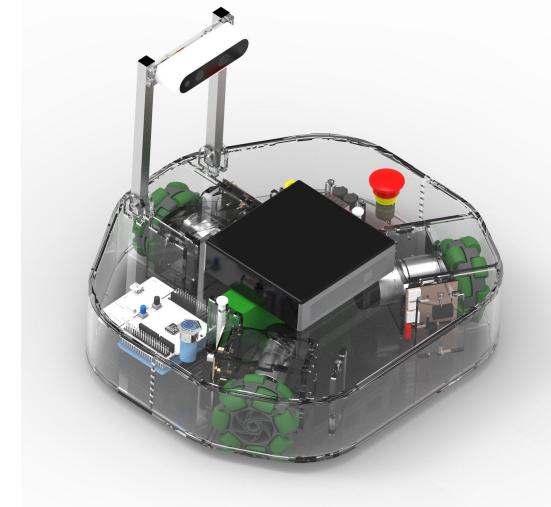




SLAM & Transforms in ROS

ROS Training for Industry: Day 4

Madis Kaspar Nigol
19.09.2019
Tartu, Estonia



Agenda: Day 4 (19.09)



- 09:15 **Transforms** in ROS, Gazebo
- 10:15 Coffee Break
- 10:30 Workshop: **static TF, broadcaster** programming
- 12:00 Lunch Break
- 13:00 **Localization, Mapping, SLAM, Navigation with Path Planning**
- 14:30 Coffee Break
- 14:45 Workshop
 - 2D mapping in Gazebo simulation
 - 2D mapping and navigation with Clearbot
 - 3D mapping on ClearBot
- 17:00 End of Day 4

What are transforms?

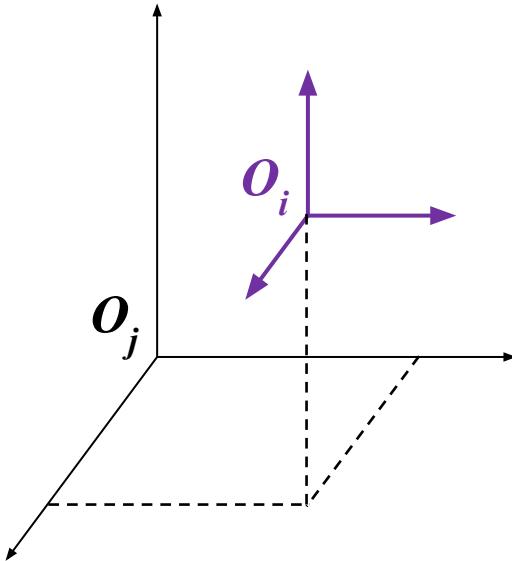
Terminology

- We will make frequent use of ***coordinate reference frames*** or simply ***frames***.
- A coordinate reference frame i consists of an origin – denoted O_i – and a triad of mutually orthogonal basis vectors – denoted $(x_i \ y_i \ z_i)$ – that are all fixed within a particular body.
- The ***pose of a body*** is always expressed ***relative to some other body***, so it can be expressed as ***the pose of one coordinate frame relative to another***.
- Similarly, ***rigid-body displacements*** can be expressed as displacements between two coordinate frames, one of which may be referred to as ***moving***, while the other may be referred to as ***fixed***.

Position and displacement

- The position of the origin of coordinate frame i relative to coordinate frame j can be denoted by the 3×1 vector

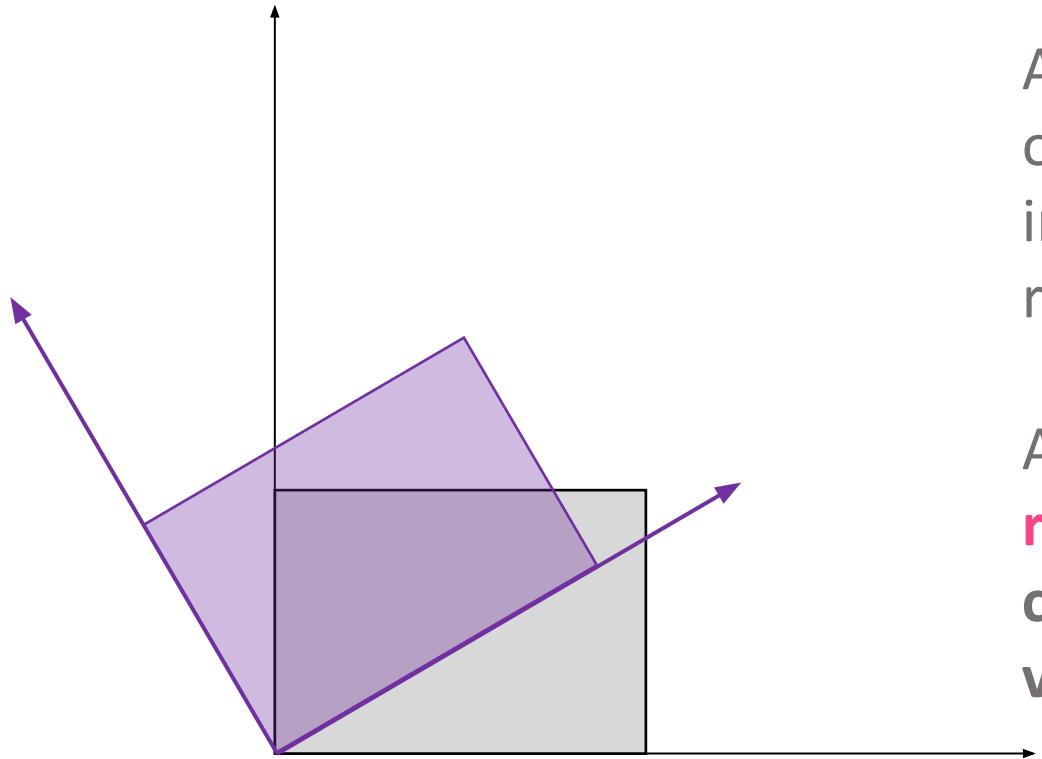
$${}^j p_i = \begin{pmatrix} {}^j p_i^x \\ {}^j p_i^y \\ {}^j p_i^z \end{pmatrix}$$



A **translation** is a displacement in which no point in the rigid body remains in its initial position and all straight lines in the rigid body remain parallel to their initial orientations.

Any **representation of position** can be used to create a **representation of displacement**, and vice versa.

Rotation and orientation



A **rotation** is a displacement in which at least one point of the rigid body remains in its initial position and not all lines in the body remain parallel to their initial orientations.

As in the case of position and translation, any **representation of orientation** can be used to create a **representation of rotation**, and vice versa.

Representing rotation and orientation

- Rotation matrix

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

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Euler angles – rotations relative to moving frame (order matters!)
- **Fixed angles**, e.g., roll-pitch-yaw (RPY) – rotations relative to fixed frame (order matters!)
- Angle-axis – single angle θ about one vector w , denoted as θw or $(\theta_{w_x} \theta_{w_y} \theta_{w_z})$
- **Quaternions**

TF in ROS

Frames in ROS

- Each part of the robot should have a **reference coordinate frame** attached to it
- These **frames** are used to determine **pose** of each part in **relation to other frames**

Static transforms



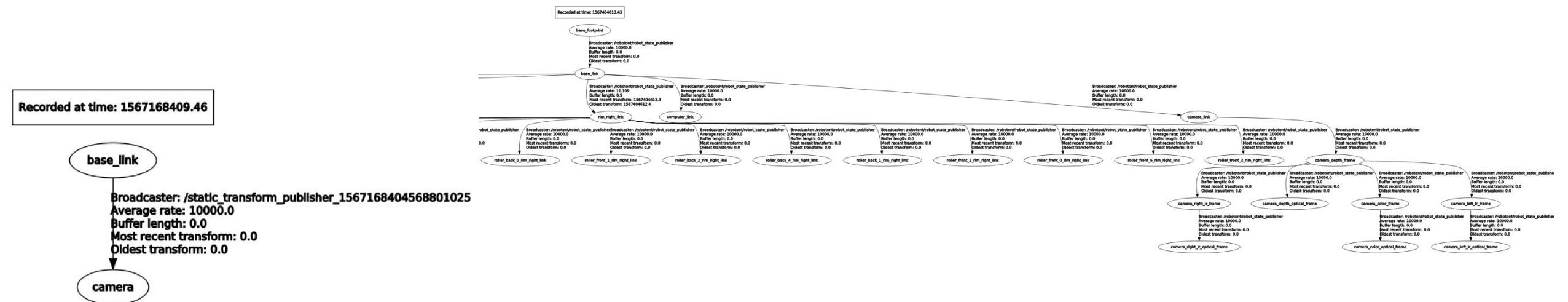
- Transform **should not** change during operation
- e.g. computer -> base_link

Dynamic transforms

- Transform **can** change during operation
- e.g. base_link -> map

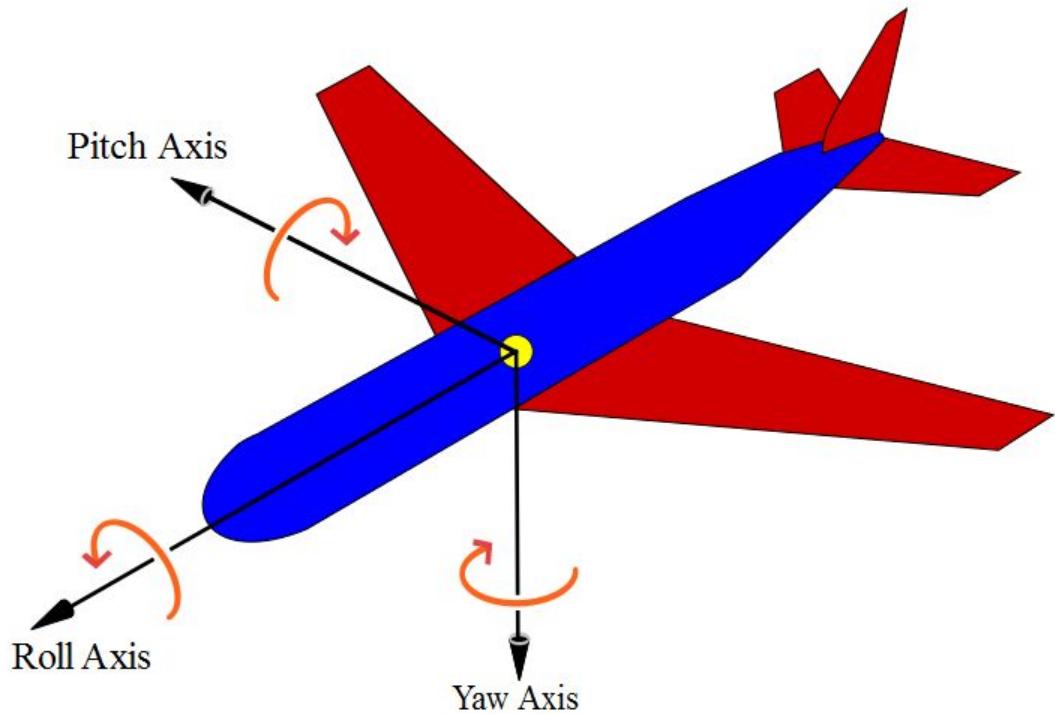
TF tree

- TF tree shows **all current transforms** and **their relations**
 - Using ***rqt***, we can visualize current TF tree



Rotations/orientations in ROS: RPY (roll-pitch-yaw)

- ROS uses **quaternions** [but RPY is also OK, as long as you know what you are doing 😊]
- RPY (roll-pitch-yaw)
 - Fairly intuitive
 - Originates from aerospace
 - **ROLL** – rotation about the axis from nose to tail
 - **PITCH** – nose up/down
 - **YAW** – nose left/right
 - Axes move with the aircraft, relative to Earth



geometry_msgs/Pose

- The position and orientation of a rigid body in space are collectively termed the **pose**.
- ROS has a **geometry_msgs/Pose** message type that consists of
 - **geometry_msgs/Point** position
 - float64 x
 - float64 y
 - float64 z
 - **geometry_msgs/Quaternion** orientation
 - float64 x
 - float64 y
 - float64 z
 - float64 w

geometry_msgs/PoseStamped



- **std_msgs/Header** header
 - **uint32** seq
 - **time** stamp
 - **string frame_id**
- **geometry_msgs/Pose** pose
 - **geometry_msgs/Point** position
 - ...
 - **geometry_msgs/Quaternion** orientation
 - ...

tf2 tutorials



- <http://wiki.ros.org/tf2/Tutorials>
- Good tutorials that cover all the basics

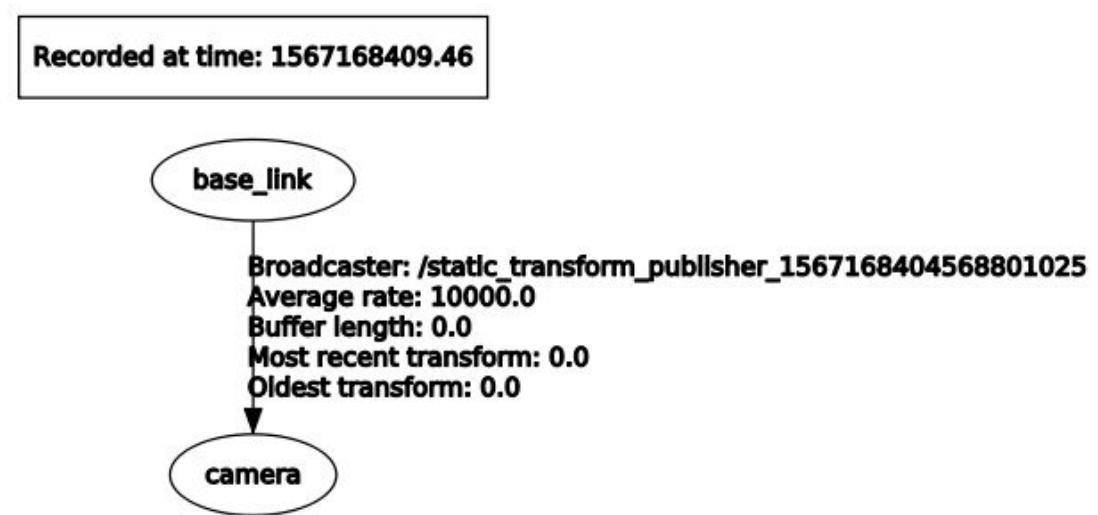
tf2_ros library examples



- void tf2::Quaternion::**setRPY** (const tf2Scalar &**roll**, const tf2Scalar &**pitch**, const tf2Scalar &**yaw**)
- Quaternion& tf2::Quaternion::**normalize** ()
- void tf2_ros::TransformBroadcaster::**sendTransform** (const geometry_msgs::TransformStamped &**transform**)

Using static_transform_publisher

- Used to quickly define a static transform between two frames
- Located in ROS package `tf2_ros`
- Syntax: `static_transform_publisher x y z yaw pitch roll frame_id child_frame_id`
- Note the order of RPY
- e.g `rosrun tf2_ros static_transform_publisher 0 0 1 0 0 0 base_link camera`



Using static_transform_publisher in roslaunch



<launch>

```
<node pkg="tf2_ros" type="static_transform_publisher"  
      name="my_tf_broadcaster" args="0 0 1 0 0 0 base_link camera" />
```

</launch>

Timing with transforms

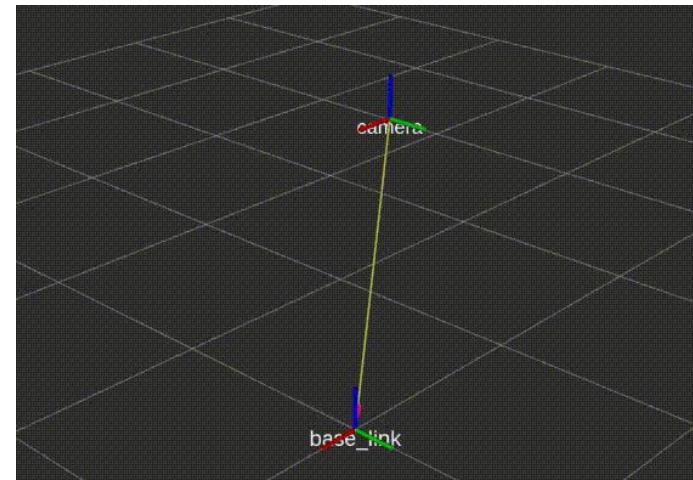


- In some cases, the timing of transforms is very important
 - mapping
 - camera calibration
- Example:
 - Camera, laserscan and odom coming from robot
 - Mapping software on external PC
 - Time not synchronized - mapping will fail
 - Use PTP or NTP (for example with Chrony) to sync
 - ...or run time critical tasks on same HW if possible

Writing a transform broadcaster in C++



- <http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20broadcaster%20%28C%2B%2B%29>
- <http://wiki.ros.org/tf2/Tutorials/Adding%20a%20frame%20%28C%2B%2B%29>



Writing a transform broadcaster in C++



```
#include <ros/ros.h>
#include <tf2_ros/transform_broadcaster.h>
#include <tf2/LinearMath/Quaternion.h>
```

```
int main(int argc, char** argv){
    ros::init(argc, argv, "my_tf2_broadcaster");
    ros::NodeHandle node;
```

```
tf2_ros::TransformBroadcaster tfb;
geometry_msgs::TransformStamped transformStamped;
```

Writing a transform broadcaster in C++



- Specify transform origin and endpoint (parent and child)

```
transformStamped.header.frame_id = "base_link";
transformStamped.child_frame_id = "camera";
```

Writing a transform broadcaster in C++



- Set transform translation

```
transformStamped.transform.translation.x = 0.0;  
transformStamped.transform.translation.y = 0.0;  
transformStamped.transform.translation.z = 1.0;
```

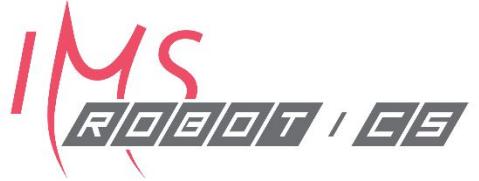
Writing a transform broadcaster in C++



- Set transform rotation

```
tf2::Quaternion q;  
    q.setRPY(0, 0, 0);  
transformStamped.transform.rotation.x = q.x();  
transformStamped.transform.rotation.y = q.y();  
transformStamped.transform.rotation.z = q.z();  
transformStamped.transform.rotation.w = q.w();
```

Writing a transform broadcaster in C++



- Publish the transform

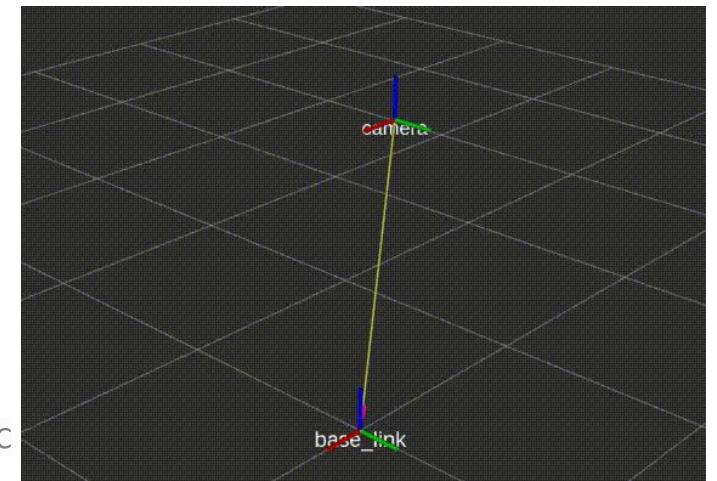
```
ros::Rate rate(10.0);
while (node.ok()){
    transformStamped.header.stamp = ros::Time::now();
    tfb.sendTransform(transformStamped);
    rate.sleep();
}
```

Writing a transform broadcaster in C++



- Publish the changing transform

```
while (node.ok()) {  
    transformStamped.header.stamp = ros::Time::now();  
    tfb.sendTransform(transformStamped);  
    i += 0.1;  
    transformStamped.transform.translation.x = 1 * sin(i);  
    transformStamped.transform.translation.y = 1 * cos(i);  
    rate.sleep();  
}
```



Gazebo

and Clearbot simulation in Gazebo



What is Gazebo?

- ROS simulation program - <http://gazebosim.org>
- Add environment
- Add robot model
 - Xacro, URDF, Day 5 topic
- Add plugins
 - Camera
 - Depth camera
 - Control
- TurtleBot3 Gazebo
 - <http://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/>

Gazebo Plugins

- Plugins enable to control the simulation environment
- Gazebo standalone
 - World, Model, Sensor, System, Visual, GUI
- Gazebo ROS
 - All previous can be connected to ROS
 - Model, Sensor, Visual can be used in URDF

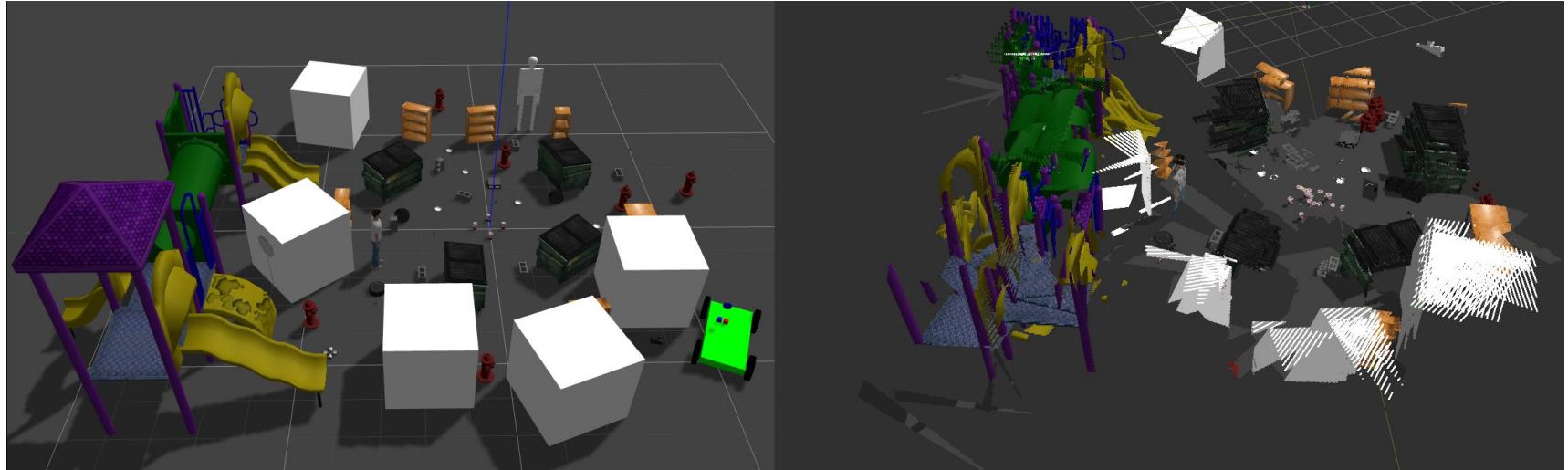
Clearbot Gazebo



- Xacro model
 - modular
 - simplified meshes
- Camera and depth camera plugins
 - Based on D435
- Control by planar_move plugin
 - Simple x, y, theta control
 - Low realism
 - Can't use ros_control on weaker PCs
- Realtime factor at or near 1

Drone Gazebo simulation

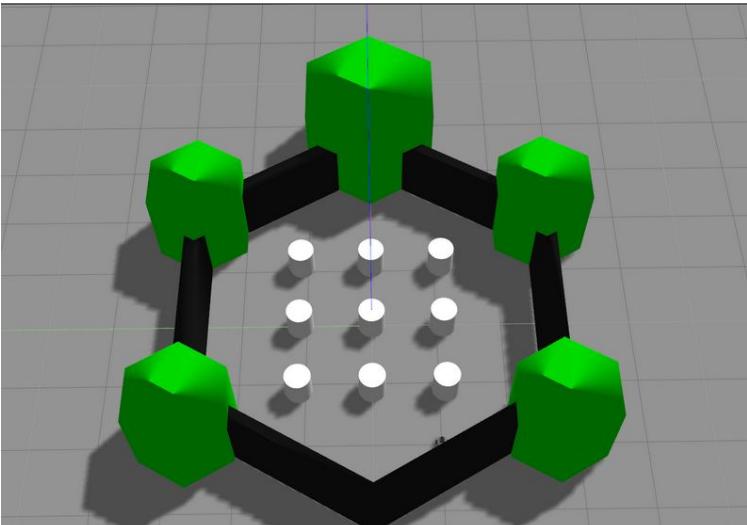
- Plugins
 - Camera
 - Mavlink
 - Controller
 - GPS
 - IMU
 - Wind, depth camera, odometry etc..
- https://github.com/jellosubmarine/droonituuled/blob/master/robotex_sim/worlds/robotex.world



Localization

Position and sensing

- Dead reckoning
 - e.g. wheel odometry, IMU
 - **subject to cumulative error** (encoder values increasing when slipping)
- Sensing
 - e.g camera, LiDAR, sonar
 - problem is **perceptual aliasing** - two different places seem the same

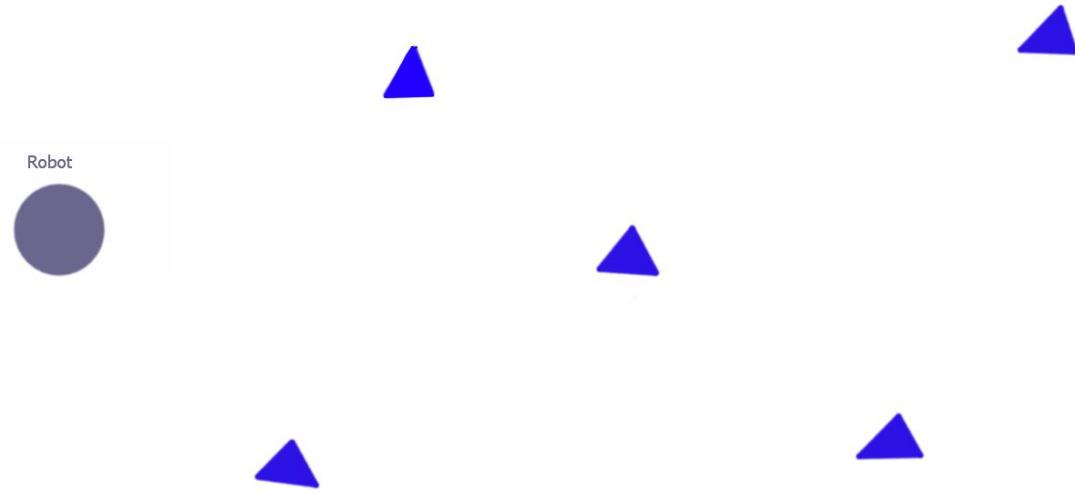


<http://emanuel.robotis.com/docs/en/platform/turtlebot3/simulation/>

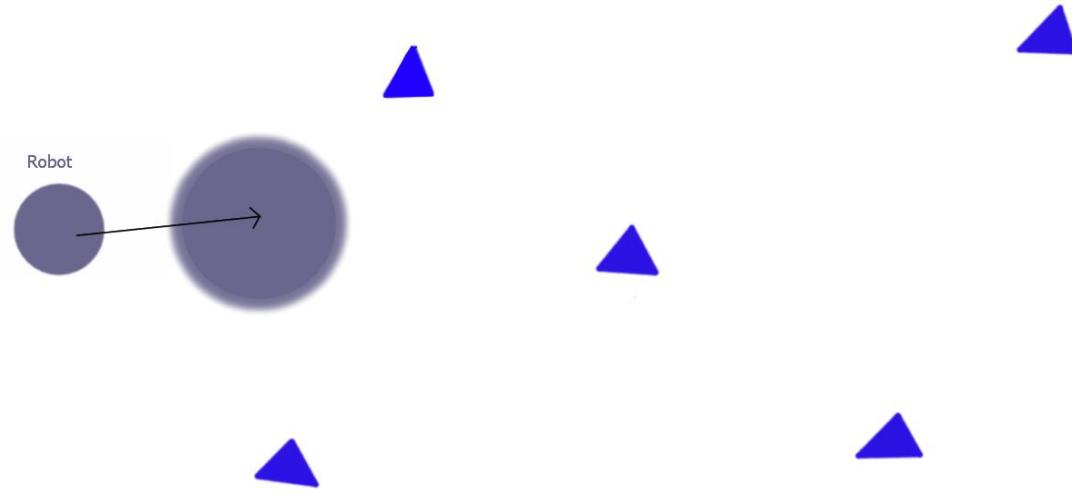
Localization

- Assume a known map
- Dead reckoning
 - Start from known place
 - Localization error increases over time
- Use landmarks and reference them to a known map

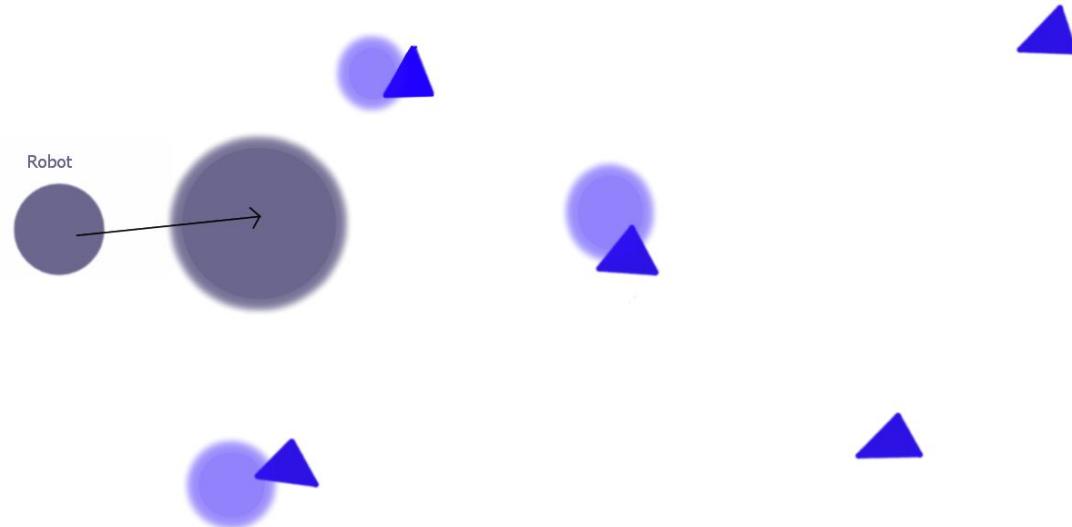
Localization illustrated



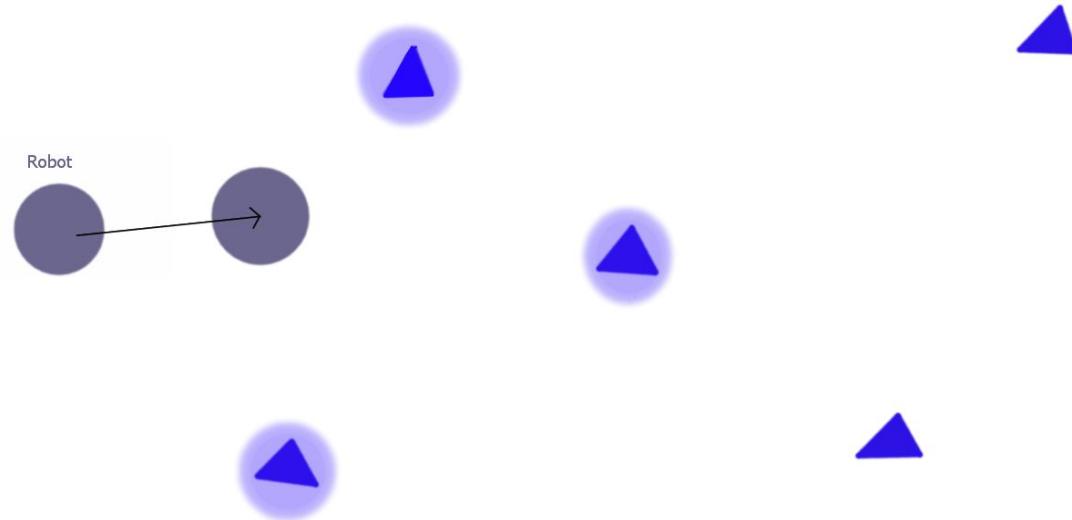
Localization illustrated



Localization illustrated



Localization illustrated



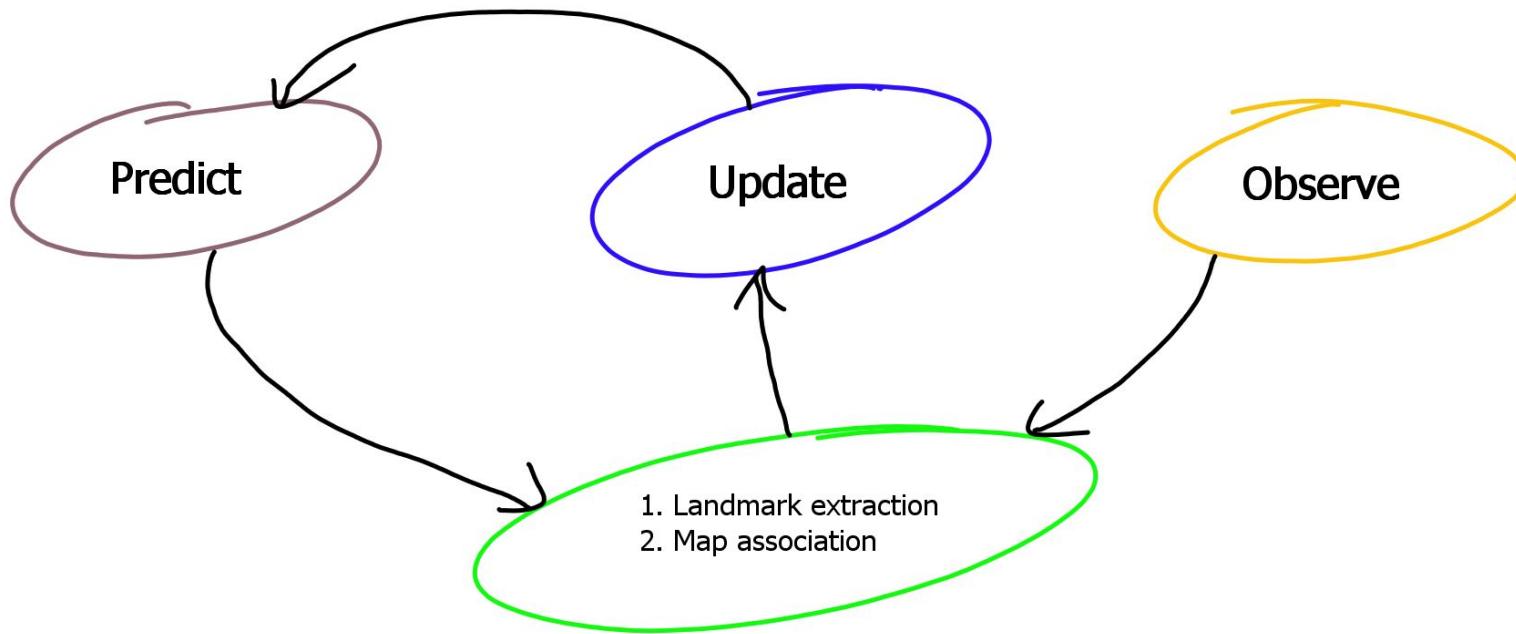
Localization TF

- map->odom->base_link
- Dead reckoning is odom->base_link
- map->odom TF fixes dead reckoning drift

Bayesian Filter

- Two main steps:
 - Prediction and update
- Prediction uses previous location and measurements and physical model to predict where the robot is going to be on the next timestep
- Update step makes an observation of the current situation
- <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python>

Bayesian filter in localization



Localization pipeline



Mapping

Clearbot sensors



- Depth camera
- Wheel odometry
- 2D scan from depth camera

General mapping



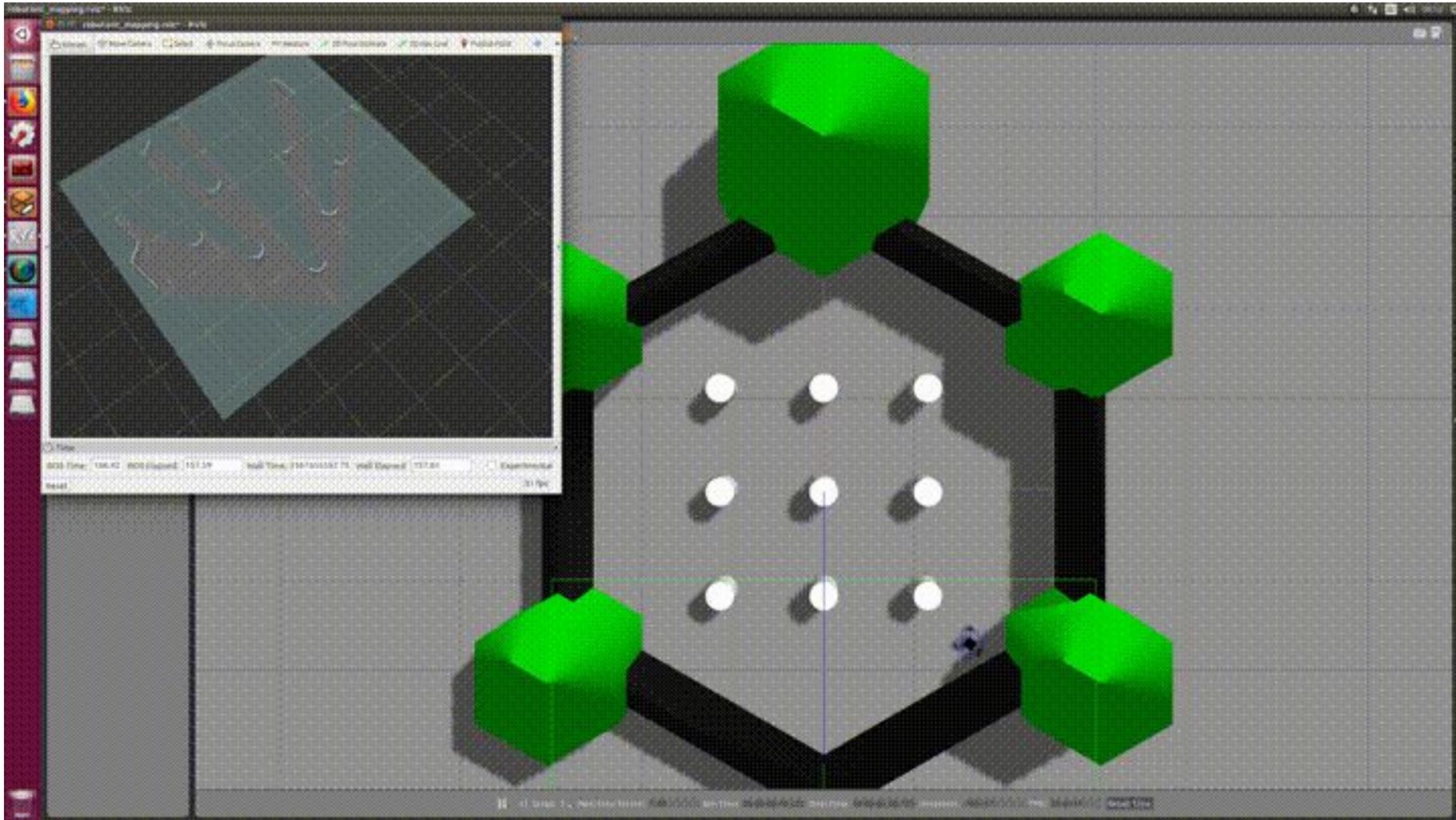
- Robot knows where it is but doesn't know where anything else is
- As with localization, the robot searches for landmarks
- When a landmark is found, it saves it in a map
- To tie all the landmarks together, a loop closure is needed
 - “Oh, I've already been here!”

SLAM

Simultaneous Localization and Mapping



Example of SLAM



Visual SLAM



- Use only cameras and visual information
- Create a depth map using one to multiple cameras
- Usually, Lidars are used to increase the accuracy of Visual SLAM
 - Tesla said that they don't need Lidars

Common ROS SLAM packages

gmapping



- <http://wiki.ros.org/gmapping>
-

Google Cartographer



- <http://wiki.ros.org/cartographer>
- Actively developed 2D SLAM package
- Recommended to build fresh version from git instead of using apt

hector_slam



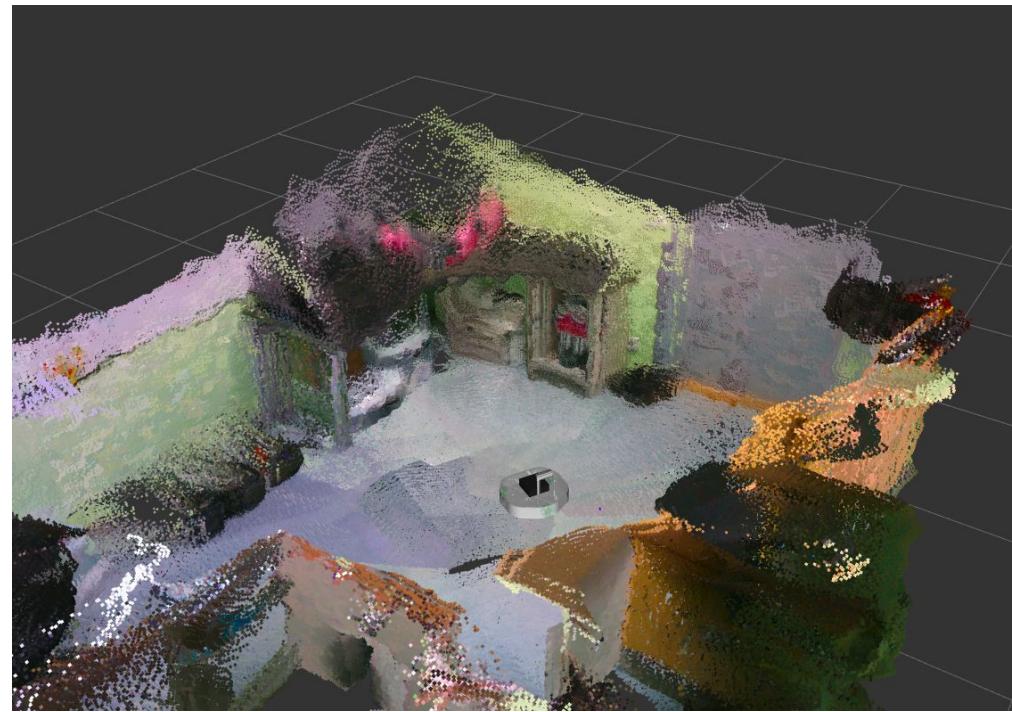
- http://wiki.ros.org/hector_slam
- You may see it a lot but it's development is quite inactive

- <http://wiki.ros.org/amcl>
- Adaptive Monte Carlo Localization
- Localization against known map

RTABMap



- http://wiki.ros.org/rtabmap_ros
- Currently one of the best packages for 3D mapping



orb_slam2_ros



- https://wiki.ros.org/orb_slam2_ros
- Visual SLAM
- 3D

Navigation with Path Planning

Overview

- Costmaps
- Local and global planners
- Requirements for navigation
- Steering mechanisms
- ROS navigation packages

Costmaps

Costmaps

- Costmap shows where it is safe to be for the robot
- Usually costmaps are **binary occupancy grids**
- Advanced costmaps can also be non-binary
 - Each part of the map has a different cost depending on how difficult it is to travel through
 - e.g. Different terrains have different “costs”

Path planning

Local planner



- Plans short paths
- Publishes cmd_vel
- Tries to follow global planner
- Can avoid obstacles unknown to global planner
 - Avoiding cars on a street vs which street to take

Global planner

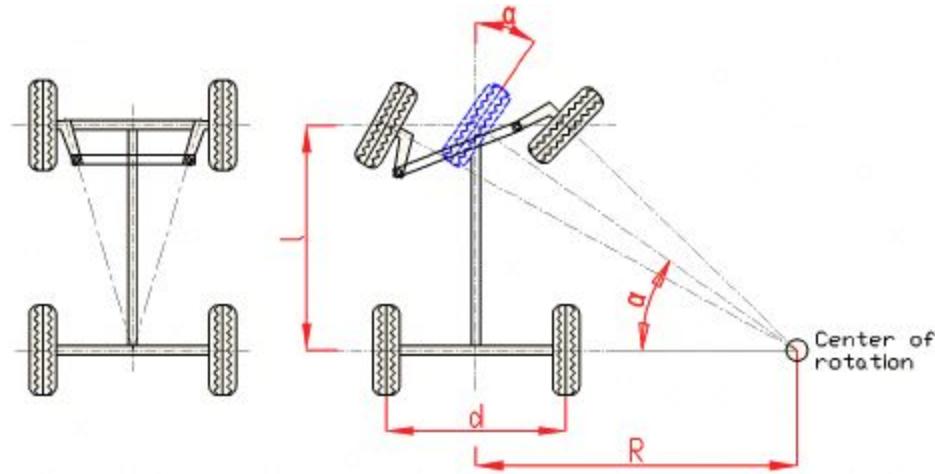
- Plans the whole trajectory
 - e.g GPS navigation
- Global planner sees the current assumed world state
- In a warehouse:
 - Global map could be the layout of the warehouse, not updated often
 - Local map is constantly updated from sensor data to avoid collisions

https://www.researchgate.net/publication/258163012_Spline-Based_RRT_Path_Planner_for_Non-Holonomic_Robots

Steering mechanisms

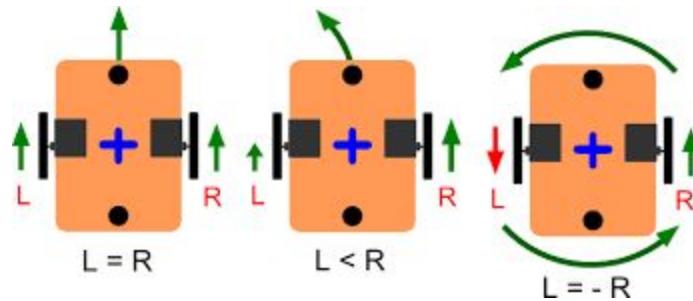
Ackermann steering

- Cars



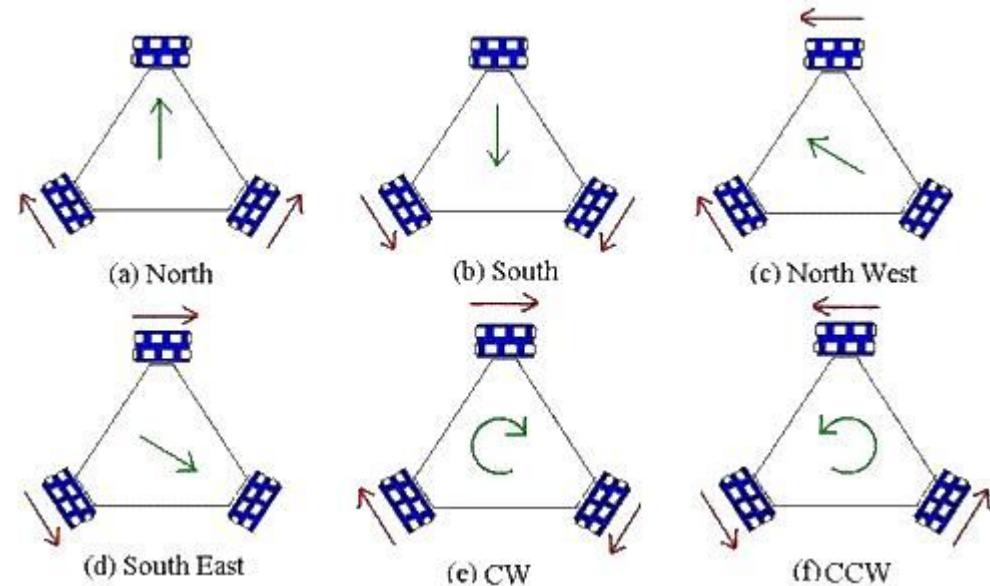
Differential steering

- Tracked vehicles



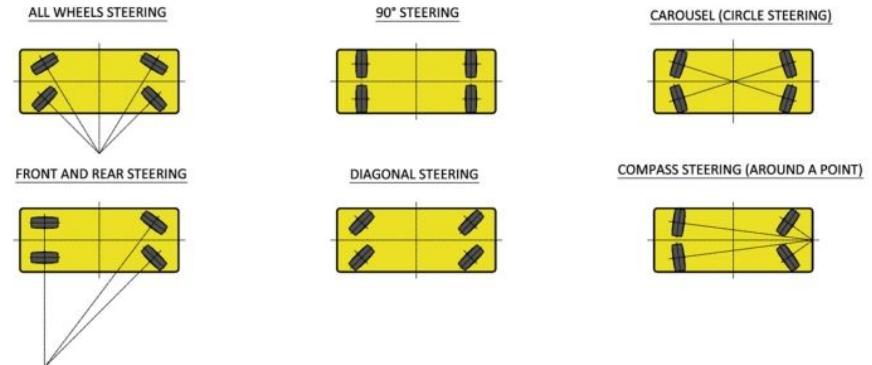
Omnidirectional steering

- Clearbot



<http://sine.ni.com/cs/app/doc/p/id/cs-14839>

STEERING TYPOLOGIES



<https://www.morellogiovanni.it/en/motorized-platforms-up-to-500-tons-capacity/>

ROS navigation

Requirements for navigation



- ROS navigation stack currently supports only differential and omnidirectional steering.
- For navigation, the robot must:
 - Accept velocity commands
 - Publish odometry messages

Common parameters

- Goal tolerance
- Maximum and minimum speed
- Local and global map size
- Numerous others

ROS navigation packages



- <http://wiki.ros.org/navigation>
- ROS navigation stack
- If your robot meets navigation requirements, it can easily be configured to autonomously navigate

3D navigation



- https://github.com/jhu-asco/dsl_gridsearch
- Mostly poor packages