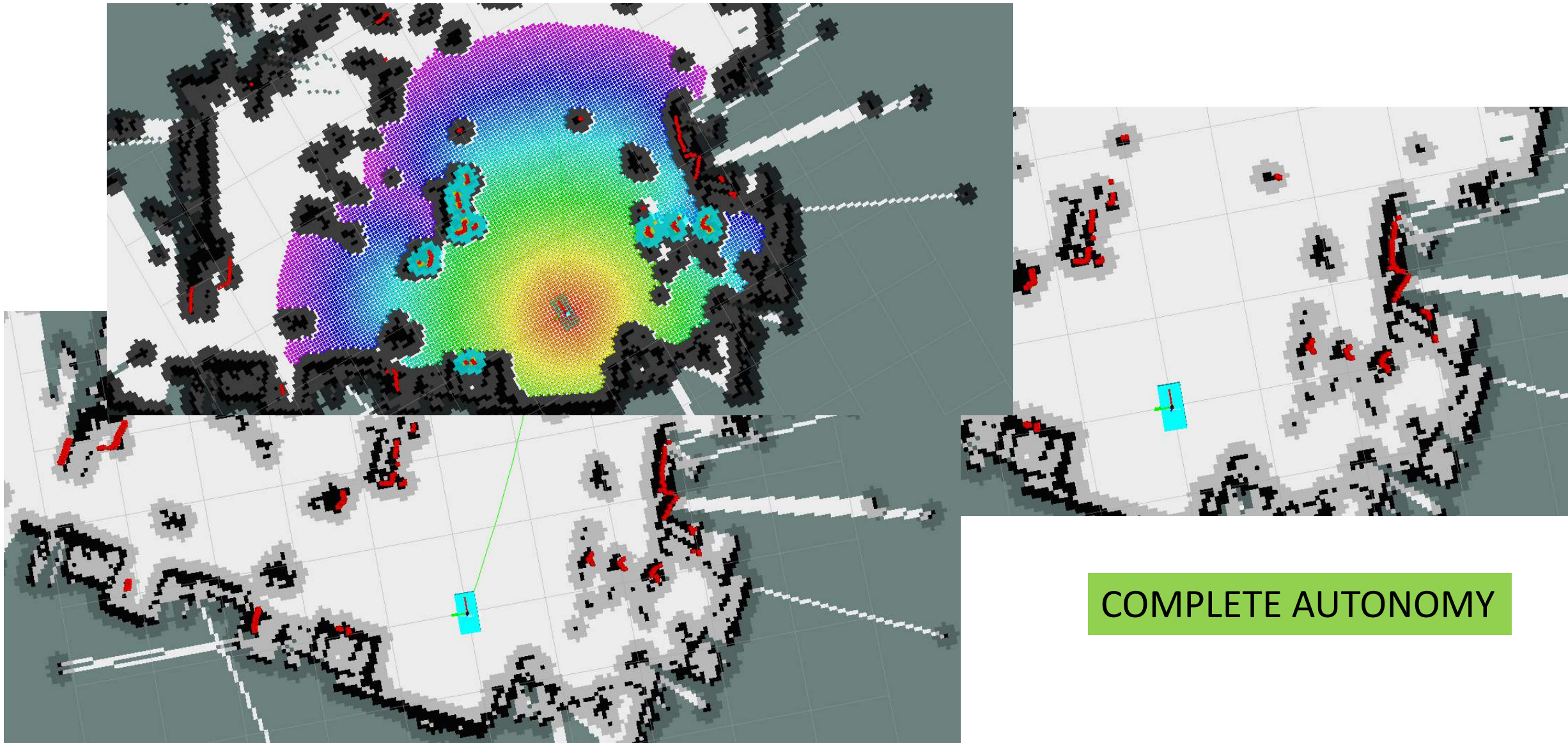
Autonomous Racing

1/10 the scale. 10 times the fun!

# ROS Transformations and frames

Madhur Behl
(University of Virginia)

# Where are we headed to



COMPLETE AUTONOMY

# Account for the forward motion of the car

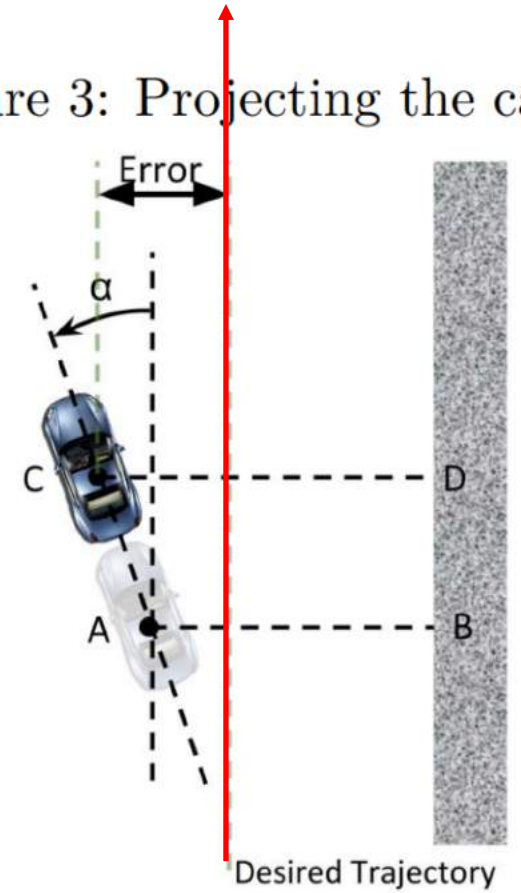



Figure 3: Projecting the car future in time

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

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

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

Error = desired trajectory – CD

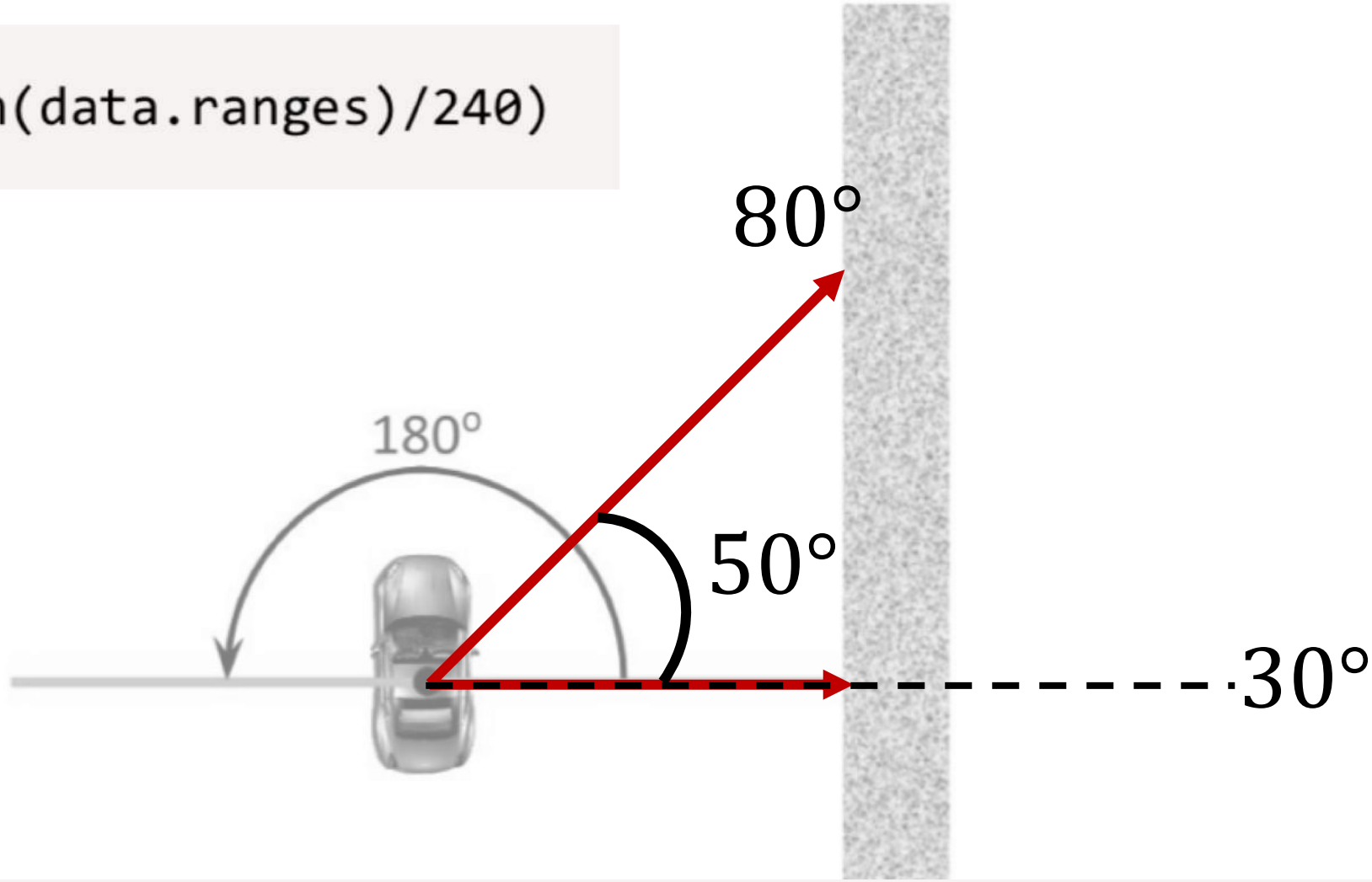
# PID Steering Control

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

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

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

```
index = theta * (len(data.ranges)/240)
```

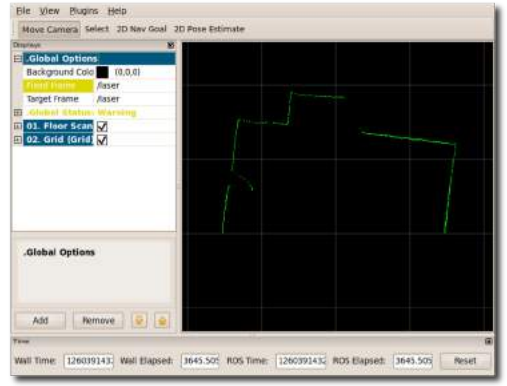



```
zero_ray_index = 30 * (len(data.ranges)/240) = 0.125 * (len(data.ranges)/240)
```

# control.py

Publishes to topic – **drive_parameters**

Subscribes to : **error**


**ROS over Network**

Subscribes to– **drive_pwm**

**Remote connection**

**SSH >> rviz visualization**

# dist_finder.py

Publishes topic – **error** | message type pid_input.msg

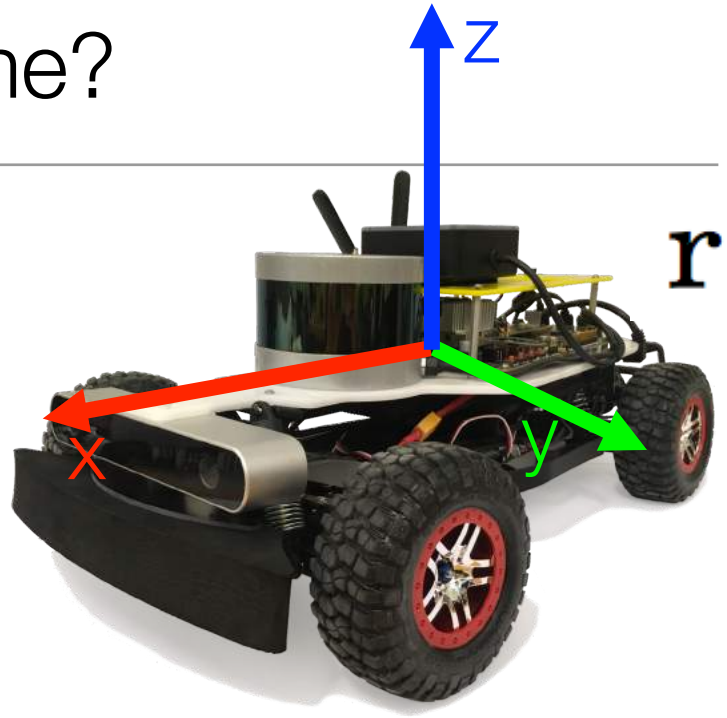
Subscribes to topic – **scan**

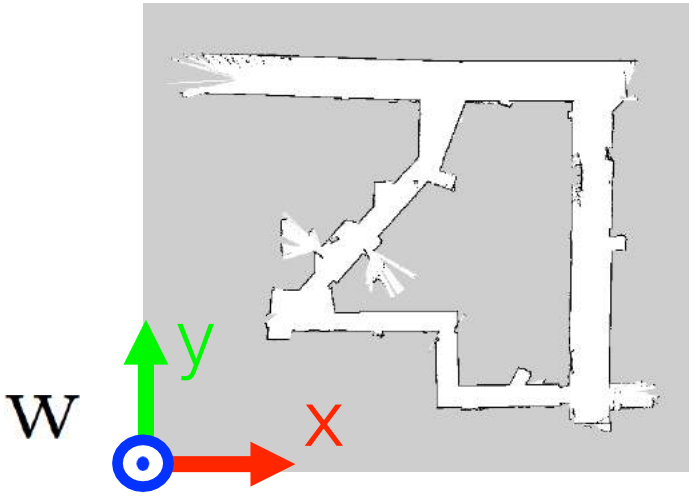
# Transformations and Frames:

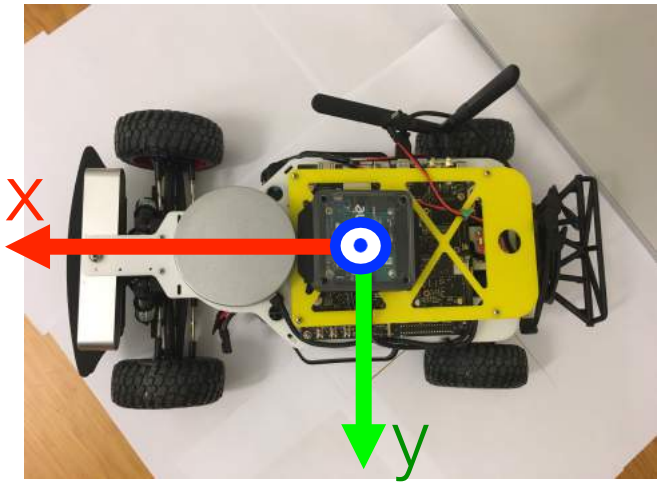
Map frame – where are you w.r.t the map – co-oridnates from origin


# 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**



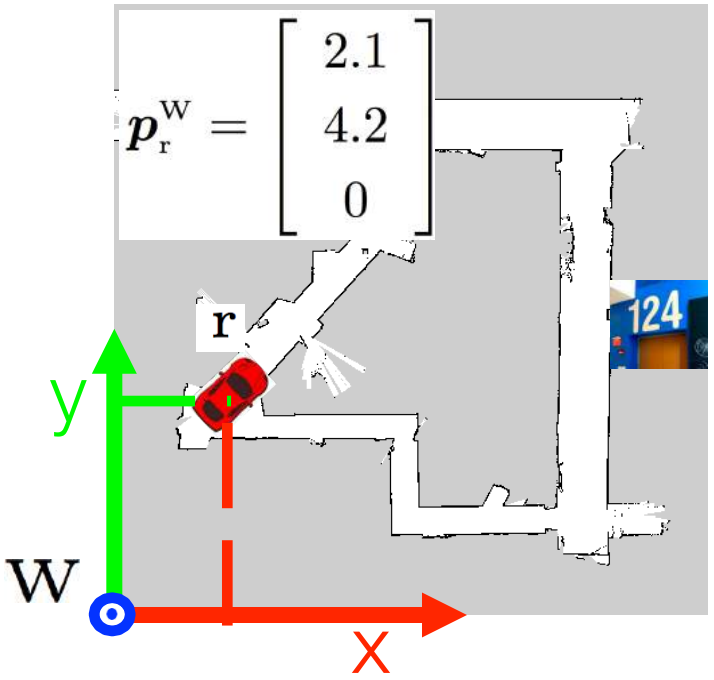



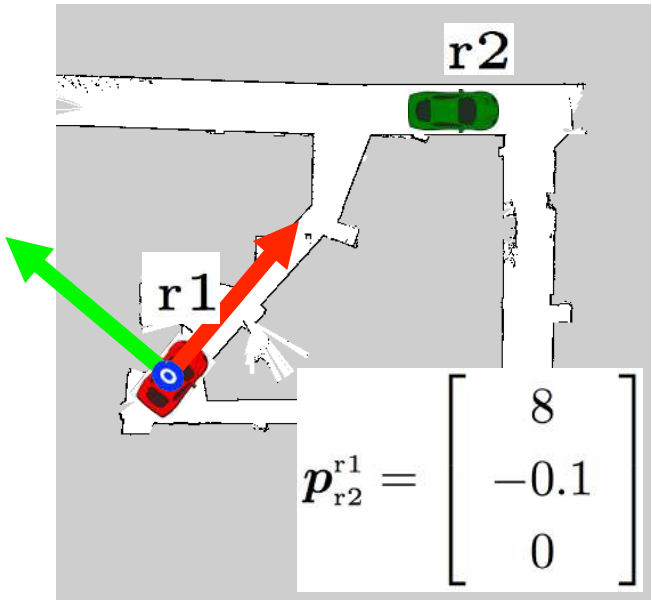




In many robotics problem, the first step is to assign a coordinate frame to all objects of interest

# Why do we need Coordinate Frames?

- compact representation of points

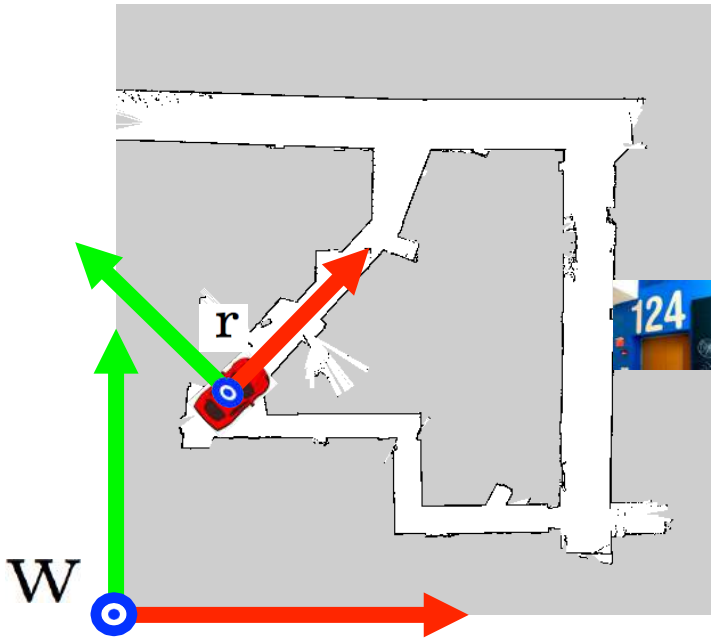


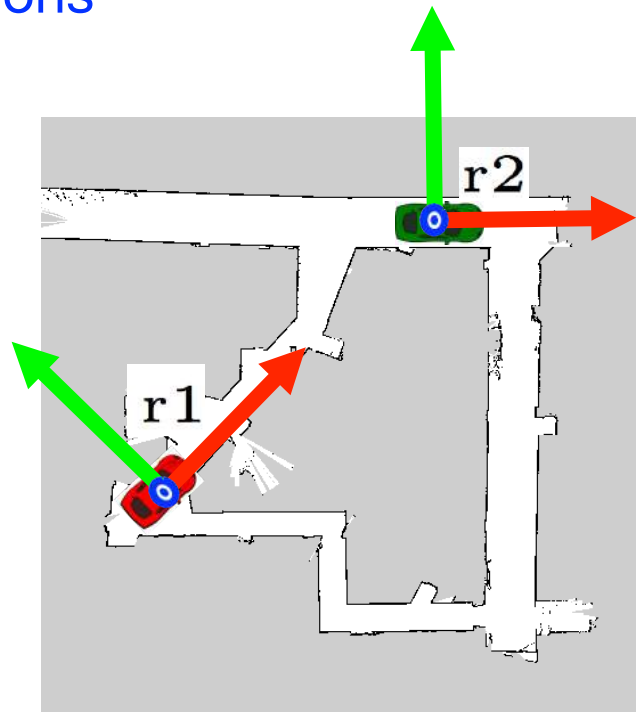




- coordinates have no meaning without specifying coordinate frame

# Why do we need Coordinate Frames?

- compact representation of points
- compact representation of orientations

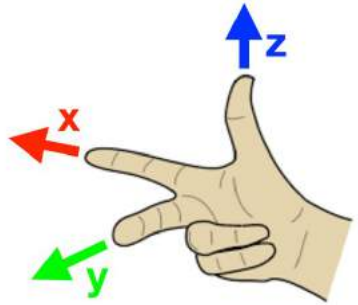


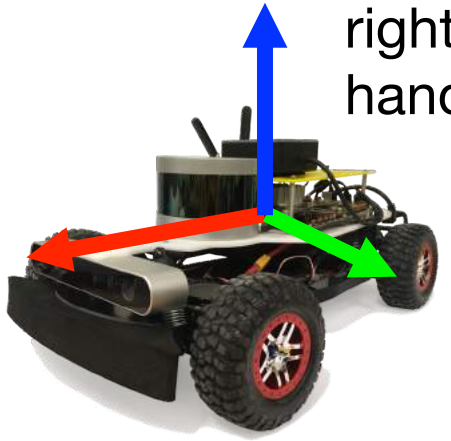




- **rotation, attitude, orientation**
- **pose** = position & rotation

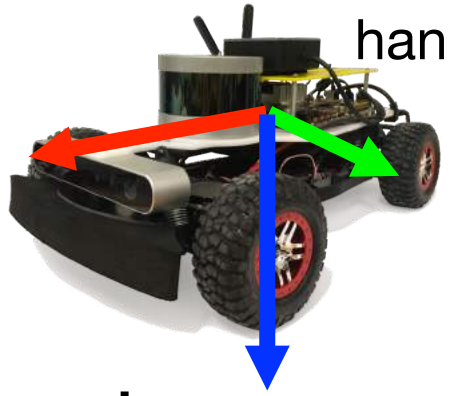
# Left-handed vs Right-handed Coordinate Frames

- **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?

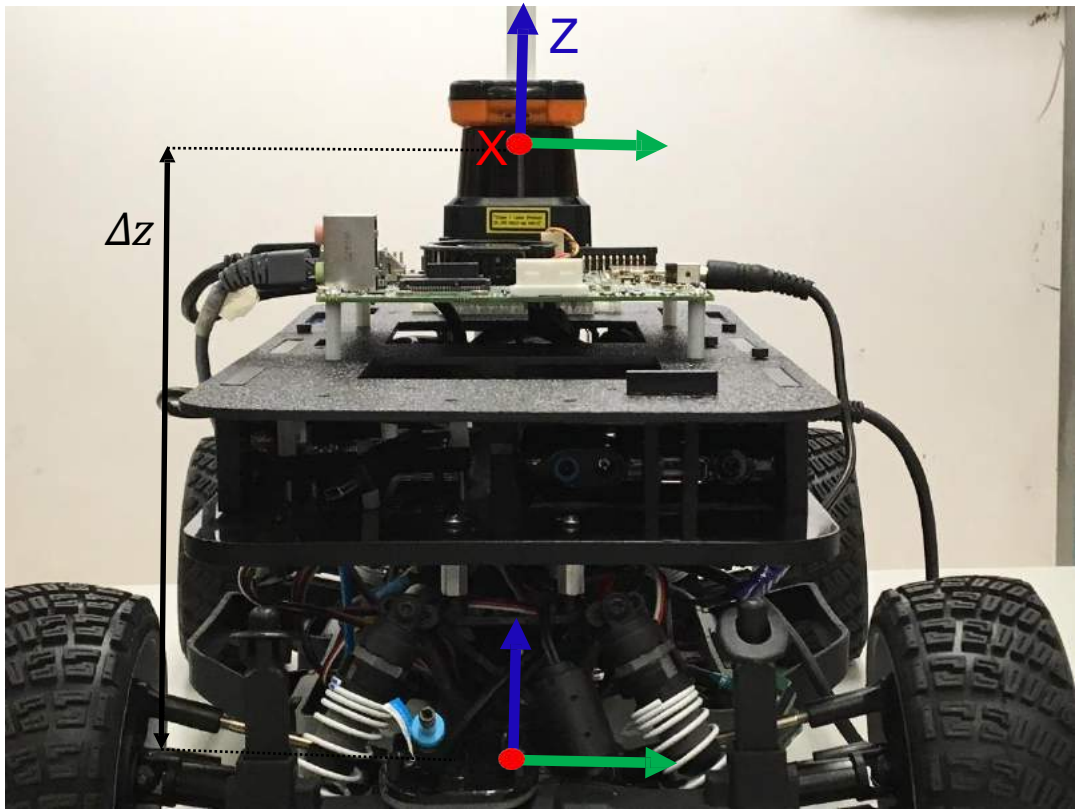
Does this tell you anything about where we are in the map?

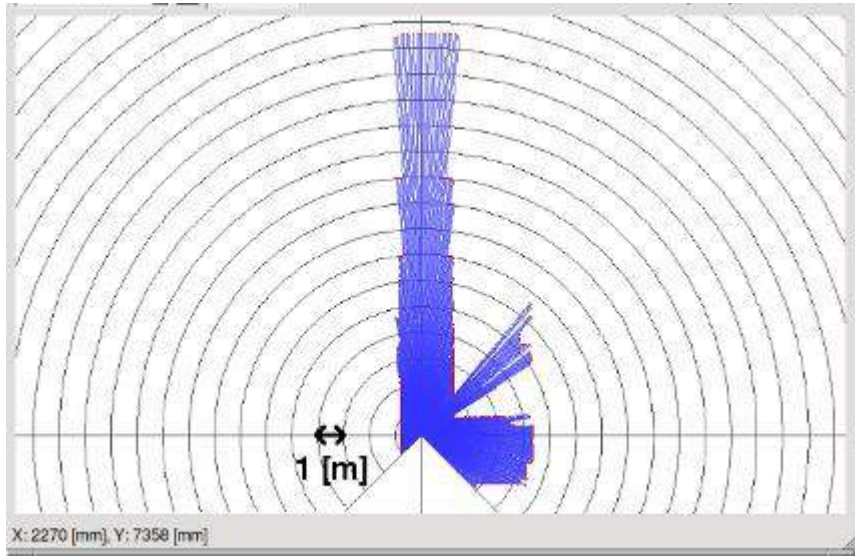
**We must link frames together**

Transformations

# Transformations and Frames

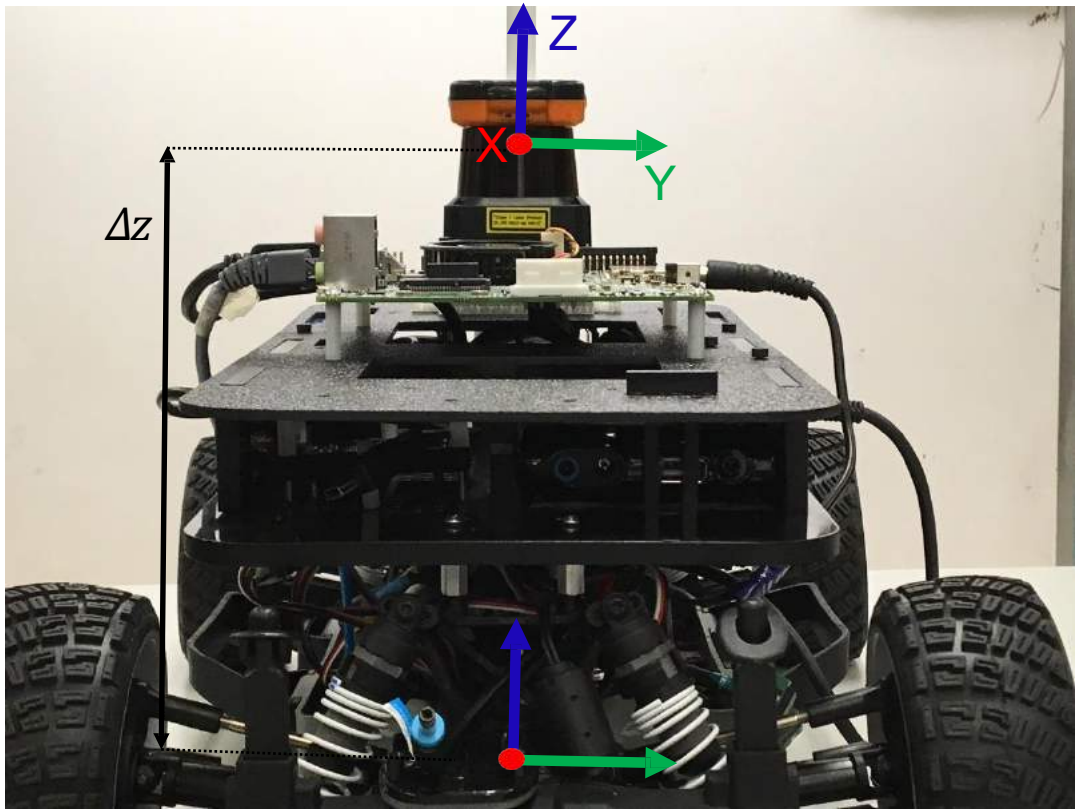
- The frame of reference in which the measurementis taken



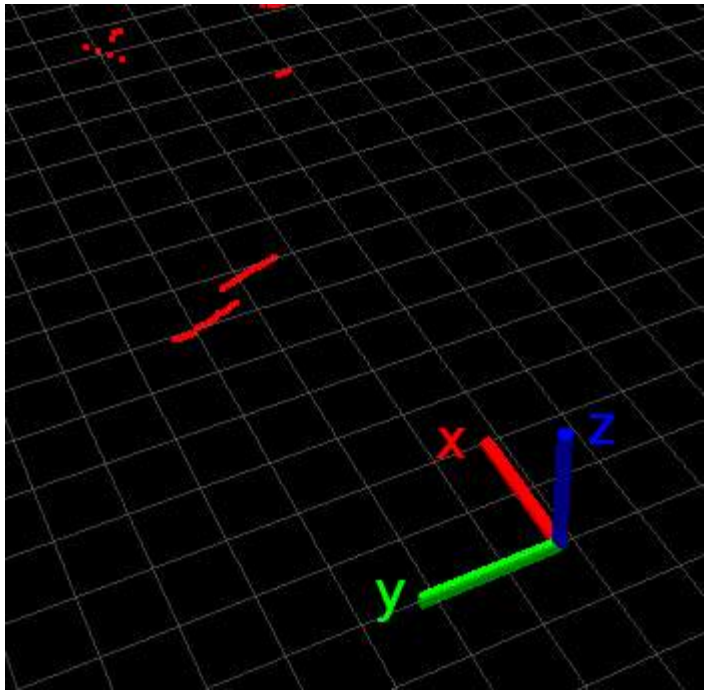



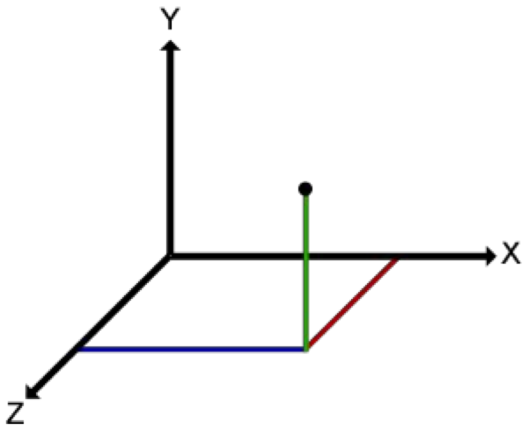

Distance measurements returned by LIDAR

# Transformations and Frames



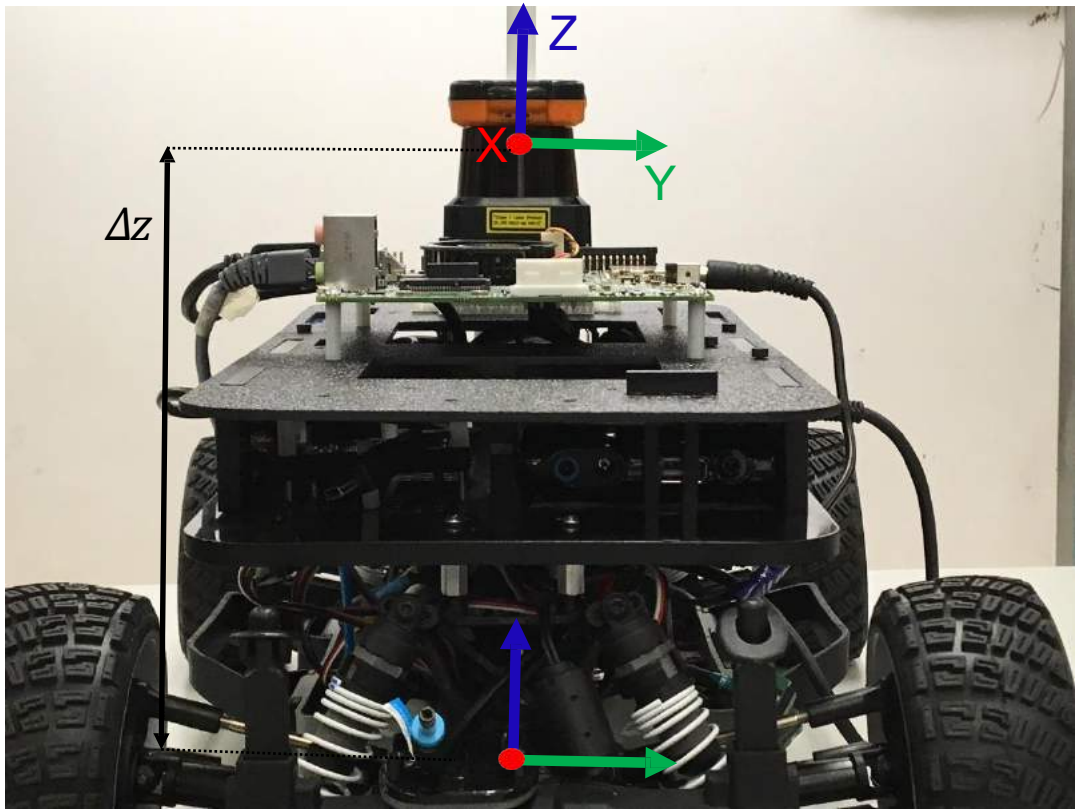

Note:
Axes X,Y,Z of Frames of Reference are orthogonal(90°) to each other.
X,Y,Z represent the axes along the 3 dimensions.



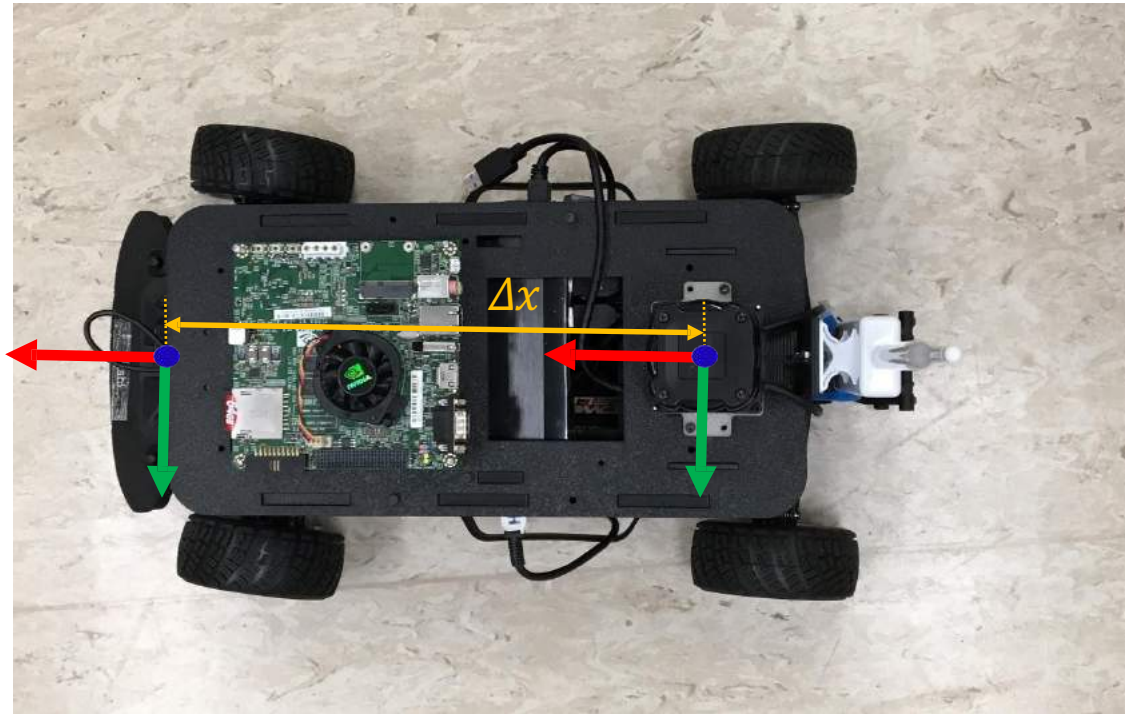



# Transformations and Frames

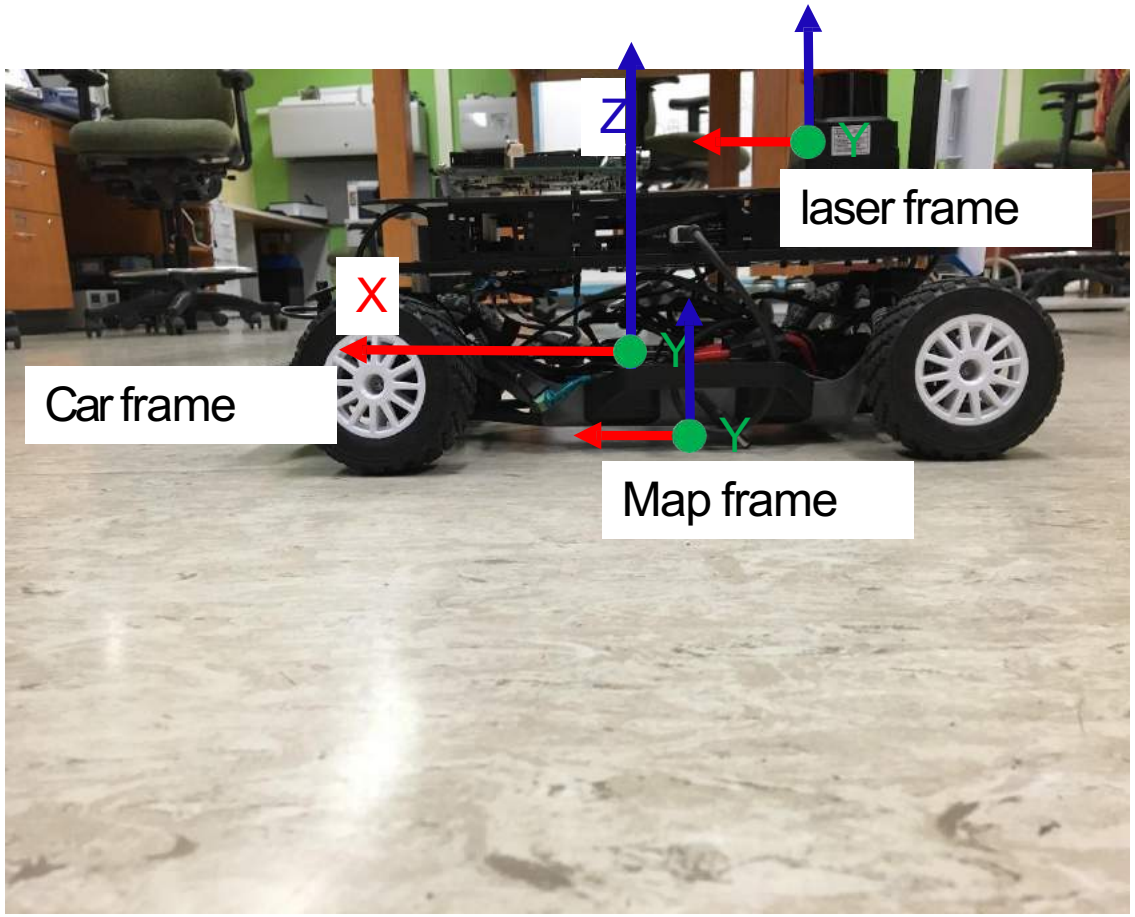
- The frame of reference in which the measurementis taken




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

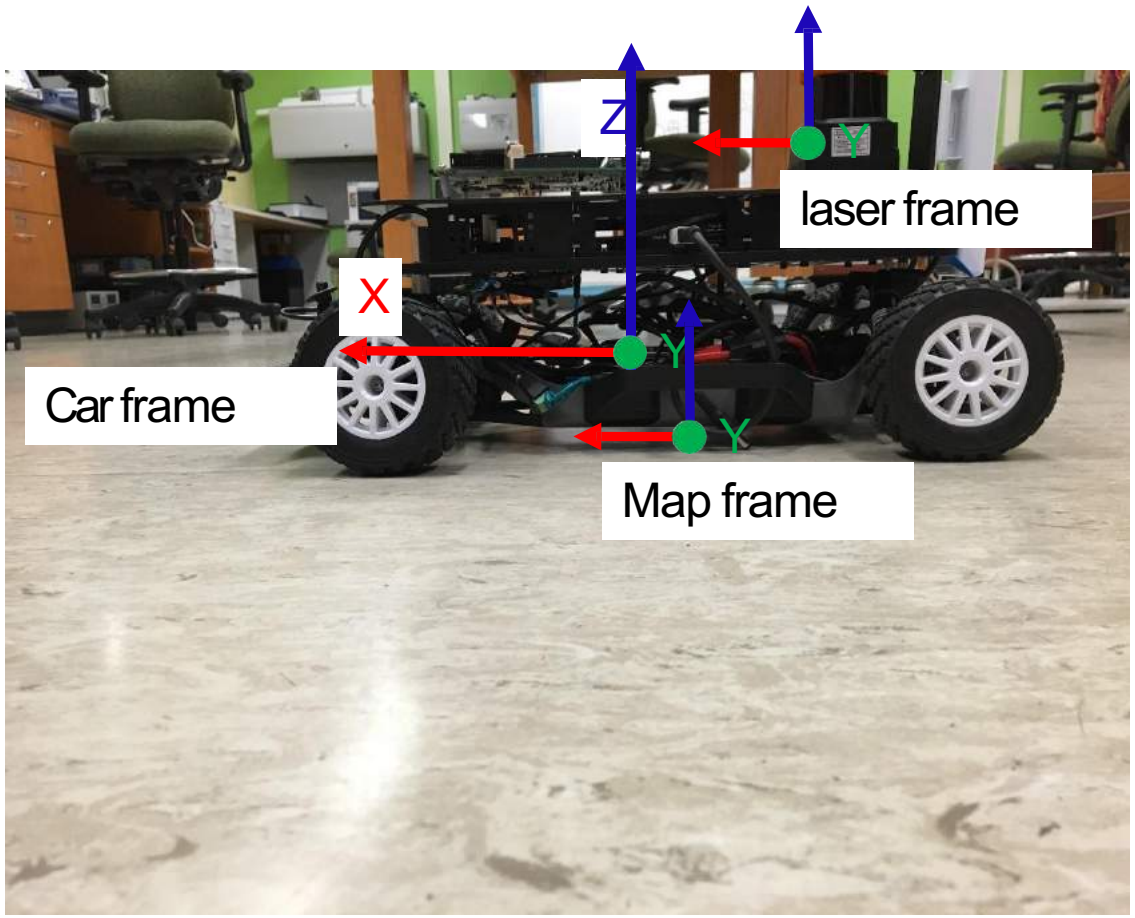


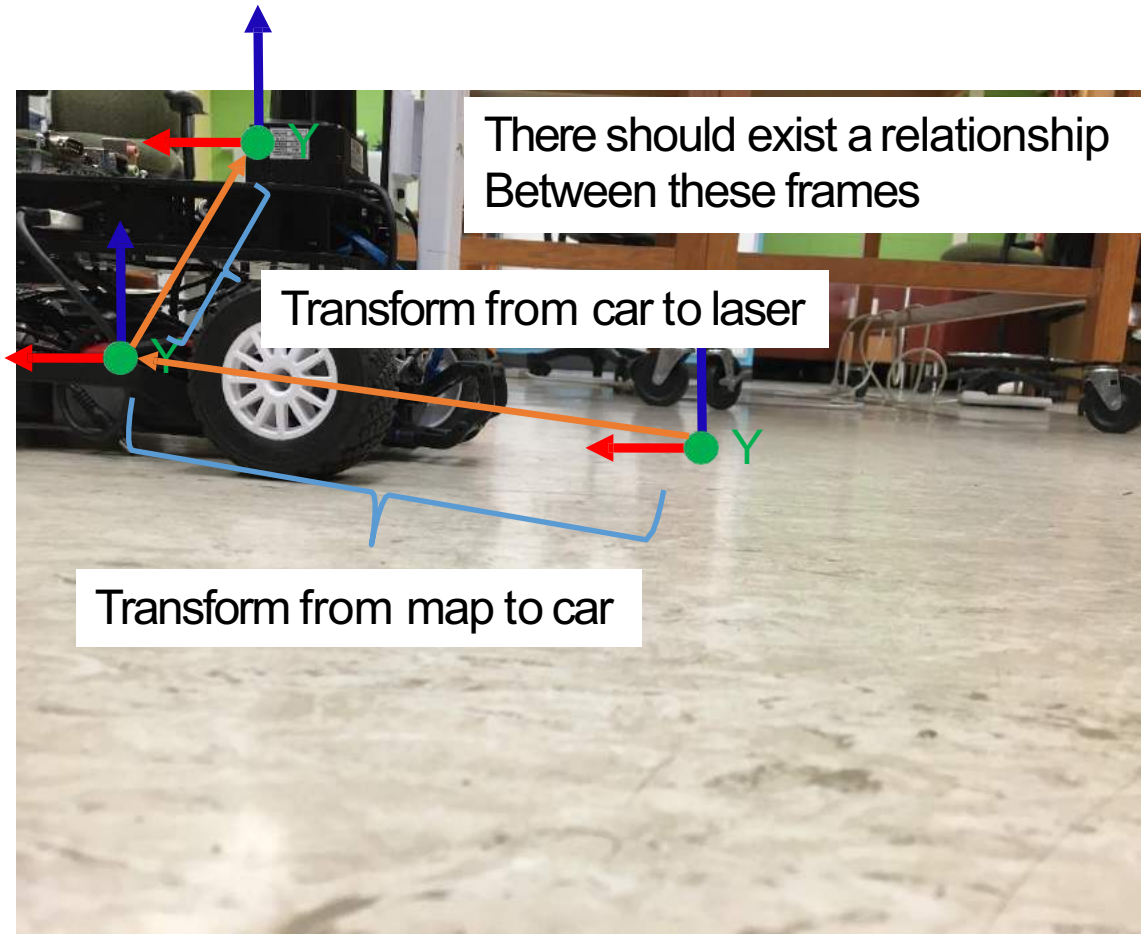

Between frames there will exist transformations that convert measurements from one frame to another

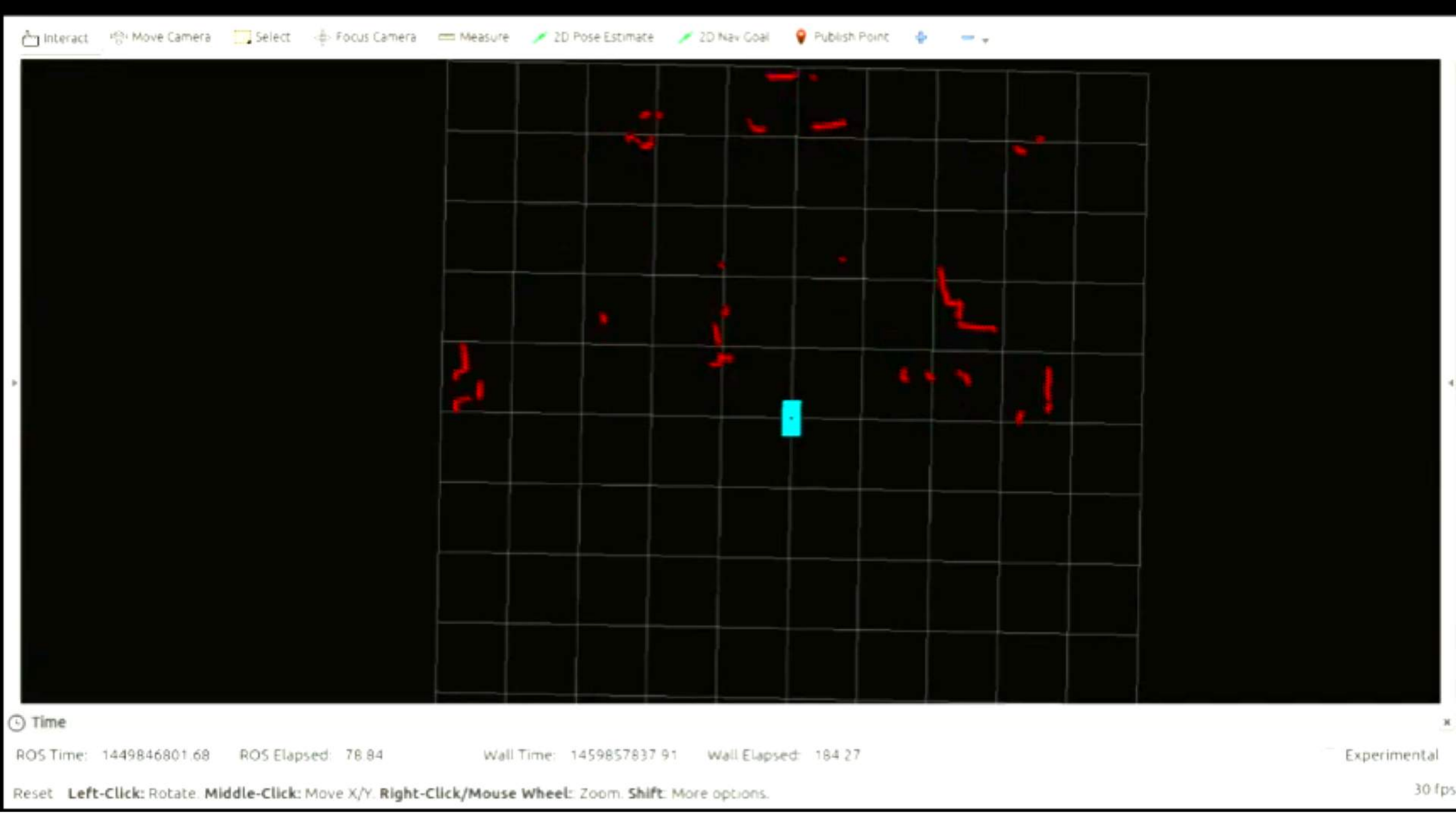
Important Point:
Note what the transformation means w.r.t frames

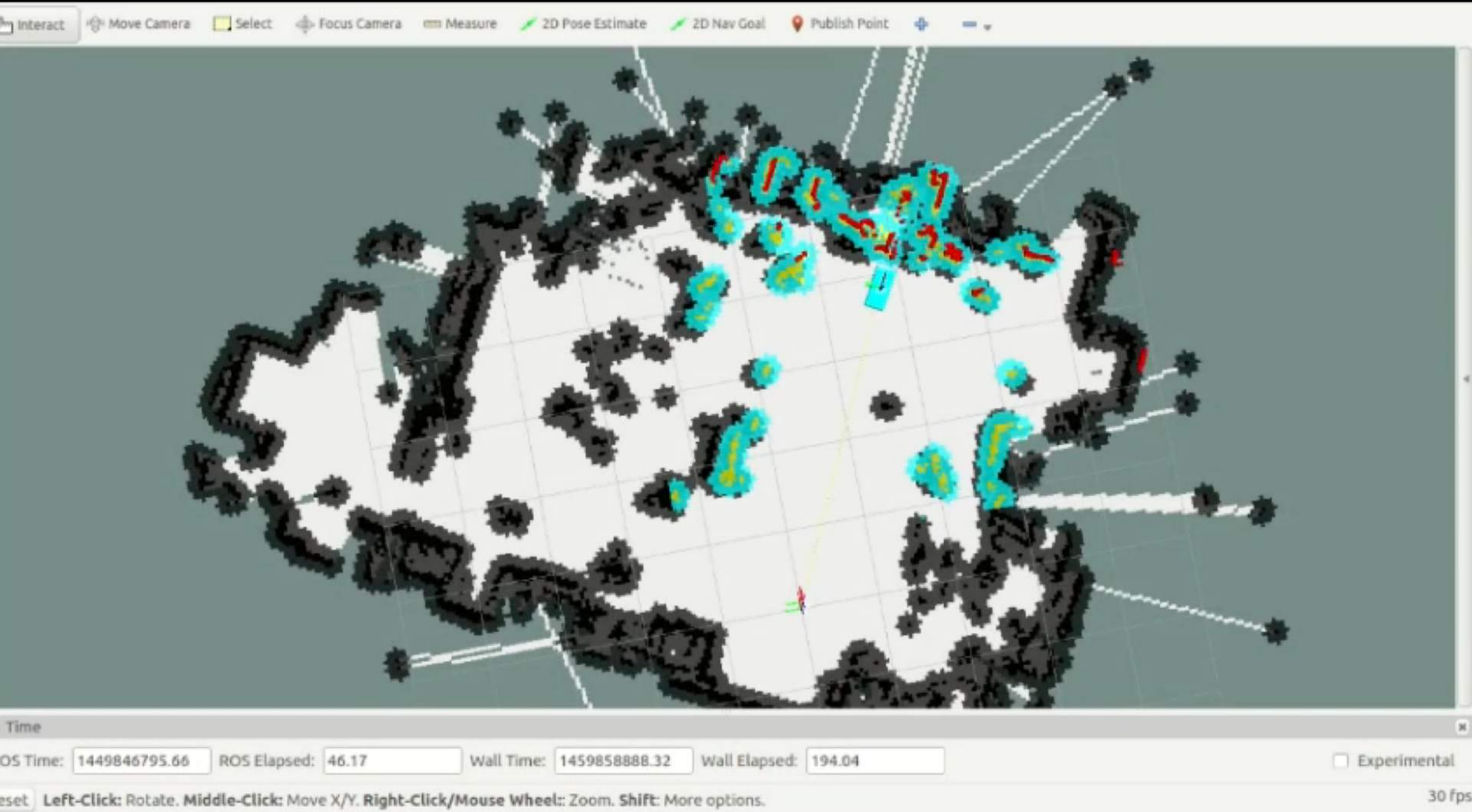
# Transformations and Frames

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



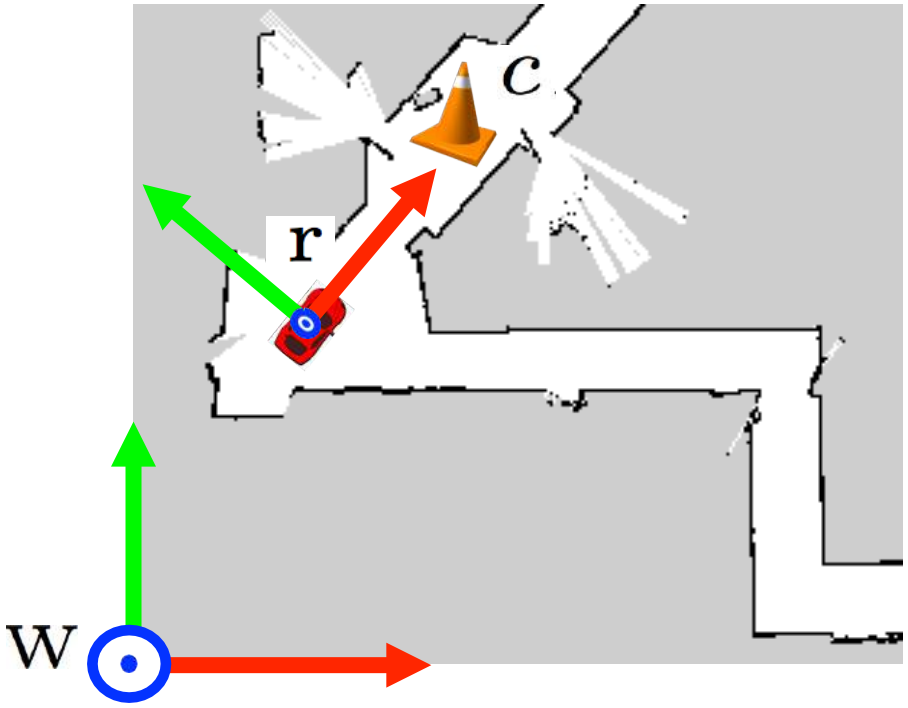




Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Time
ROS Time: 1449846801.68
ROS Elapsed: 78.84
Wall Time: 1459857837.91
Wall Elapsed: 184.27
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.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

# Rigid-body Transformations

- How to transform points from one coordinate frame to another?

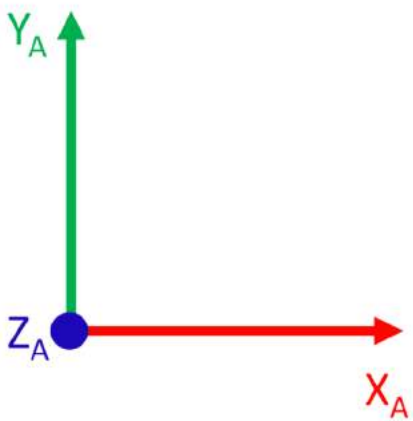



Given point “c” in coordinate frame “r” what is the position of the point in frame “w”?

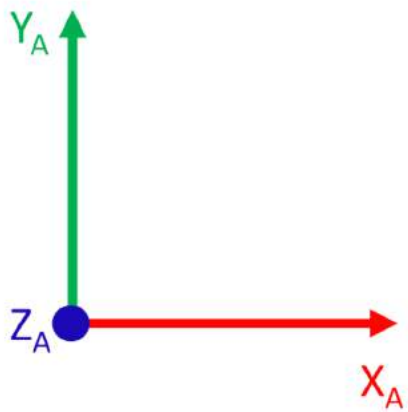
$$\boldsymbol{p}_{\mathrm{c}}^{\mathrm{w}} = \boldsymbol{R}_{\mathrm{r}}^{\mathrm{w}} \boldsymbol{p}_{\mathrm{c}}^{\mathrm{r}} + \boldsymbol{p}_{\mathrm{r}}^{\mathrm{w}}$$

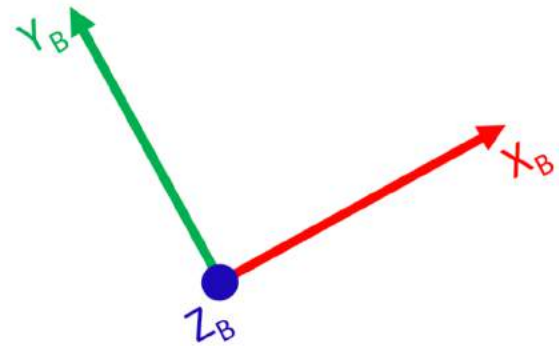
- **why rigid**? transformation does not change distance between points (does not deform shapes)
- translations, rotations, (reflections)

# Rigid Body Transforms

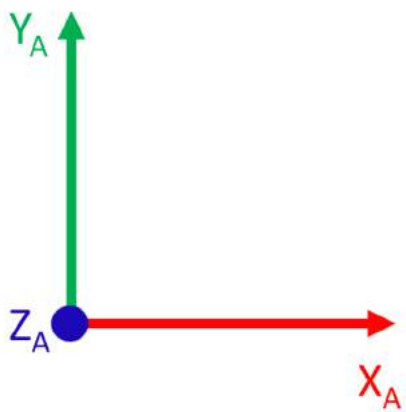


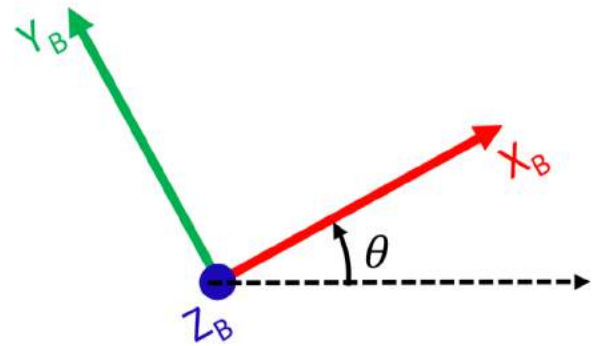
# Rigid Body Transforms



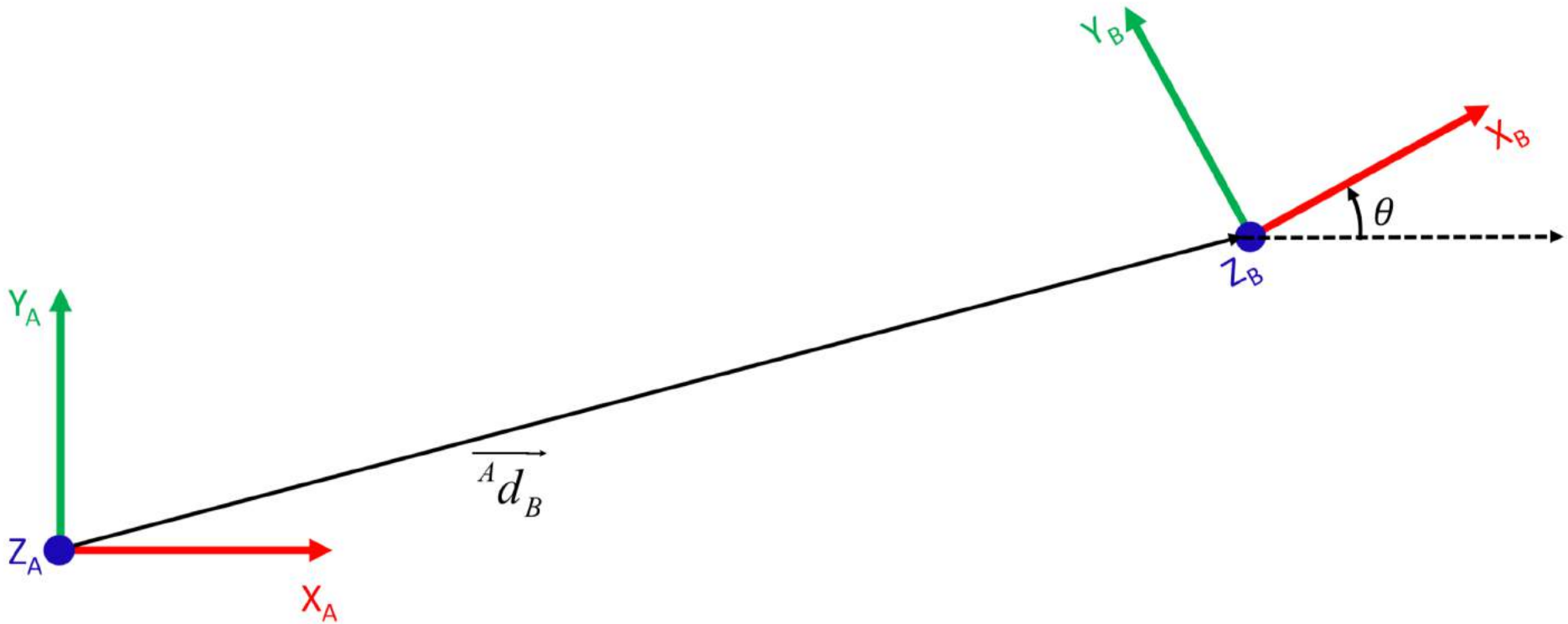



# Rigid Body Transforms

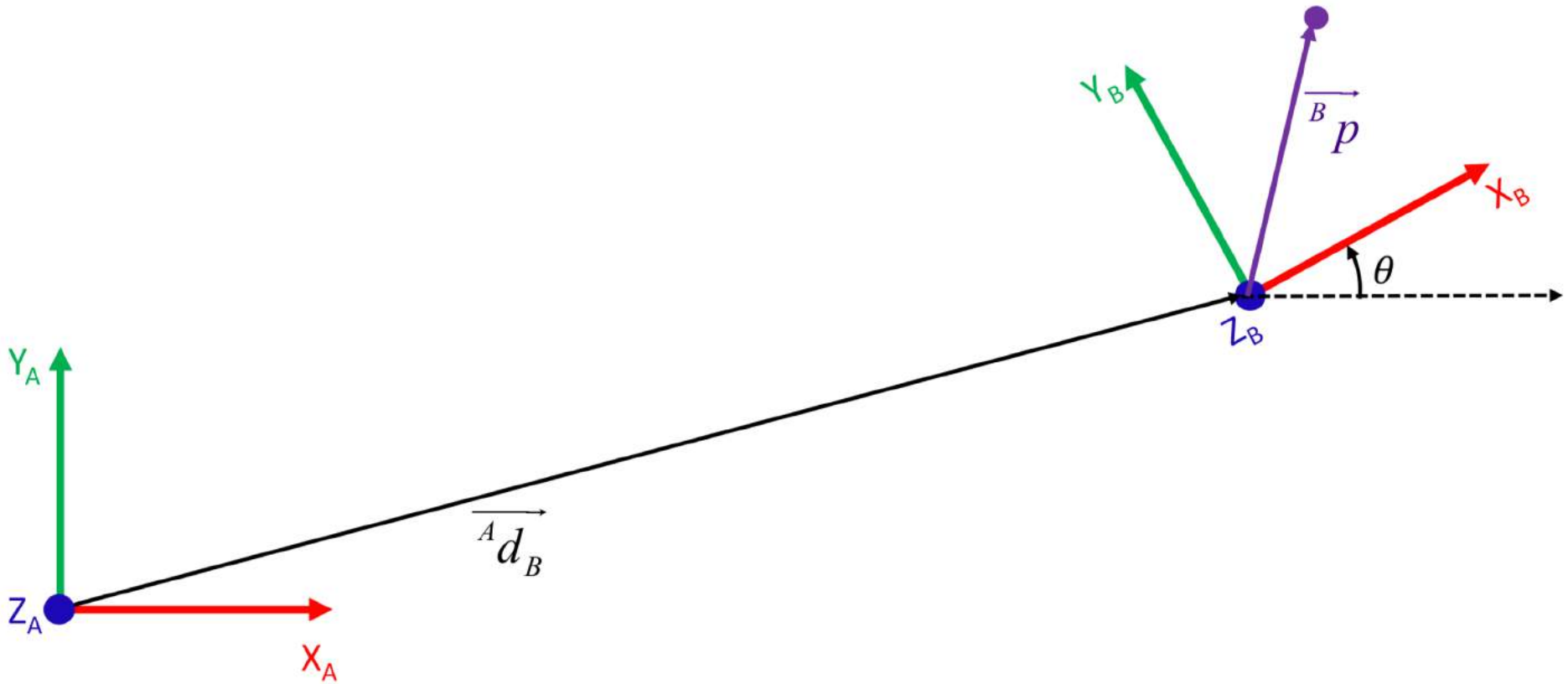


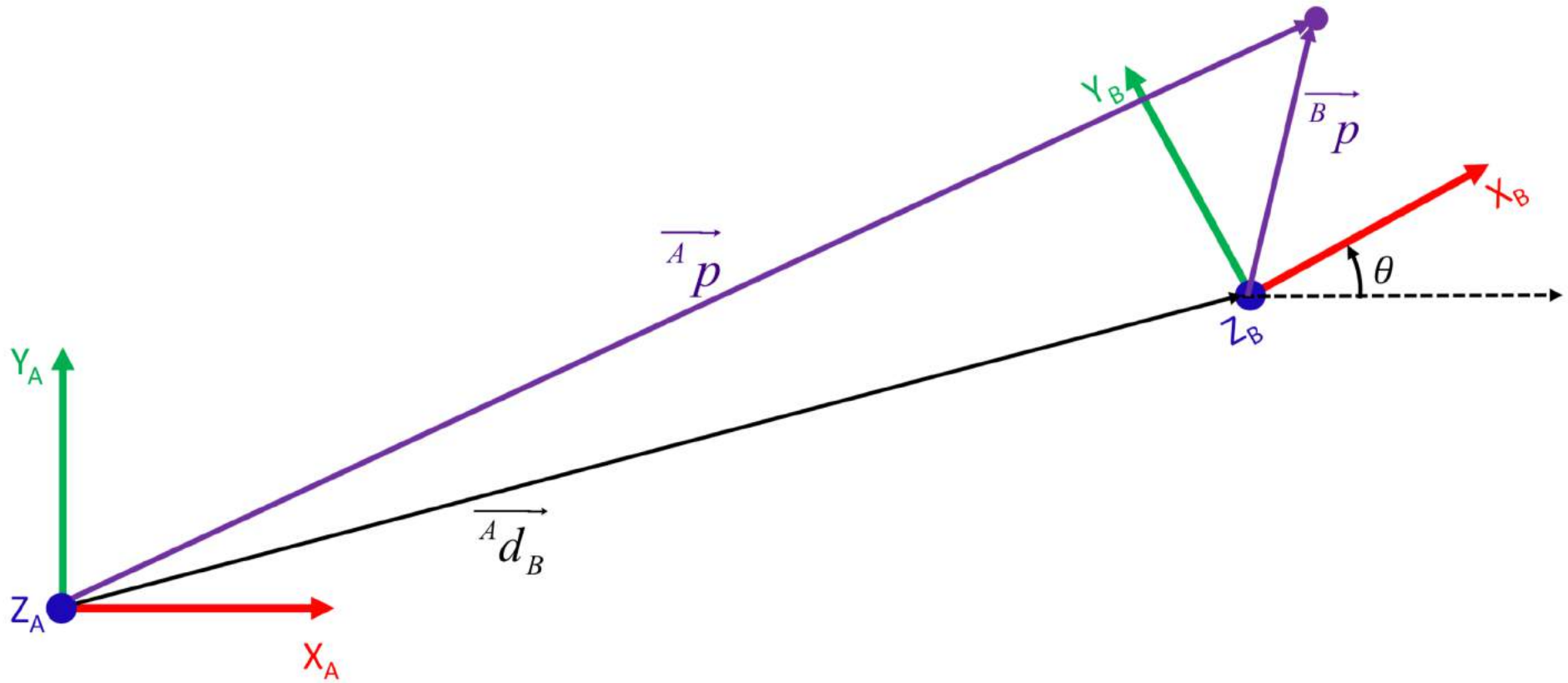
# Rigid Body Transforms



# Rigid Body Transforms

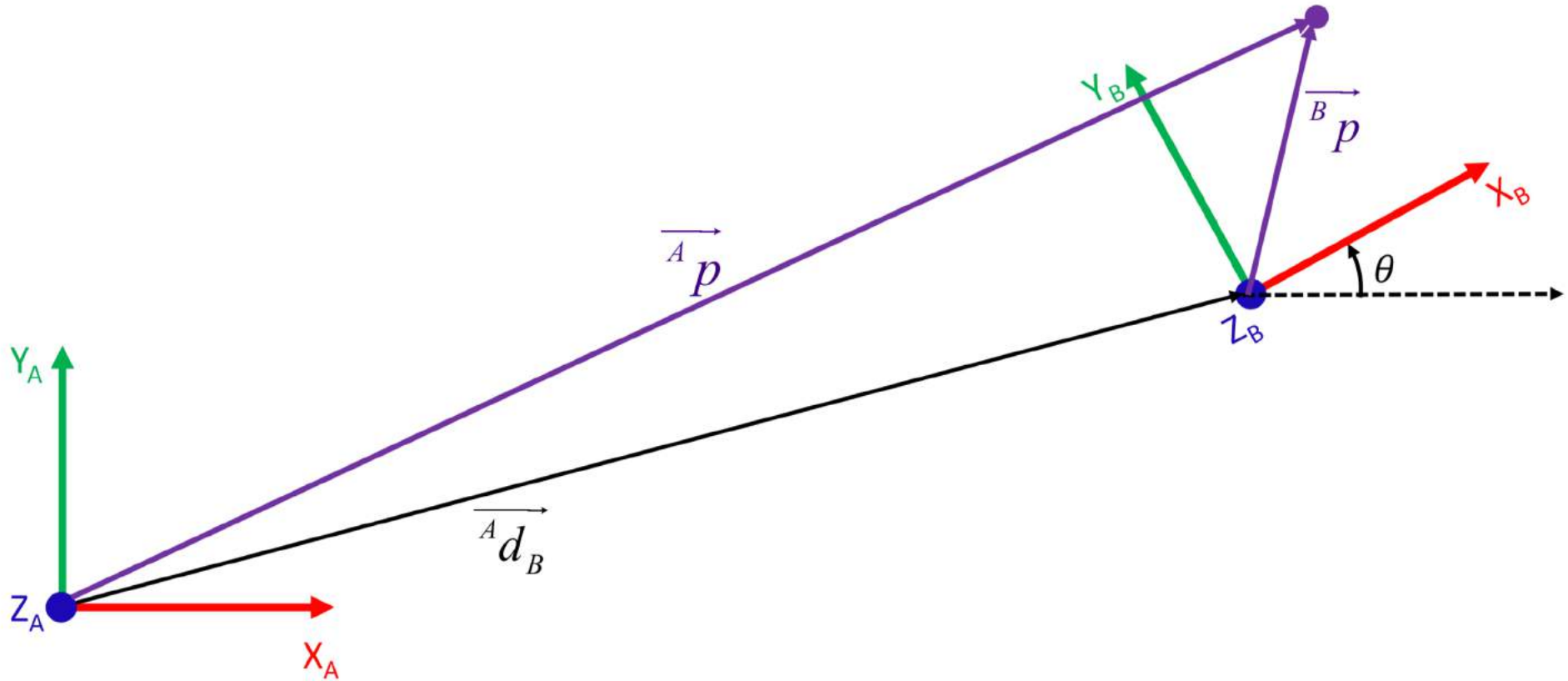


# Rigid Body Transforms



# Rigid Body Transforms

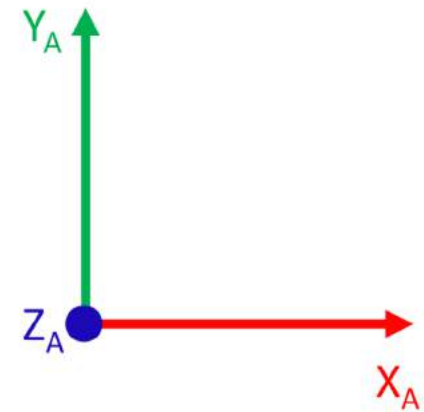
- What we need is Point p with respect to Frame A, given its pose in Frame B

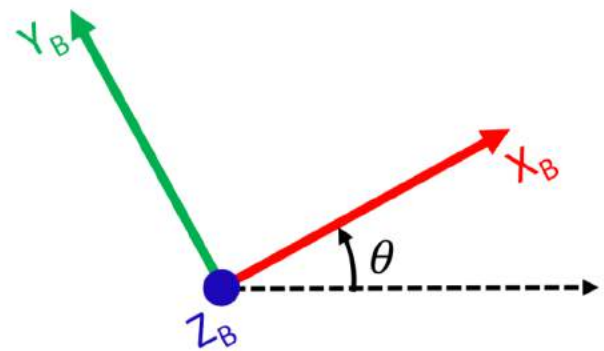


# Rigid Body Transforms

- Special type of matrices called Rotation matrices

$$\widehat{X_B} = R_{11}\widehat{X_A} + R_{21}\widehat{Y_A} + R_{31}\widehat{Z_A}$$
$$\widehat{Y_B} = R_{12}\widehat{X_A} + R_{22}\widehat{Y_A} + R_{32}\widehat{Z_A}$$
$$\widehat{Z_B} = R_{13}\widehat{X_A} + R_{23}\widehat{Y_A} + R_{33}\widehat{Z_A}$$



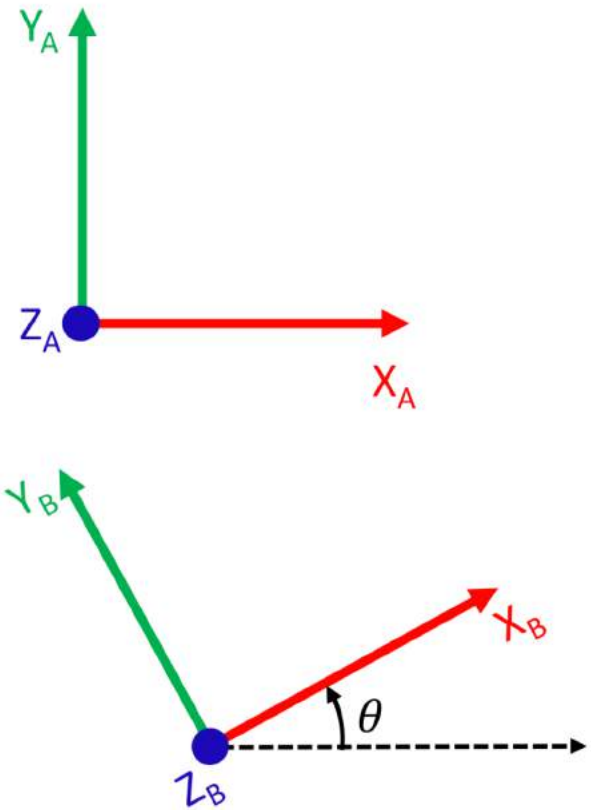
# Rigid Body Transforms

- Special type of matrices called Rotation matrices

$$\widehat{X_B} = R_{11}\widehat{X_A} + R_{21}\widehat{Y_A} + R_{31}\widehat{Z_A}$$
$$\widehat{Y_B} = R_{12}\widehat{X_A} + R_{22}\widehat{Y_A} + R_{32}\widehat{Z_A}$$
$$\widehat{Z_B} = R_{13}\widehat{X_A} + R_{23}\widehat{Y_A} + R_{33}\widehat{Z_A}$$

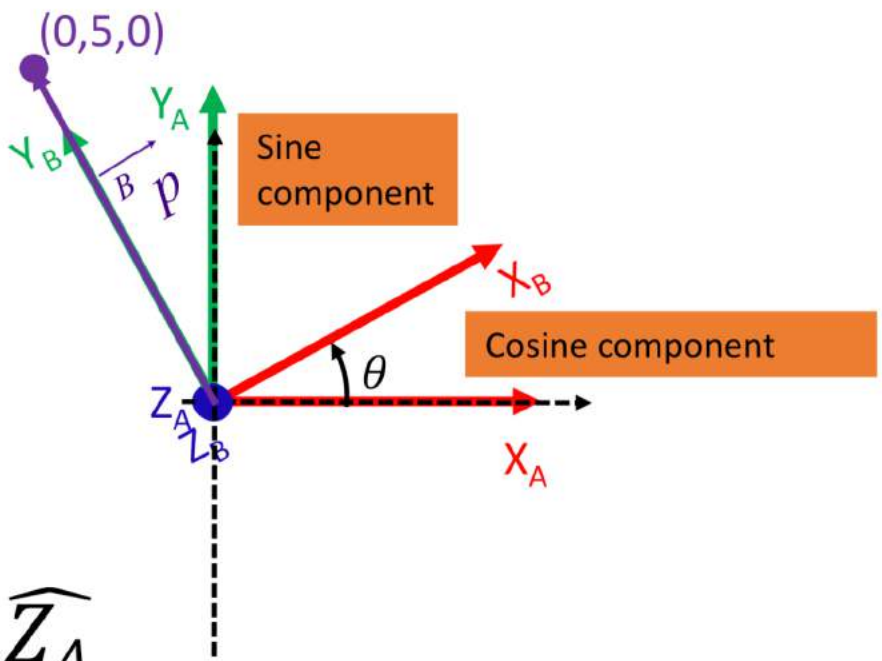
$${}^{A}R_{B} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

Takes points in frame B and represents their orientation in frame A



# Rigid Body Transforms: Rotation Matrices

$$\widehat{X_B} = R_{11}\widehat{X_A} + R_{21}\widehat{Y_A} + R_{31}\widehat{Z_A}$$
$$\widehat{Y_B} = R_{12}\widehat{X_A} + R_{22}\widehat{Y_A} + R_{32}\widehat{Z_A}$$
$$\widehat{Z_B} = R_{13}\widehat{X_A} + R_{23}\widehat{Y_A} + R_{33}\widehat{Z_A}$$




$$\widehat{X_B} = \underbrace{\cos(\theta)}_{R_{11}} \times \widehat{X_A} + \underbrace{\sin(\theta)}_{R_{21}} \times \widehat{Y_A} + \underbrace{0}_{R_{31}} \times \widehat{Z_A}$$

# Rigid Body Transforms: Rotation Matrices

$$
{}^{A}R_{B} = \begin{bmatrix} \mathrm{Cos}(\theta) & R_{12} & R_{13} \\ \mathrm{Sin}(\theta) & R_{22} & R_{23} \\ 0 & R_{32} & R_{33} \end{bmatrix}
\qquad
\begin{matrix} \widehat{X_A} \\ \widehat{Y_A} \\ \widehat{Z_A} \end{matrix}
$$

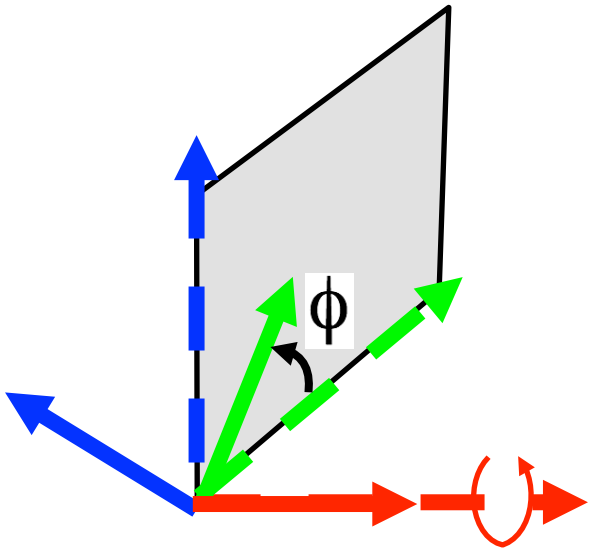
$\widehat{X_B}$ $\widehat{Y_B}$ $\widehat{Z_B}$

$$
{}^{A}R_{B} = \begin{bmatrix} C_\theta & -S_\theta & 0 \\ S_\theta & C_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

$$
C_\theta = \mathrm{Cos}(\theta)
$$
$$
S_\theta = \mathrm{Sin}(\theta)
$$

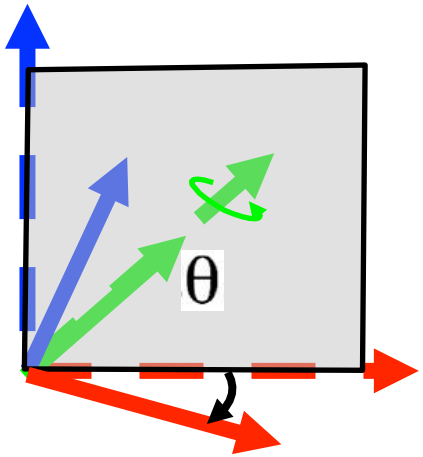
# Elementary Rotations

## Rotation around the **x** axis



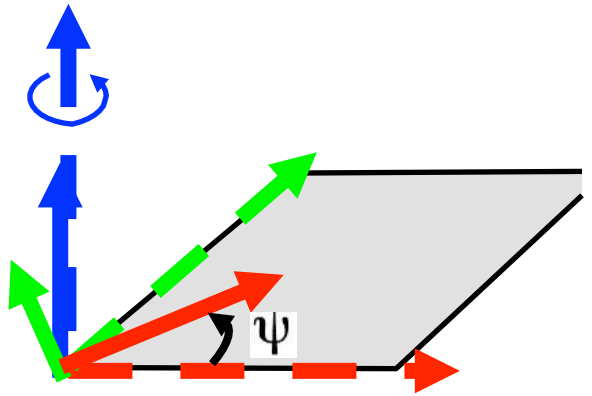

$$R_x(\varphi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\varphi & -\sin\varphi\\ 0 & \sin\varphi & \cos\varphi\end{bmatrix}$$

## Rotation around the **y** axis




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

## Rotation around the **z** axis




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

# Euler Angles

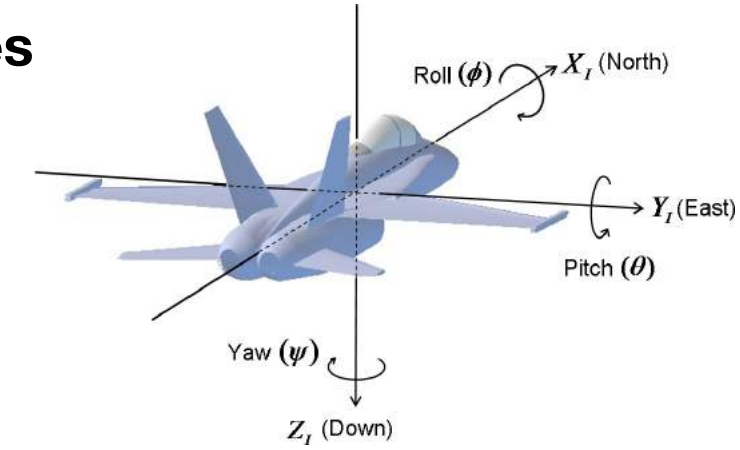
**Euler's rotation theorem (1.0)**: Any rotation can be written as the product of no more than 3 elementary rotations (no two consecutive rotations along the same axis).

$$\boldsymbol{R} = \boldsymbol{R}_z(\gamma)\boldsymbol{R}_y(\beta)\boldsymbol{R}_x(\alpha) \qquad \boldsymbol{R} = \boldsymbol{R}_y(\gamma)\boldsymbol{R}_z(\beta)\boldsymbol{R}_y(\alpha)$$

- why not x-y-z, y-x-z, x-x-z, …?
- $\alpha, \beta, \gamma$ are generally called **Euler angles**

$$\boldsymbol{R} = \boldsymbol{R}_z(\gamma)\boldsymbol{R}_y(\beta)\boldsymbol{R}_x(\alpha)$$

yaw pitch roll

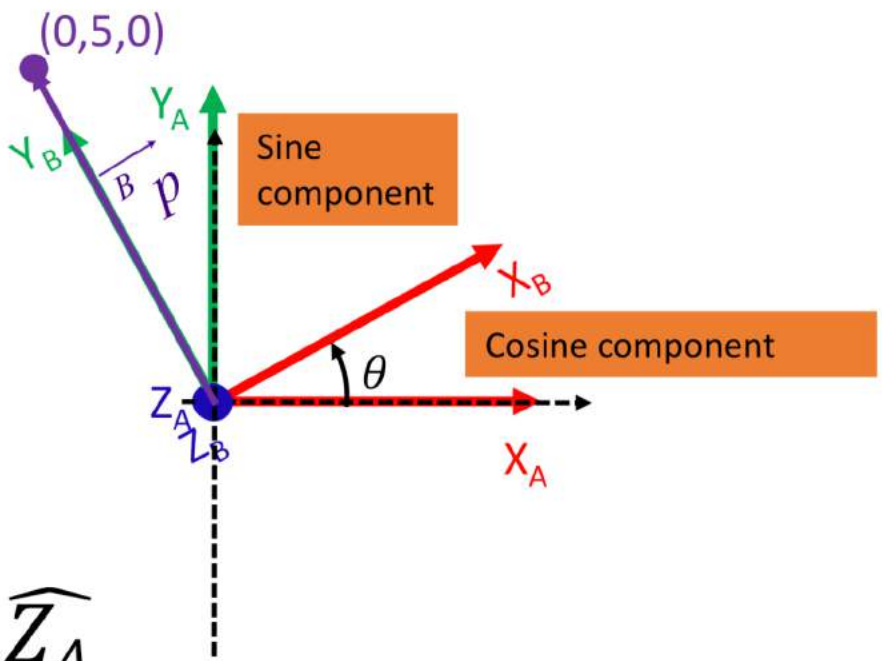



- We only need 3 numbers to represent a rotation!

http://petercorke.com/wordpress/toolboxes/robotics-toolbox

# Rigid Body Transforms: Rotation Matrices

$$\widehat{X_B} = R_{11}\widehat{X_A} + R_{21}\widehat{Y_A} + R_{31}\widehat{Z_A}$$
$$\widehat{Y_B} = R_{12}\widehat{X_A} + R_{22}\widehat{Y_A} + R_{32}\widehat{Z_A}$$
$$\widehat{Z_B} = R_{13}\widehat{X_A} + R_{23}\widehat{Y_A} + R_{33}\widehat{Z_A}$$




$$\widehat{X_B} = \underbrace{\cos(\theta)}_{R_{11}} \times \widehat{X_A} + \underbrace{\sin(\theta)}_{R_{21}} \times \widehat{Y_A} + \underbrace{0}_{R_{31}} \times \widehat{Z_A}$$

# We have the Rotation Matrix, so now what?

$$R = \begin{bmatrix} 0.86 & -0.5 & 0 \\ 0.5 & 0.86 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

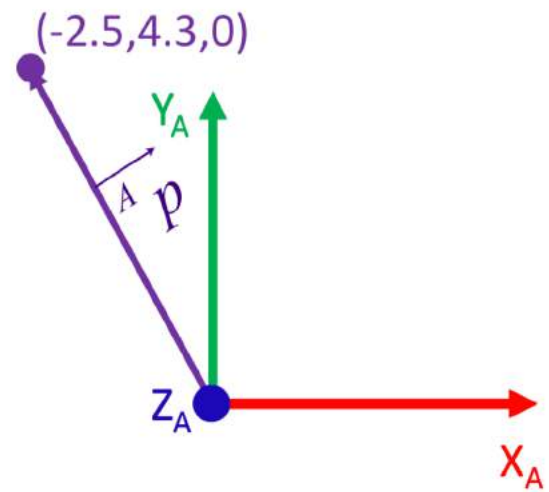
$$\overrightarrow{{}^{A}p} = {}^{A}R_{B} \times \overrightarrow{{}^{B}p}$$

Known Known

We now have the point P as referenced in frame A

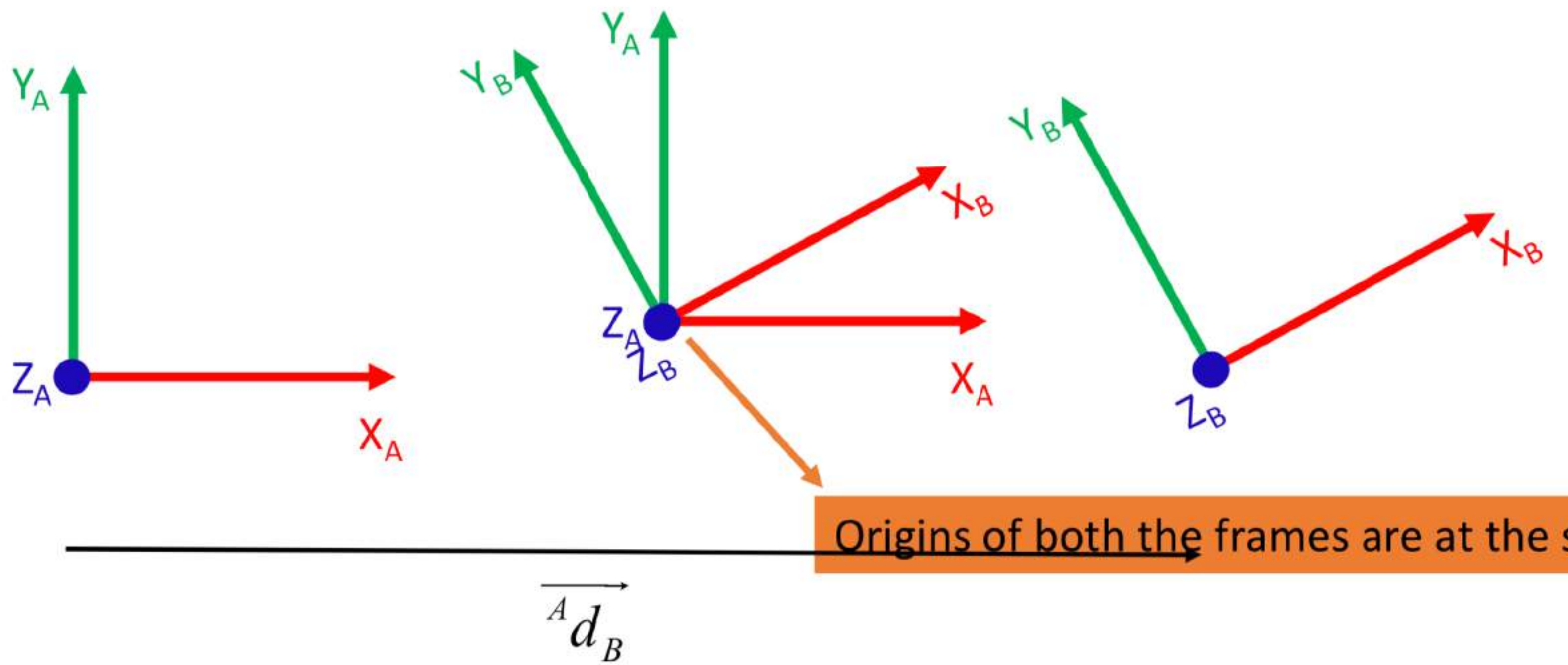
For example

$$\theta = \pi/6 \quad \Rightarrow \quad \overrightarrow{{}^{A}p} = (-2.5, 4.3, 0)$$

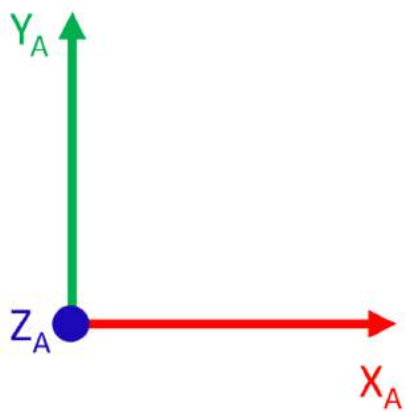


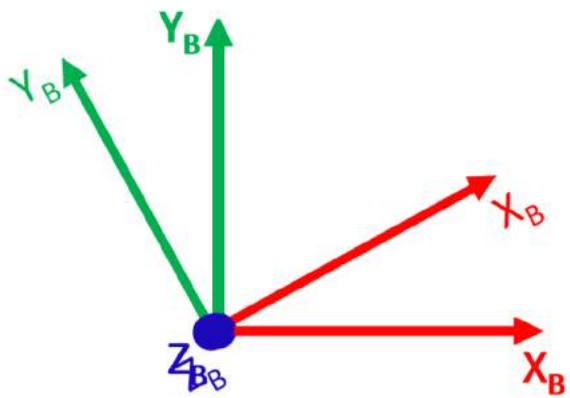
# Important point to remember

- The rotation matrix will take care of perspectives of orientation, what about displacement?



$$\overrightarrow{{}^{A}p} = {}^{A}R_{B} \times {}^{B}p + {}^{A}d_{B}$$

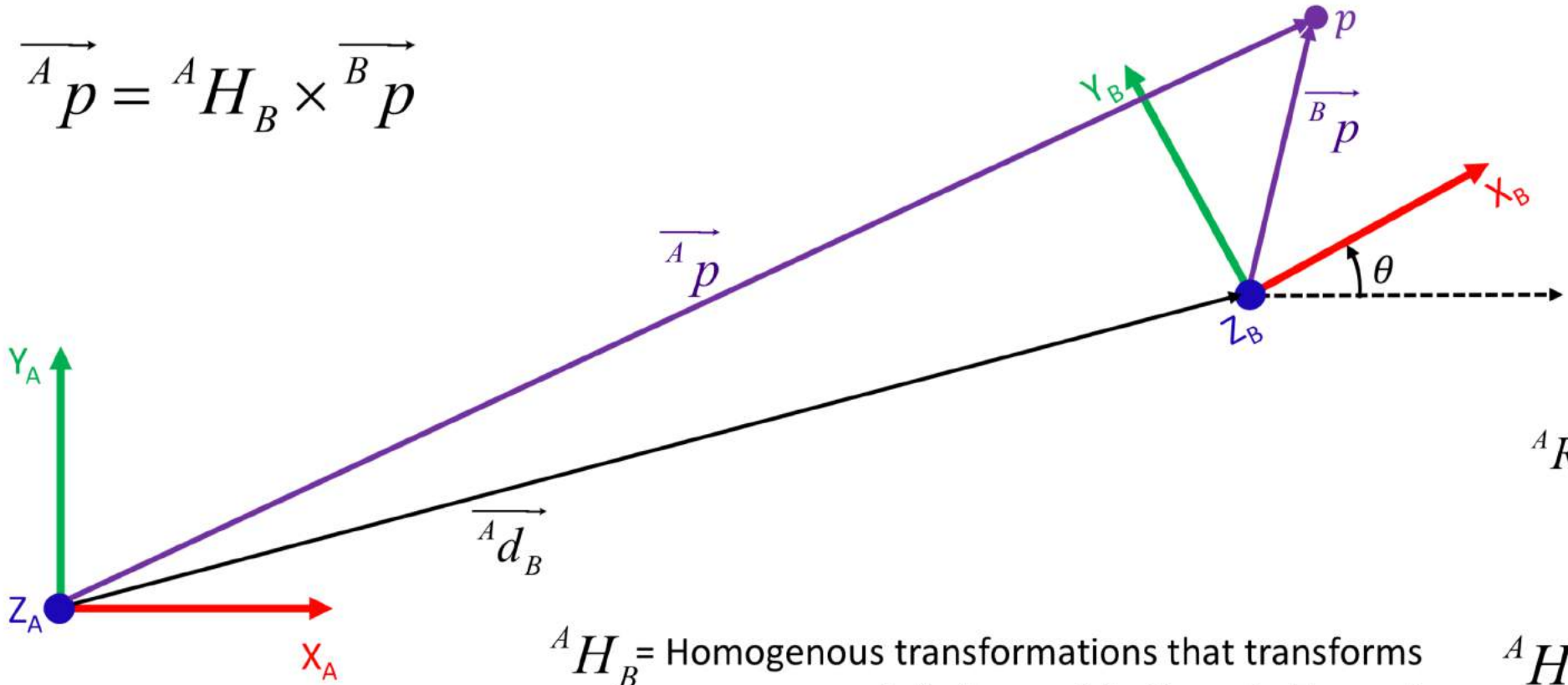


# Rigid Body Transforms

- What we need is Point p with respect to Frame A, given its pose in Frame B

$$\overrightarrow{{}^{A}p} = {}^{A}H_{B} \times \overrightarrow{{}^{B}p}$$

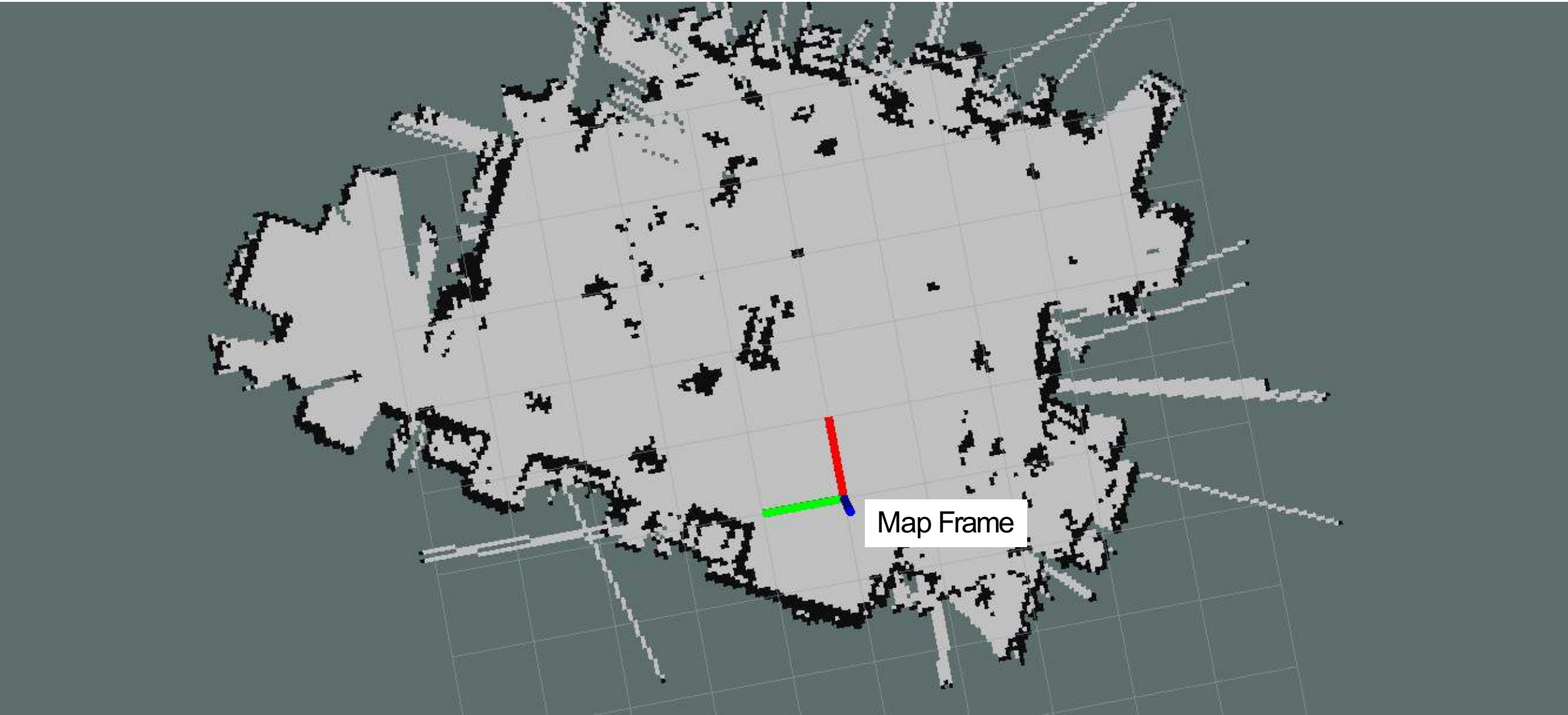



$$ {}^{A}R_{B} = \begin{bmatrix} C_\theta & -S_\theta & 0 \\ S_\theta & C_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

${}^{A}H_{B}$ = Homogenous transformations that transforms measurements in Frame B to those in Frame A

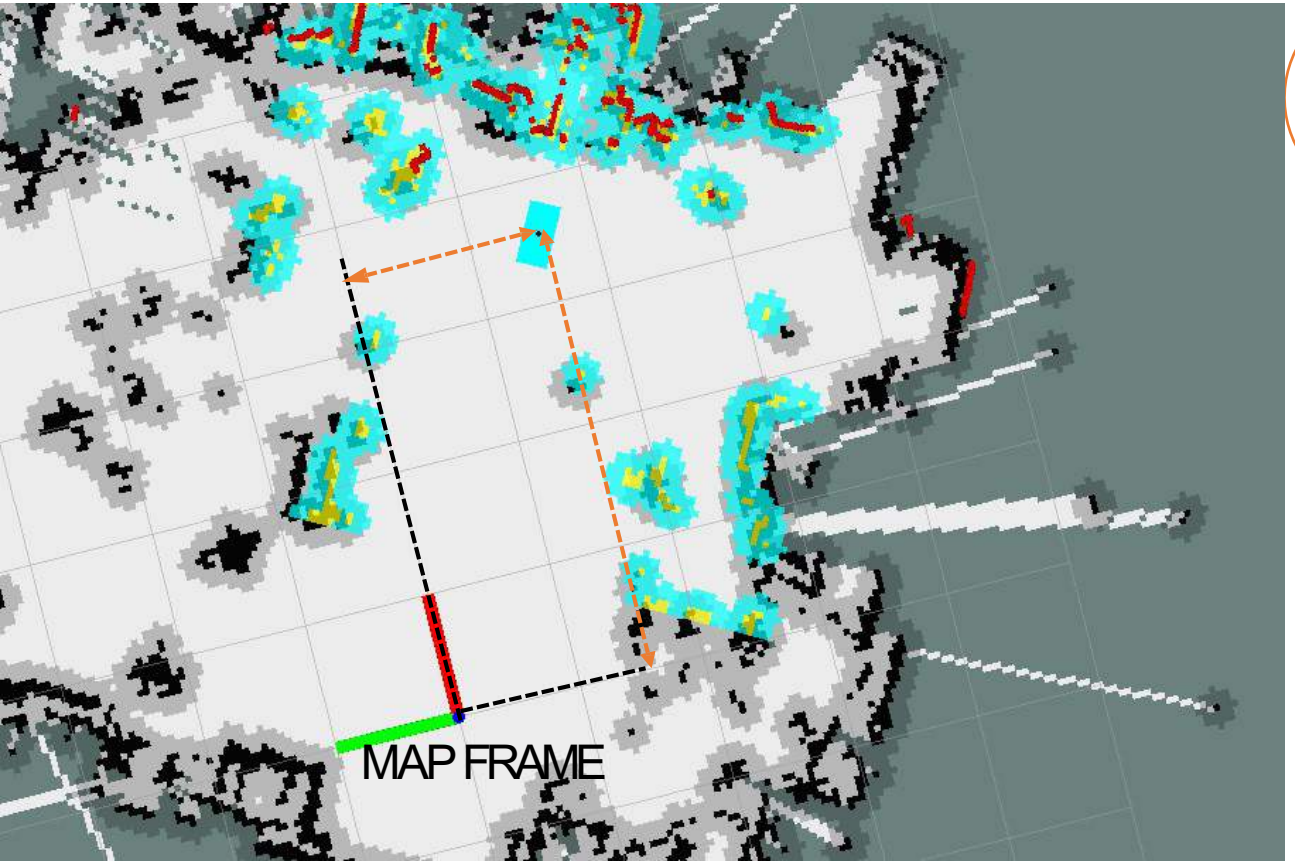
$$ {}^{A}H_{B} = \begin{bmatrix} {}^{A}R_{B} & {}^{A}d_{B} \\ 0 & 1 \end{bmatrix} $$

# Map frame



# Map frame: Importance

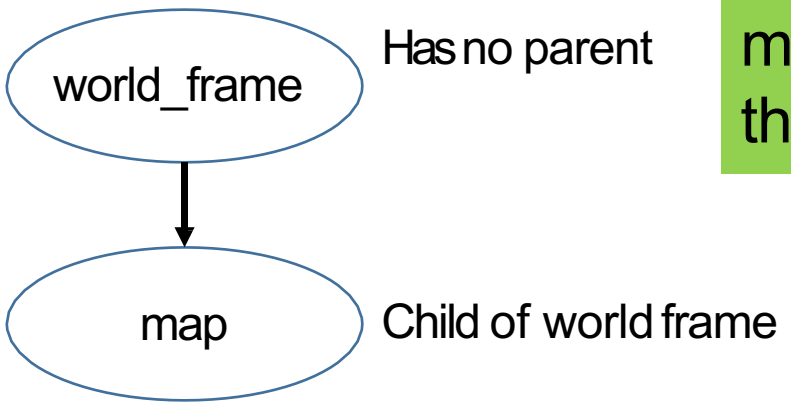
- Position with respect to map




```
paritoshkelkar@paritoshkelkar-system: ~
paritoshkelkar@paritoshkelkar-system:~/xlab_ws/src/xlab_race/bags$ rostopic echo /amcl
/amcl/parameter_descriptions  /amcl/parameter_updates       /amcl_pose
paritoshkelkar@paritoshkelkar-system:~/xlab_ws/src/xlab_race/bags$ rostopic echo /amcl_pose
header:
  seq: 19
  stamp:
    secs: 1449846796
    nsecs: 79412250
  frame_id: map
pose:
  pose:
    position:
      x: 3.64072439486
      y: -1.60239534849
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.236966624977
      w: 0.971517791215
  covariance: [0.014441140554119158, -0.005220206392102078, 0.0, 0.0, 0.0, 0.0, -0.005220206392102078, 0.01649975537
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0024897152112678023]
---
^Cparitoshkelkar@paritoshkelkar-system:~/xlab_ws/src/xlab_race/bags$
[1] 0:bags- 1:traxxas  2:move_base  3:posInMap*
```

# Map Frame: ROS

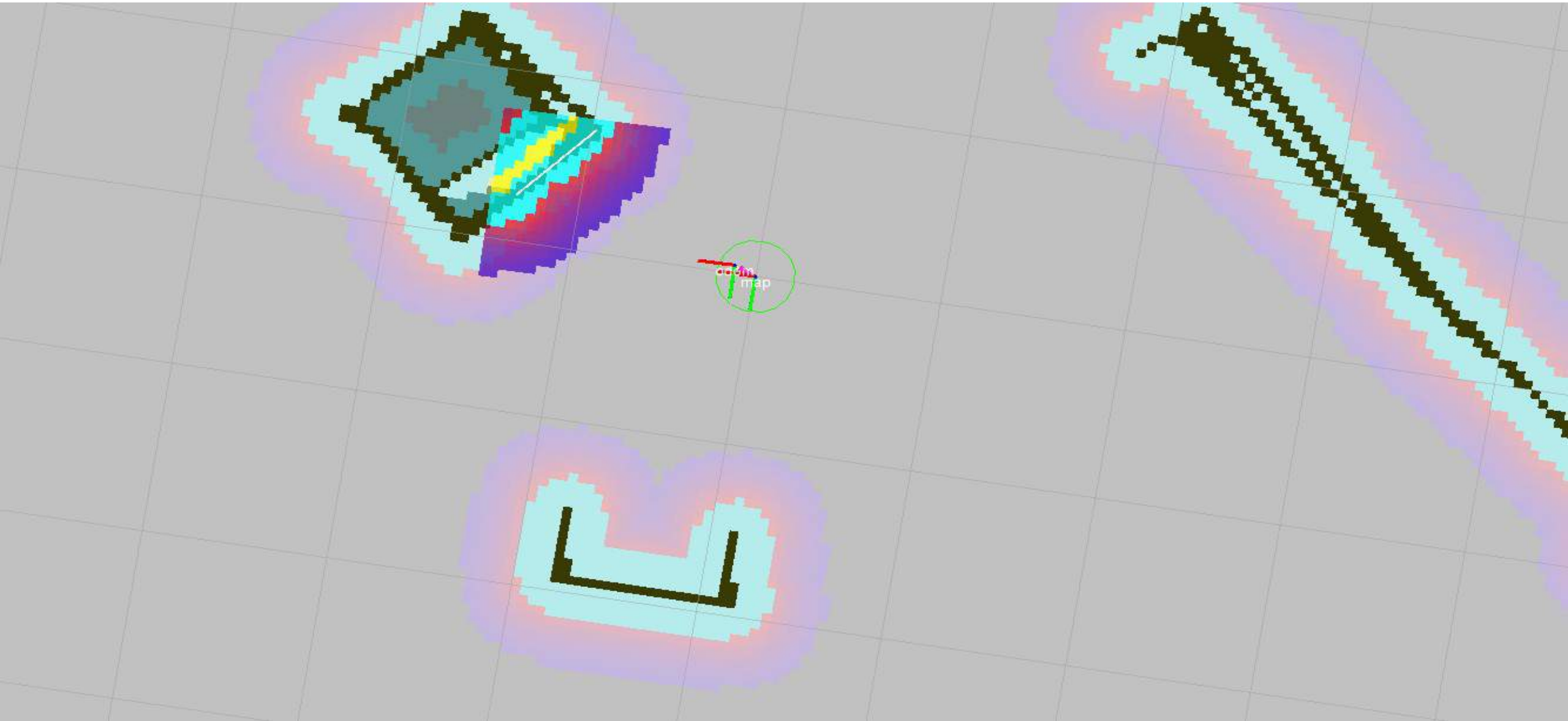
- The tf package – tracks multiple 3D coordinate frames - maintains a tree structure b/w frames – access relationship b/w any 2 frames at any point of time
- ROSREP (ROS Enhancement Proposals) 105 describes the various frames involved
- Normal hierarchy




A tf tree is a structure that maintains relations between the linked frames.

Note:
Tf = transformer class

# Odom Frame

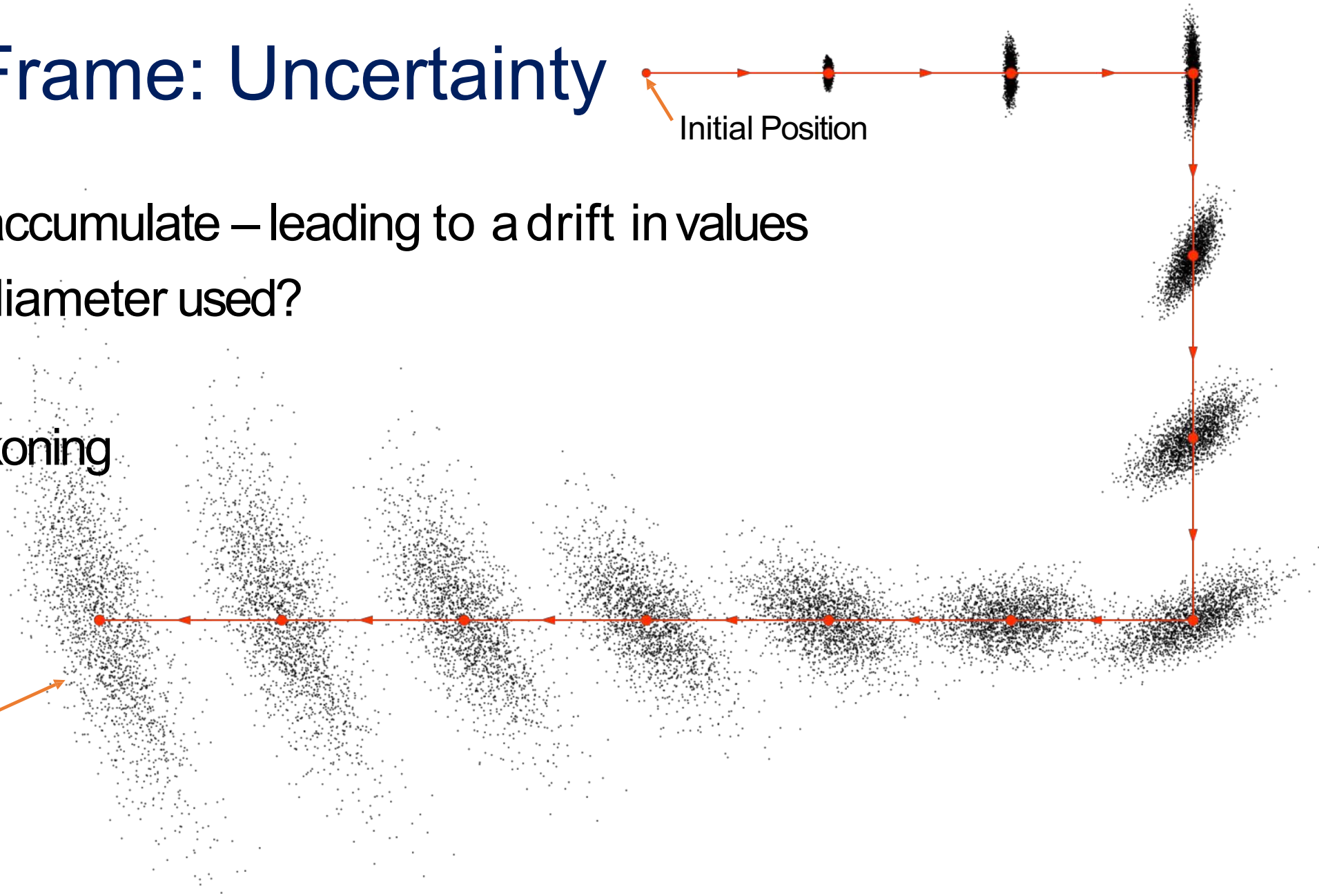


# Odom Frame: Calculation

- Difference in count of ticks of wheels – orientation
- Integrating the commanded velocities/accelarations
- Integrating values from IMU
- Scan Matching

# Odom Frame: Uncertainty

- Error can accumulate – leading to a drift in values
- Incorrect diameter used?
- Slippage?
- Dead Reckoning

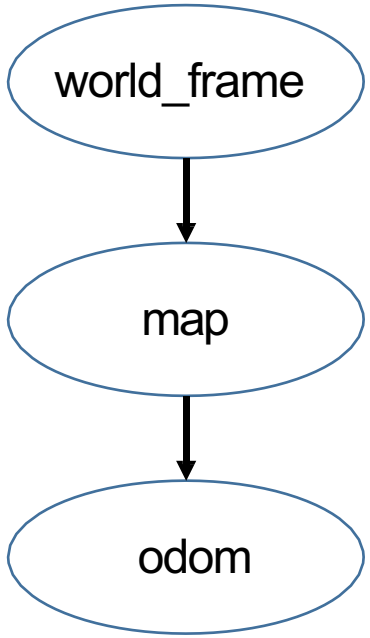


# Odom Frame: Properties

- Continuous – actual data from actuators/motors
- Evolves in a smooth manner, without discrete jumps
- Short term ; accurate local reference

# Odom Frame: ROS

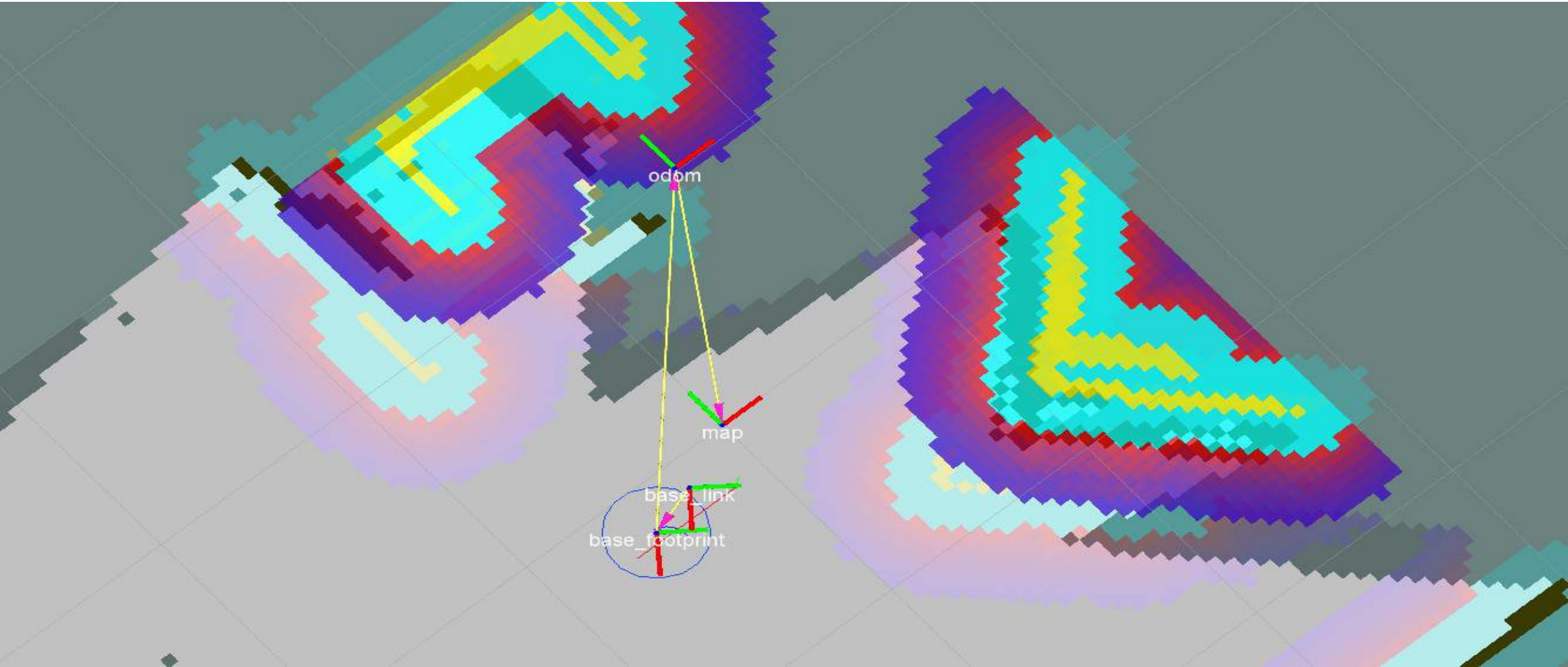
- General ROS frame hierarchy




Tf tree

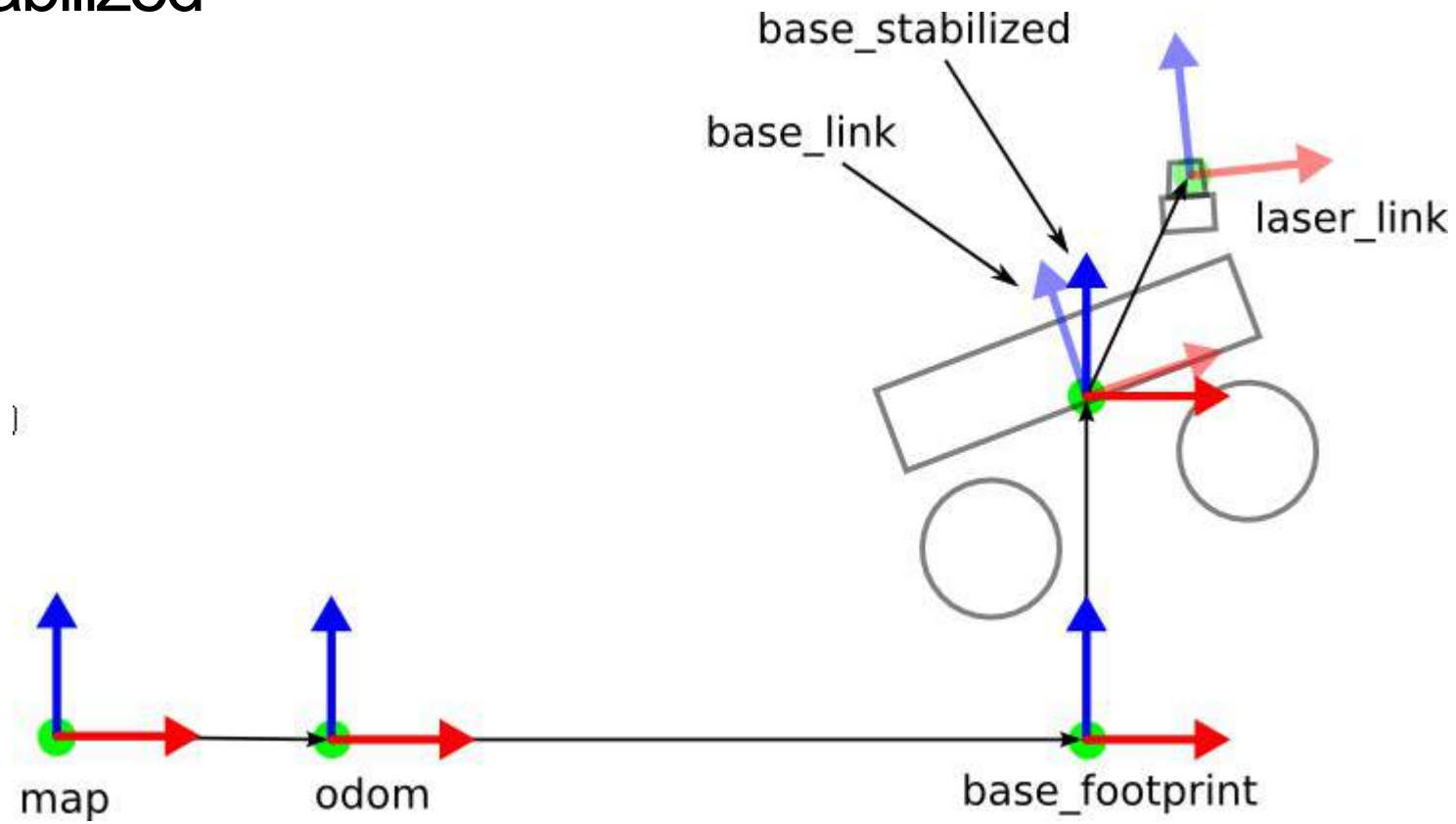
Note that if the frame is connected in the tf tree, we can obtain a representation of that frame with any other frame in the tree

# Base_link and fixed frames attached to the robot



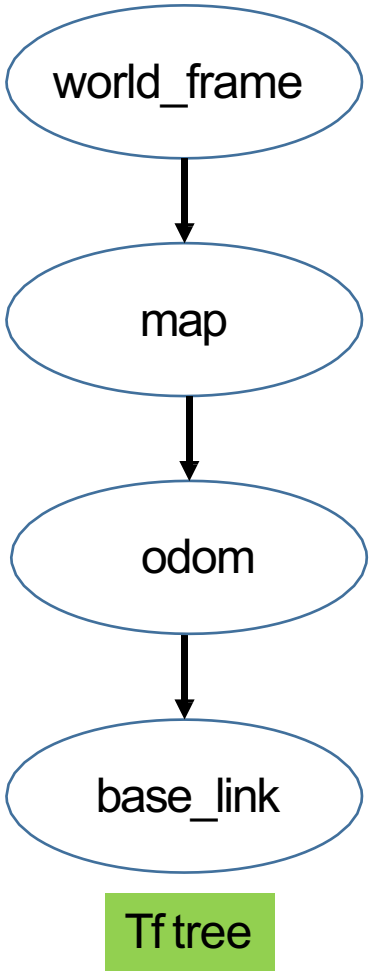
# Base link: What is it

- Attached to the robot itself – base_footprint; base_link; base_stabilized



# Base_link Frame: ROS

- General ROS frame hierarchy



# Base link: Properties

- Odom -> base link transform provided by Odometry source
- Map -> base_link transform provided by localization component

# ROS.W.T.F

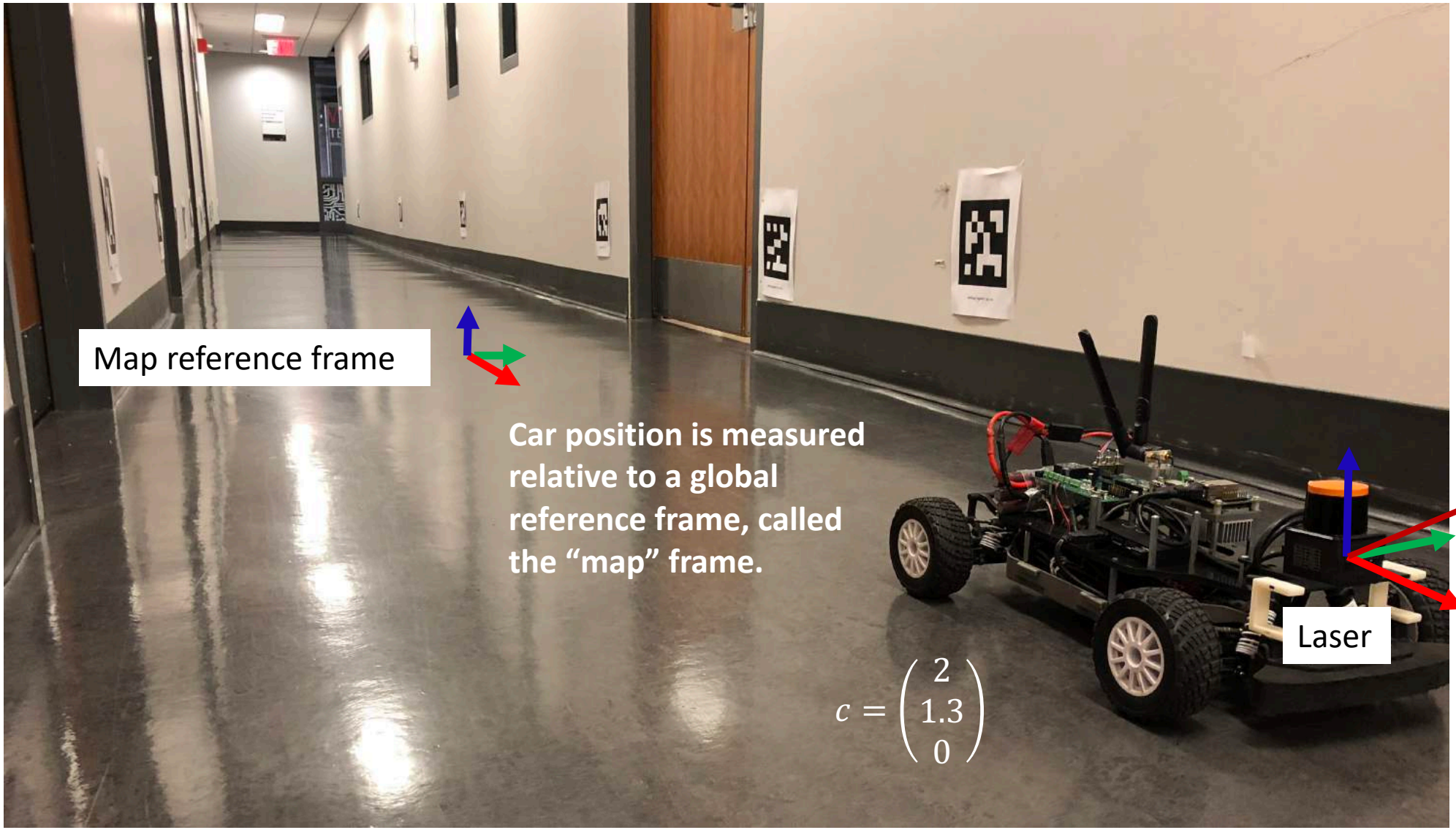
- Host of tf debugging tools provided by ROS
- ROS tf tutorial for further details

```
$ rosrun tf view_frames
```

```
$ rosrun tf view_monitor
```

```
$ roswtf
```

# Why do we have multiple frames?

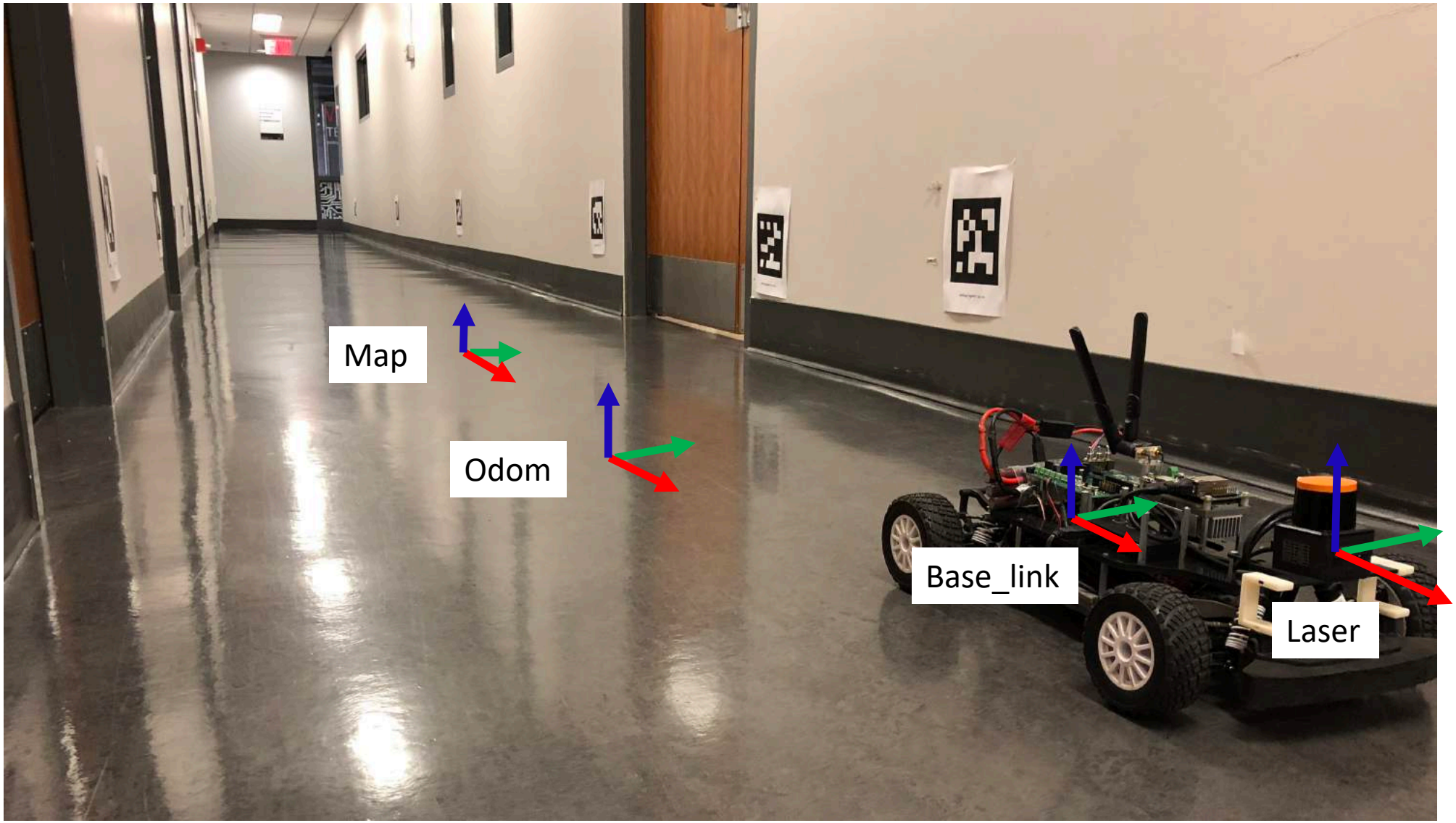


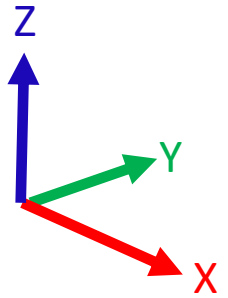

Lidar measurements are relative to the LiDAR itself, i.e., in the laser reference frame. (After all, LiDAR doesn’t know where the car is)

$o = \begin{pmatrix} 2 \\ 3 \\ 2 \end{pmatrix}$

What other frames might be useful?

# Other frames we will use



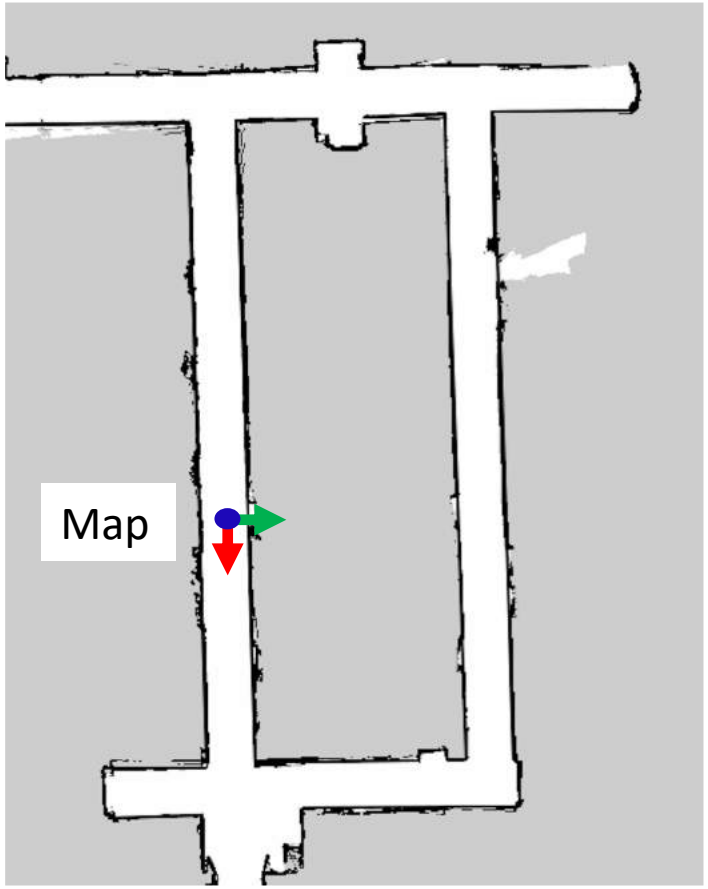



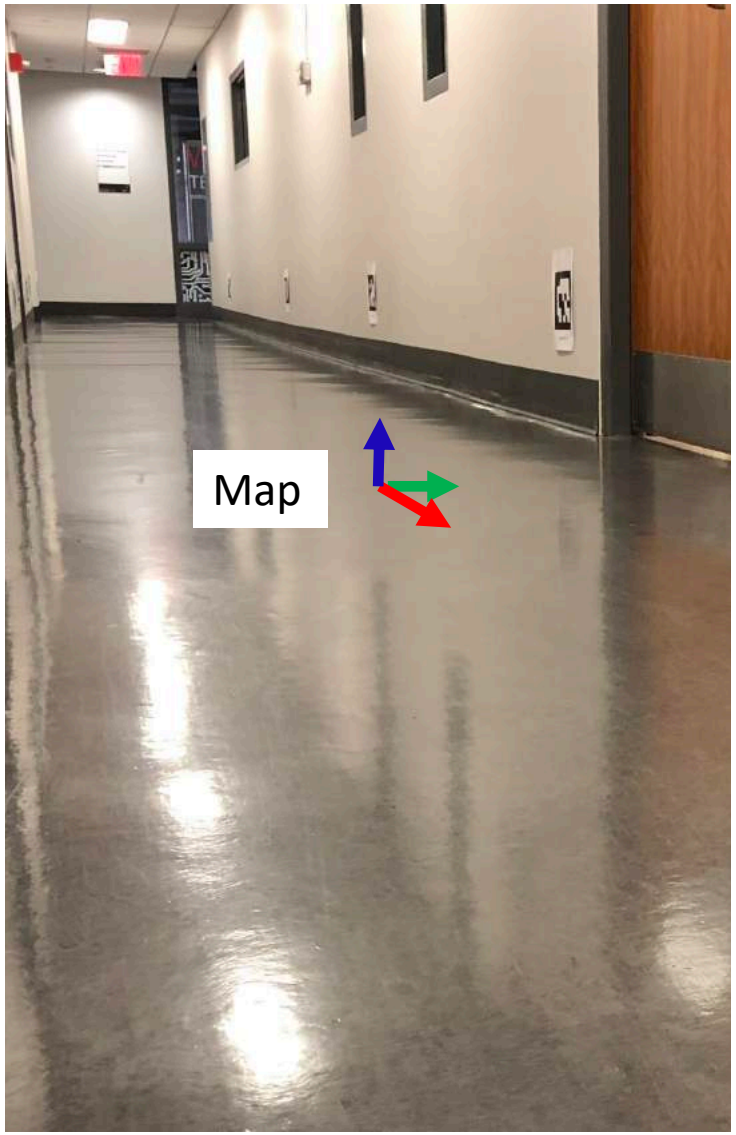

Memory trick:
RGB -> XYZ

* ROS convention is having positive x direction in front of the link and positive y direction to left of the link. [REP 103]

# Map Frame

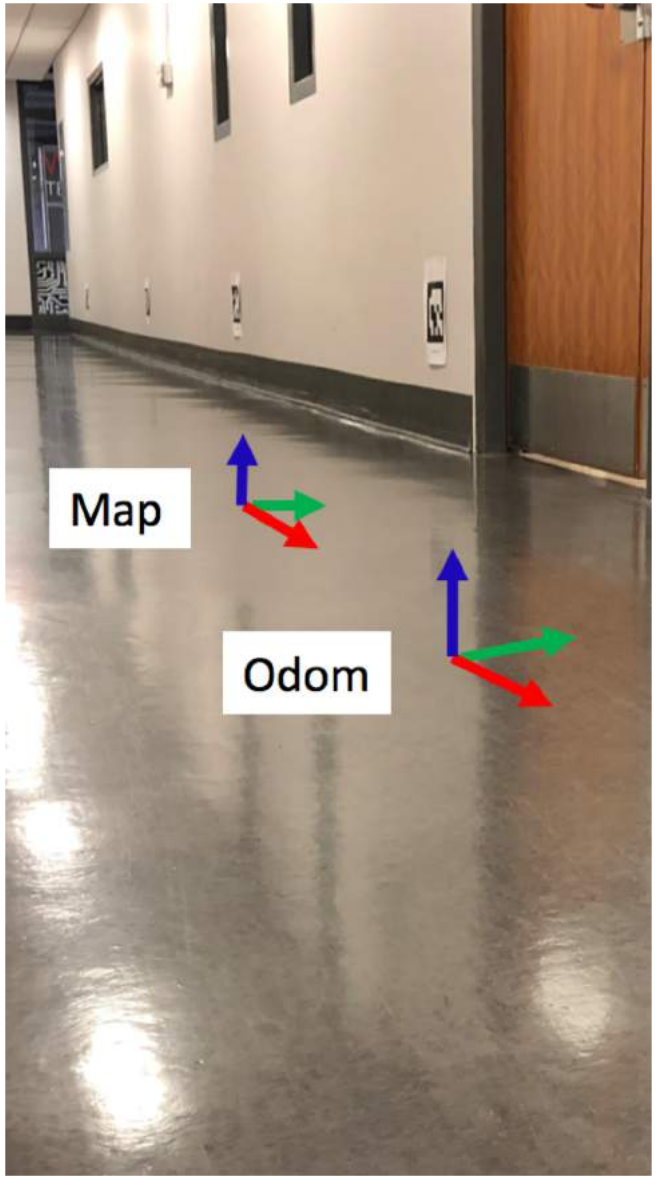
- Map frame origin is set arbitrarily, meaning we can choose it.



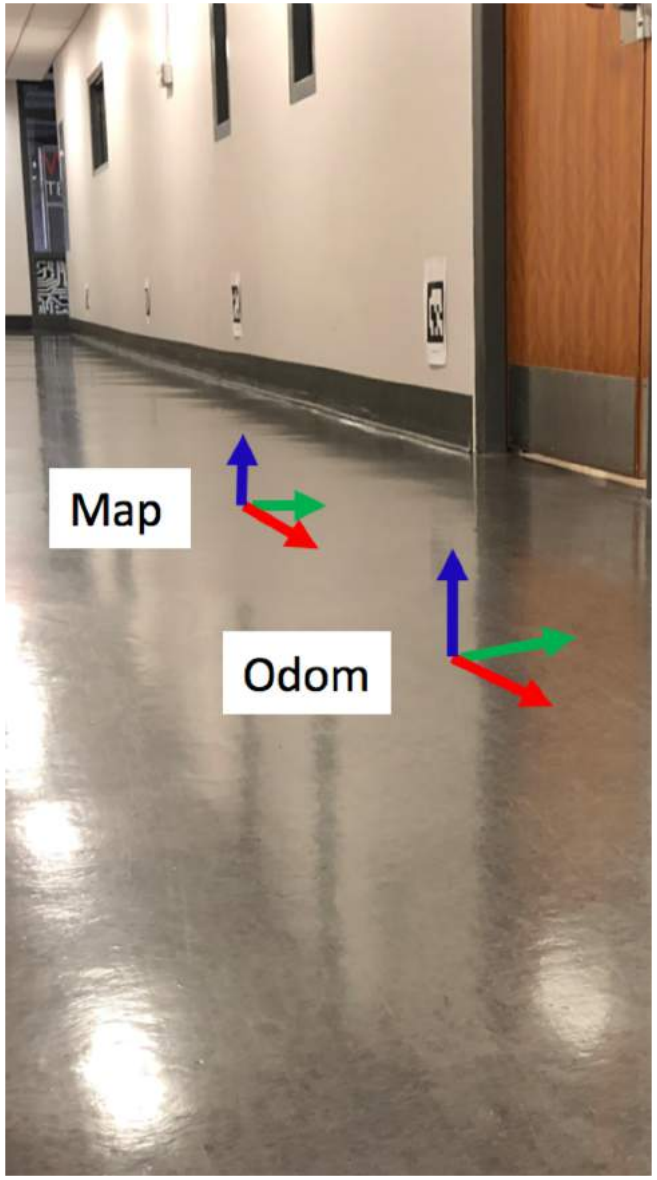
# Odom Frame

- Odom frame originates where the car starts at. In this case the car started a meter in front of the map origin. Hence map -> odom transform is 1m in x direction.
- Odom frame stays fixed – it does not move with the robot.



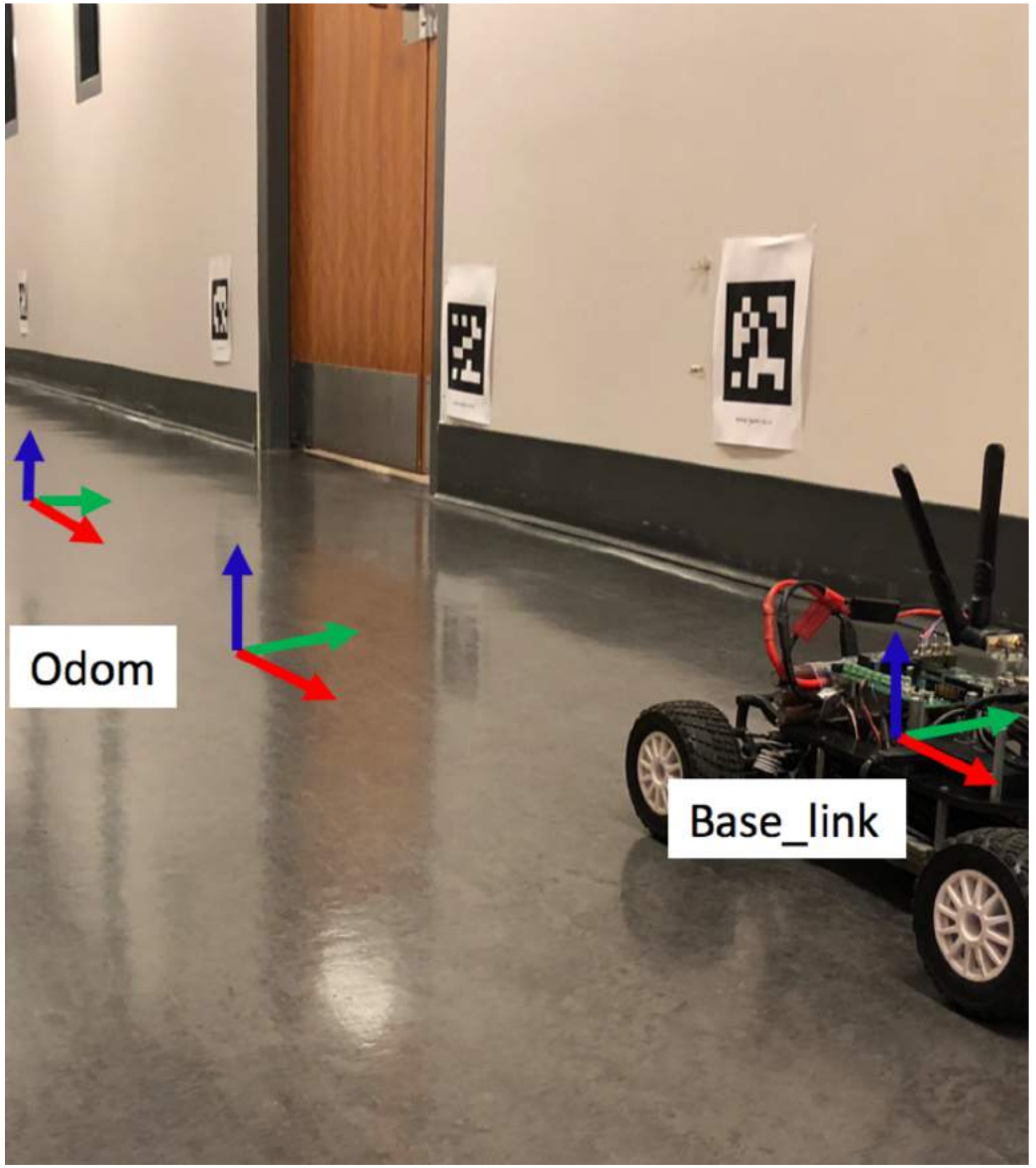
# Burdened by history

- *Historically*, localization is done relative to map frame.
- Terminology: Localization = fusing multiple sources of location information, including odometry, into one best estimate
- Pose of a robot in map frame can have jumps, meaning discontinuous, because the best estimate (e.g., provided by an Extended Kalman Filter) can jump

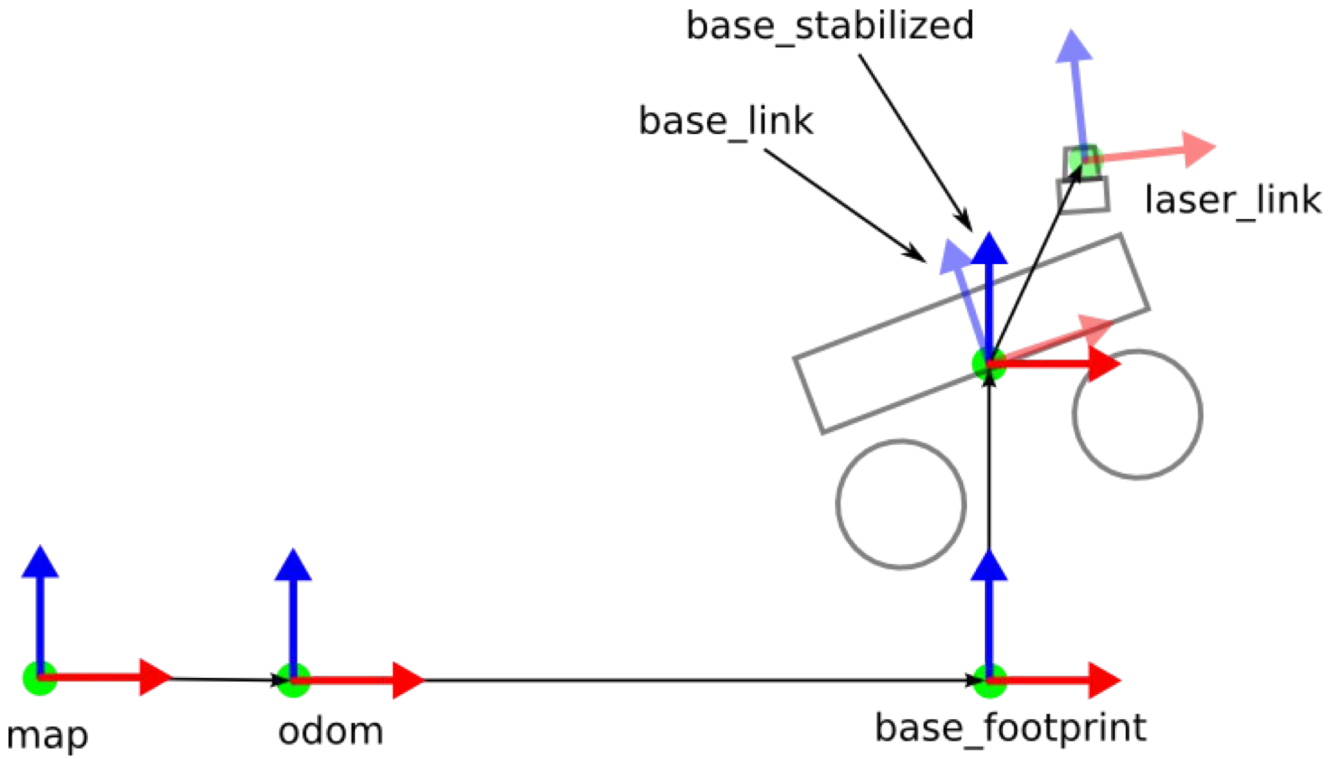


# Base_link Frame

- Defined as the center of the car's rear axle
- Base_link frame moves with the car (it is not fixed)
- You really control where to place the base_link on the car. Just be consistent, and be aware that algorithms you re-use have their assumptions (usually, center of car's rear axle).



# 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).

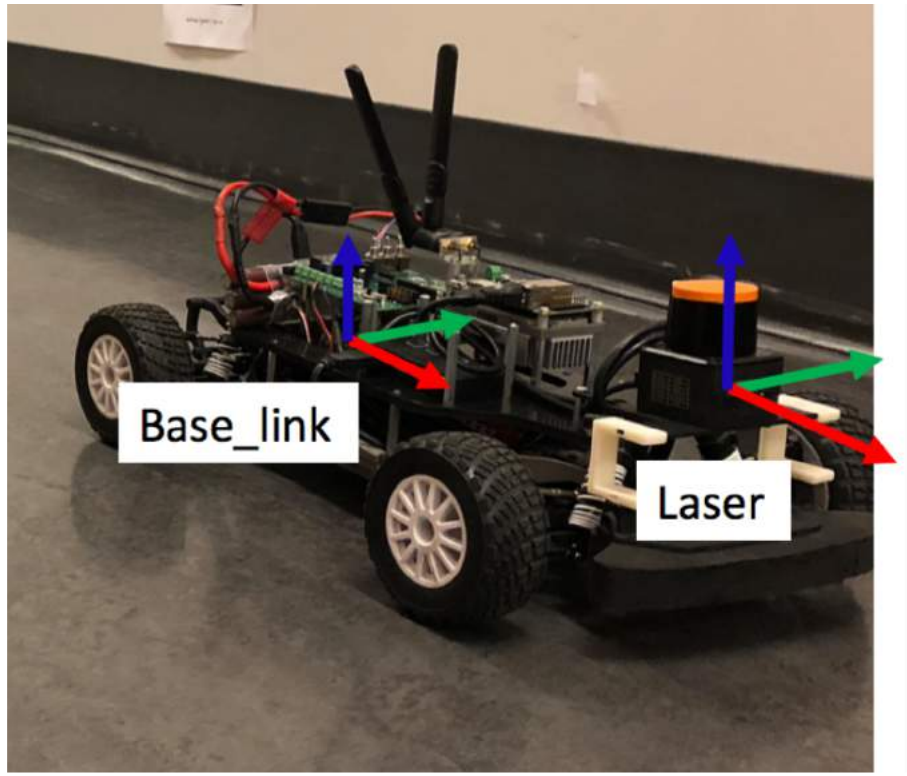
If we assume a flat world for our car, we only need **map**, **odom**, **base_link**, and **laser** (no motion in Z direction)

# Why do we need to transform between frames?

- Data is usually provided in the most convenient frame to the data source (e.g. laser scans).
- If an obstacle is detected in the *base_link,* we want to know where it is in the world, i.e. in *map*.
- When we do laser scanning, we need to know the transformation between the lidar position (*laser*) and the *base_link* of the car.
- If we had two disconnected maps, we might want to know their location with respect to each other within a global world frame.

# Static transform

- There exists a static transform between laser  and base_link (why?)
- What are the translation and rotation of laser relative to base_link?

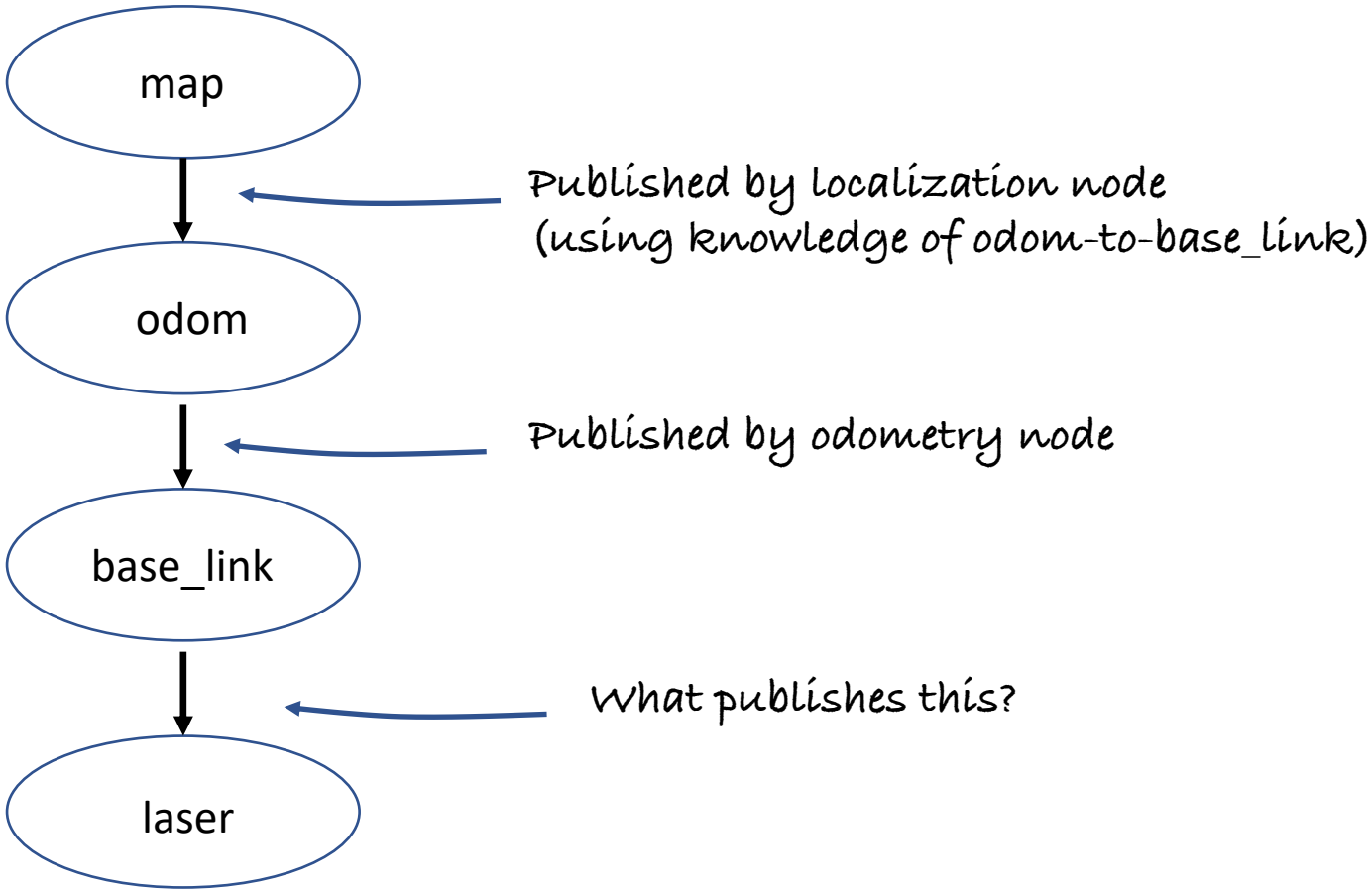


# TF Tree

- The TF tree in ROS gives us all the frames and their transforms.
- We can visualize the TF Tree by running

```
>> rosrun rqt_tf_tree rqt_tf_tree
```

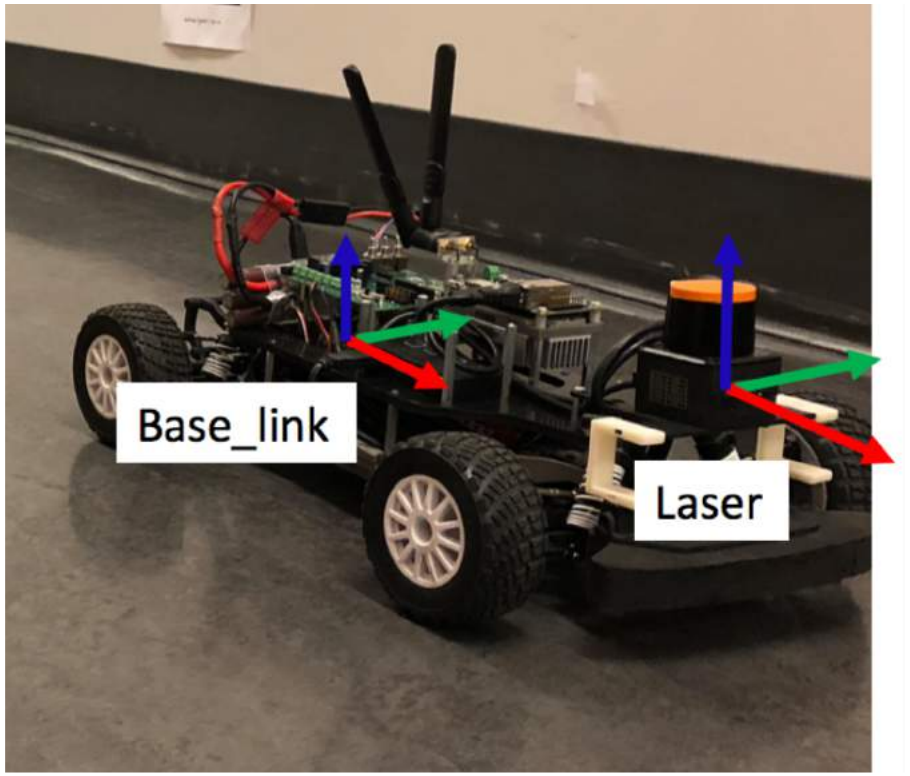
- If a localization package doesn't work, check if the ROS TF tree is connected.
- If a frame is connected in the tf tree, we can transform between it and any other frame in the tree.




Tf tree

# 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