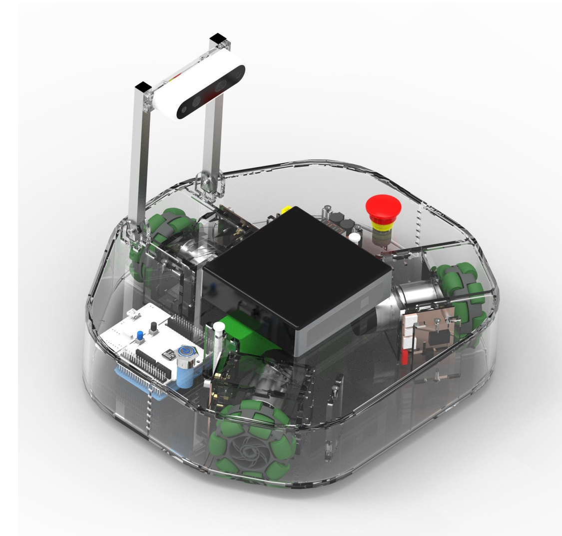
SLAM & Transforms in ROS

ROS Training for Industry: Day 4

Madis Kaspar Nigol

19.09.2019

Tartu, Estonia



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

What are transforms?

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Terminology

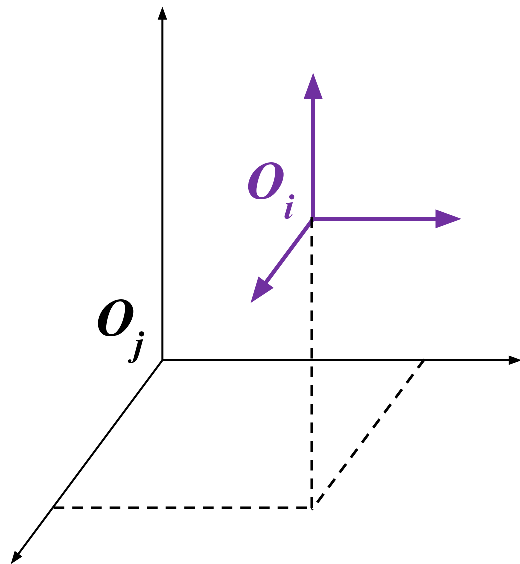
- We will make frequent use of ***coordinate reference frames*** or simply ***frames***.
- A coordinate reference frame $\boldsymbol{i}$ consists of an origin – denoted $\boldsymbol{O_i}$ – and a triad of mutually orthogonal basis vectors – denoted $(x_i\ y_i\ z_i)$ – that are all fixed within a particular 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***.

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Position and displacement

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

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

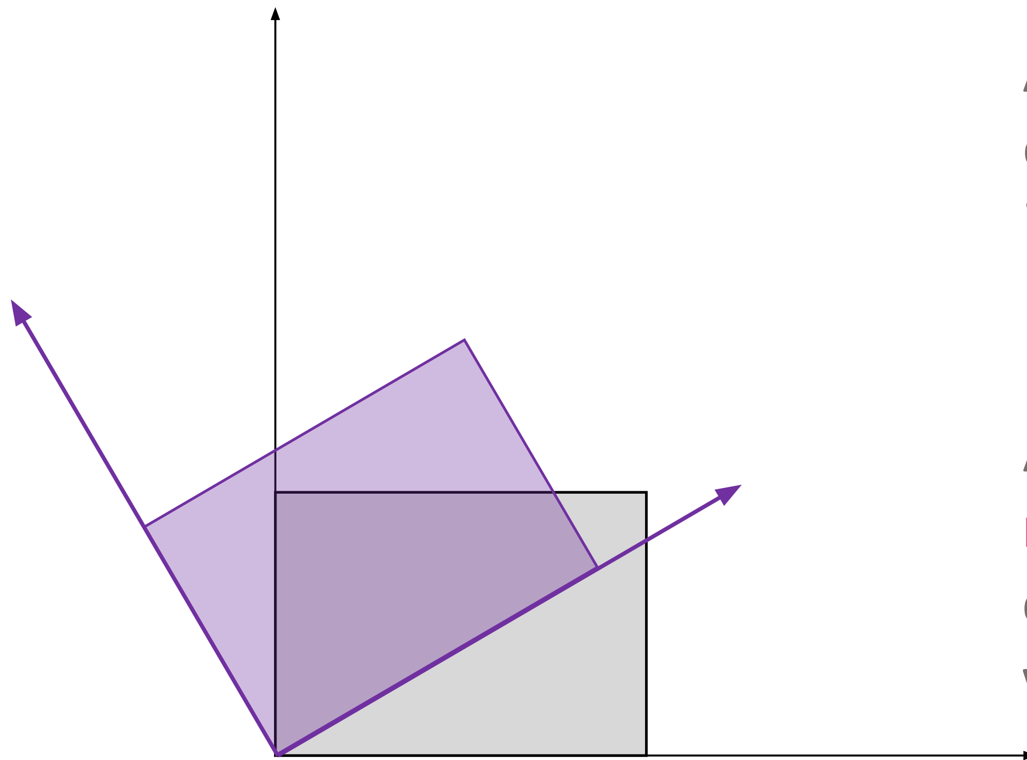


A **translation** is a displacement in which no point in the rigid body remains in its initial position and all straight lines in the rigid body remain parallel to their initial orientations.

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Rotation and orientation



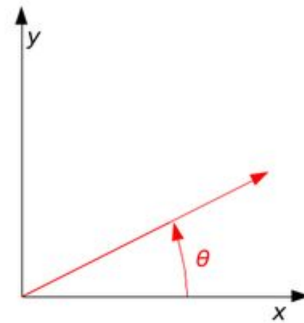
A **rotation** is a displacement in which at least one point of the rigid body remains in its initial position and not all lines in the body remain parallel to their initial orientations.

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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 $\boldsymbol{w}$, denoted as $\theta\boldsymbol{w}$ or $(\theta w_x\ \theta w_y\ \theta w_z)$
- **Quaternions**

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

TF in ROS

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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**

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Static transforms

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

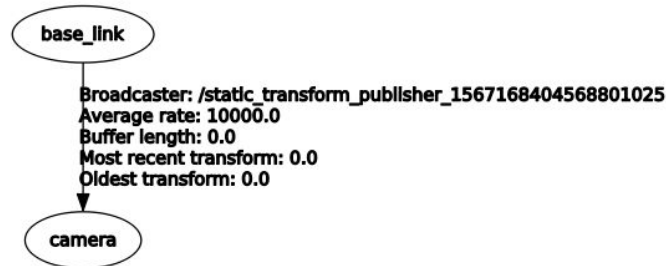
Dynamic transforms

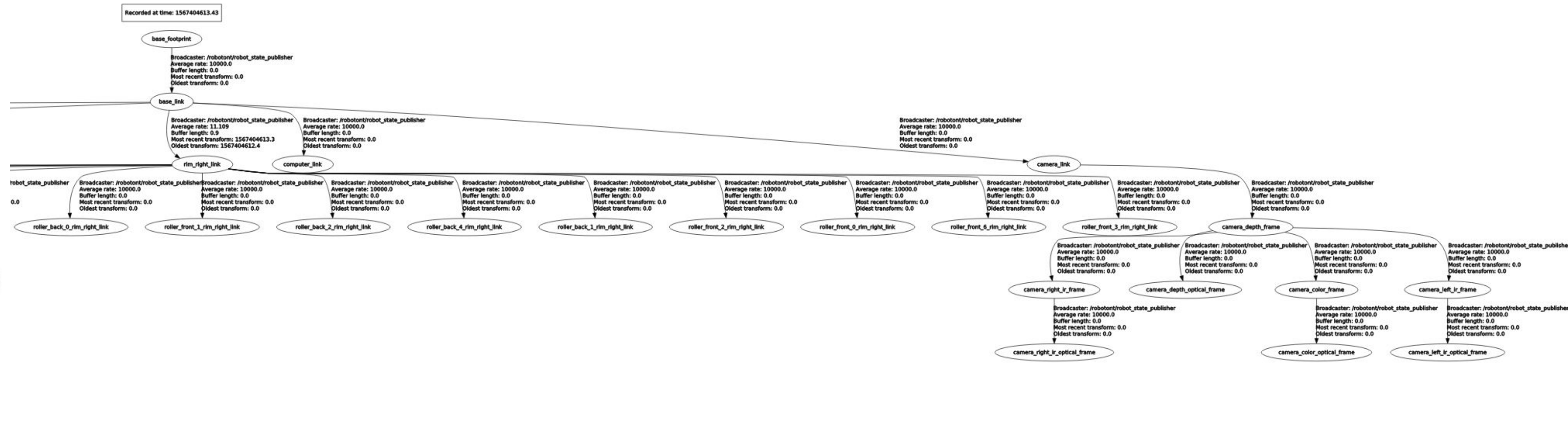
- Transform **can** change during operation
- e.g. base_link -> map

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

TF tree

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

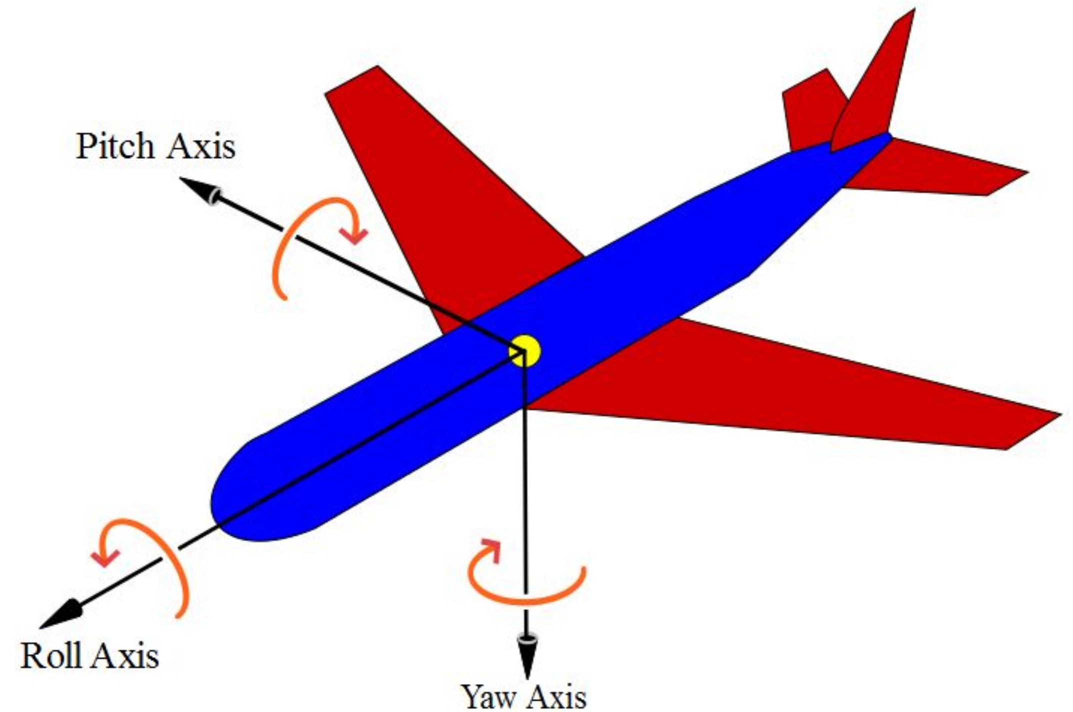




Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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, <u>relative to Earth</u>



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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
    - …

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

tf2 tutorials

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

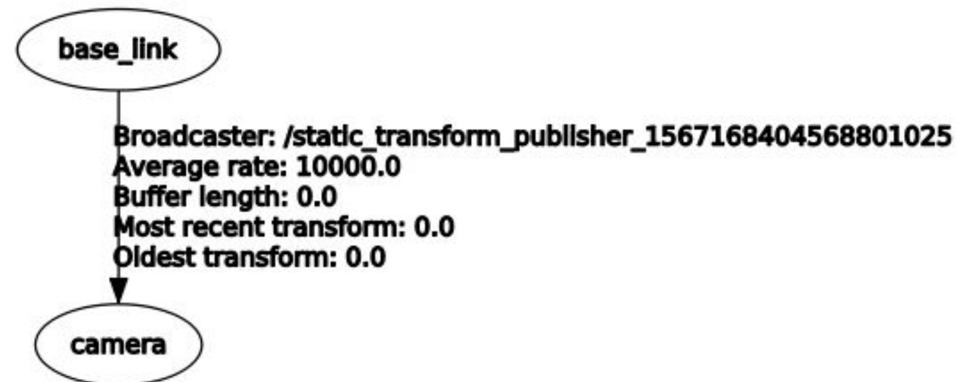
tf2_ros library examples

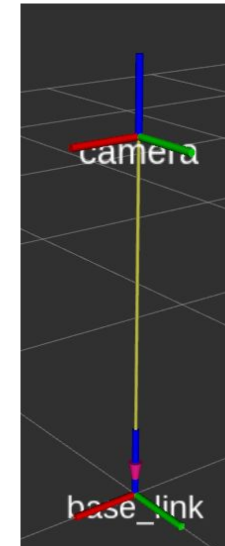
- void tf2::Quaternion::**setRPY** (const tf2Scalar &**roll**, const tf2Scalar &**pitch**, const tf2Scalar &**yaw**)
- Quaternion& tf2::Quaternion::**normalize** ()
- void tf2_ros::TransformBroadcaster::**sendTransform** (const geometry_msgs::TransformStamped &**transform**)

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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***





Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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>
```

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

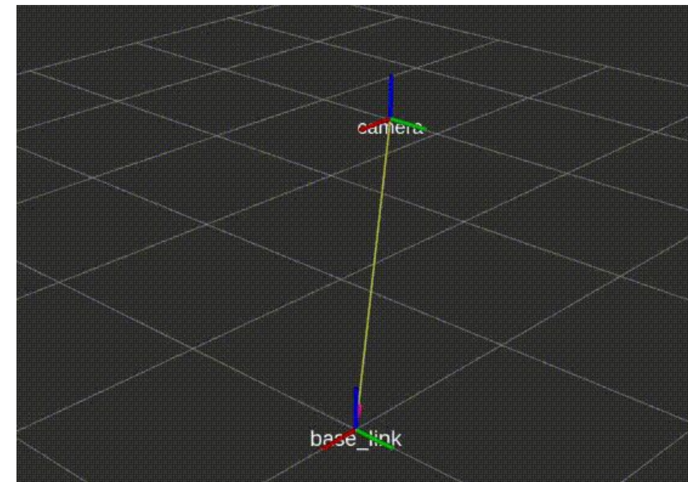
Timing with transforms

- 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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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;
```

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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";
```

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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;
```

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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();
```

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Writing a transform broadcaster in C++

- Publish the transform

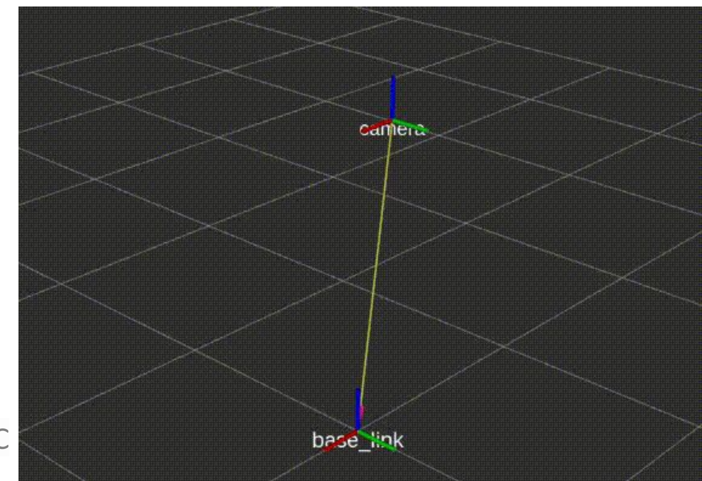
```
  ros::Rate rate(10.0);
  while (node.ok()){
    transformStamped.header.stamp = ros::Time::now();
    tfb.sendTransform(transformStamped);
    rate.sleep();
  }
```

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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();
}
```



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Gazebo

and Clearbot simulation in Gazebo

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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/

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

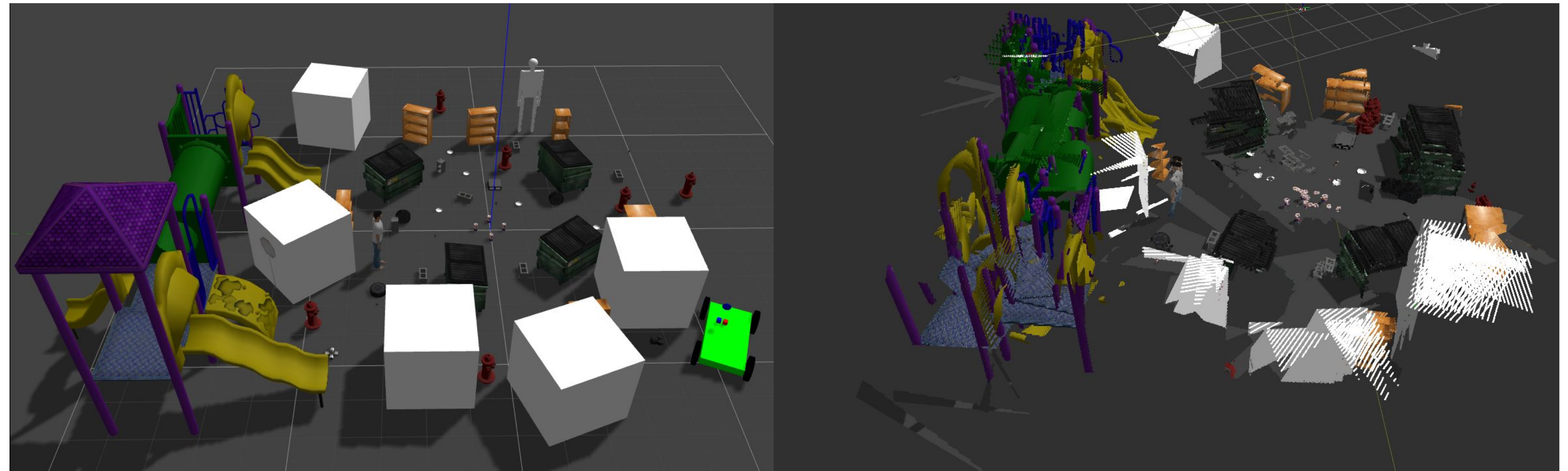
Clearbot Gazebo

- 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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

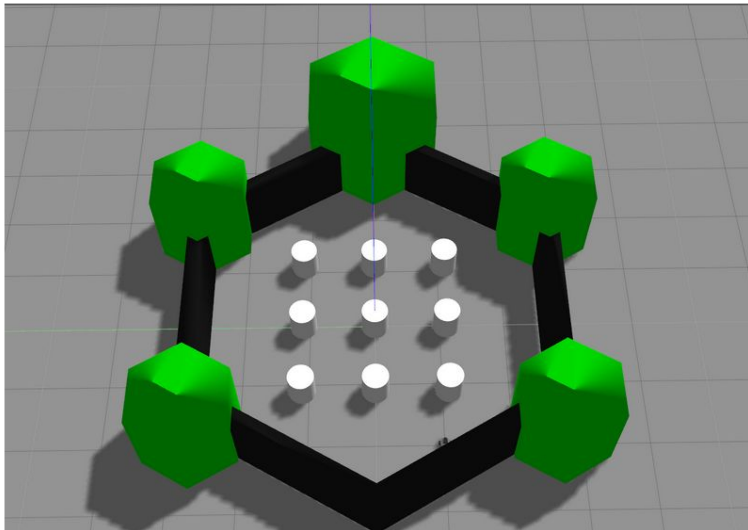


Localization

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/

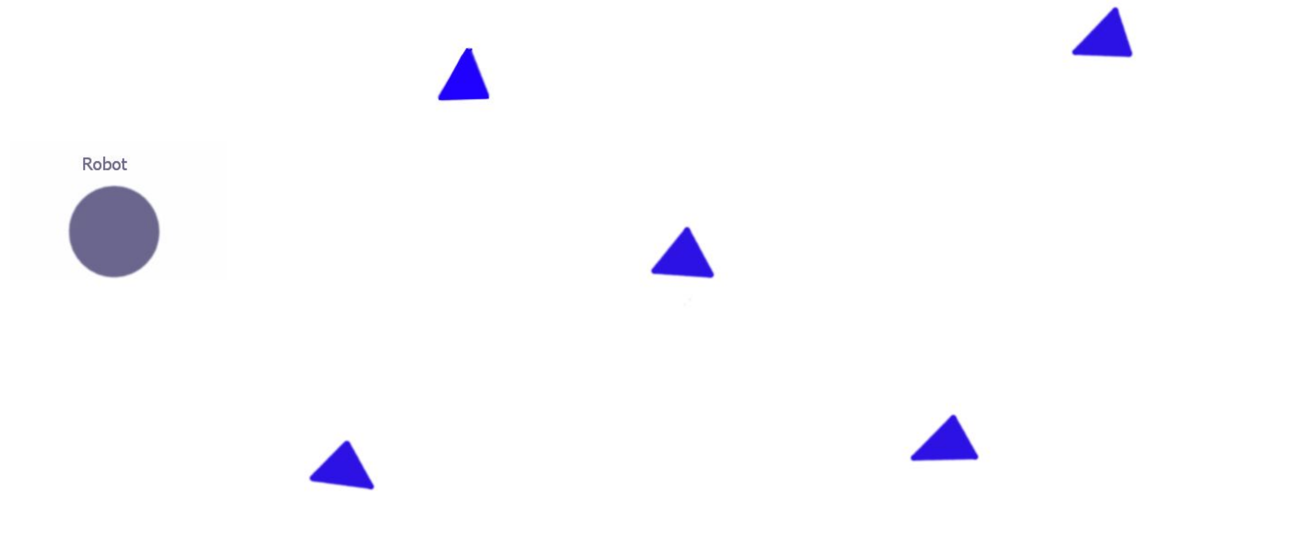
Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

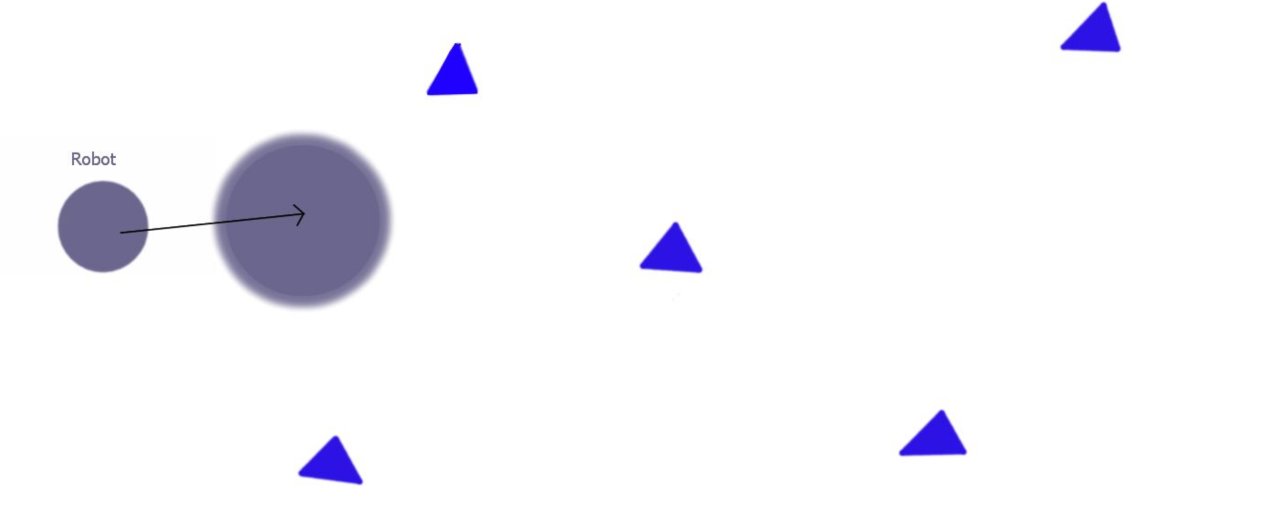
Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Localization illustrated



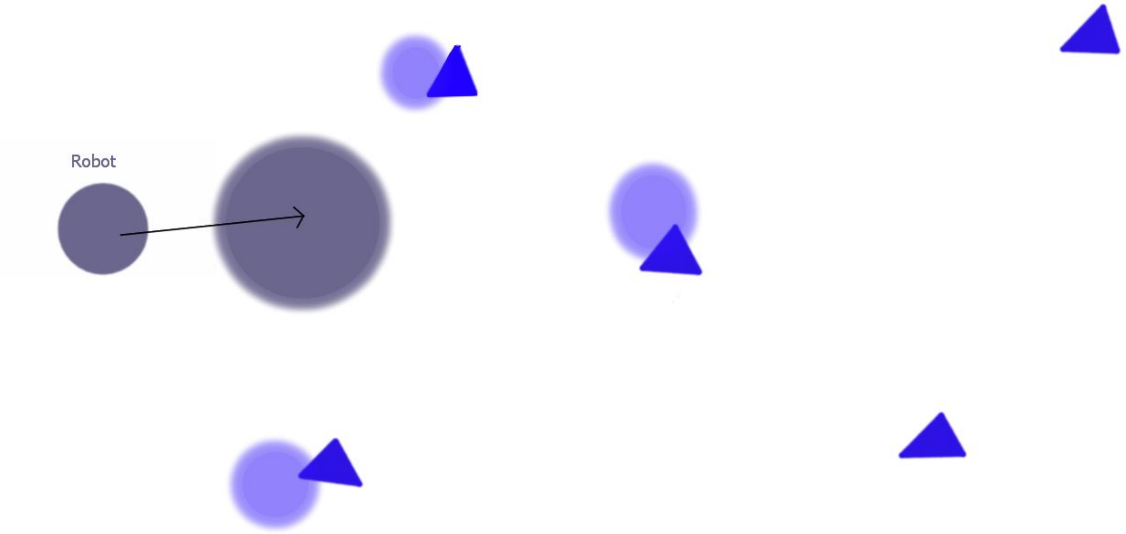
Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Localization illustrated



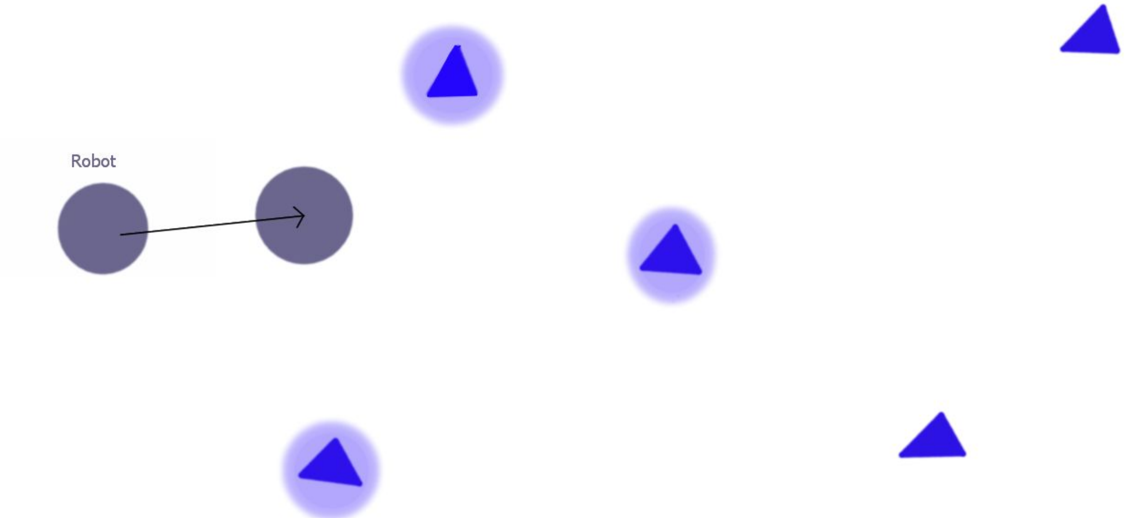
Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Localization illustrated



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Localization illustrated



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Localization TF

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

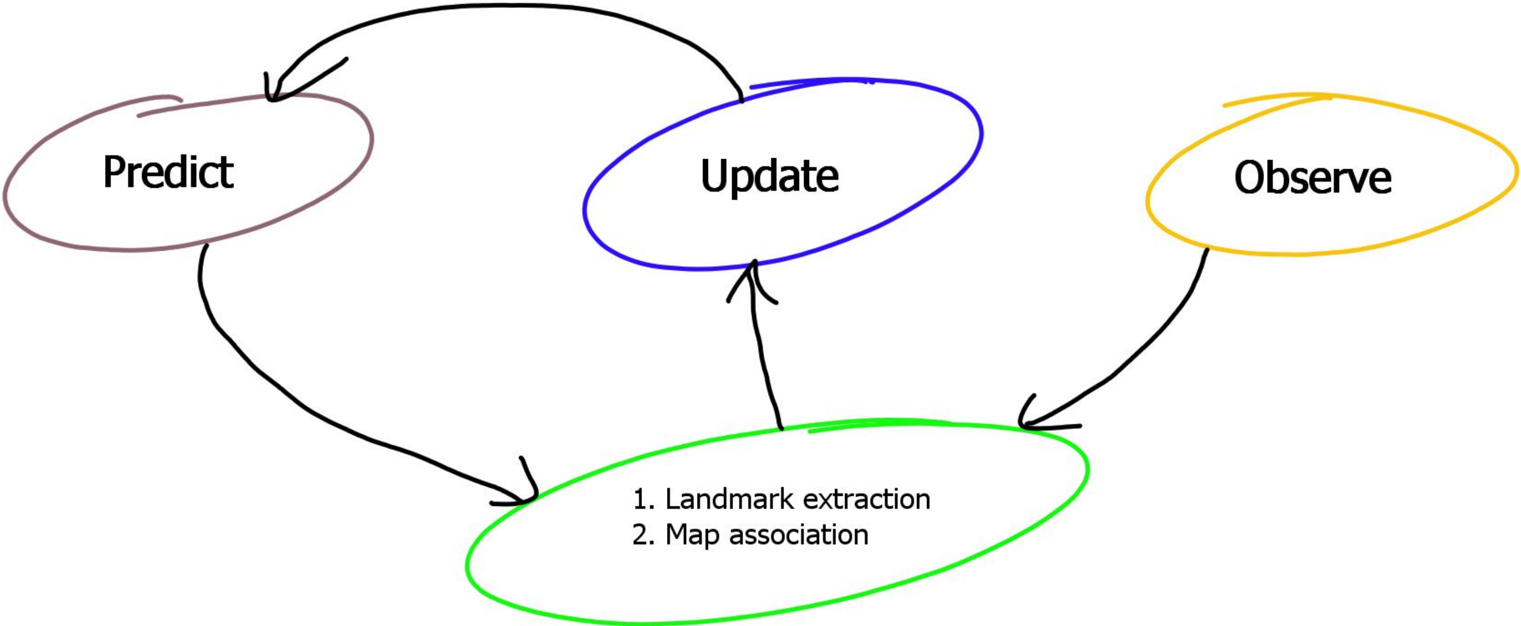
Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Bayesian filter in localization



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Mapping

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Clearbot sensors

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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!”

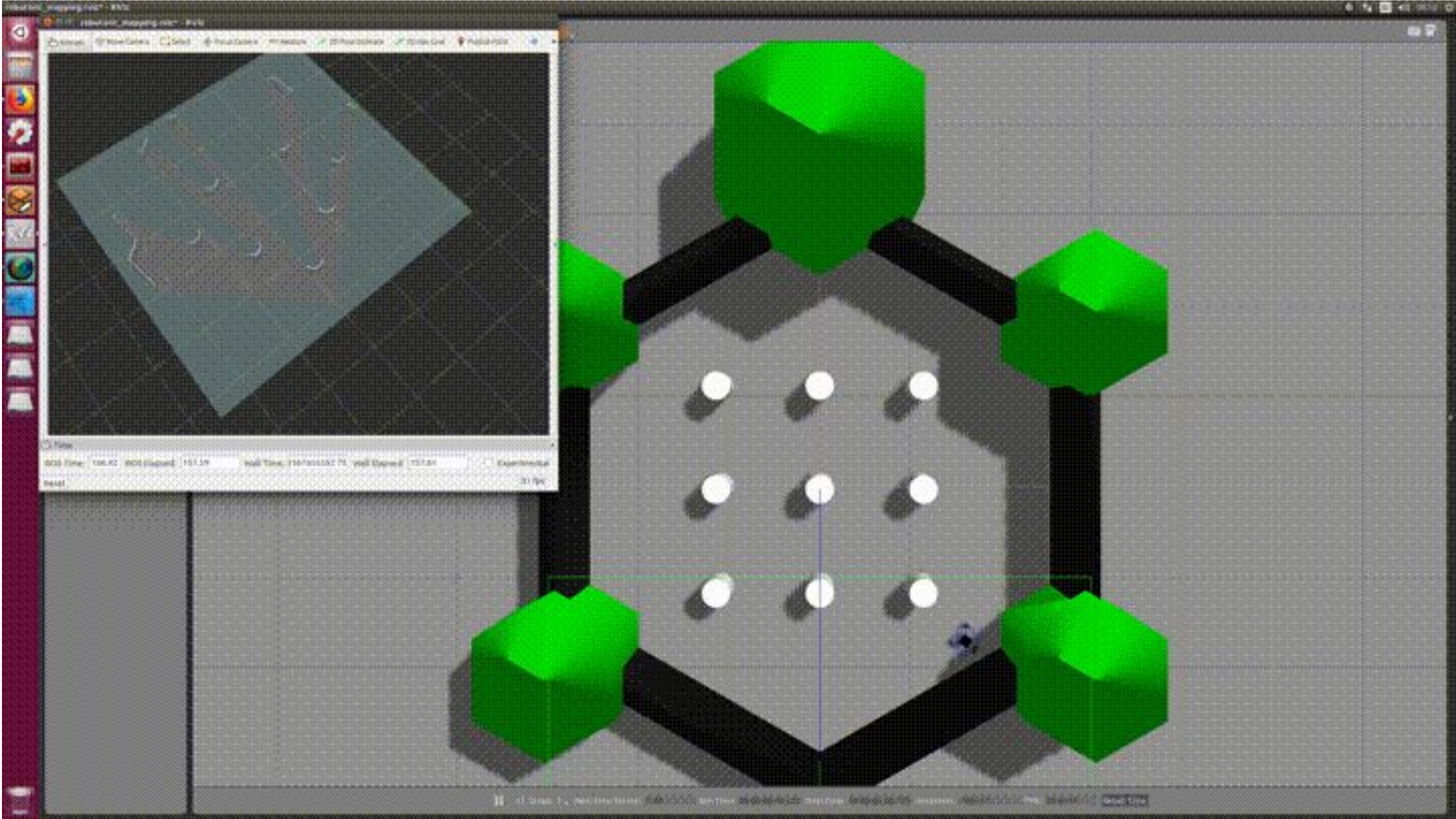
Copyright 2019, University of Tartu, Licence CC BY-ND-NC

SLAM

Simultaneous Localization and Mapping

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Example of SLAM



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Common ROS SLAM packages

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

gmapping

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

TARTU ÜLIKOOL · UNIVERSITAS TARTUENSIS · 1632

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Google Cartographer

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

hector_slam

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

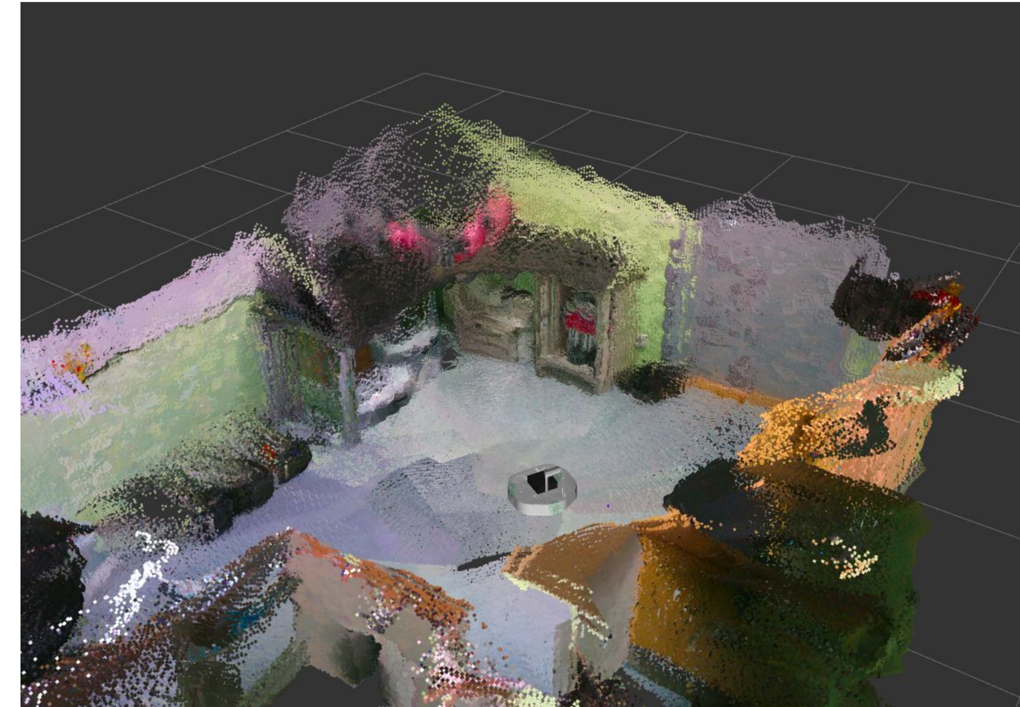
amcl

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

RTABMap

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



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

orb_slam2_ros

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

TARTU ÜLIKOOL · UNIVERSITAS TARTUENSIS · 1632

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Navigation with Path Planning

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Overview

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Costmaps

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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”

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Path planning

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

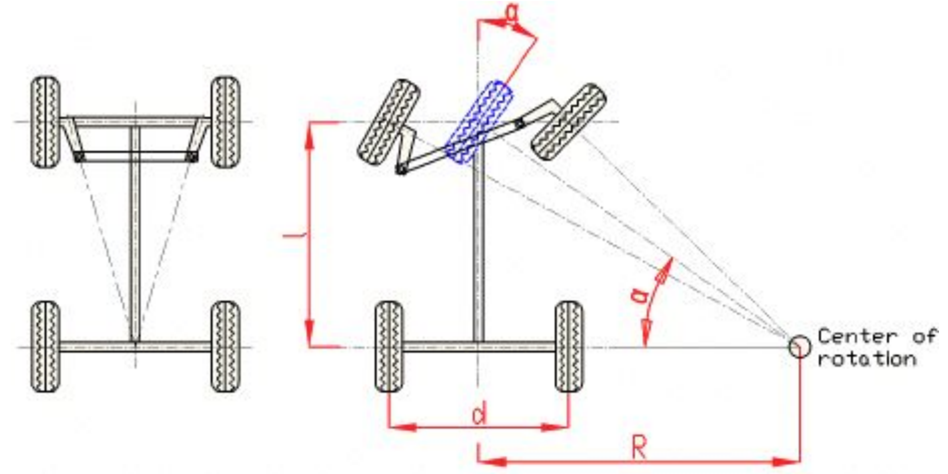
Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Steering mechanisms

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Ackermann steering

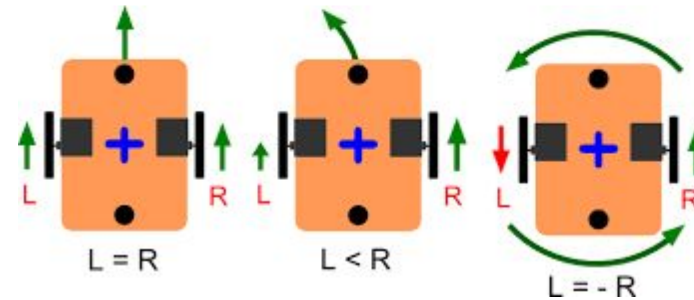
- Cars



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Differential steering

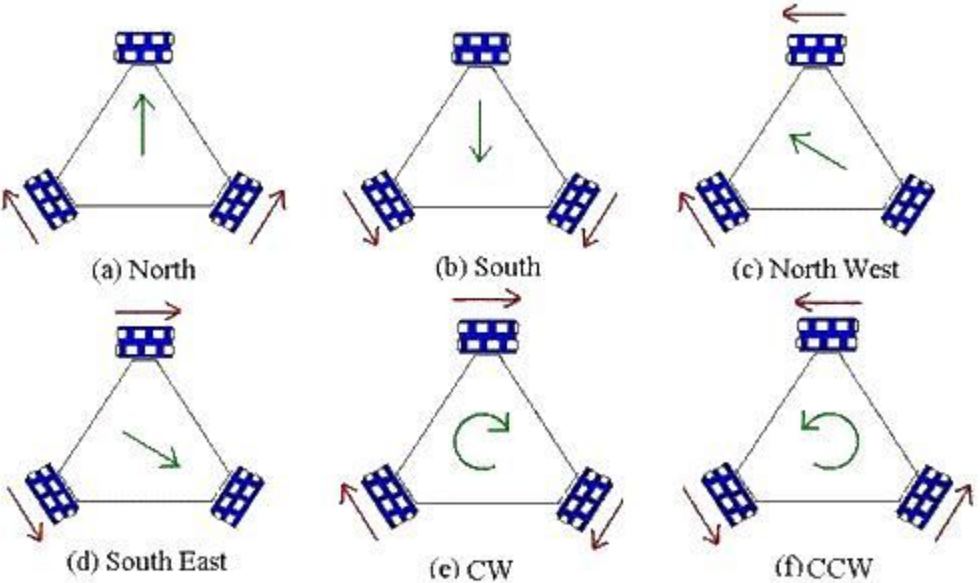
- Tracked vehicles



Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Omnidirectional steering

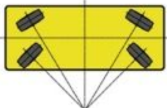
- Clearbot



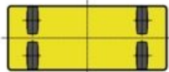
http://sine.ni.com/cs/app/doc/p/id/cs-14839

STEERING TYPOLOGIES

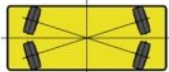
ALL WHEELS STEERING



90° STEERING

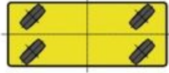


CAROUSEL (CIRCLE STEERING)

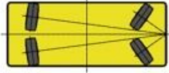


FRONT AND REAR STEERING

DIAGONAL STEERING



COMPASS STEERING (AROUND A POINT)



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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

ROS navigation

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Requirements for navigation

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

Common parameters

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC

3D navigation

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

Copyright 2019, University of Tartu, Licence CC BY-ND-NC