F1/10

Autonomous Racing

1/10 the scale. 10 times the fun!

ROS Transformations and frames

Madhur Behl

(University of Virginia)

Course Announcements

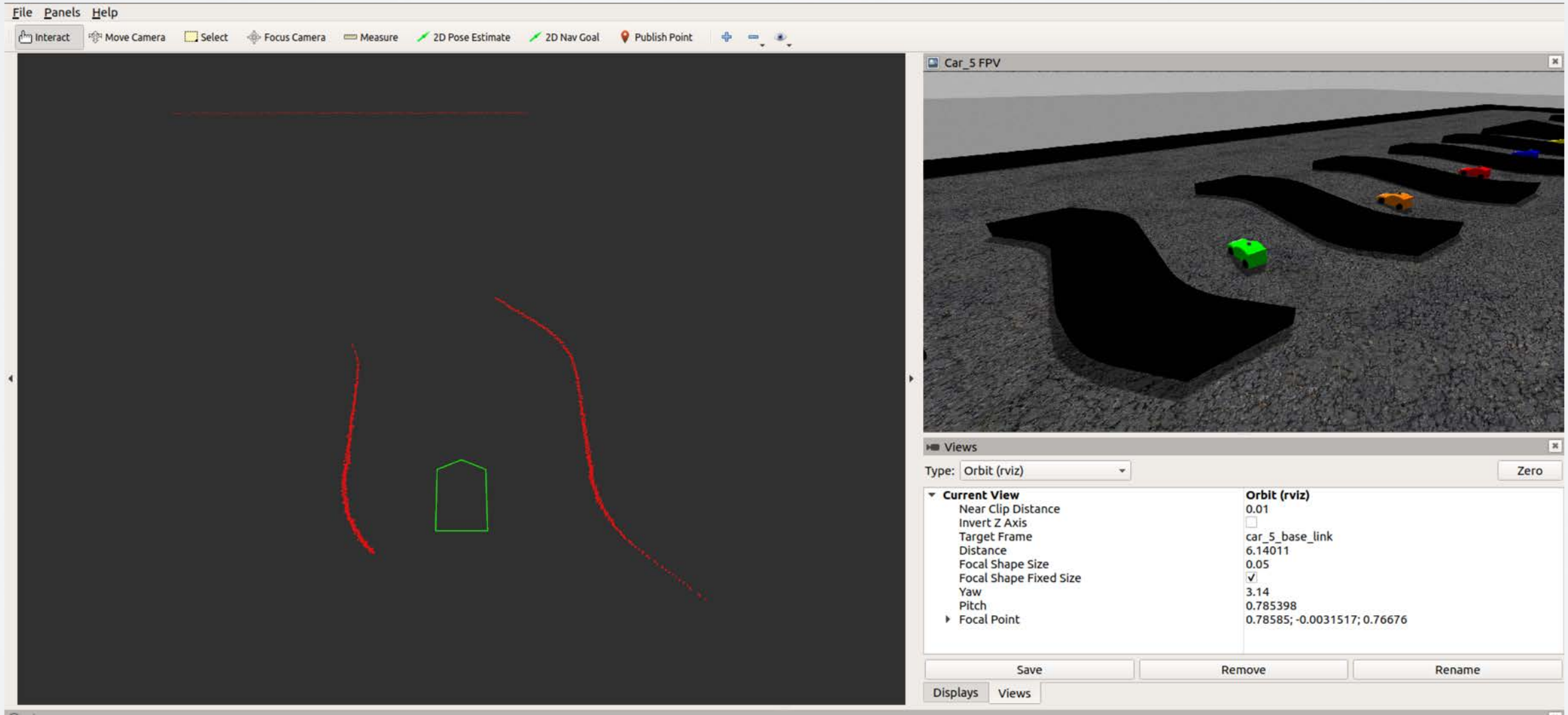
- All lectures and labs will be held online
  - Recordings will be posted for lectures on Youtube – announcements will go out for each posting.
  - Lab handouts will be posted on the course webpage: You are expected to run the lab handouts on your own.
  - Live office hours will be held routinely.

Course Announcements

- No more F1/10 hardware will be used.
- We will switch to the ROS F1/10 Simulator for the remainder of the course.
- You will still work in teams, as before
- Some more individual lab assignments will be added.

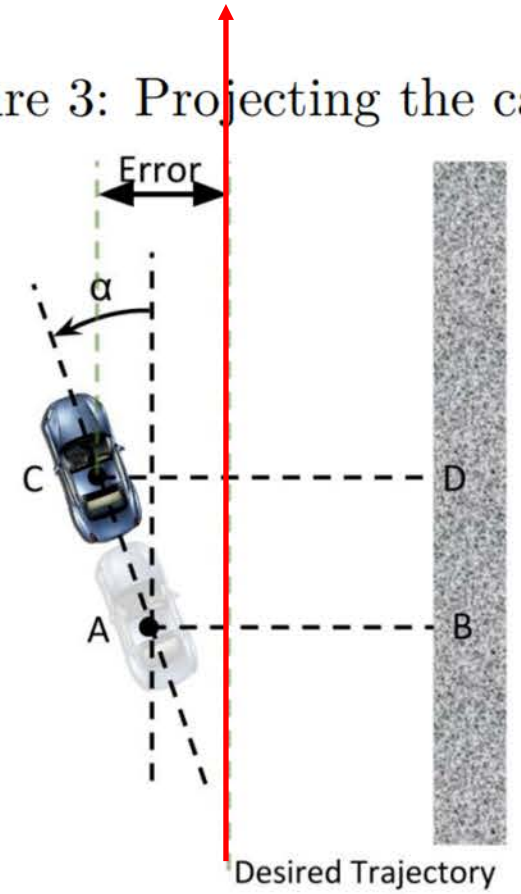
Course Announcements

- We are working on providing shared access to the simulator online:



Account for the forward motion of the car

Figure 3: Projecting the car future in time



$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta)-b}{a\sin(\theta)}\right)$$

$$AB = b\cos(\alpha)$$

$$CD = AB + AC\sin(\alpha)$$

Error = desired trajectory – CD

PID Steering Control

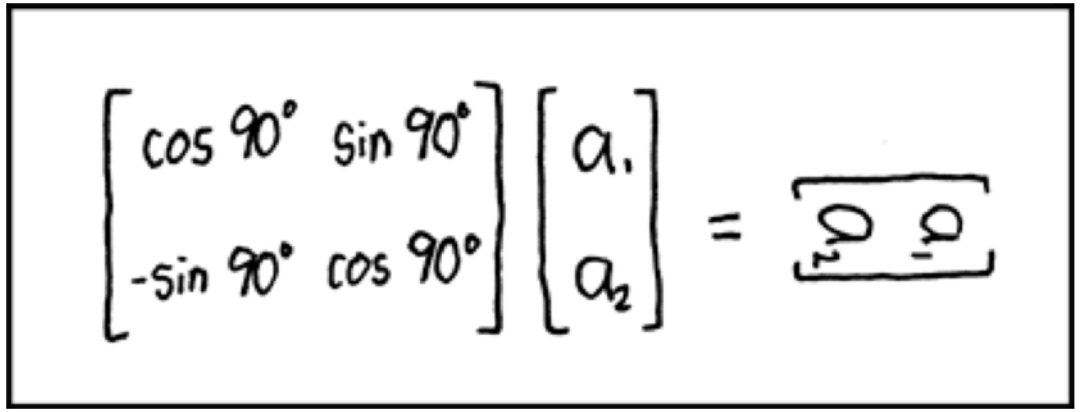
$$V_\theta = K_p \times e(t) + K_d \frac{de(t)}{dt}$$

$$V_\theta = K_p \times error + K_d \times previous\ error\ \text{-}\ current\ error$$

$$steering\ angle = steering\ angle\ - V_\theta$$

What we'll cover today

- Coordinate Frames and Transformations: why?
- Reference Frames
- Rigid Body Transformations
- tf/tf2_ros (ROS specific frames and transformations)

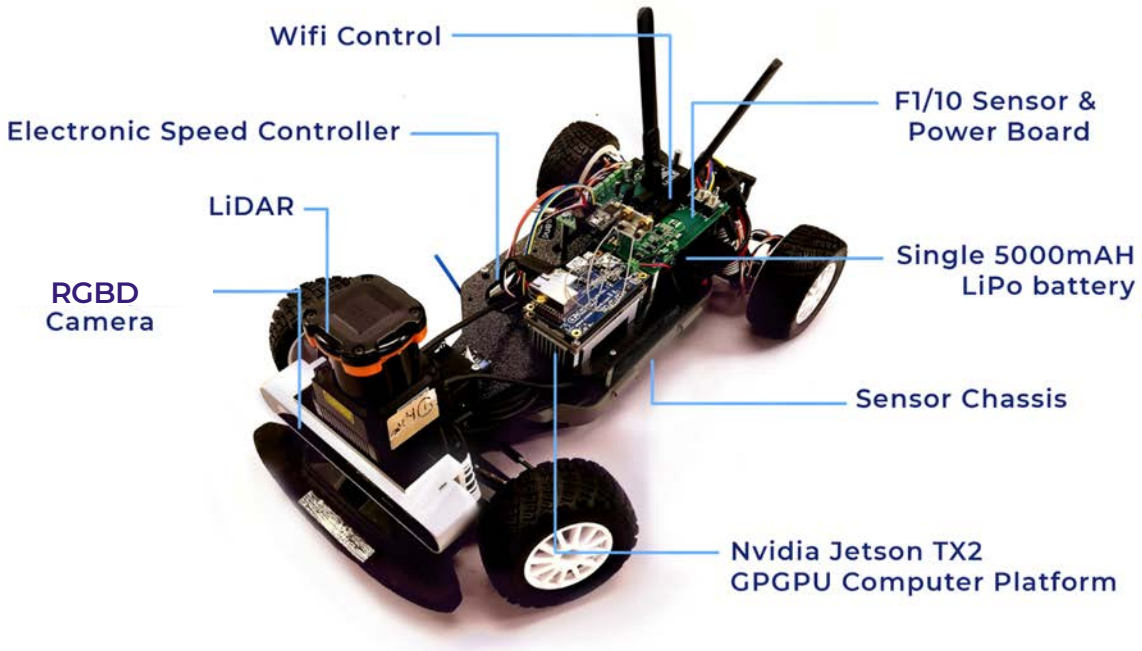


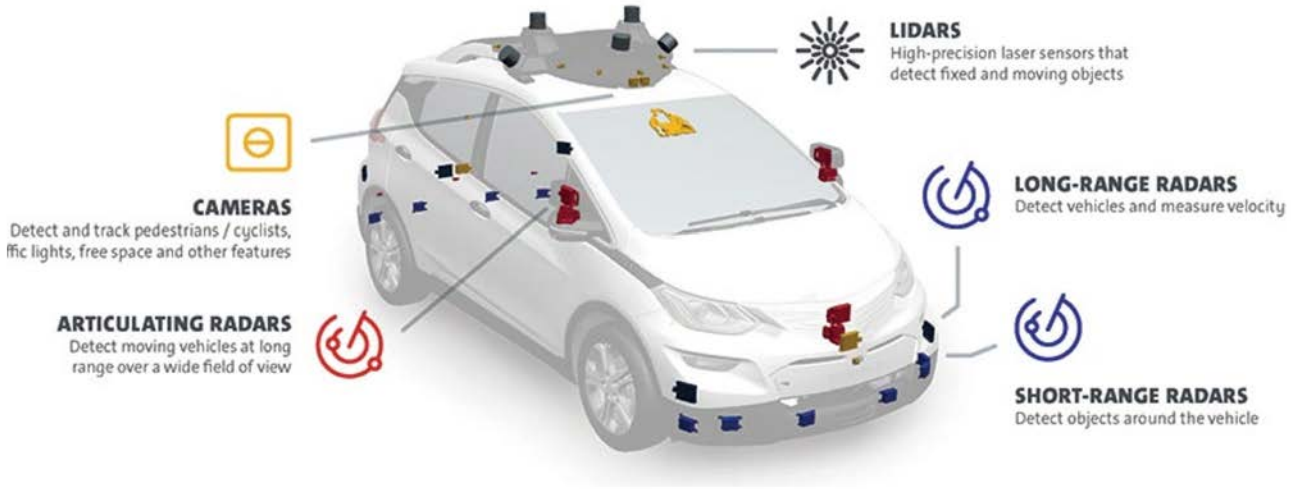
https://imgs.xkcd.com/comics/matrix_transform.png

Frames and Transformations on AV

Why do we need Frame Transformations on AV

Sensors provides measurement in the frame of reference specific to that sensor

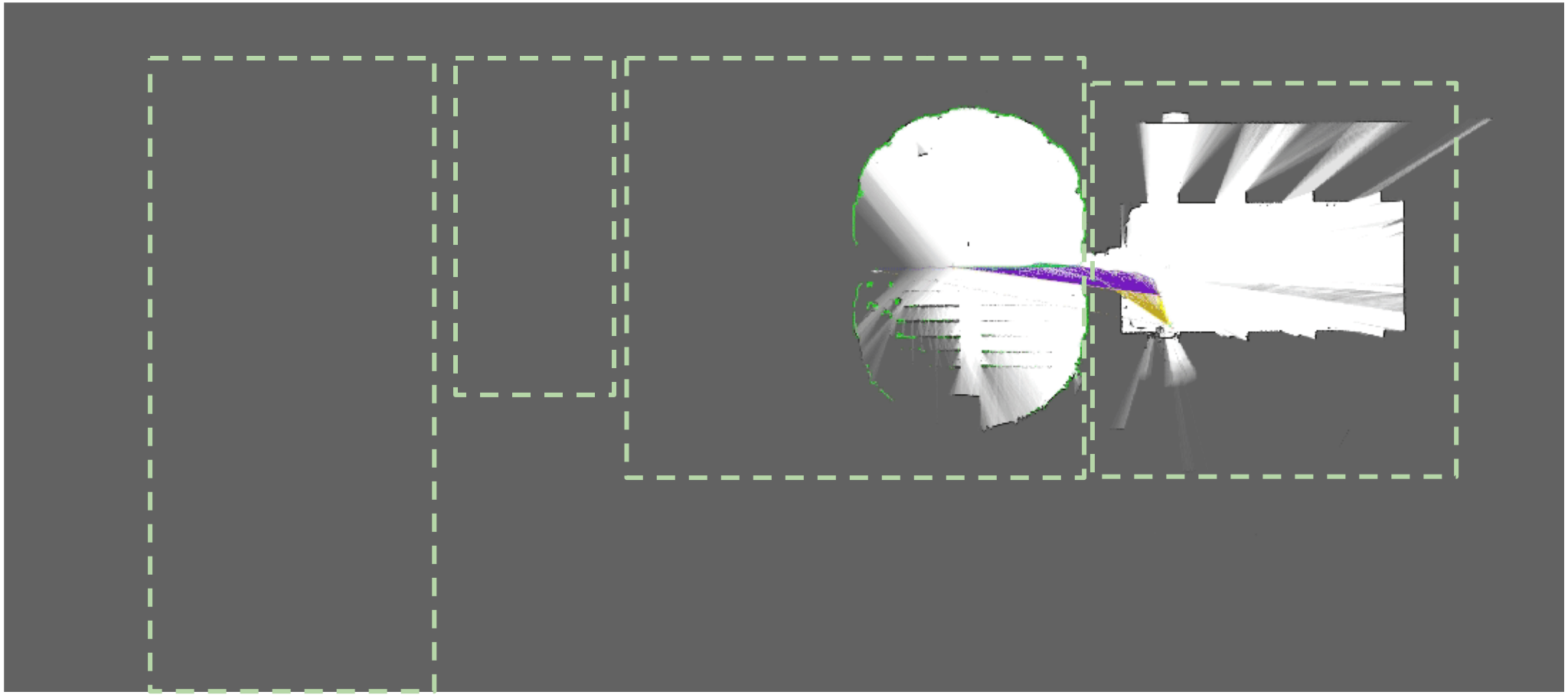




www.autonomousvehicletech.com

Why do we need Frame Transformations on AV

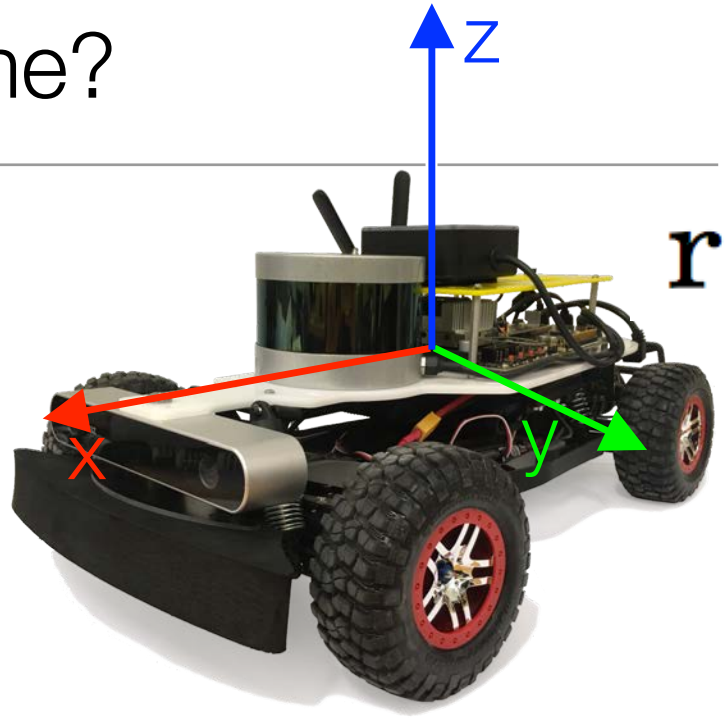
Mapping is usually done by tying submaps together

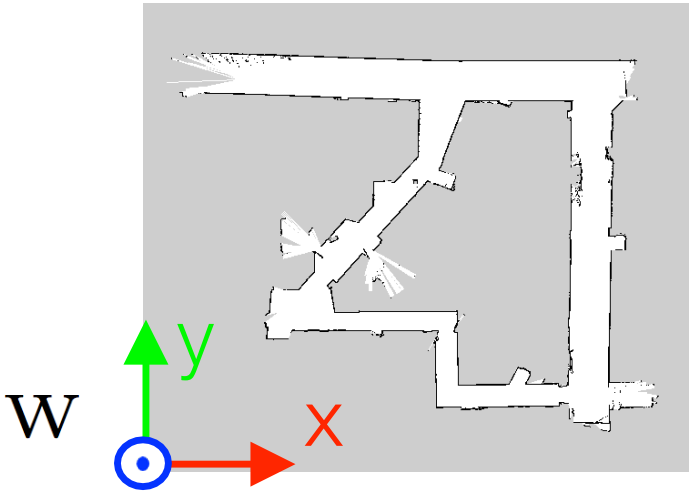


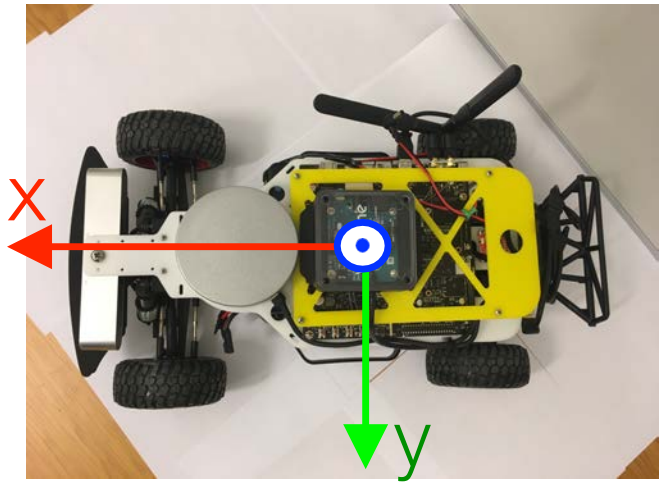
https://github.com/googlecartographer/cartographer_ros

What is a Coordinate Frame?

- "set of orthogonal axes attached to a body that serves to describe position of points relative to that body"
- **axes** intersect at a point known as the **origin**



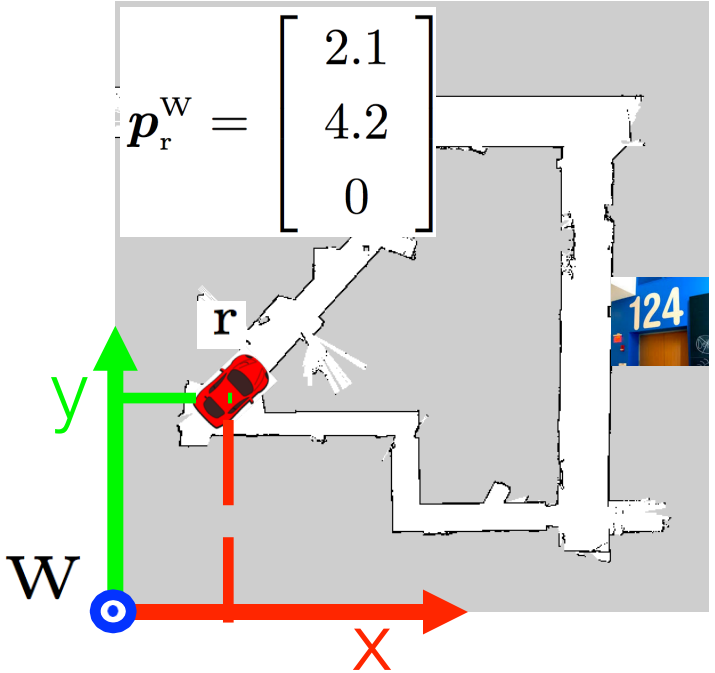


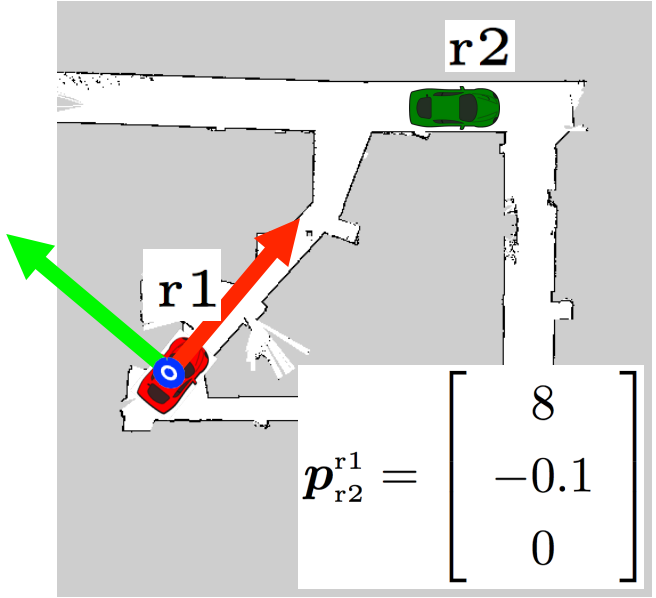


In many robotics problem, the first step is to assign a coordinate frame to all objects of interest

Why do we need Coordinate Frames?

- compact representation of points

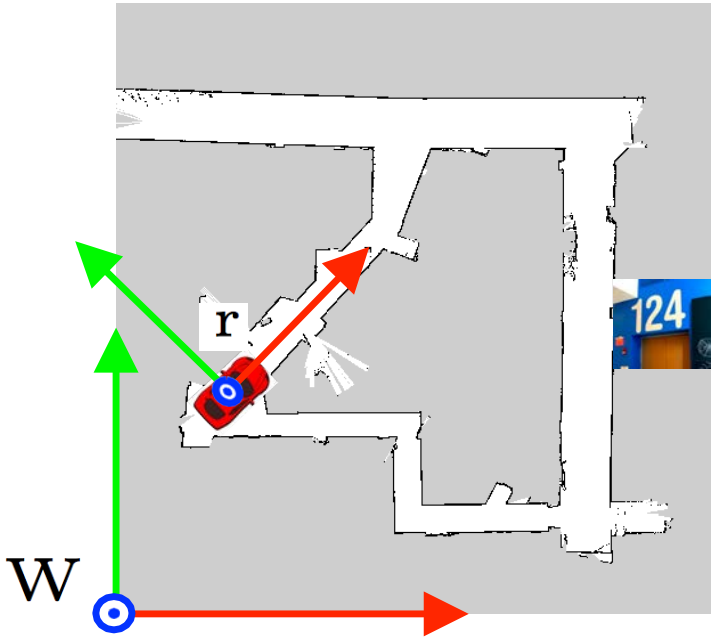


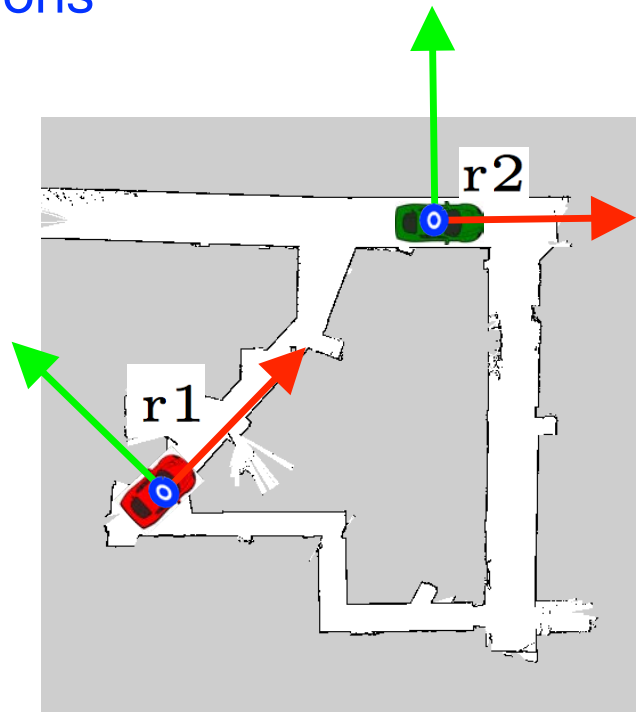


- coordinates have no meaning without specifying coordinate frame

Why do we need Coordinate Frames?

- compact representation of points
- compact representation of orientations

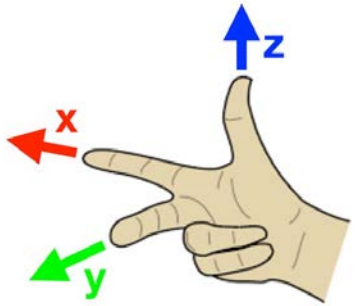


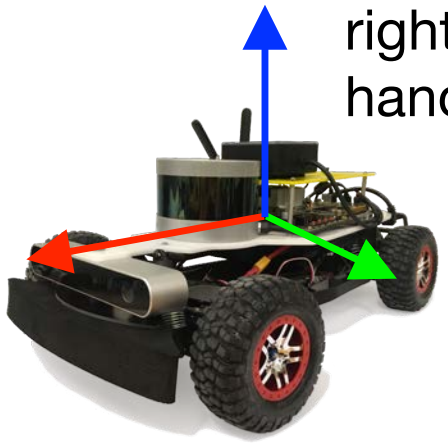


- **rotation, attitude, orientation**
- **pose** = position & rotation

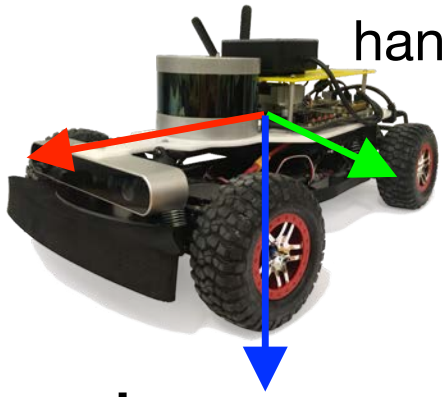
Left-handed vs Right-handed Coordinate Frames

- **Direction of axes is important!**
  - There is a common 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.





right-handed



left-handed

- **robotics always uses right-handed coordinate systems**
- left-handed systems are sometimes used in graphics

[slide courtesy of Prof. Roy]

Transformations and Frames:

- The Sensor frame – how does the world look w.r.t the sensor

Does this tell you anything about where obstacles are in the map?

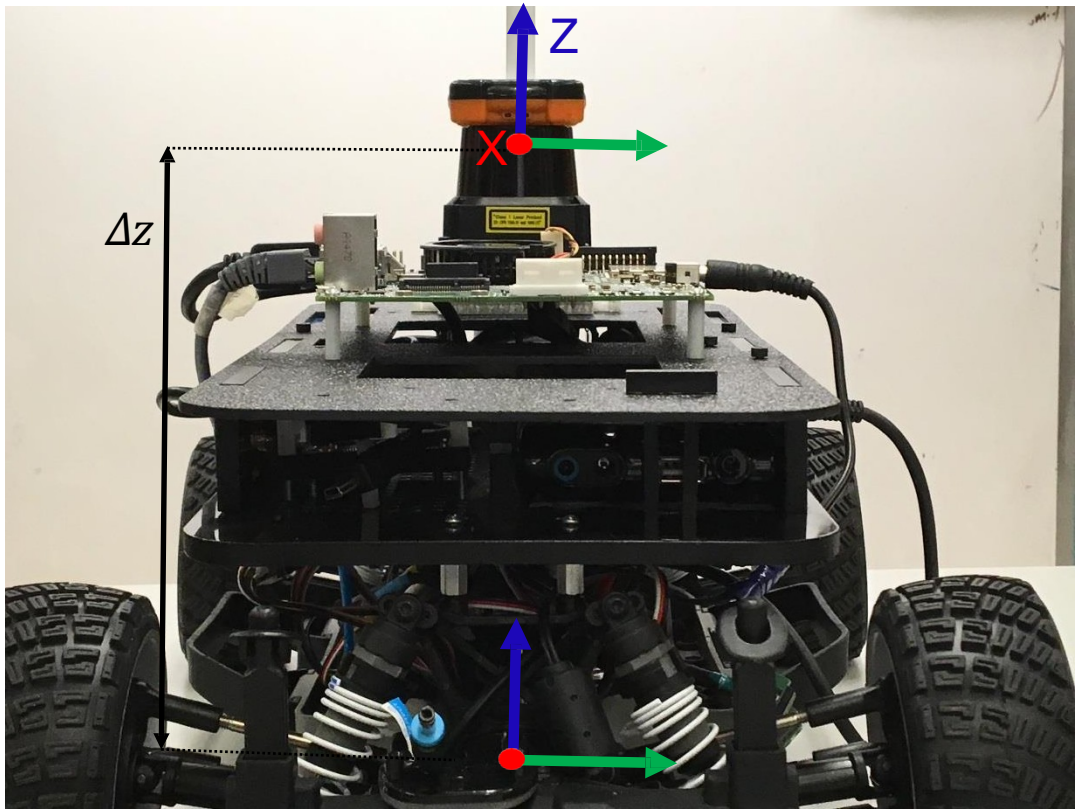
Does this tell you anything about where we are in the map?

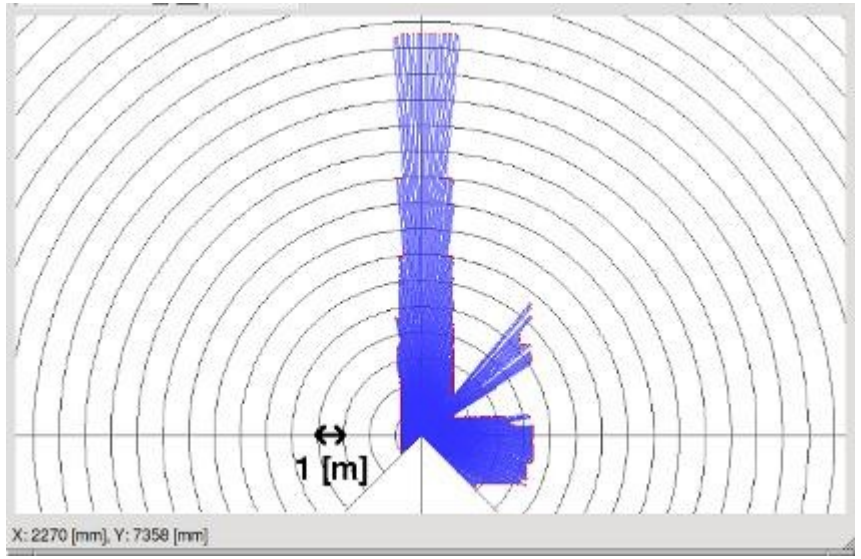
We must link frames together

Transformations

Transformations and Frames

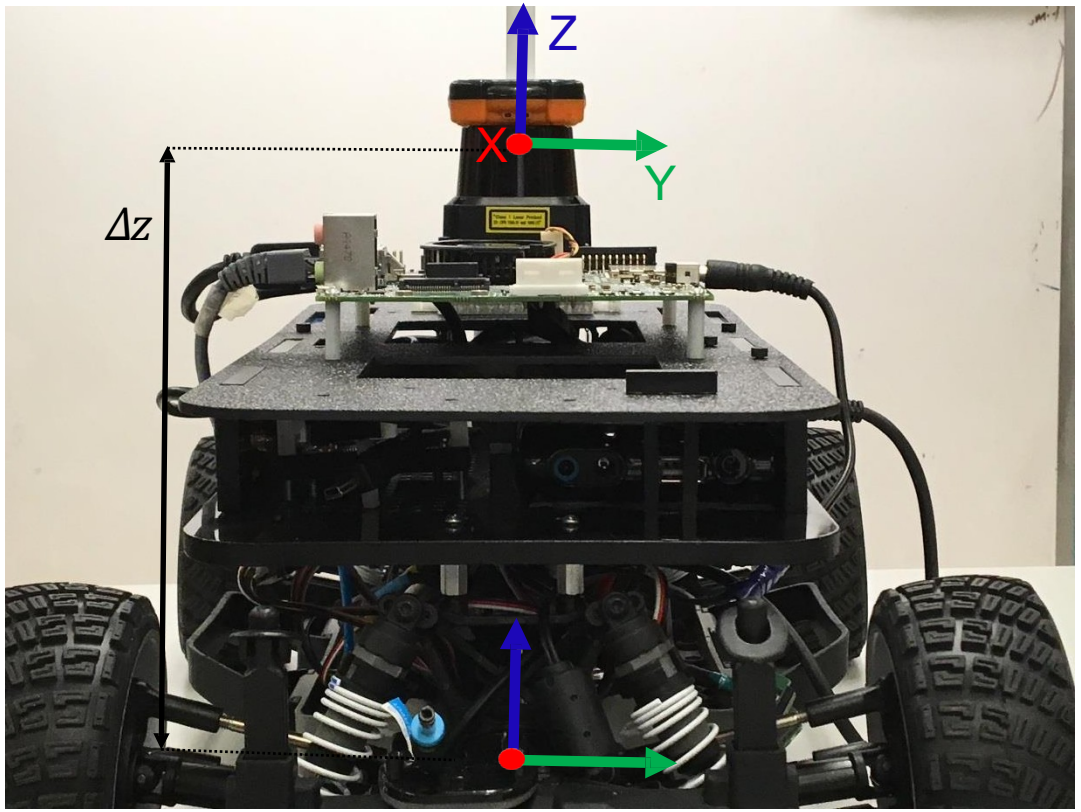
- The frame of reference in which the measurementis taken



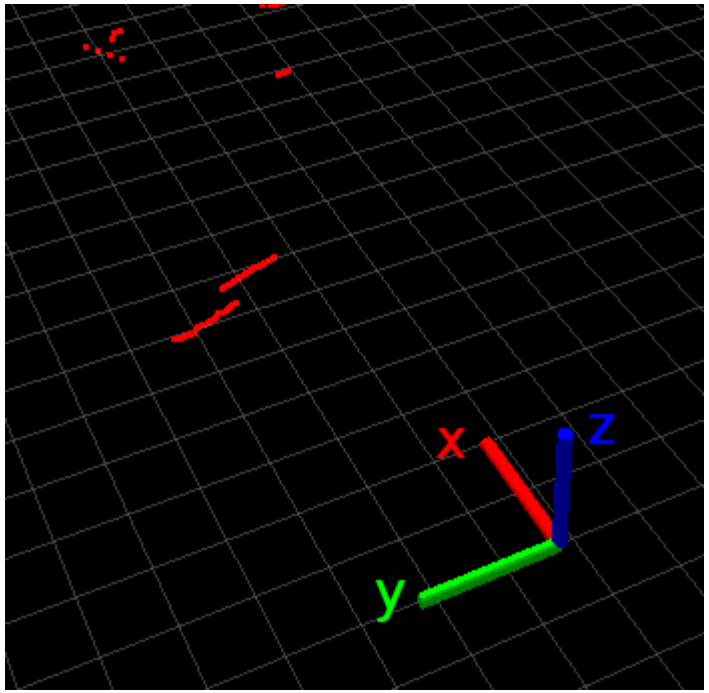


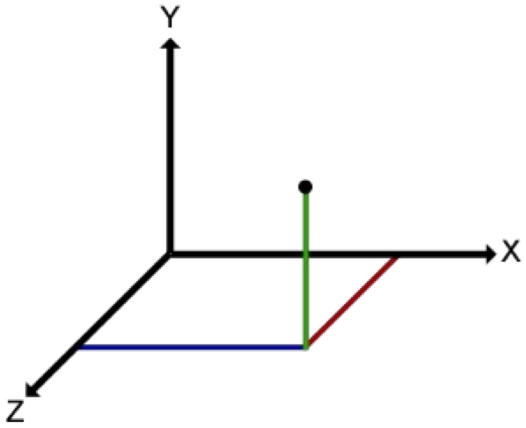
Distance measurements returned by LIDAR

Transformations and Frames



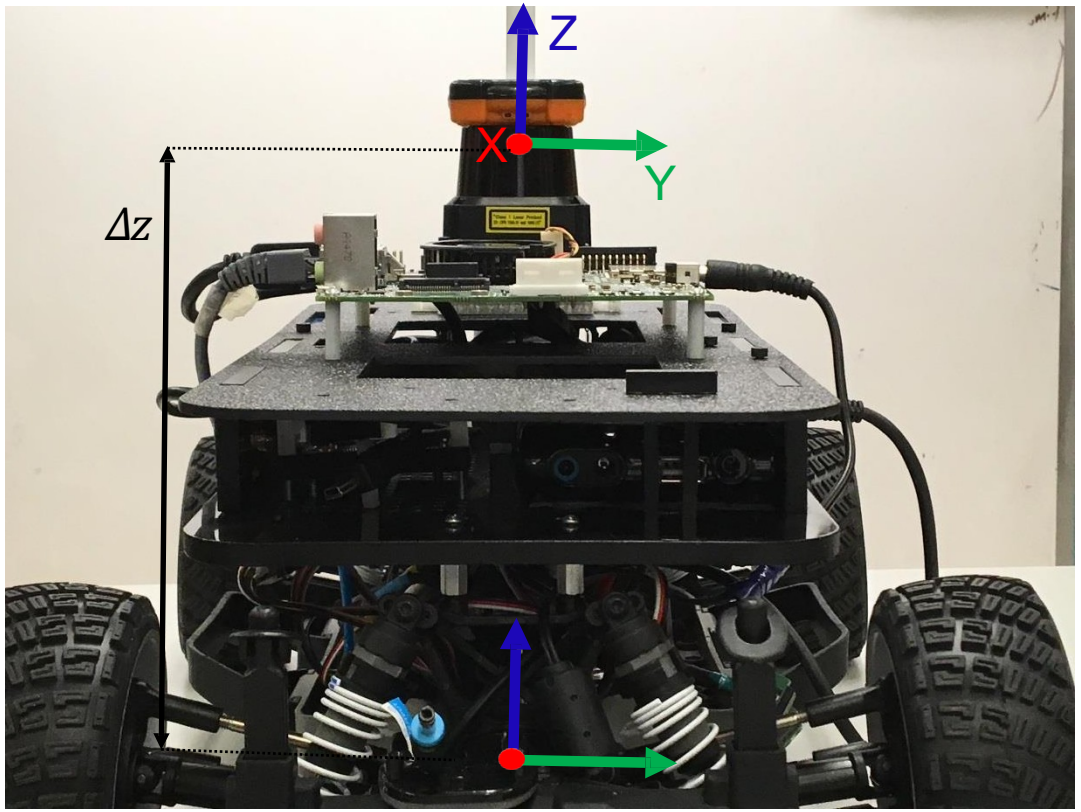
Note:
Axes X,Y,Z of Frames of Reference are orthogonal(90°) to each other.
X,Y,Z represent the axes along the 3 dimensions.



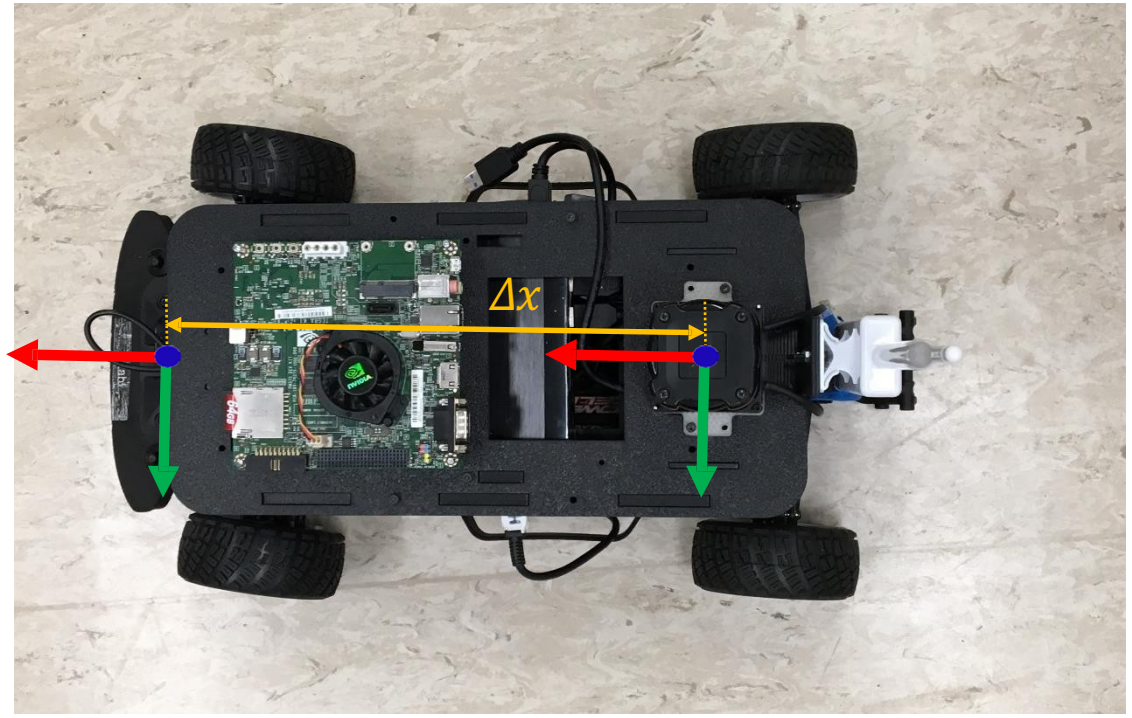


Transformations and Frames

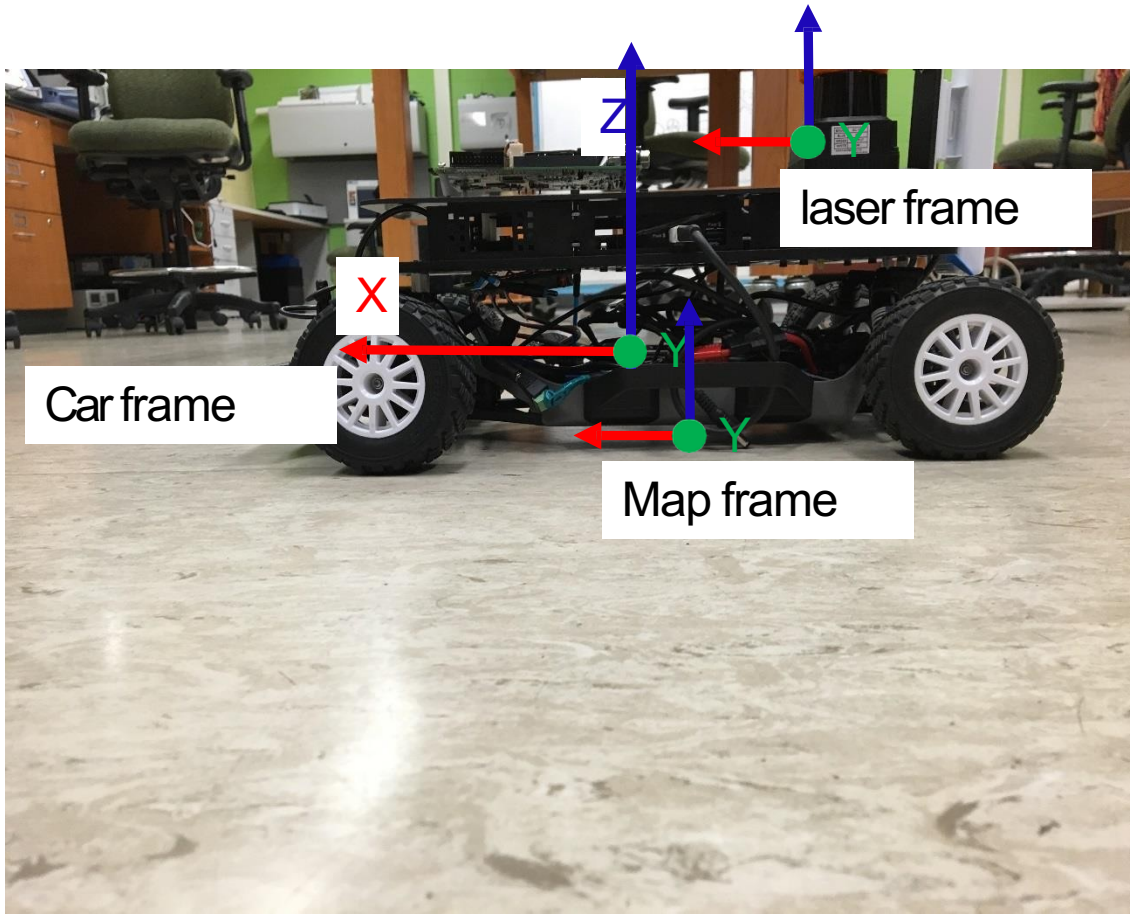
- The frame of reference in which the measurementis taken



The scan Values from the LIDARwill not tell us how far away are the obstacles. We must take care of the offsets



Transformations and Frames

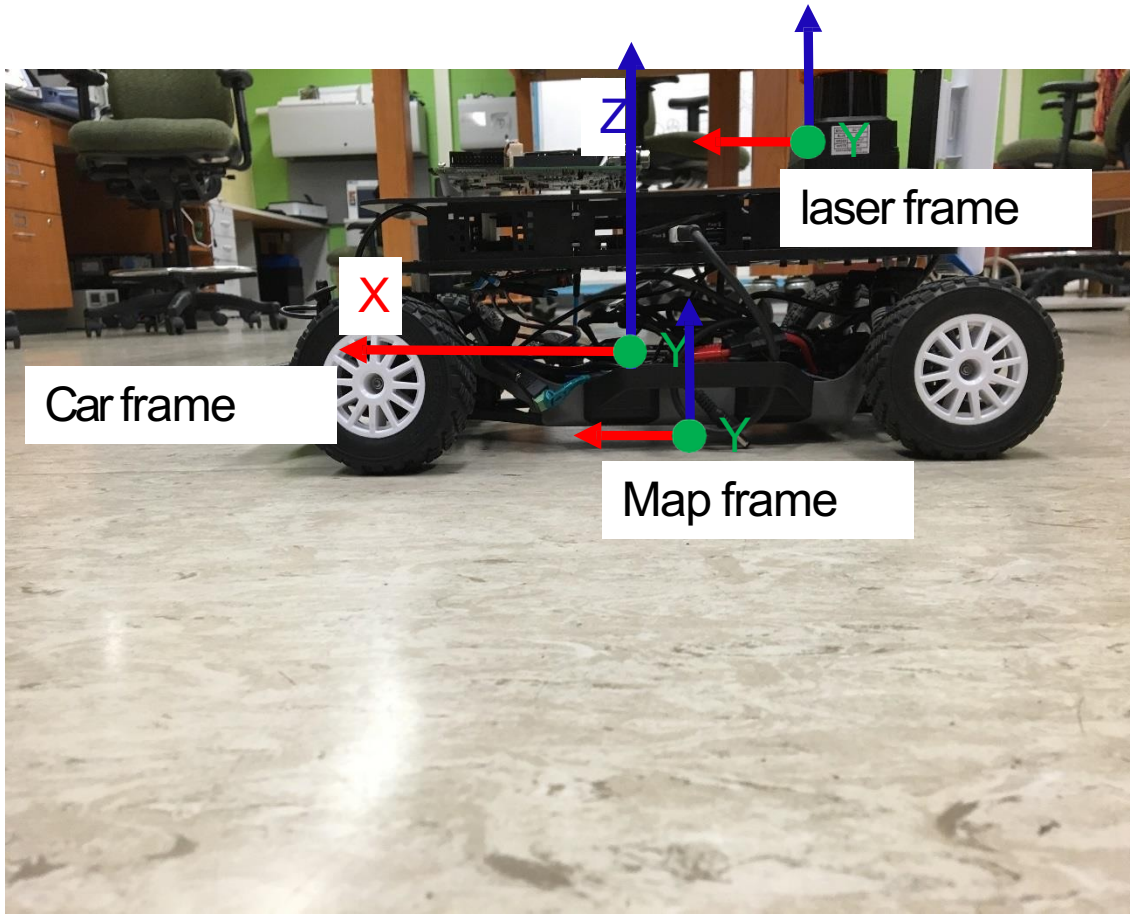


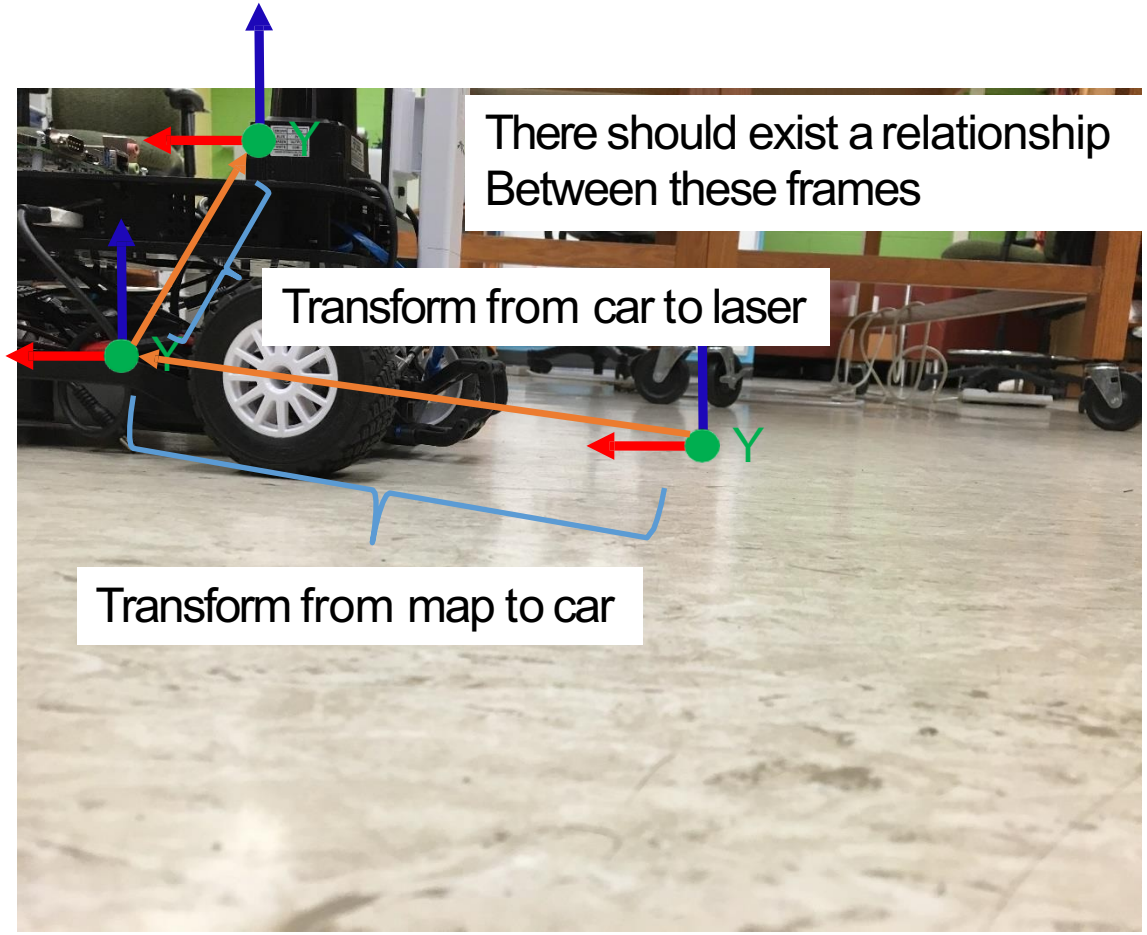
Between frames there will exist transformations that convert measurements from one frame to another

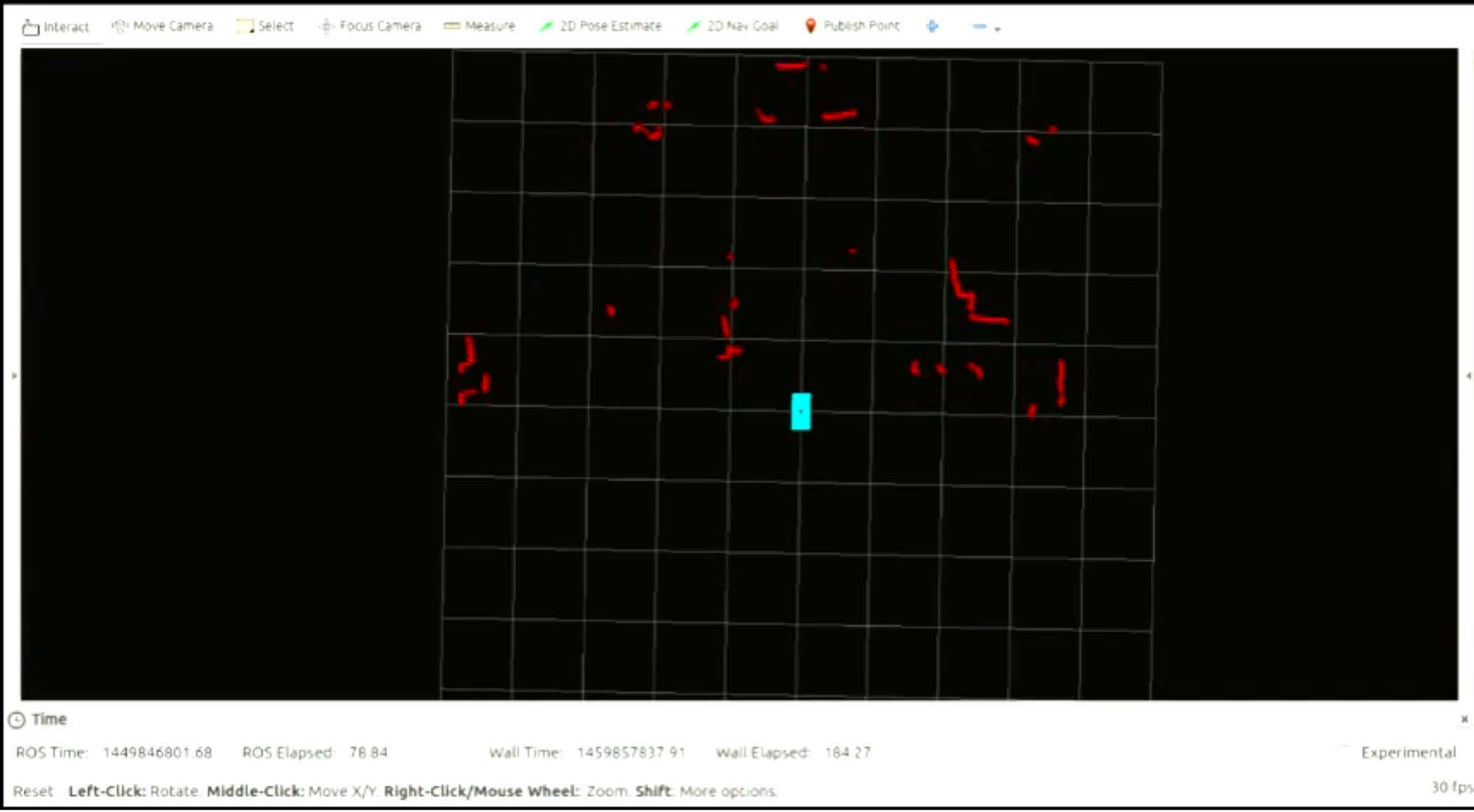
Important Point:
Note what the transformation means w.r.t frames

Transformations and Frames

Between frames there will exist transformations that convert measurements from one frame to another






Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Time
ROS Time: 1449846801.68
ROS Elapsed: 78.84
Wall Time: 1459857837.91
Wall Elapsed: 184.27
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
30 fps

Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Time
OS Time: 1449846795.66
ROS Elapsed: 46.17
Wall Time: 1459858888.32
Wall Elapsed: 194.04
Experimental
eset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
30 fps

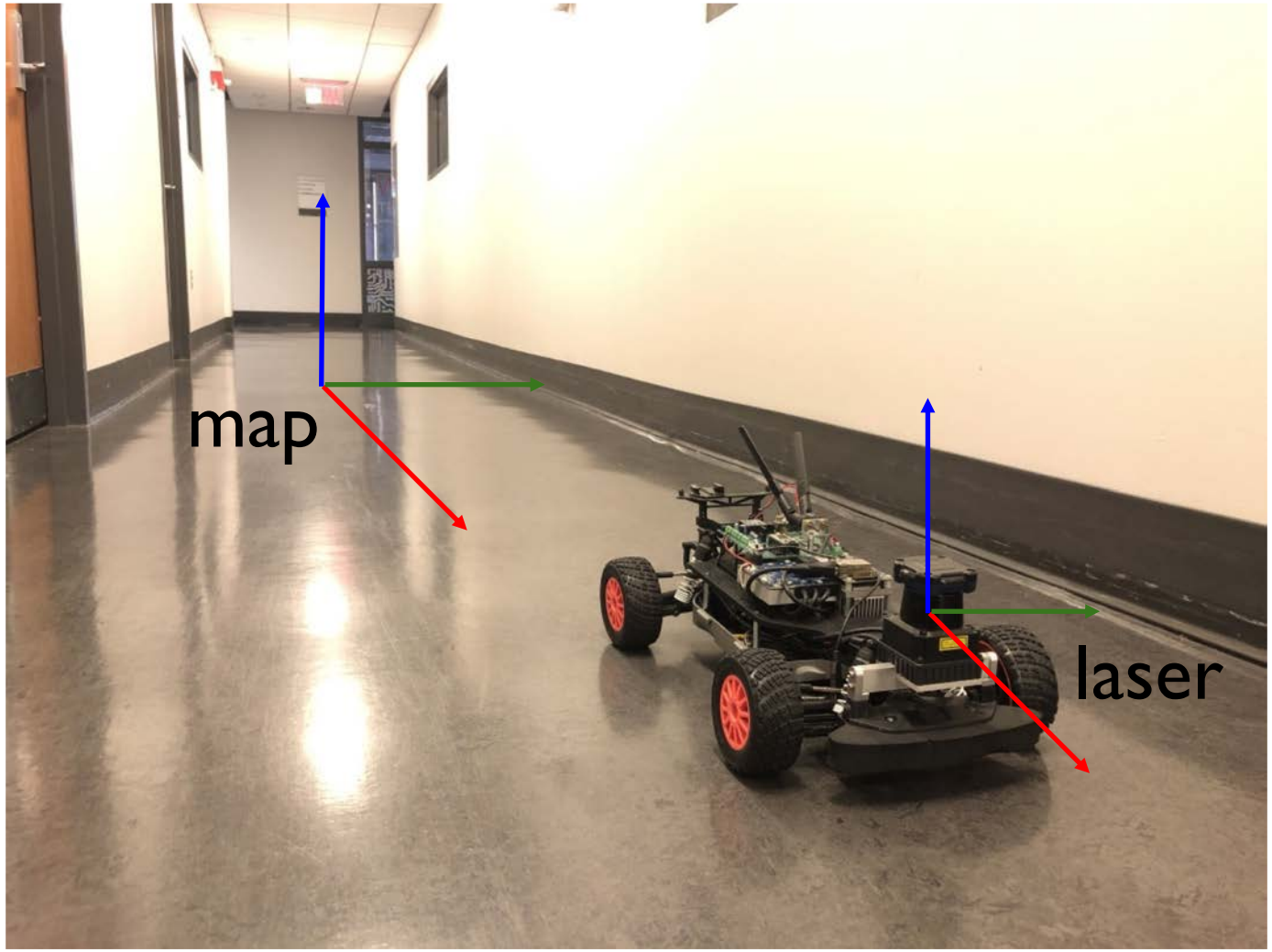
Reference Frames on F1/10

Multiple Reference Frames on F1/10

Why?

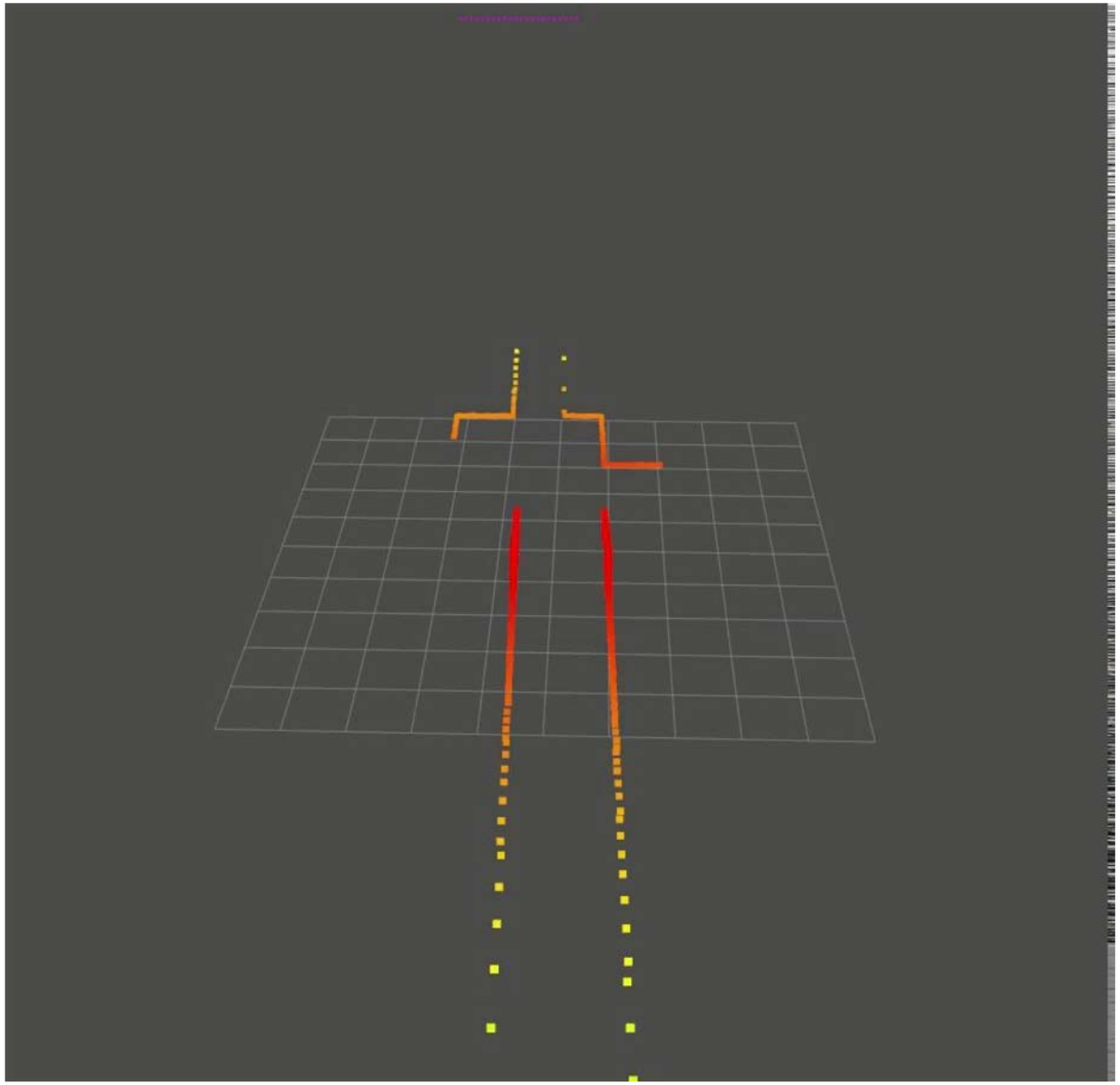
We need to know where the robot is in the world.

We need to know the relationship between the measurement frame and the actuation frame.



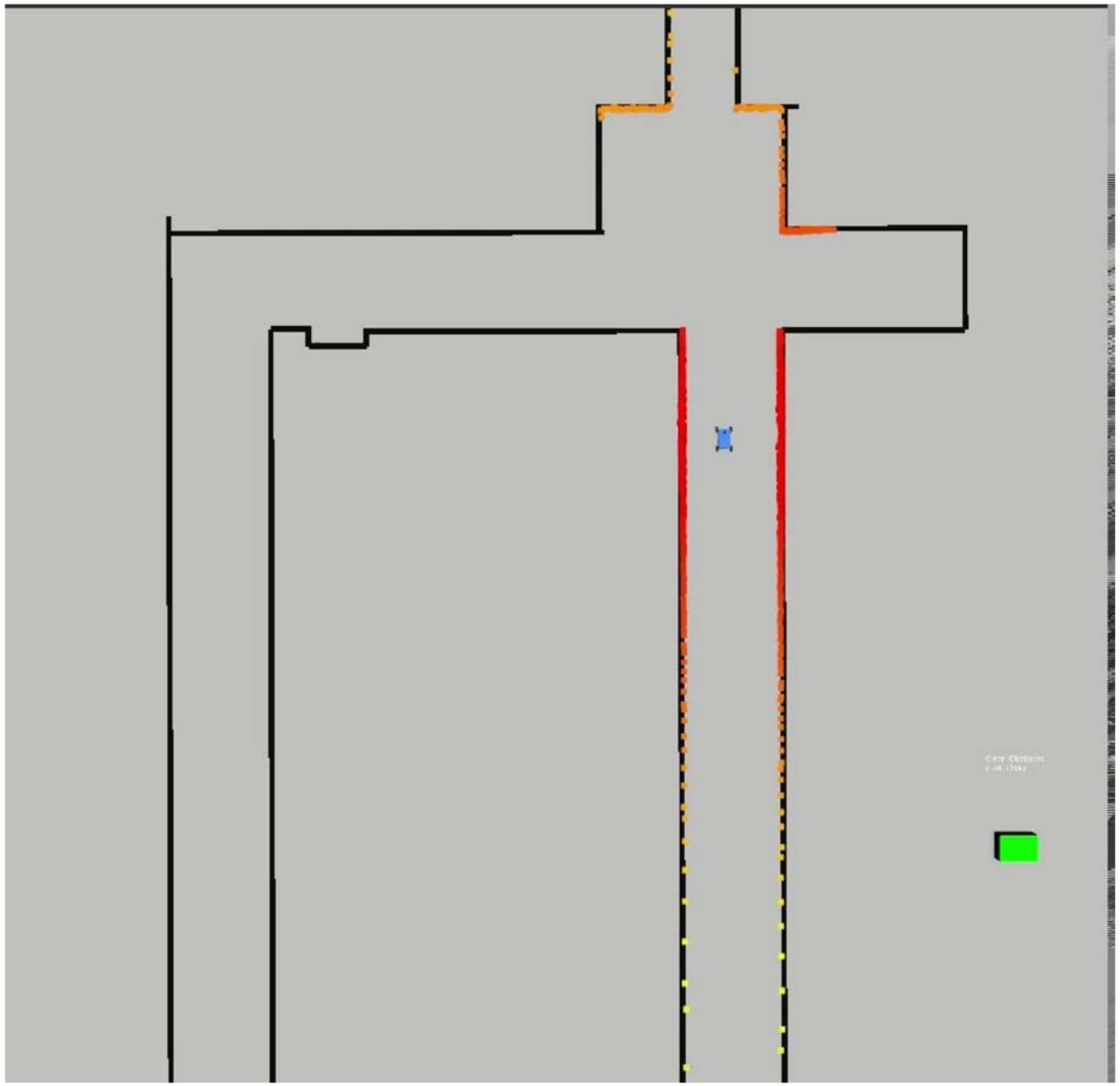
Multiple Reference Frames

Measurement in **laser frame**: can see some information about the environment, and some information on the positional relationship between the robot and the world.

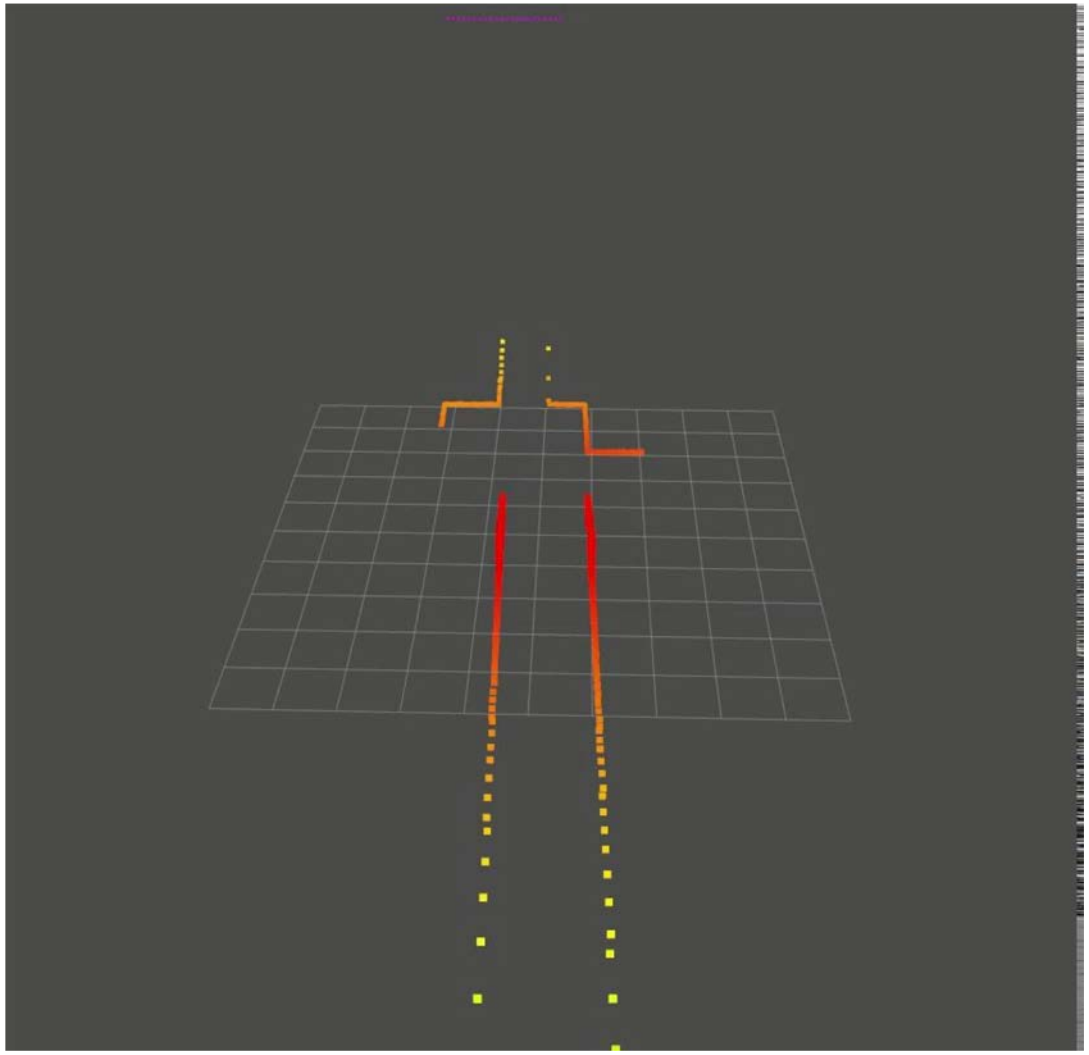


Multiple Reference Frames

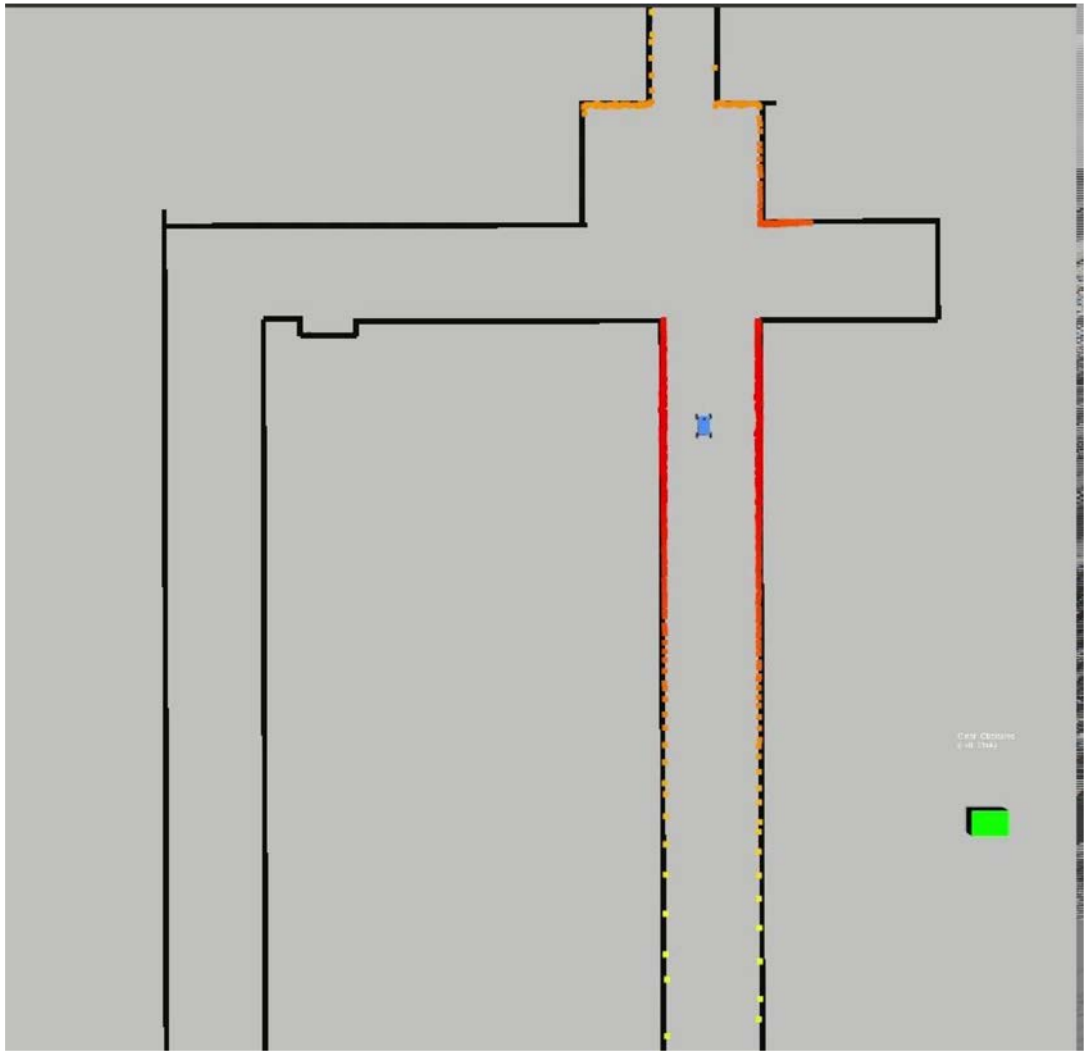
Measurement in **map frame**: the range measurement we see fully makes sense. The observation matches up with the environment. We know where the robot is in the environment.



Multiple Reference Frames



Laser frame



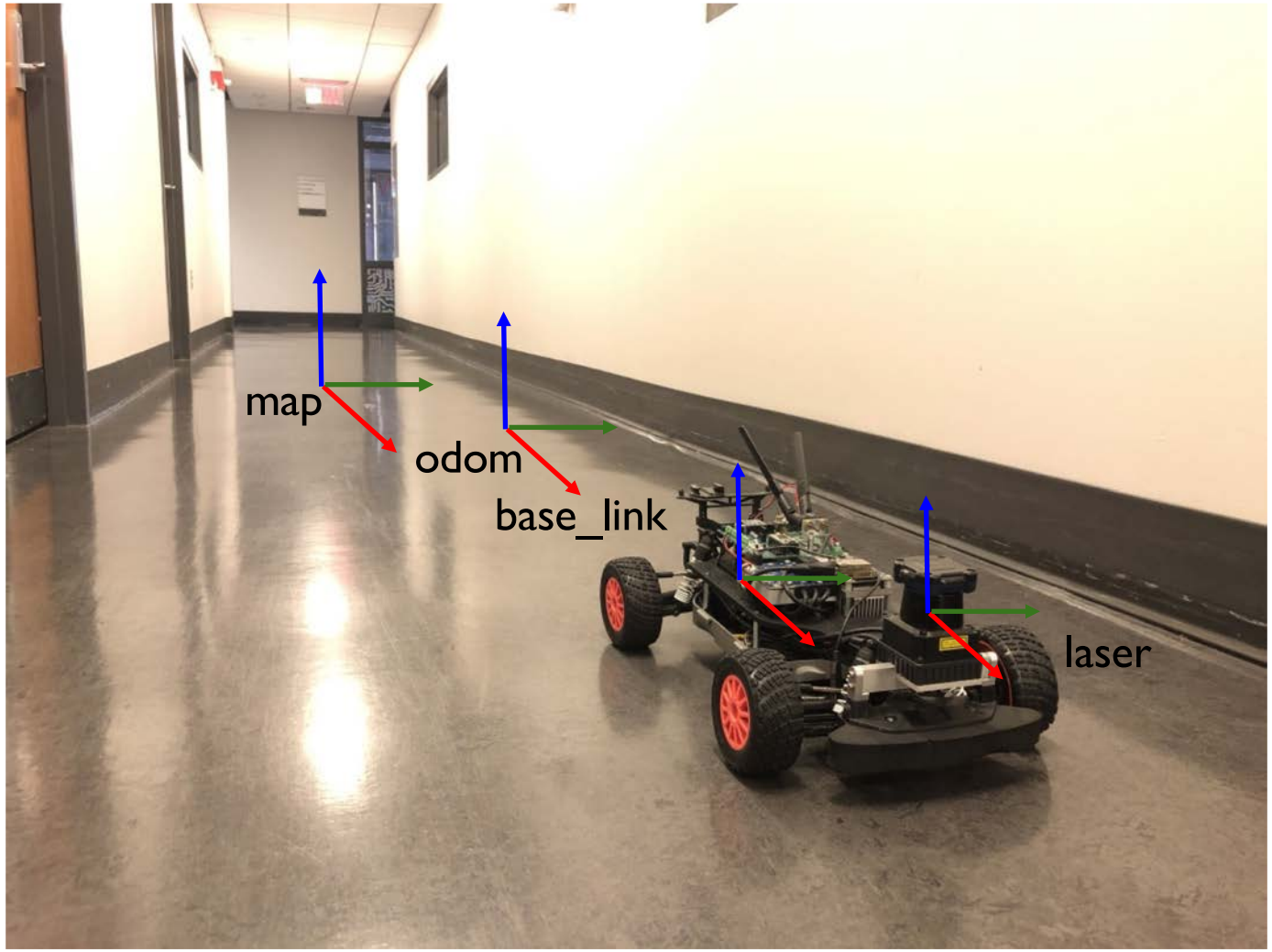
Map frame

Summary: why do we need transforms between frames?

- Data is usually provided in the most convenient frame to the data source
- If we had two disconnected maps (e.g. submaps), we might want to know their location w.r.t. Each other within a global world frame
- We want to localize ourselves on a map
- If an obstacle is detected in the *laser* frame, maybe we want to know where it is in the world, i.e. in the *map* frame
- We need to know the transforms between frames of components on the car for more accurate actuation
- ...

Frames we will use

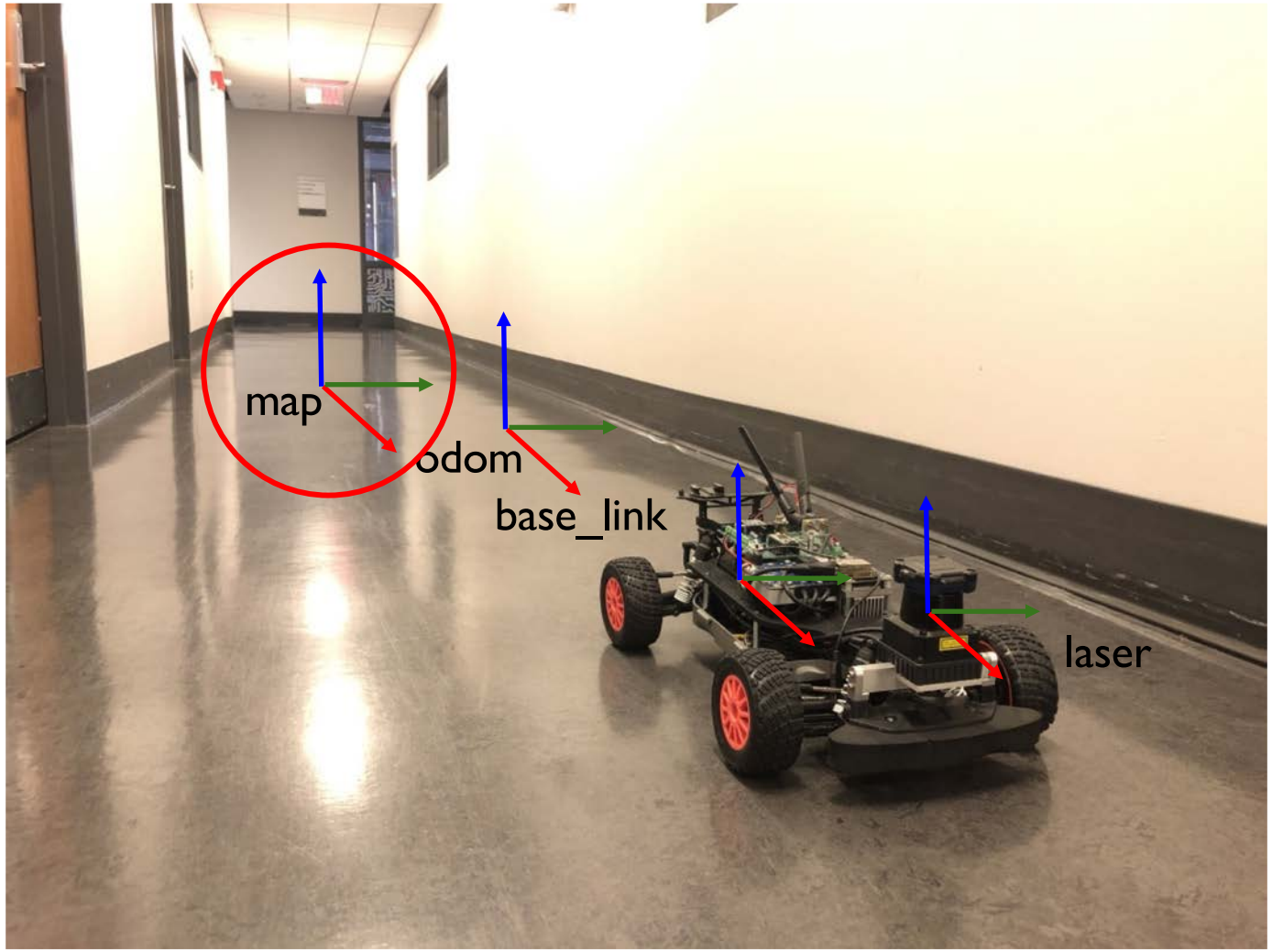
1. map
2. base_link
3. laser
4. (odom)



P.S. In ROS, RGB → XYZ

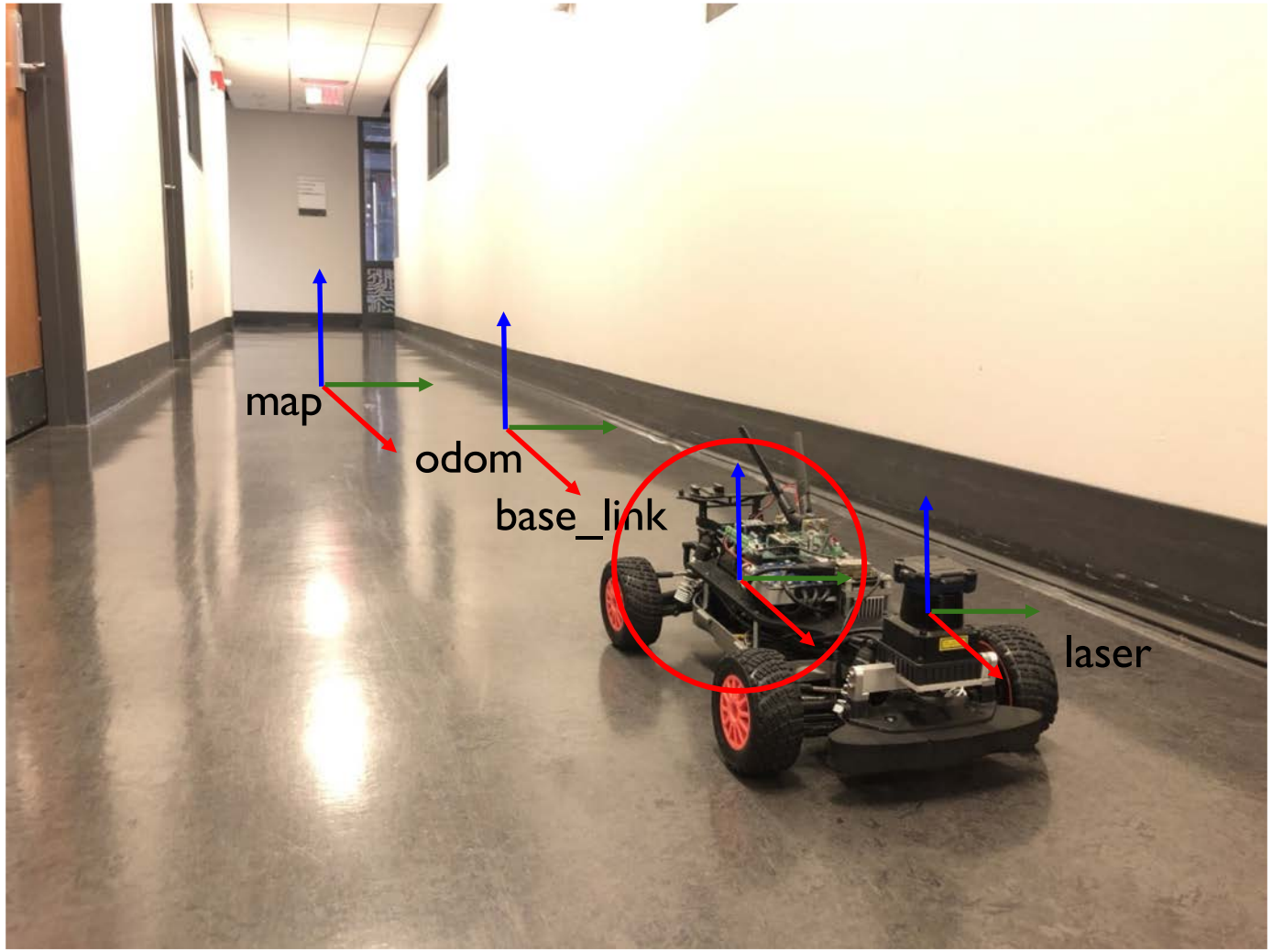
Frames we will use

1. map
   - Map frame origin can be set arbitrarily when we make the map
   - Represents the environment around the robot



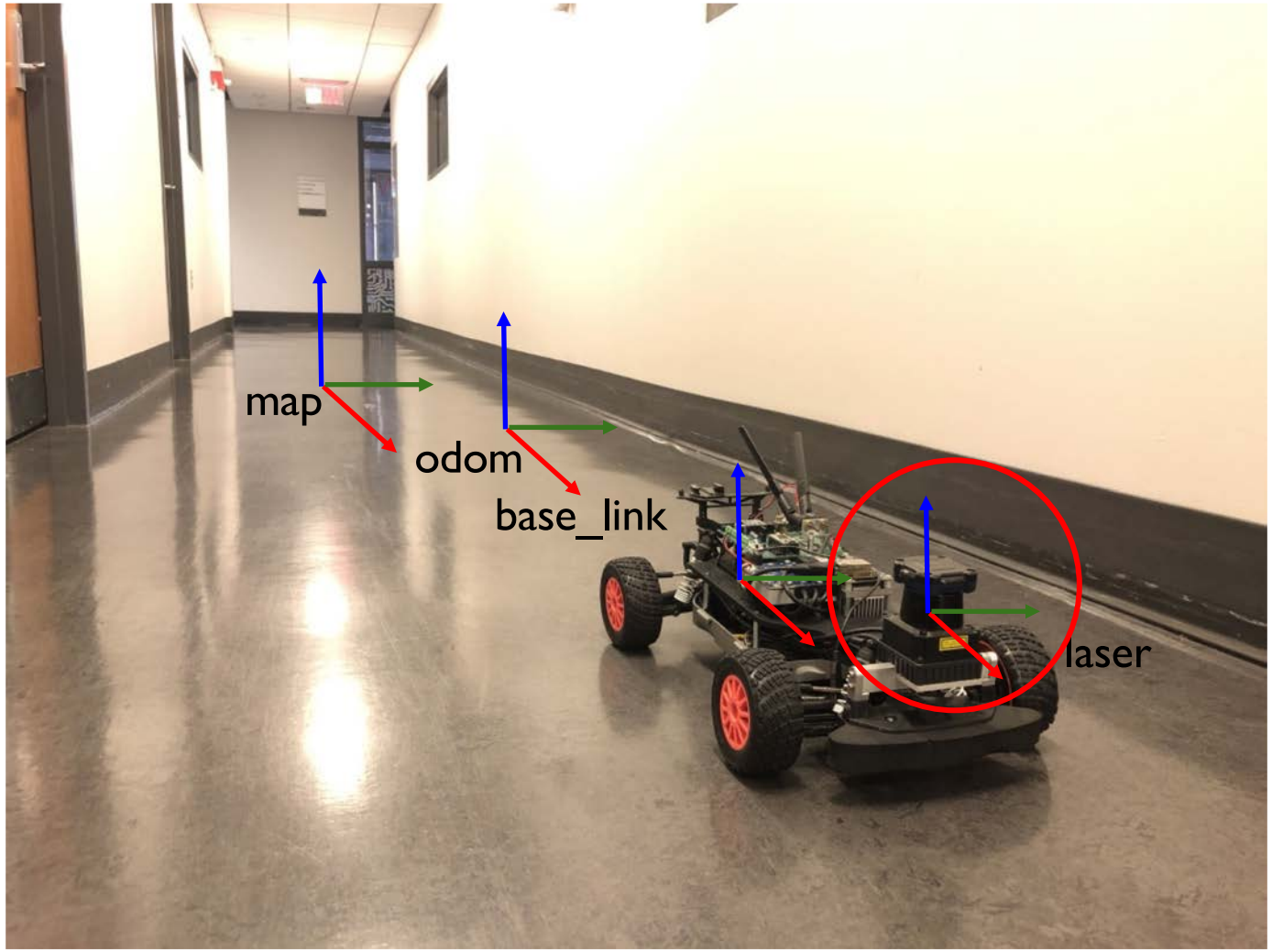
Frames we will use

2. base_link
   - Defined as the center of the car's rear axle
   - Moves with the car relative to the map frame



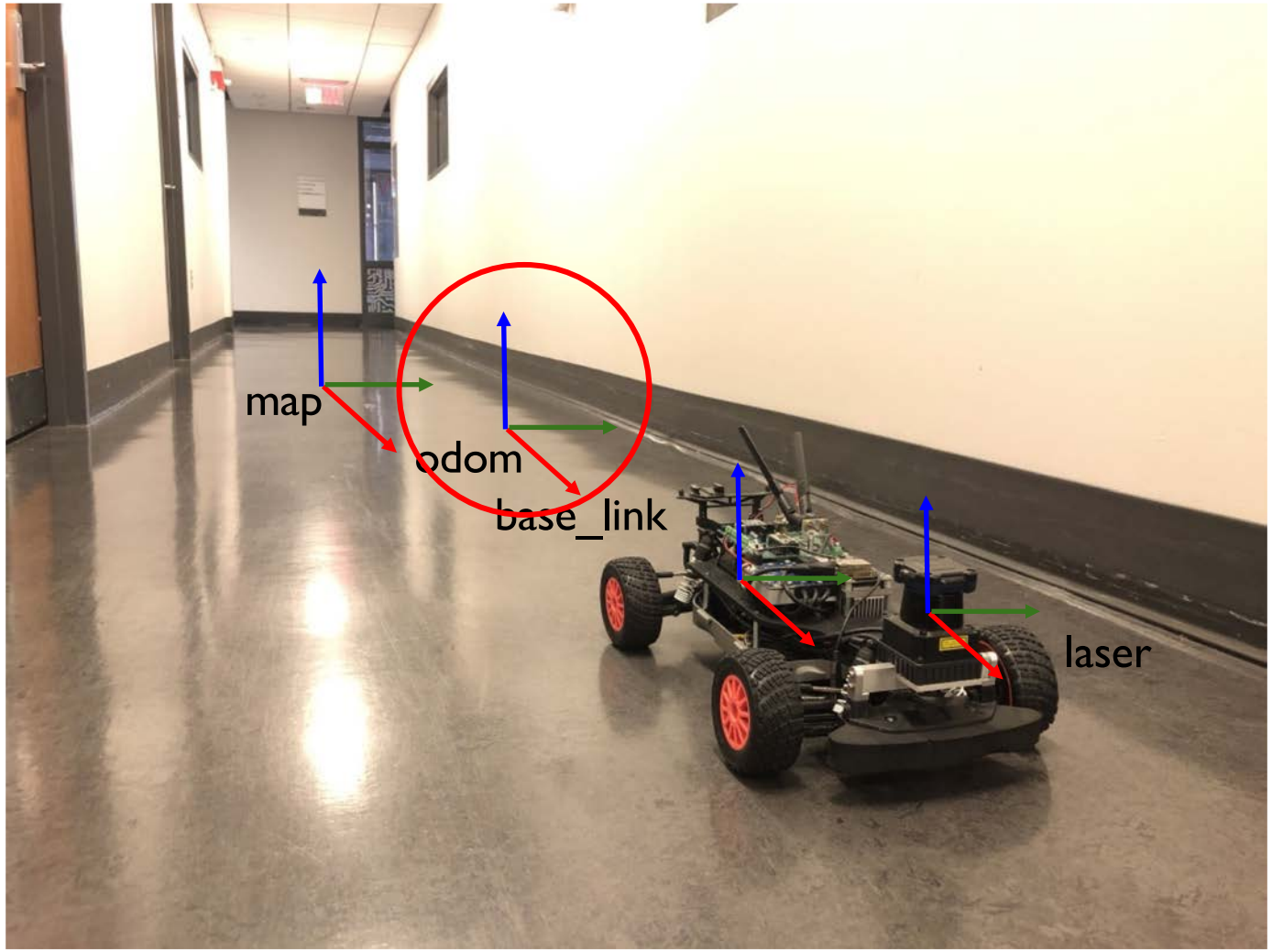
Frames we will use

3. laser
   - The frame in which the laser scan measurement is taken
   - Also moves with the car relative to the map frame



Frames we will use

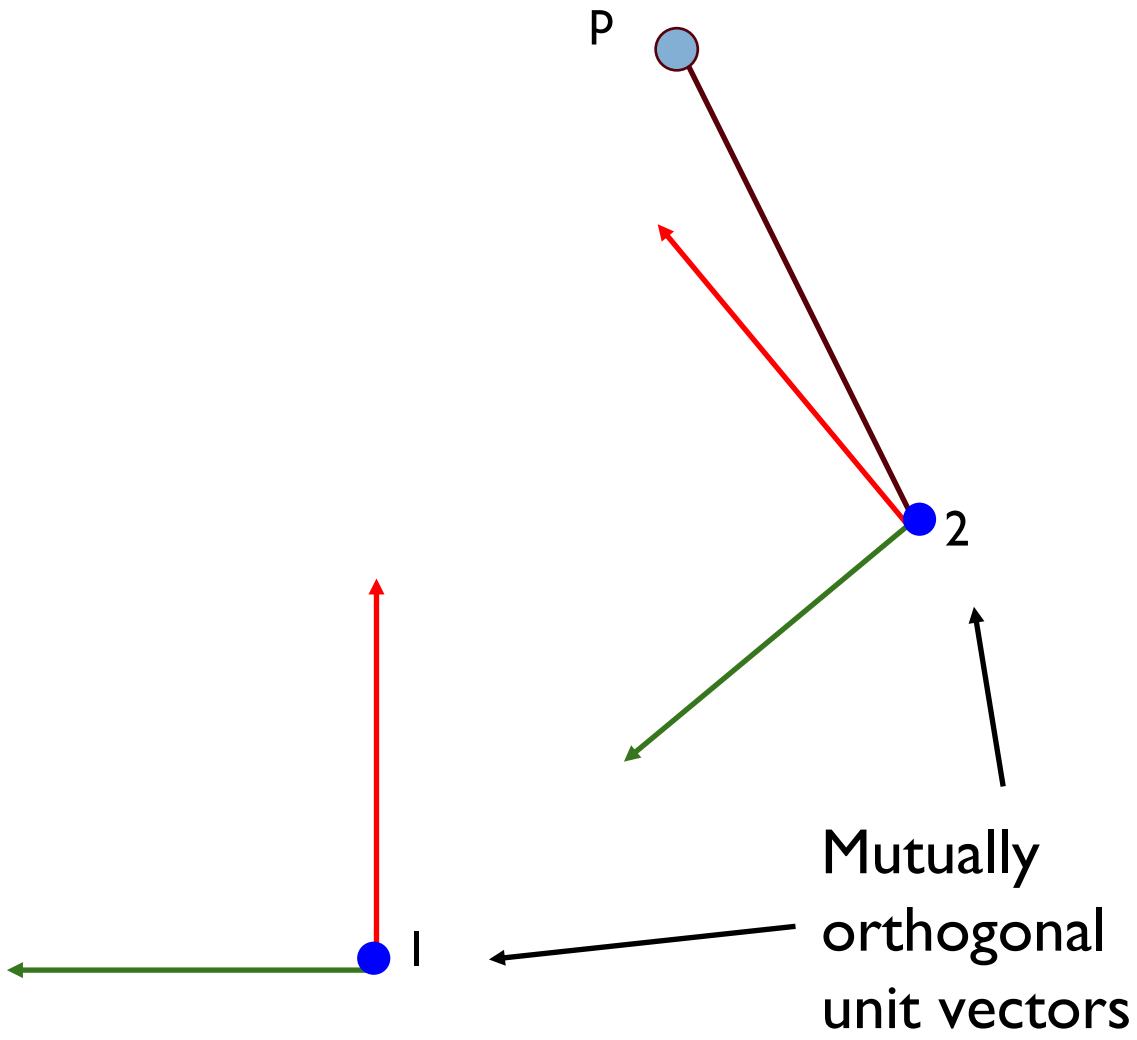
4. odom
   - The frame in which the odometry measurement is taken
   - Fixed relative to the map, origin is where the car started
   - Not explicitly defined



Transformations

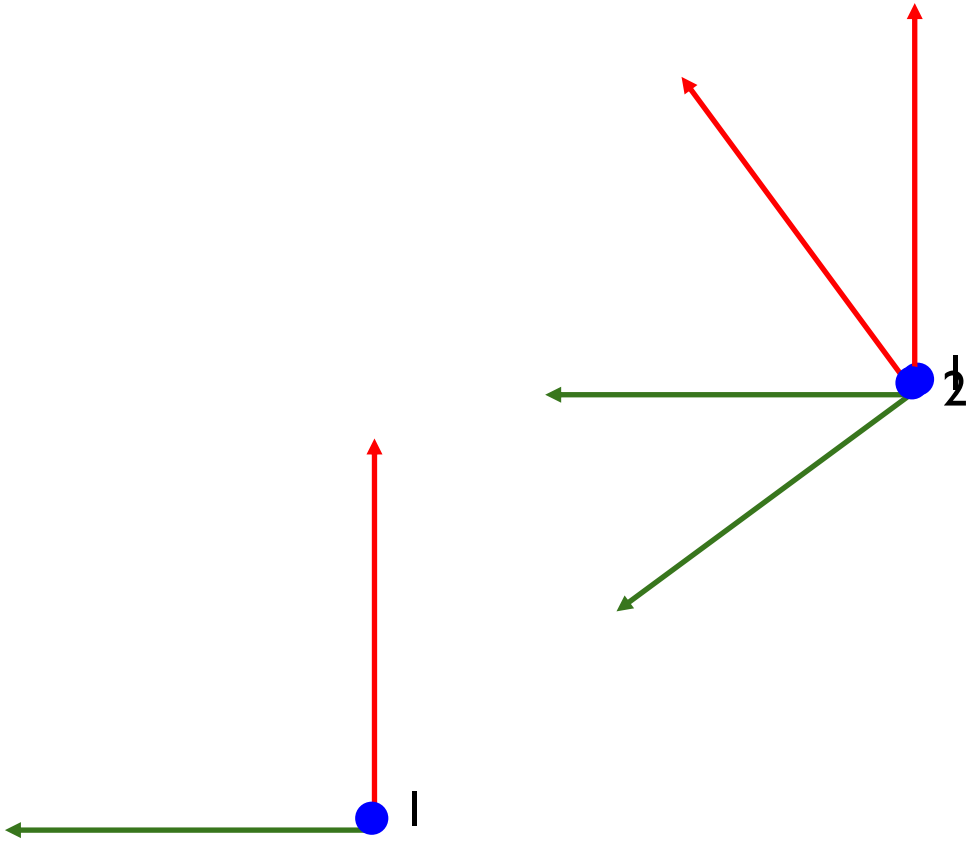
Rigid Body Transforms

Goal: We want the coordinate of point p in frame 2, given the coordinate of point p in frame 1.



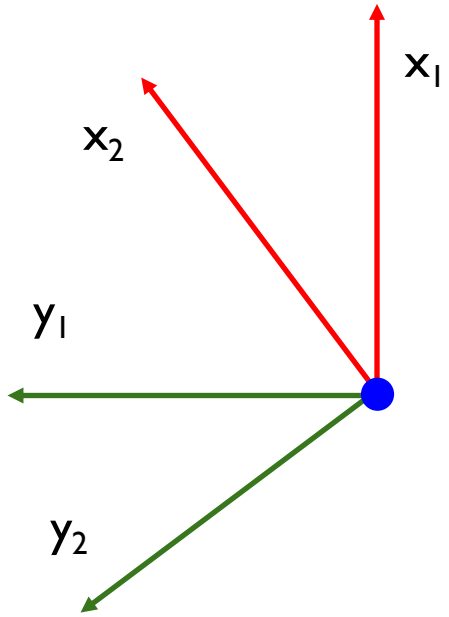
Rigid Body Transforms

Goal: We want the coordinate of point p in frame 2, given the coordinate of point p in frame 1.



Rotation Matrix

We can write the mutually orthogonal unit vectors in one frame, as linear combinations of the mutually orthogonal unit vectors in the other frame.



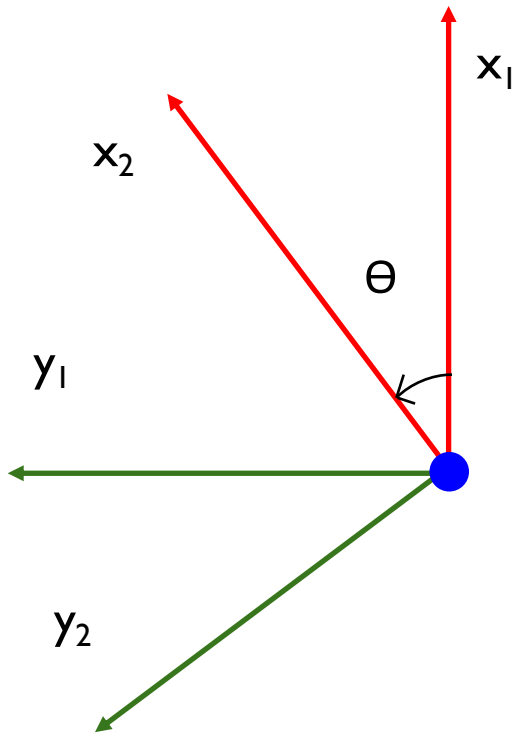
$$\mathbf{x}_2 = R_{11}\mathbf{x}_1 + R_{21}\mathbf{y}_1$$
$$\mathbf{y}_2 = R_{12}\mathbf{x}_1 + R_{22}\mathbf{y}_1$$

Rotation Matrix

Gather everything into a matrix:

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

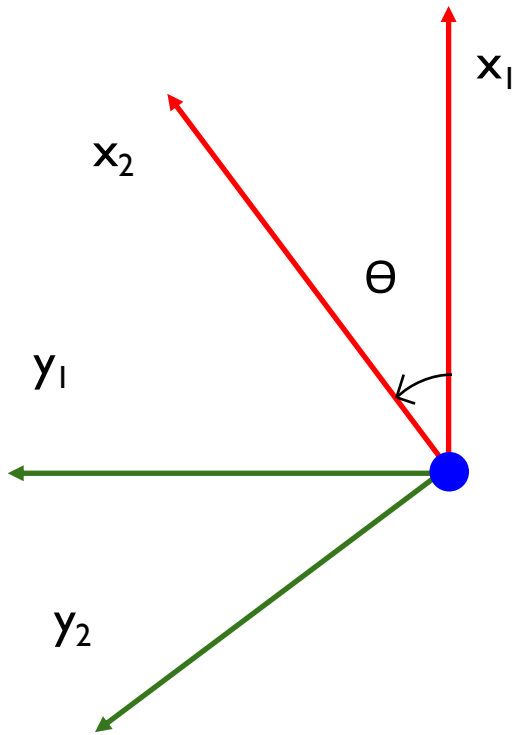
$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix}$$



Rotation Matrix

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

Question: What are the coefficients?

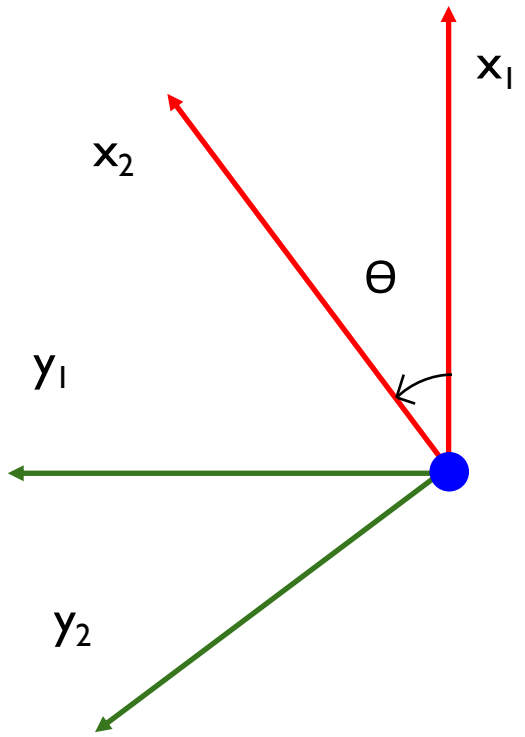


Rotation Matrix

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

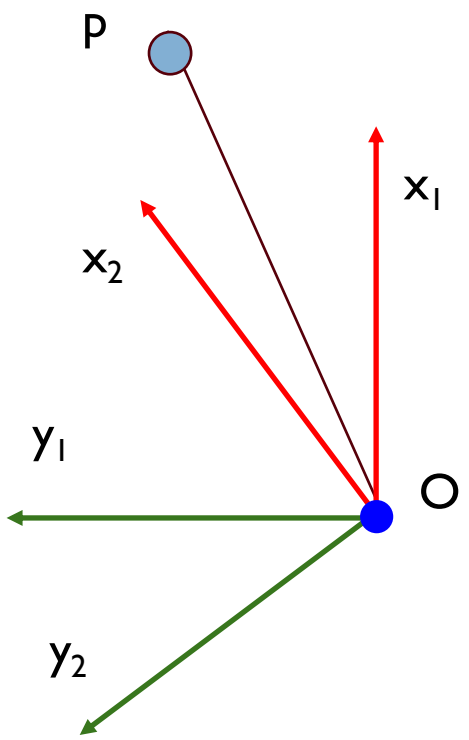
$$\mathbf{R} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

x y



Rotation Matrix

What if we need to find a point's coordinates in two frames?

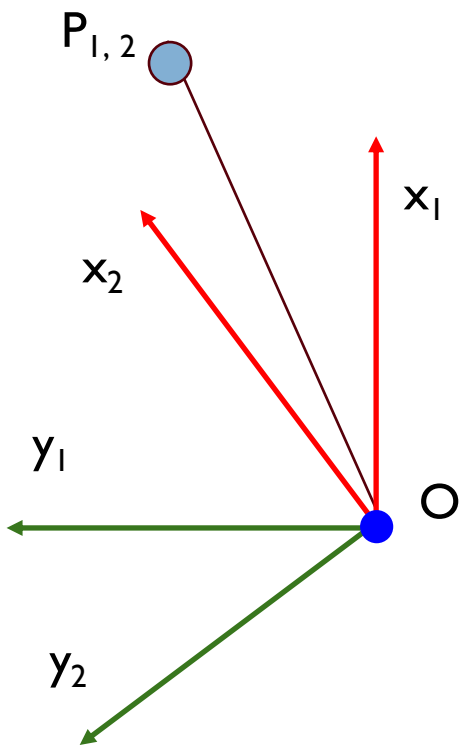


Rotation Matrix

Recall we represented the basis vectors of one frame as linear combinations of the other frame's basis vector. We can do the same here.

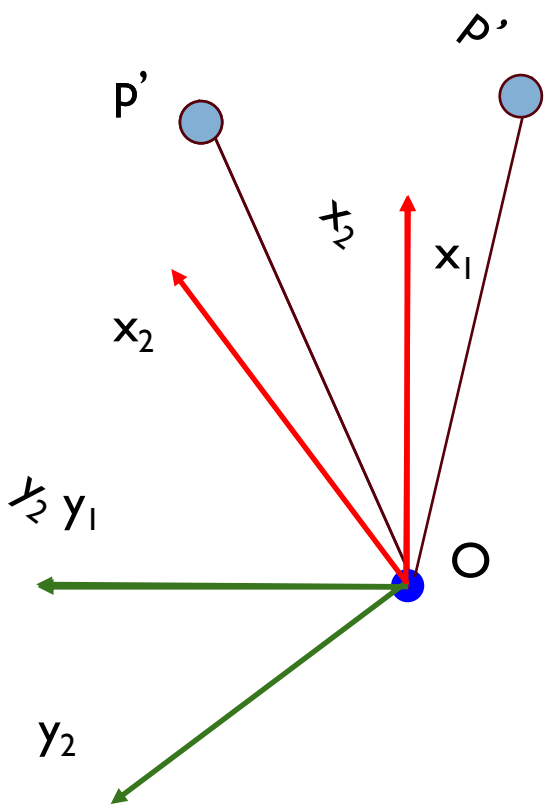
$$OP_1 = p_{x1}x_1 + p_{y1}y_1$$

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



Rotation Matrix

How do we relate the two?

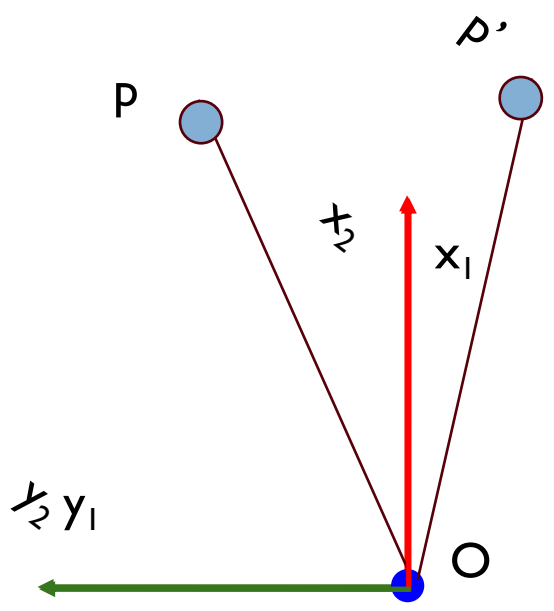


$$OP_1 = p_{x1}x_1 + p_{y1}y_1$$

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

Rotation Matrix

Instead of using two frames, we can represent these two vectors in the same basis vector now.



$$OP = p_x x_1 + p_y y_1$$

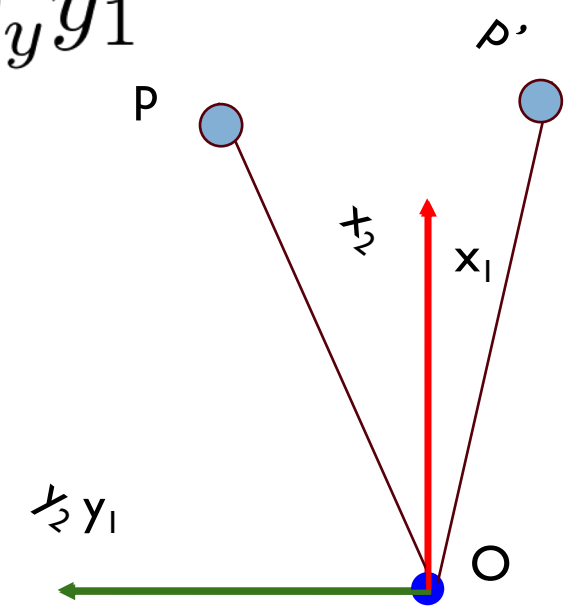
$$OP' = p'_x x_1 + p'_y y_1$$

Rotation Matrix

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

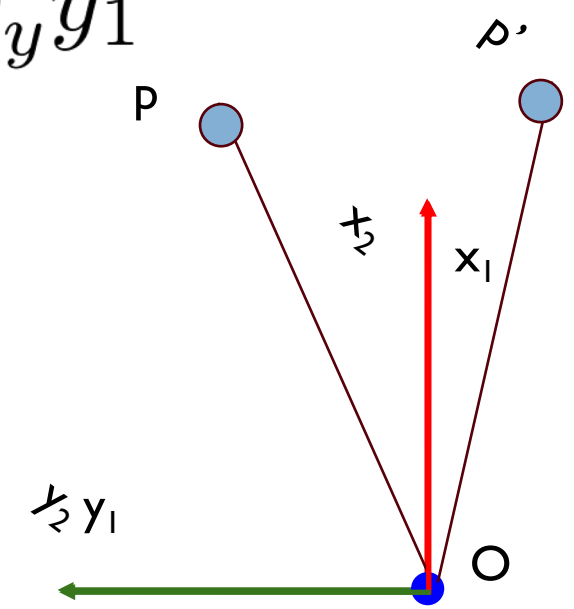
$$\begin{bmatrix} p'_x \\ p'_y \end{bmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

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



Rotation Matrix

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



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

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

Example: Rotations in 3D

Rotation matrices in 3D around each axis. See something similar?

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

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

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

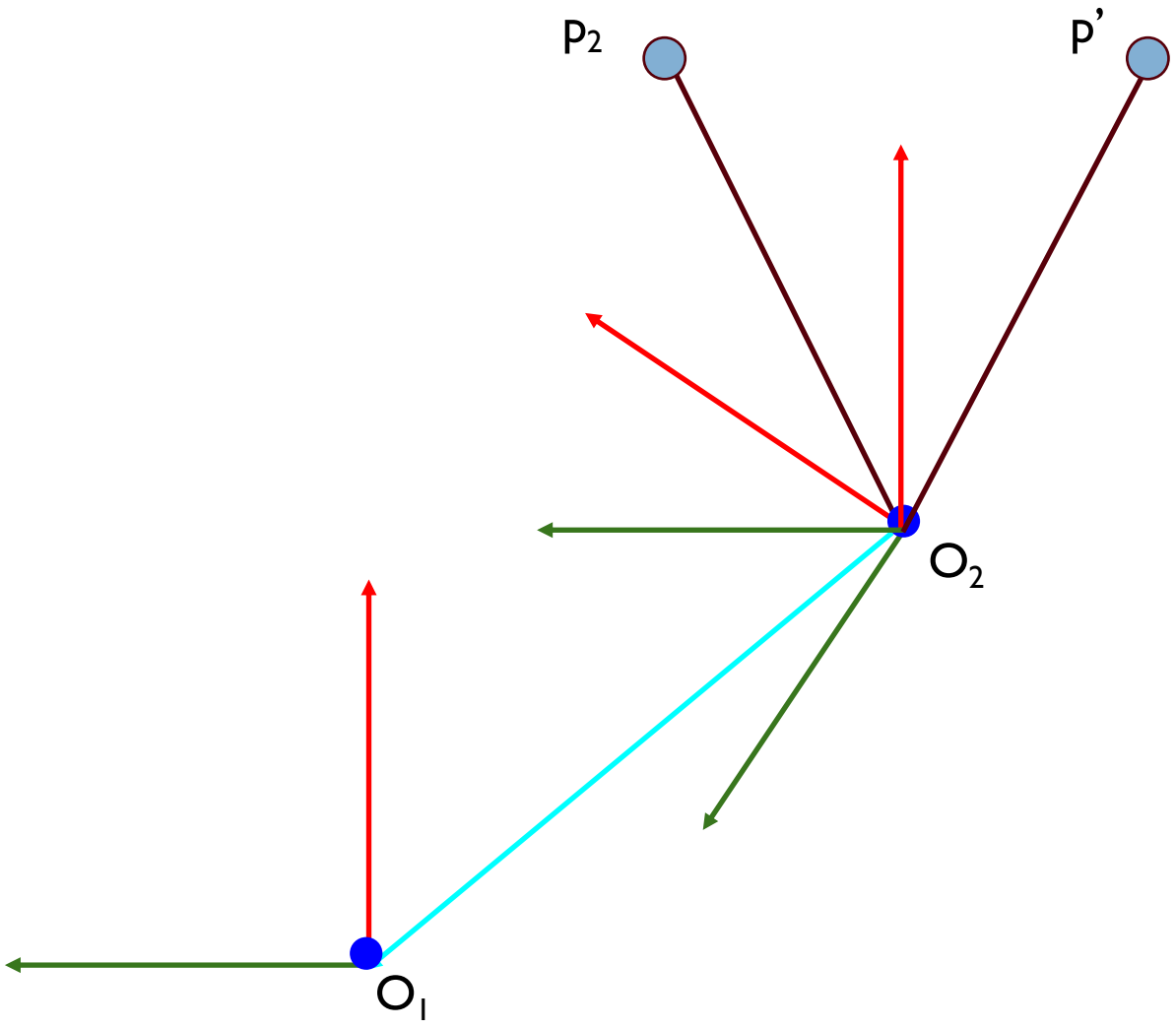
Other representations of rotations

- Quaternions: (ROS standard)
  - Supplementary materials in LaValle's Virtual Reality lectures.
- Euler angles
- Axis angle

Putting everything together (with translation)

First we apply the rotation

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

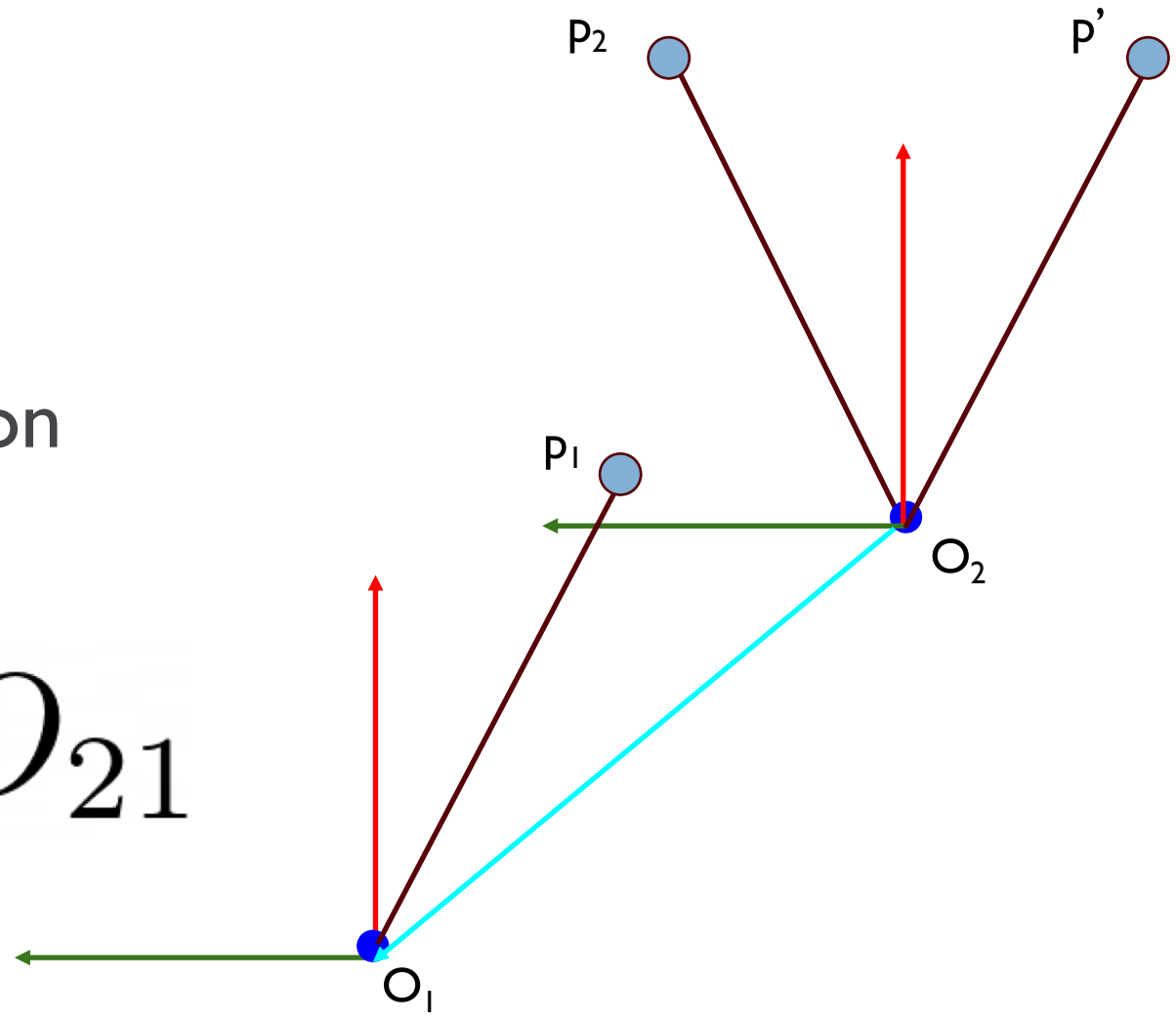


Putting everything together (with translation)

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

Then we apply the translation

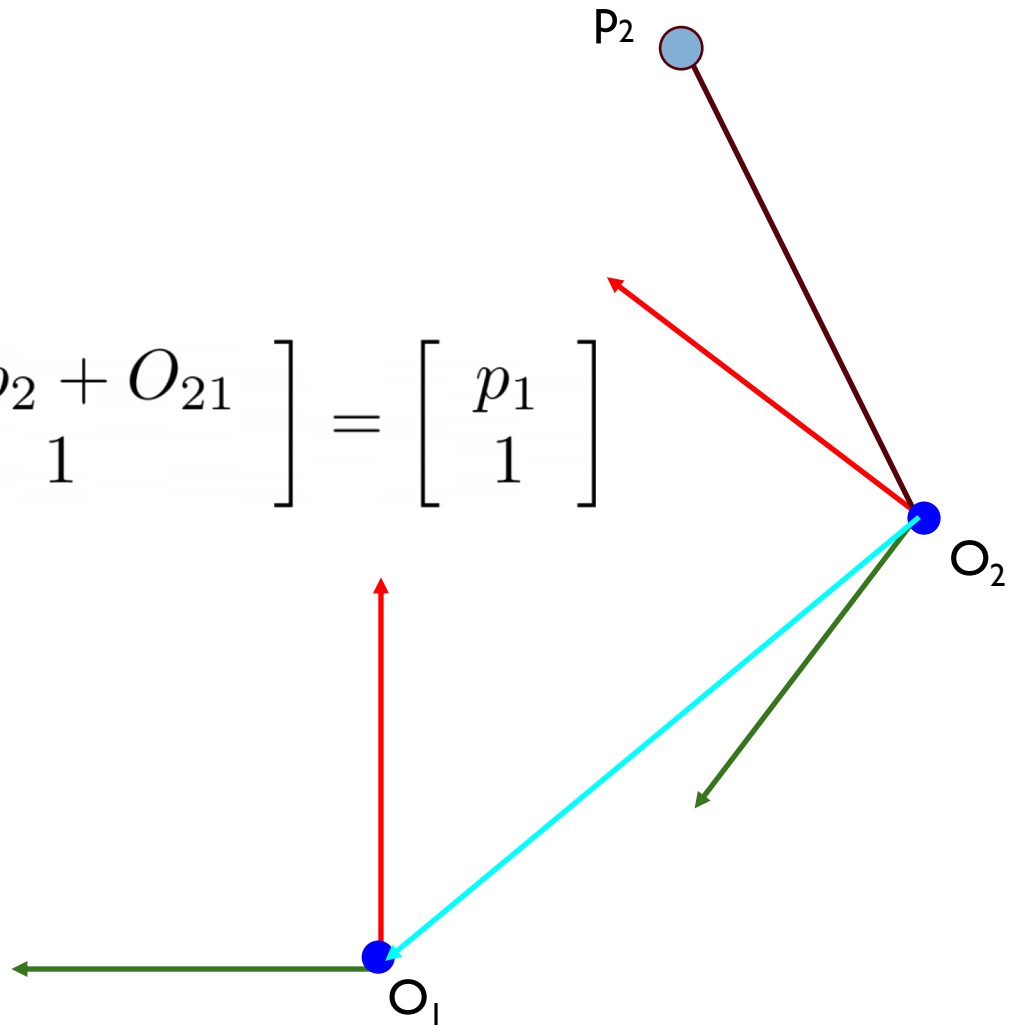
$$p_1 = \mathbf{R}_2^1 p_2 + O_{21}$$



Putting everything together

What if we want a more compact representation?

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



Homogeneous Transformations

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

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

- Here 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:

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

- Here p is a 2x1 (2D) / 3x1 (3D) vector

Transformations using ROS
(the main point of this lecture)

The ROS tf/tf2 Package

- Goal: Maintain relationship between reference frames overtime. And transform points/poses between two coordinates.
- Broadcast this information across ROS so that any node can subscribe to it.
- Will be very useful for all your future labs.

Highly Recommended Tutorials

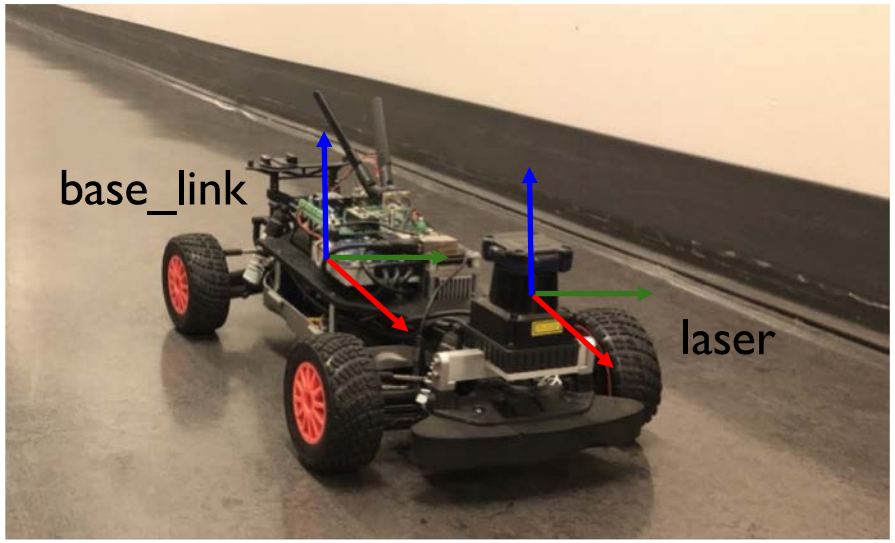
- http://wiki.ros.org/tf2/Tutorials
- http://wiki.ros.org/tf/Tutorials
- http://wiki.ros.org/tf2/Migration

ROS tf Conventions

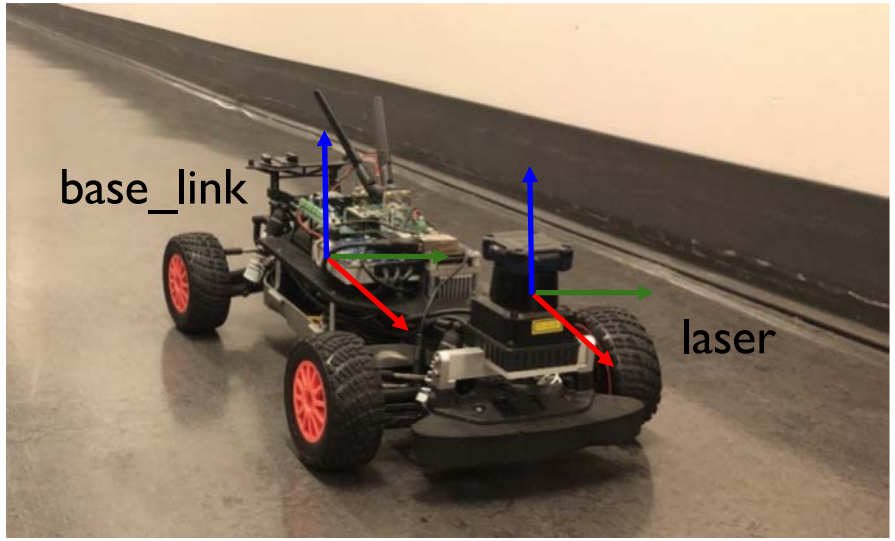
- Units, orientation conventions, rotation representations, and covariance representations: https://www.ros.org/reps/rep-0103.html
- Standard names for mobile base coordinate frames: https://www.ros.org/reps/rep-0105.html

Static Transforms

- Frame for various hardware components
- Usually defined by robot description/urdf files
- You can also broadcast a static transform
- There exists a static transform between laser and base_link (why?)



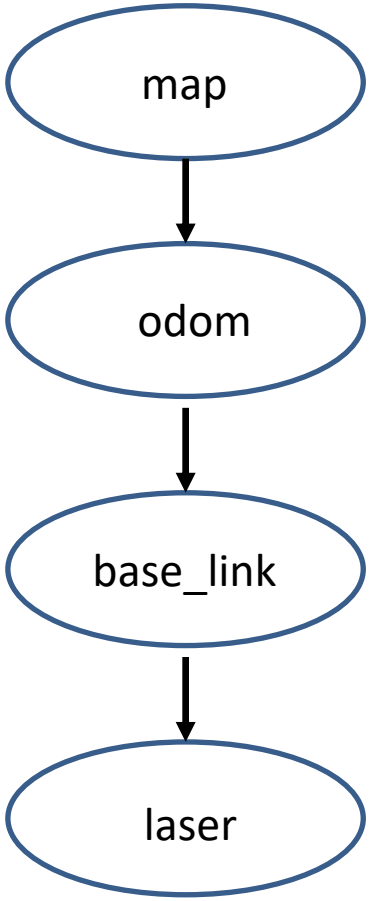
Example of a robot description




```
<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>
```

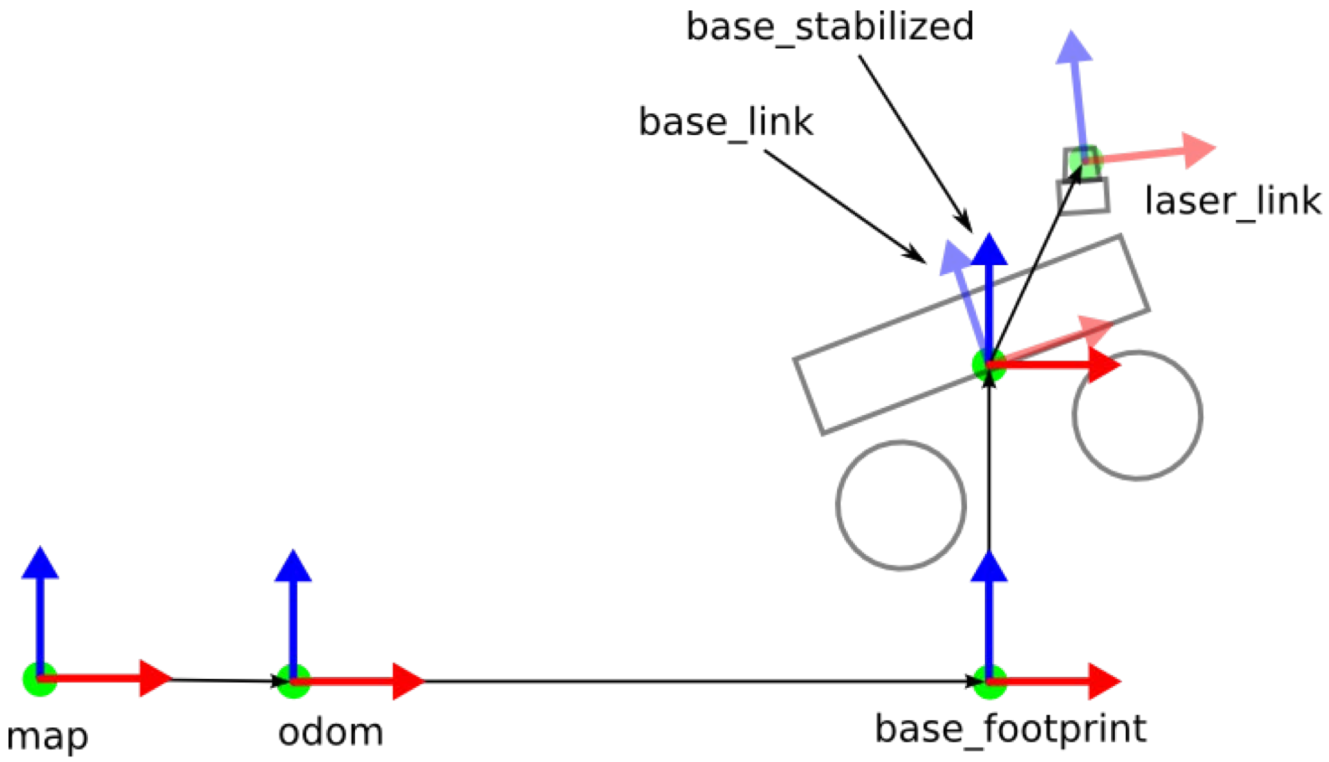
`model.xacro` **in the** `simulator` **package**

F1/10 : ROS Tf



Tf tree

All the single frames, all the single frames

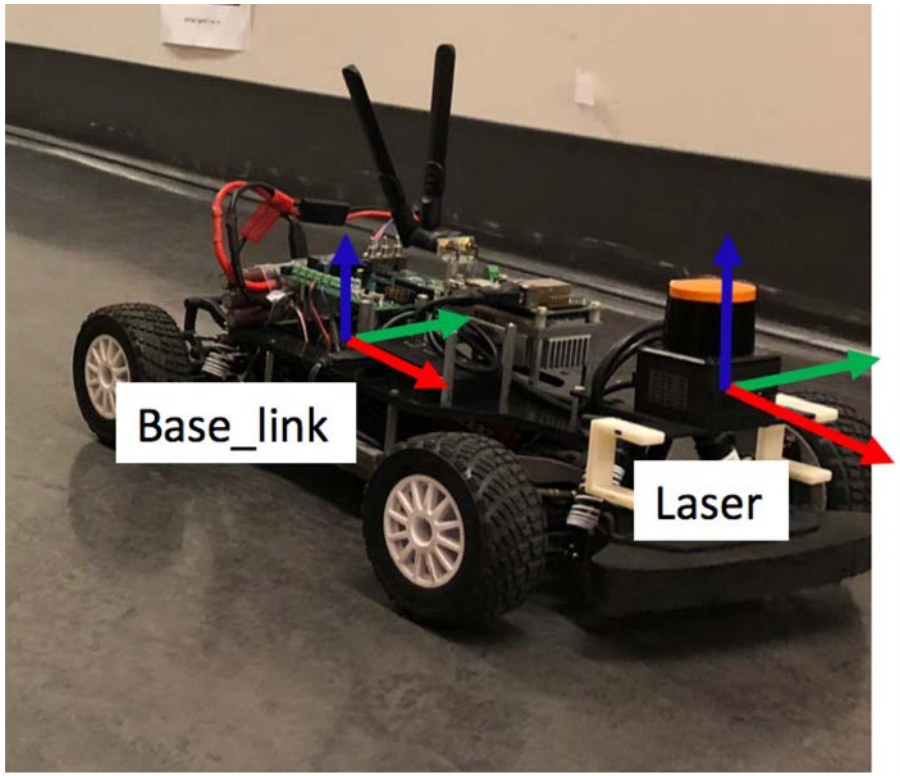


You might also see: base_stabilized (no roll or pitch) and base_footprint. (E.g. these are used by hector_slam).

If we assume a flat world for our car, we only need **map**, **odom**, **base_link**, and **laser** (no motion in Z direction)

Static transform

- There exists a static transform between laser and base_link (why?)
- Here, relative to base_link, laser has transform of 0.285 in x direction and 0.127 in z direction. No rotation in quaternions.




```
<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`

ROS Implementation of Static Transform Publisher

```
<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
```

- Placing this line above in the launch file creates a static transform between base_link and laser.
- Published on the topic /tf_static
- X offset of 0.285 meters and z offset of 0.127
- No rotation in quaternions (qx, qy, qz, and qw)
- http://wiki.ros.org/tf2_ros

tf Tree

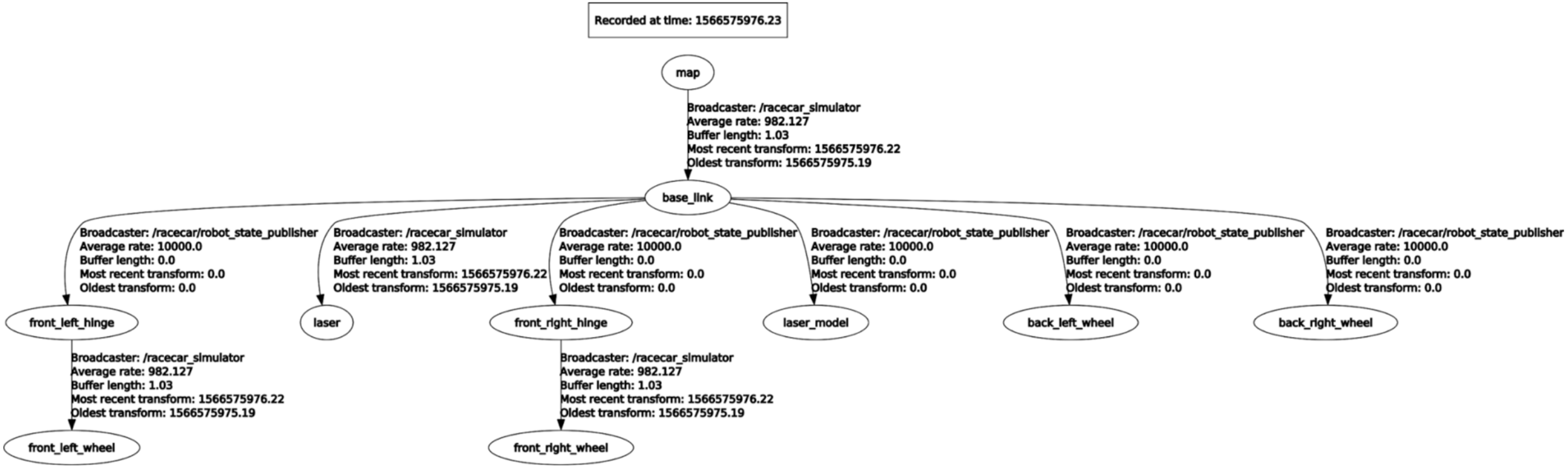
- The tf tree in ROS gives us all the frames, their relative transforms, who is broadcasting each transform, and how fast they're being broadcasted
- If a frame is connected to the tf tree, we can obtain all transforms between this frame and all other frames in the tree via tf
- Visualizing it is a great way to debug

tf Tree

- To see the current tf tree
  - `$ rosrun tf view_frames`
  - `$ rosrun rqt_tf_tree rqt_tf_tree`

tf Tree

Example: frames in the simulator



Some other useful Command Line Tools

Display debug information on where tf is coming from etc.

- `$ 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
```

Some other useful Command Line Tools

Echo tf information between two frames

- `$ rosrun tf tf_echo <source_frame> <target_frame> [echo_rate]`

```
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]
```

TF API

1. Relevant ROS messages
2. TF broadcasters
3. TF listeners
4. Coordinate transformations

TF Relevant ROS messages

The following information are in Python, there are equivalent C++ API too.

ROS has many messages for describing spatial information, especially within the geometry_msgs package.

- Useful for communicating spatial information between nodes
- Often used within other messages (e.g. the Odometry message)
- Some are natively visualizable in RViz
- Most message types have stamped and unstamped variants
- Point, PointStamped, Pose, PoseStamped…

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

TF API - Transform Listener

In ROS, a `tf.TransformListener` object can receive and cache transformations from broadcasters for querying. Some extremely useful methods:

- `listener.canTransform(target_frame, source_frame, time)`
- `listener.frameExists(frame_id)`
- `listener.getLatestCommonTime(source_frame, target_frame)`
- `listener.waitForTransform(target_frame, source_frame, time)`
- `listener.lookupTransform(target_frame, source_frame, time)`

(READ THE DOCS)

TF API - Transform Broadcaster

In ROS, a `tf.TransformBroadcaster` object can be used in a node to broadcast up-to-date transformation matrices, or quaternions between frames.

- `braodcaster.sendTransform(trans, rot, child_frame, parent_frame)`

(AGAIN, READ THE DOCS)

This is used extensively in our simulator.

Coordinate transformations

In ROS, a `tf.TransformListener` object can also be used to transform some of the geometry_msgs types.

- `listener.transformPoint(target_frame, point_stamped)`
- `listener.transformPose(target_frame, pose_stamped)`
- `...`

Note that the point/pose used by these methods must be stamped in the header by the source frame using the frame_id.

ROS also provides some handy helper functions for transformations:

- https://github.com/ros/geometry/blob/hydro-devel/tf/src/tf/transformations.py
- `roll,pitch,yaw = tf.transformations.euler_from_quaternion((x,y,z,w))`
- `x,y,z,w = tf.transformations.quaternion_from_euler((roll,pitch,yaw))`
- `...`

(AGAIN, READ THE DOCS)

What have we learned so far

- Coordinate Frames and Transformations: why do we need them?
- Reference Frames on F1/10
- Rigid Body Transformations
- tf/tf2_ros (Transformations in ROS)

Tf exercise in Turtlesim

Demo