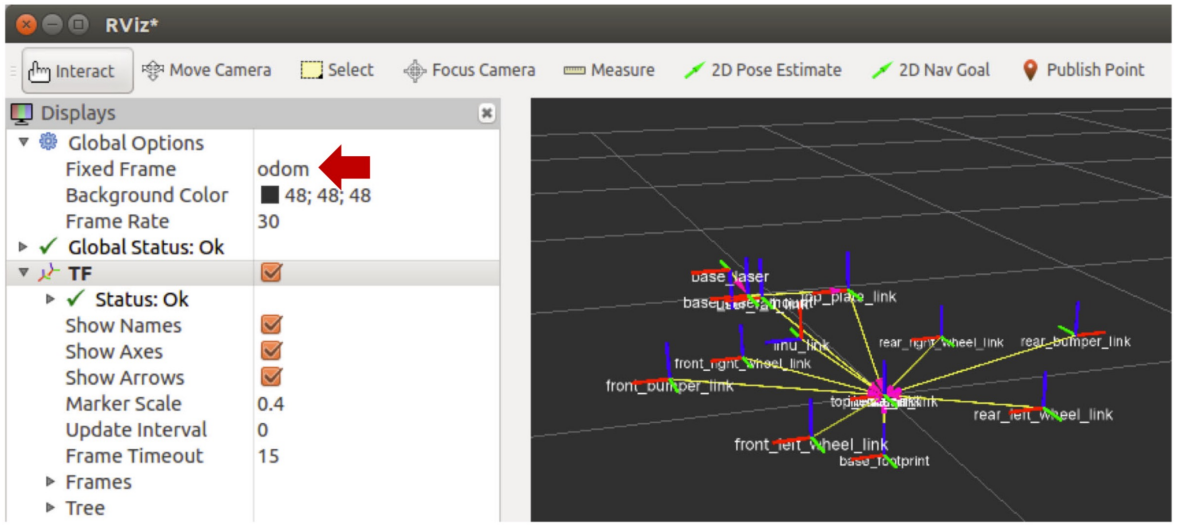
husky_empty_world.launch

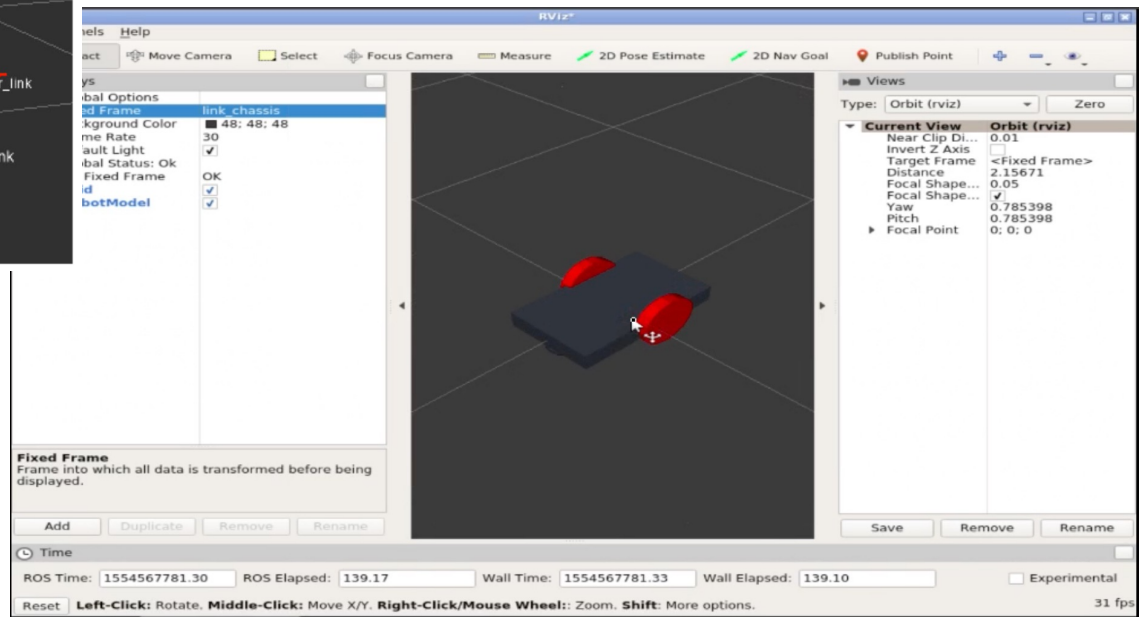
```
…
<include file="$(find husky_gazebo)/launch/spawn_husky.launch">
  <arg name="laser_enabled" value="$(arg laser_enabled)"/>
  <arg name="ur5_enabled" value="$(arg ur5_enabled)"/>
  <arg name="kinect_enabled" value="$(arg kinect_enabled)"/>
</include>
…
```

spawn_husky.launch

```
…
<param name="robot_description" command="$(find xacro)/xacro.py
'$(arg husky_gazebo_description)'
   laser_enabled:=$(arg laser_enabled)
   ur5_enabled:=$(arg ur5_enabled)
   kinect_enabled:=$(arg kinect_enabled)" />
…
```

Visualization - RViz





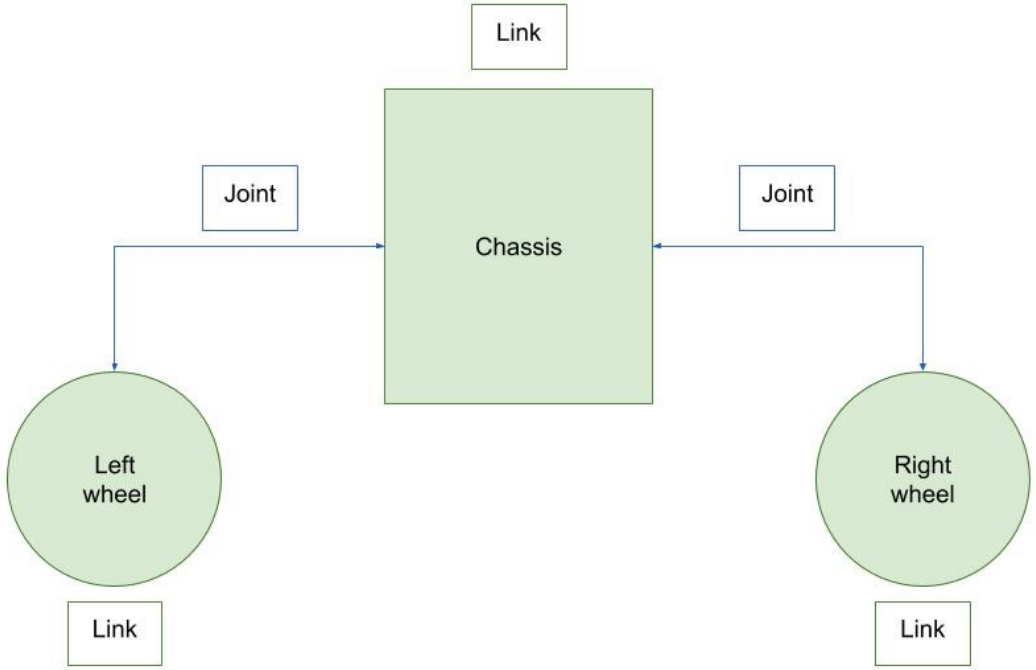
Simulation Description Format

- The simulation description format (SDF) is an XML that can describe:
  - Environments (lighting, gravity etc.)
  - Objects (static and dynamic)
  - Sensors
  - Robots
- SDF is the standard format for Gazebo
- Gazebo converts a URDF to SDF automatically

http://sdformat.org

Example

- A robot composed by 3 links and 2 joints
- Every robot needs a base link => the chassis is in charge of connecting all the parts of the robot
- Links in green, Joints in blue.



Example

```
<?xml version="1.0" ?>
<robot name="m2wr" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <material name="black">
    <color rgba="0.0 0.0 0.0 1.0"/>
  </material>
  <material name="blue">
    <color rgba="0.203125 0.23828125 0.28515625 1.0"/>
  </material>


  <gazebo reference="link_chassis">
    <material>Gazebo/Orange</material>
  </gazebo>
  <gazebo reference="link_left_wheel">
    <material>Gazebo/Blue</material>
  </gazebo>
  <gazebo reference="link_right_wheel">
    <material>Gazebo/Blue</material>
  </gazebo>
```

```
<link name="link_chassis">
  <!-- pose and inertial -->
  <pose>0 0 0.1 0 0 0</pose>
  <inertial>
    <mass value="5"/>
    <origin rpy="0 0 0" xyz="0 0 0.1"/>
    <inertia ixx="0.0395416666667" ixy="0" ixz="0" iyy="0.106208333333" iyz="0" izz="0.106208333333"/>
  </inertial>
  <!-- body -->
  <collision name="collision_chassis">
    <geometry>
      <box size="0.5 0.3 0.07"/>
    </geometry>
  </collision>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <box size="0.5 0.3 0.07"/>
    </geometry>
    <material name="blue"/>
  </visual>
  <!-- caster front -->
  <collision name="caster_front_collision">
    <origin rpy=" 0 0 0" xyz="0.35 0 -0.05"/>
    <geometry>
      <sphere radius="0.05"/>
    </geometry>
    <surface>
      <friction>
        <ode>
          <mu>0</mu>
          <mu2>0</mu2>
          <slip1>1.0</slip1>
          <slip2>1.0</slip2>
        </ode>
      </friction>
    </surface>
  </collision>
  <visual name="caster_front_visual">
    <origin rpy=" 0 0 0" xyz="0.2 0 -0.05"/>
    <geometry>
      <sphere radius="0.05"/>
    </geometry>
  </visual>
</link>
```

Example

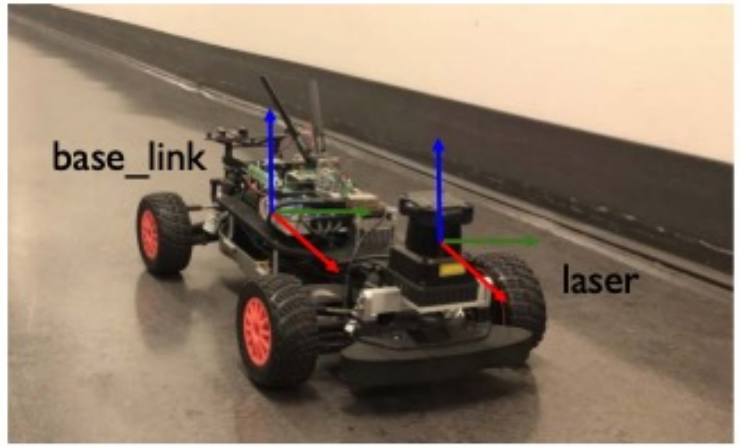
```
<link name="link_right_wheel">
  <inertial>
    <mass value="0.2"/>
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="0.000526666666667" iyz="0" izz="0.001"/>
  </inertial>
  <collision name="link_right_wheel_collision">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>
      <cylinder length="0.04" radius="0.1"/>
    </geometry>
  </collision>
  <visual name="link_right_wheel_visual">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>
      <cylinder length="0.04" radius="0.1"/>
    </geometry>
  </visual>
</link>

<joint name="joint_right_wheel" type="continuous">
  <origin rpy="0 0 0" xyz="-0.05 0.15 0"/>
  <child link="link_right_wheel"/>
  <parent link="link_chassis"/>
  <axis rpy="0 0 0" xyz="0 1 0"/>
  <limit effort="10000" velocity="1000"/>
  <joint_properties damping="1.0" friction="1.0"/>
</joint>
```

```
<link name="link_left_wheel">
  <inertial>
    <mass value="0.2"/>
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="0.000526666666667" iyz="0" izz="0.001"/>
  </inertial>
  <collision name="link_left_wheel_collision">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>
      <cylinder length="0.04" radius="0.1"/>
    </geometry>
  </collision>
  <visual name="link_left_wheel_visual">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>
      <cylinder length="0.04" radius="0.1"/>
    </geometry>
  </visual>
</link>

<joint name="joint_left_wheel" type="continuous">
  <origin rpy="0 0 0" xyz="-0.05 -0.15 0"/>
  <child link="link_left_wheel"/>
  <parent link="link_chassis"/>
  <axis rpy="0 0 0" xyz="0 1 0"/>
  <limit effort="10000" velocity="1000"/>
  <joint_properties damping="1.0" friction="1.0"/>
</joint>
</robot>
```

Example 2

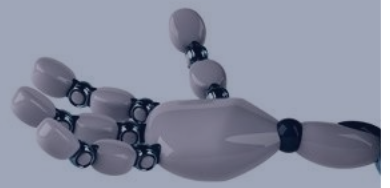



```
<joint name="base_to_laser_model" type="fixed">
  <parent link="base_link"/>
  <child link="laser_model"/>
  <origin xyz="${laser_distance_from_base_link} 0 ${ground_offset+height+(laser_height/2)}"/>
</joint>
```

Tutorials

- http://wiki.ros.org/urdf/Tutorials
- http://gazebosim.org/tutorials?tut=build_model

ROS Transform Frame



TF Transformation System

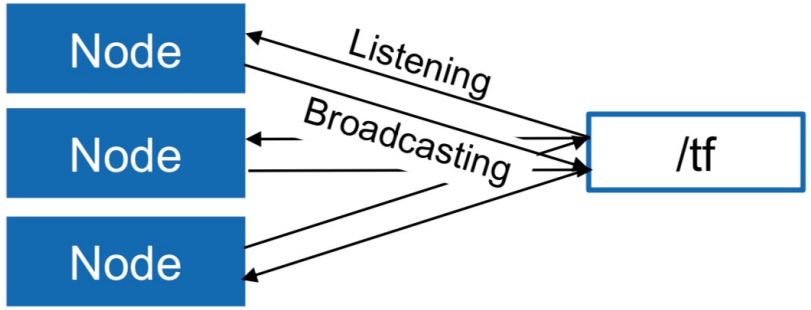
Usual questions

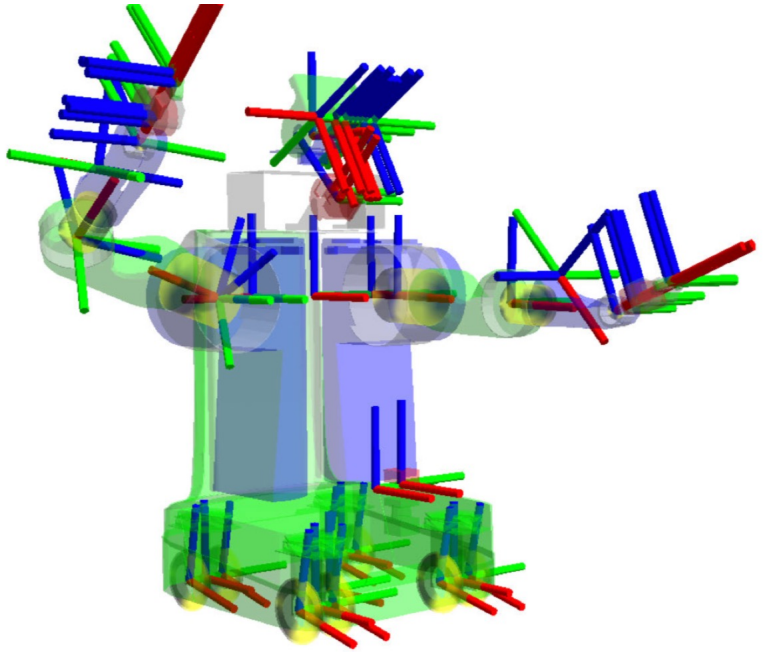
- Where was the head frame relative to the world frame, 5 seconds ago?
- What is the pose of the object in my gripper relative to my base?
- What is the current pose of the base frame in the map frame?

TF Transformation System

- A standard method in order to keep track of coordinate frames and transform data within the entire system over time
- Individual component users can be confident about the consistency of their data in a particular coordinate frame without requiring knowledge about all the other coordinate frames in the system and their associations
- Coordinate frames relationship in a tree structure buffered in time
- Publisher/subscriber using /tf and /tf_static

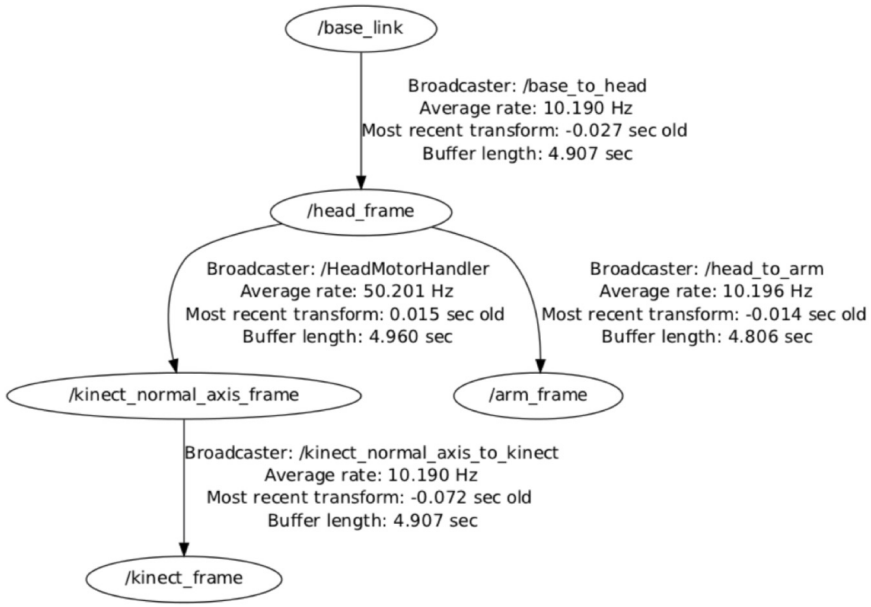
http://wiki.ros.org/tf2





Transform Tree

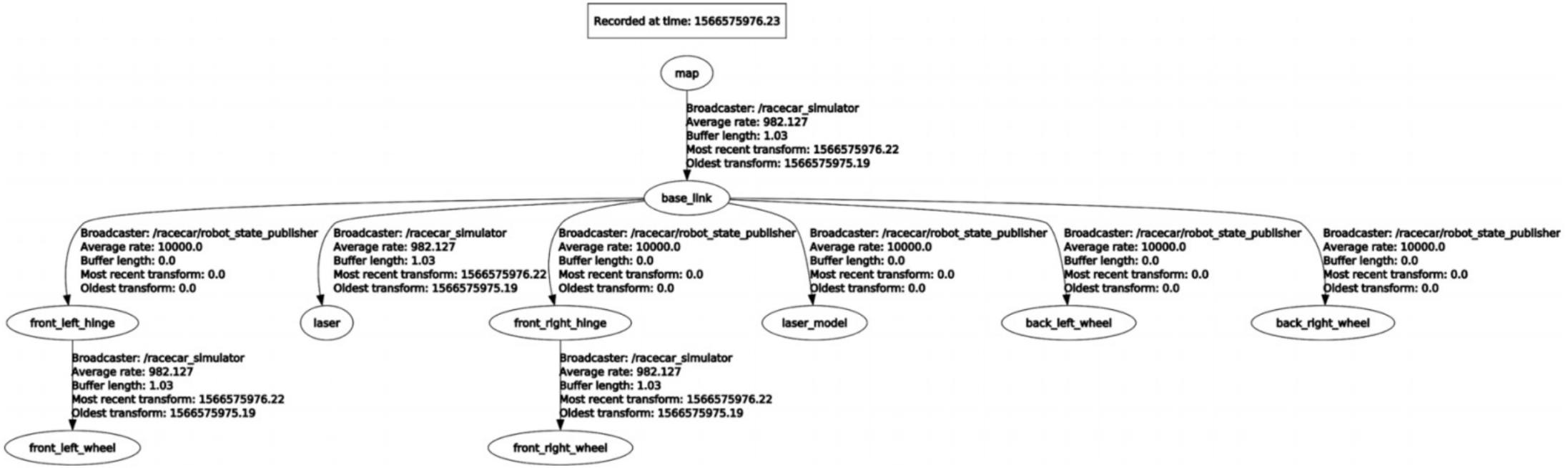
- TF listeners use a buffer to listen to all broadcasted transforms
- Query for specific transforms from the transform tree



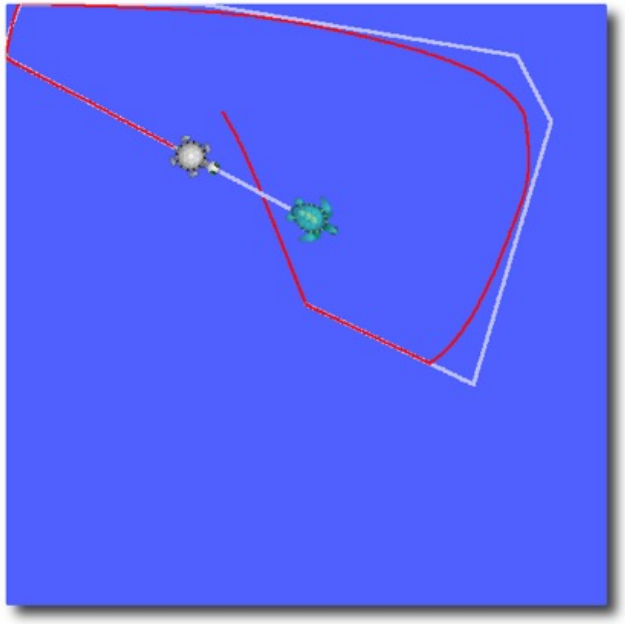
tf2_msgs/TFMessage.msg

```
geometry_msgs/TransformStamped[] transforms
  std_msgs/Header header
    uint32 seqtime stamp
    string frame_id
  string child_frame_id
  geometry_msgs/Transform transform
    geometry_msgs/Vector3 translation
    geometry_msgs/Quaternion rotation
```

Car example

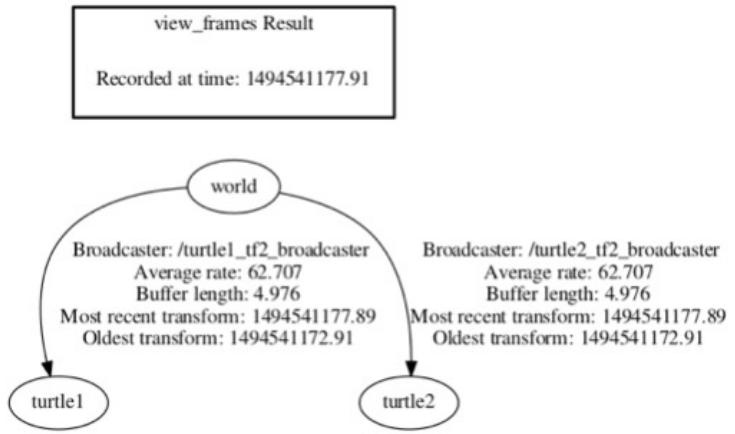

Recorded at time: 1566575976.23
map
Broadcaster: /racecar_simulator
Average rate: 982.127
Buffer length: 1.03
Most recent transform: 1566575976.22
Oldest transform: 1566575975.19
base_link
Broadcaster: /racecar/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /racecar_simulator
Average rate: 982.127
Buffer length: 1.03
Most recent transform: 1566575976.22
Oldest transform: 1566575975.19
Broadcaster: /racecar/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /racecar/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /racecar/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /racecar/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
front_left_hinge
laser
front_right_hinge
laser_model
back_left_wheel
back_right_wheel
Broadcaster: /racecar_simulator
Average rate: 982.127
Buffer length: 1.03
Most recent transform: 1566575976.22
Oldest transform: 1566575975.19
Broadcaster: /racecar_simulator
Average rate: 982.127
Buffer length: 1.03
Most recent transform: 1566575976.22
Oldest transform: 1566575975.19
front_left_wheel
front_right_wheel

Turtle Example



$$\mathbf{T}_{turtle1\_turtle2} = \mathbf{T}_{turtle1\_world} * \mathbf{T}_{world\_turtle2}$$

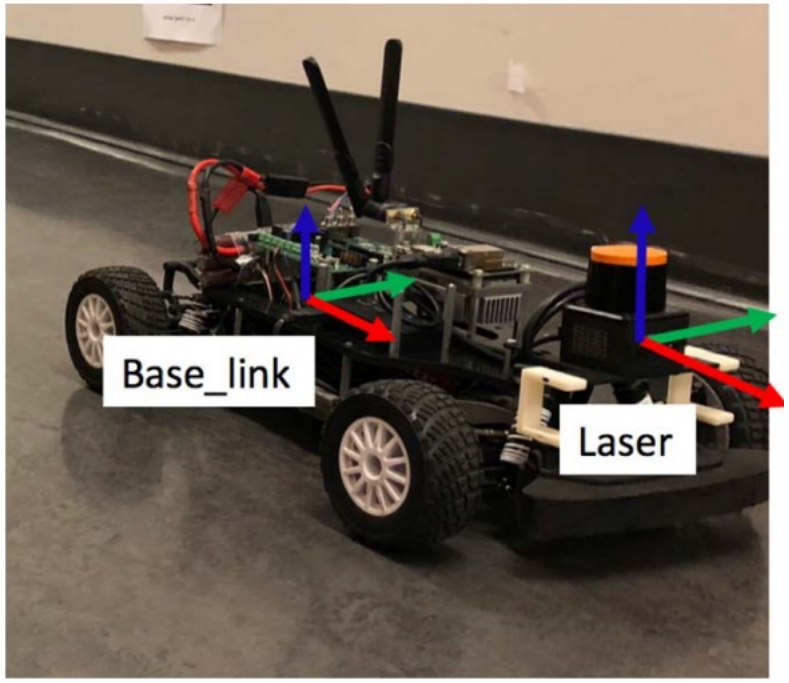
```
rosrun tf tf_echo turtle1 turtle2
```




```
[ INFO] 1253585683.529245000: Started node [/tf2_echo_1253585683508144000], pid [24418], bo
und on [aqy], xmlrpc port [41125], tcpros port [57369], logging to [~/ros/ros/log/tf2_echo_
1253585683508144000_24418.log], using [real] time
Exception thrown:Frame id /turtle1 does not exist! When trying to transform between /turtle
2 and /turtle1.
The current list of frames is:

Success at 1253585684.557003974
[0.000000 0.000000 0.140754 0.990045] Euler(0.282446 -0.000000 0.000000)
Translation: [-0.000036 -0.000010 0.000000]
Success at 1253585685.544698953
[0.000000 0.000000 0.140754 0.990045] Euler(0.282446 -0.000000 0.000000)
Translation: [-0.000036 -0.000010 0.000000]
Success at 1253585686.557049989
[0.000000 0.000000 0.140754 0.990045] Euler(0.282446 -0.000000 0.000000)
Translation: [-0.000036 -0.000010 0.000000]
Success at 1253585687.552628993
[0.000000 0.000000 0.140754 0.990045] Euler(0.282446 -0.000000 0.000000)
Translation: [-0.000036 -0.000010 0.000000]
Success at 1253585688.553683042
[0.000000 0.000000 0.140754 0.990045] Euler(0.282446 -0.000000 0.000000)
Translation: [-0.000036 -0.000010 0.000000]
Success at 1253585688.910640001
[0.000000 0.000000 0.140754 0.990045] Euler(0.282446 -0.000000 0.000000)
Translation: [-0.000036 -0.000010 0.000000]
```

Static transforms



- Laser vs base_link => constant postion => static transformation
- static_transform_publisher node

```
<node pkg="tf2_ros" type="static_transform_publisher" name="base_link_to_laser"
      args="0.285 0.0 0.127 0.0 0.0 0.0 1.0 /base_link /laser" />

static_transform_publisher x y z qx qy qz qw frame_id child_frame_id
                                                  Child_frame relative to frame
```

Tools

- Print information about the current tranform tree

  rosrun tf tf_monitor

```
RESULTS: for all Frames

Frames:
Frame: back_left_wheel published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: back_right_wheel published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: base_link published by unknown_publisher Average Delay: 0.000256327 Max Delay: 0.00292399
Frame: front_left_hinge published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: front_left_wheel published by unknown_publisher Average Delay: 0.000264519 Max Delay: 0.00293304
Frame: front_right_hinge published by unknown_publisher(static) Average Delay: 0 Max Delay: 0
Frame: front_right_wheel published by unknown_publisher Average Delay: 0.000269633 Max Delay: 0.0029375
Frame: laser published by unknown_publisher Average Delay: 0.000700335 Max Delay: 0.00294158
Frame: laser_model published by unknown_publisher(static) Average Delay: 0 Max Delay: 0

All Broadcasters:
Node: unknown_publisher 401.529 Hz, Average Delay: 0.000368705 Max Delay: 0.00234121
Node: unknown_publisher(static) 1e+08 Hz, Average Delay: 0 Max Delay: 0
```

Tools

- Print information about the transform between two frames

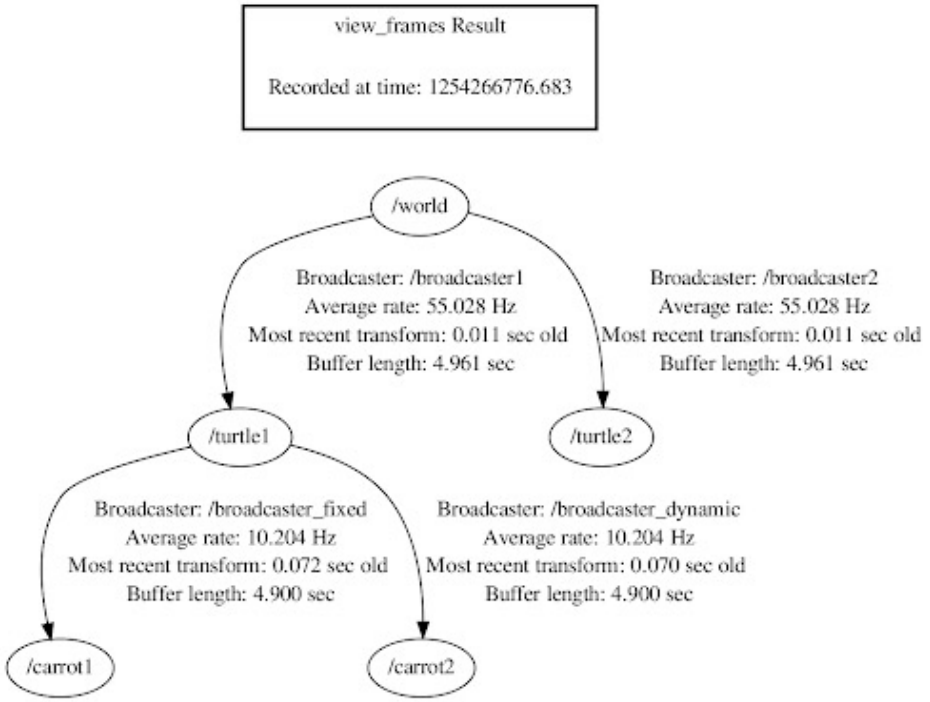
  rosrun tf tf_echo source_frame target_frame

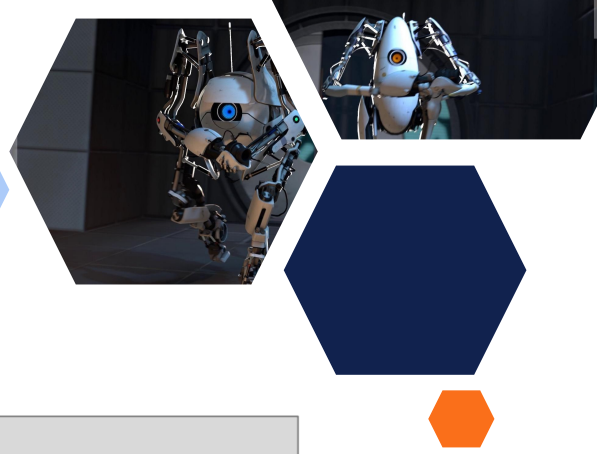
```
At time 1568046719.660
- Translation: [0.275, 0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
            in RPY (radian) [0.000, -0.000, 0.000]
            in RPY (degree) [0.000, -0.000, 0.000]
At time 1568046719.759
- Translation: [0.275, 0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.000, 1.000]
            in RPY (radian) [0.000, -0.000, 0.000]
            in RPY (degree) [0.000, -0.000, 0.000]
```

Tools

- Creates a visual graph (PDF) of the transform tree

  rosrun tf view_frames





C++ Example

- Create a TF listener to fill up a buffer

```
tf2_ros::Buffer tfBuffer;
tf2_ros::TransformListener tfListener(tfBuffer);
```

- Make sure, that the `listener` does not run out of scope!
- To lookup transformations, use

```
geometry_msgs::TransformStamped transformStamped =
tfBuffer.lookupTransform(target_frame_id,
                         source_frame_id, time);
```

- For `time`, use `ros::Time(0)` to get the latest available transform

```
#include <ros/ros.h>
#include <tf2_ros/transform_listener.h>
#include <geometry_msgs/TransformStamped.h>

int main(int argc, char** argv) {
  ros::init(argc, argv, "tf2_listener");
  ros::NodeHandle nodeHandle;
  tf2_ros::Buffer tfBuffer;
  tf2_ros::TransformListener tfListener(tfBuffer);

  ros::Rate rate(10.0);
  while (nodeHandle.ok()) {
    geometry_msgs::TransformStamped transformStamped;
    try {
      transformStamped = tfBuffer.lookupTransform("base",
                                 "odom", ros::Time(0));
    } catch (tf2::TransformException &exception) {
      ROS_WARN("%s", exception.what());
      ros::Duration(1.0).sleep();
      continue;
    }
    rate.sleep();
  }
  return 0;
};
```

More info + Tutorials

- http://wiki.ros.org/tf/Tutorials
- http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF
- http://wiki.ros.org/tf2/Tutorials
- http://wiki.ros.org/tf2/Migration

Thank you

Question Time