DTU

Rasmus Andersen

34761 – Robot Autonomy

Mapping

Overview of 34761 – Robot Autonomy

- 3 lectures on software frameworks
- 7 lectures on building your own autonomy stack for a mobile robot
- 1 lecture on DL/RL – an overview of black-box approaches to what you have done
- 2 lectures of project work before hand in + guest lectures

Software frameworks | Robot autonomy stack | Project work + guest lectures

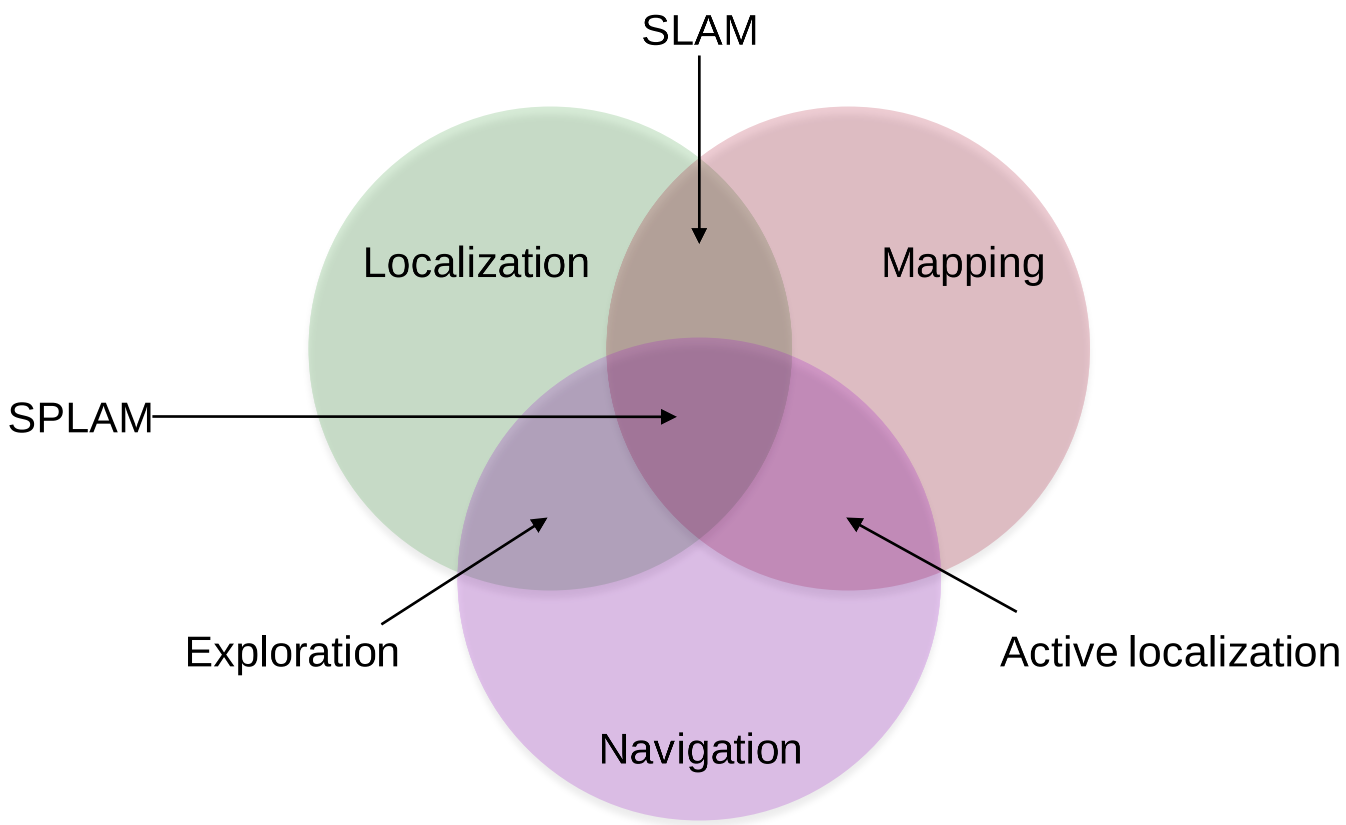
30th Jan	6th Feb	13th Feb	20th Feb	27th Feb	6th Mar	13th Mar	20th Mar	27th Mar	3rd Apr	10th Apr	17th Apr	24th Apr	1st May
Software architecture	Intro to ROS2	More on ROS2	Localization	Mapping	Localization wrt. maps	SLAM	Navigation	Easter 🐇	Exploration	Behavior Trees	Black Box Approaches	Group work	

Today

Outline for the next 7 weeks

- Our own autonomy stack:
  1. Localization
  2. Mapping
  3. Navigation
  4. Exploration
  5. Behaviour trees

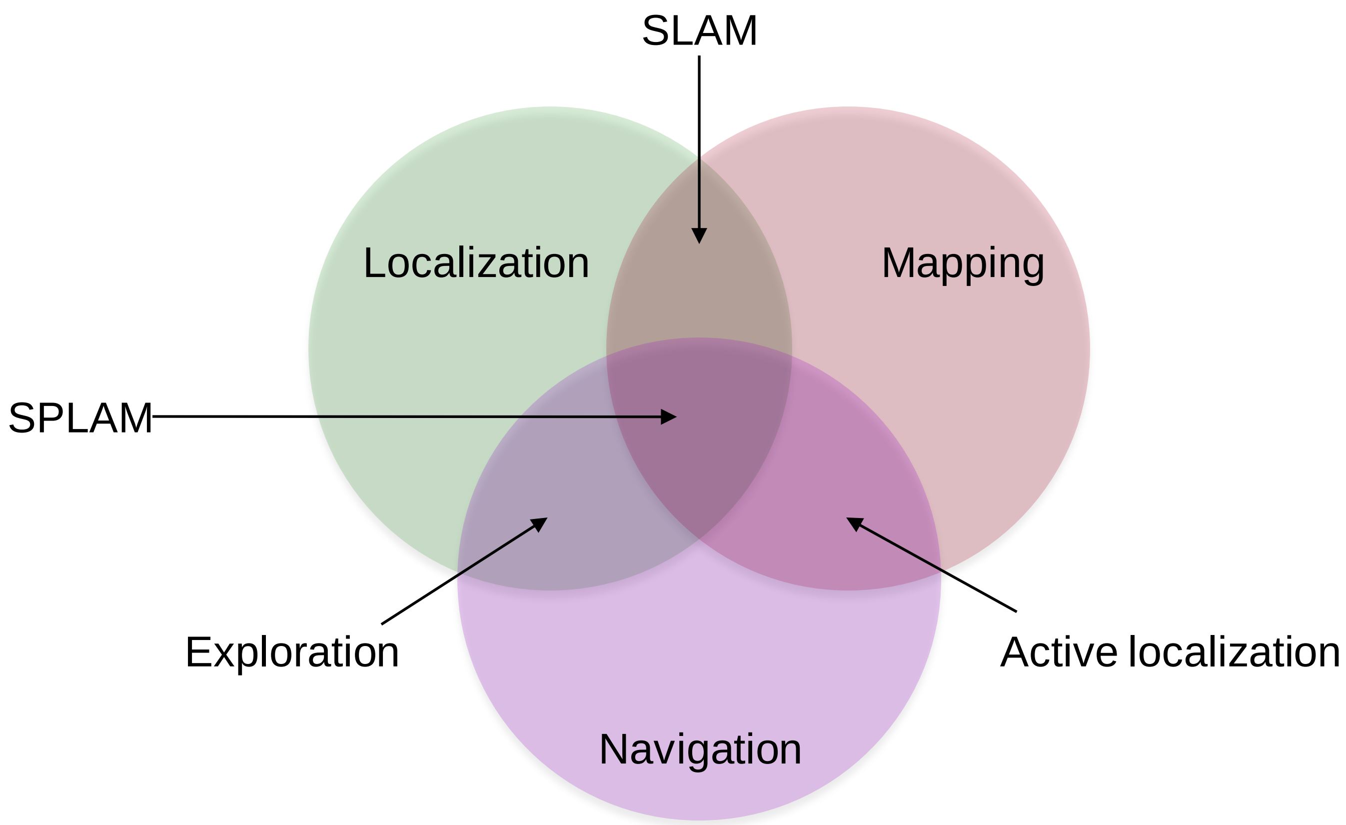
Topic of today



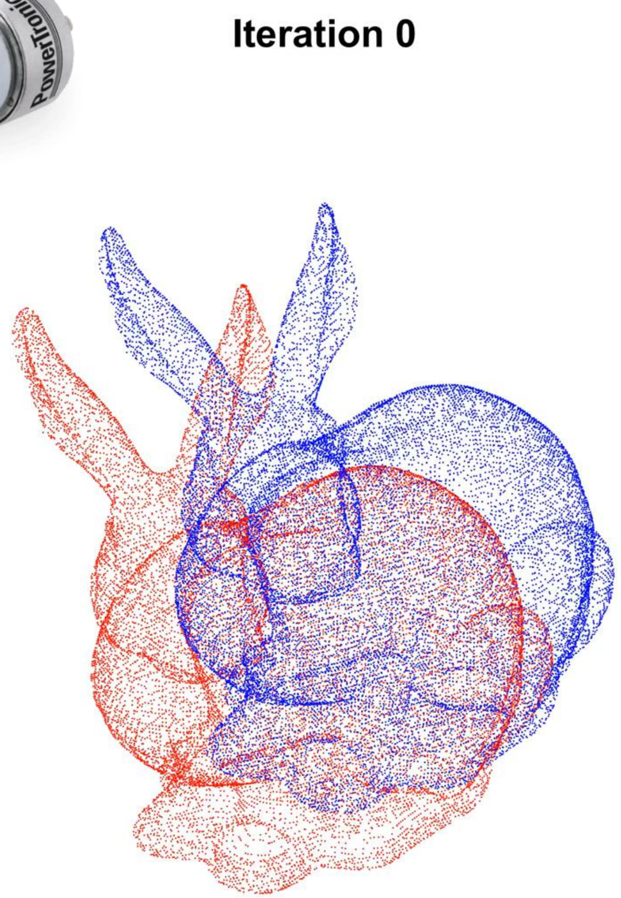
Outline for the next 7 weeks

- Our own autonomy stack:
  1. Localization
  2. Mapping
     - Map representations
     - Making a map
     - Challenges
  3. Navigation
  4. Exploration
  5. Behaviour trees

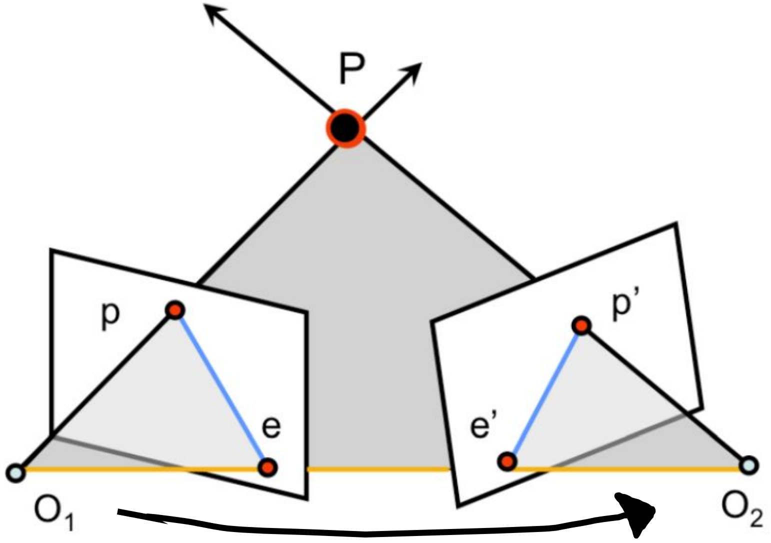
Topic of today



Recall from last lecture



- Wheel Odometry
- Three types of Visual Odometry – can you name them all?
  - 2D to 2D
  - 3D to 2D
  - 3D to 3D
- LIDAR Odometry
  - Iterative closest point
  - LIDAR Odometry



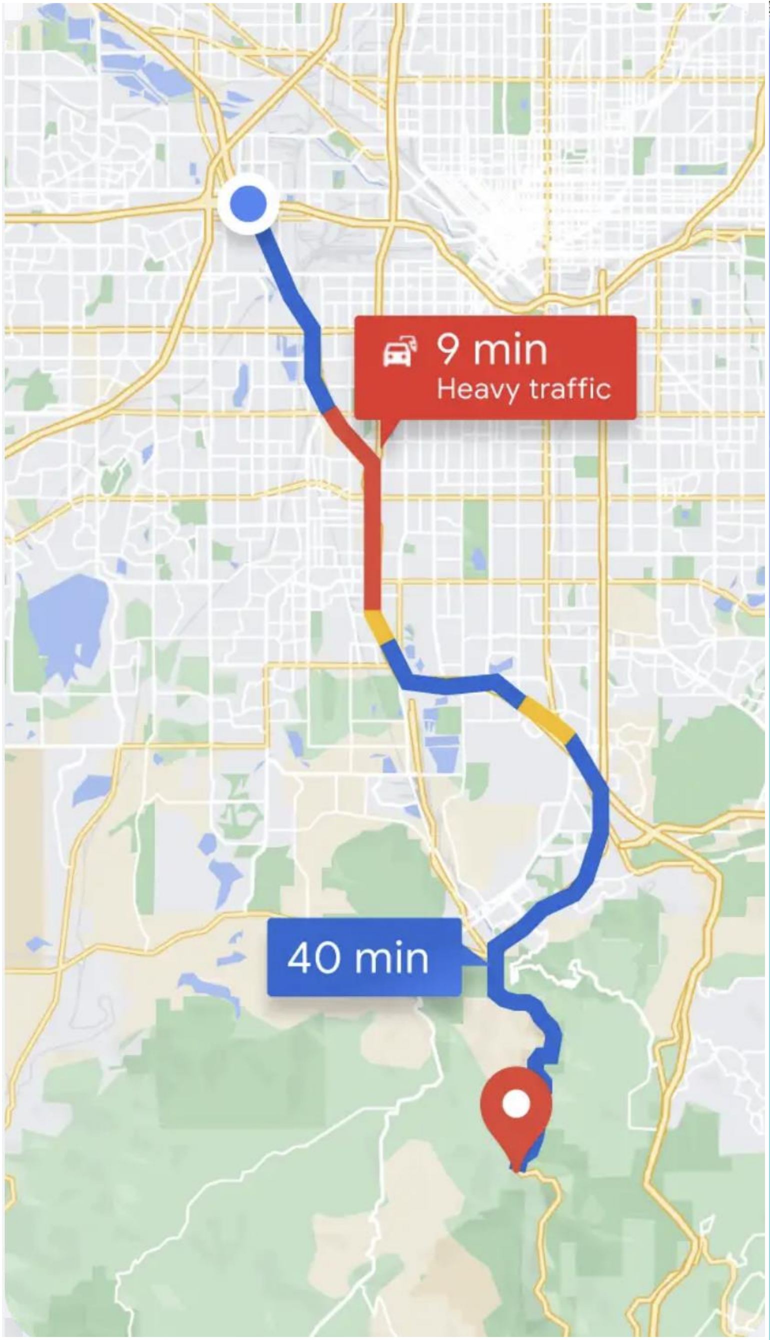
$$T_k = \begin{bmatrix} R_{k-1,k} & t_{k-1,k} \\ 0 & 1 \end{bmatrix}$$

Recall from last lecture

- Creating a localization ROS node for your turtlebot

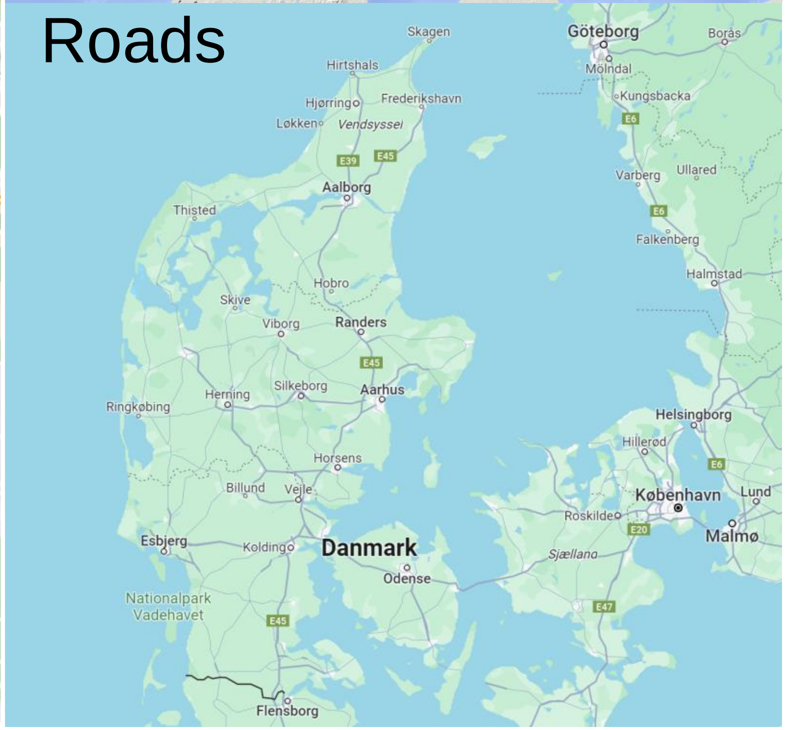
What is a map?

S-tog
Urban rail
Metro
Underground
Øvrige tog
Other rail services
Hillerød
Farum
Holte
Lyngby
Buddinge
Klampenborg
Frederikssund
Ballerup
Hellerup
Orientkaj
Østerport
Vanløse
Nørreport
Kongens Nytorv
Christianshavn
Vesterport
København H
Frederiksberg
Flintholm
Valby
Høje Taastrup
Sydhavn
Amagerbro
Lergravsparken
Øresund
Kastrup
Københavns Lufthavn CPH
Vestamager
København Syd
Hundige
Køge
ØRESUND
KØGE BUGT
København
Copenhagen



9 min
Heavy traffic
40 min

Elevation
Adobe Stock | #12792507



Roads
Skagen
Göteborg
Borås
Hirtshals
Mölndal
Hjørring
Frederikshavn
Kungsbacka
Løkken
Vendsyssel
Aalborg
Varberg
Ullared
Thisted
Falkenberg
Halmstad
Hobro
Skive
Viborg
Randers
Silkeborg
Herning
Aarhus
Ringkøbing
Helsingborg
Hillerød
Horsens
Billund
Vejle
København
Lund
Roskilde
Malmö
Esbjerg
Kolding
Danmark
Sjælland
Odense
Nationalpark Vadehavet
Flensborg

What can we use a map for?

- Internally on the robot:
  - Localization
    - Where is the robot
  - Path planning
    - How does the robot go from A to B without collisions
  - Navigation
    - How does the robot execute a path without collisions

- Externally
  - Recreation of otherwise inaccessible areas
  - Inspection – finding abnormalities or other deviations from your expectation

What are the challenges with a map?

- Updating dynamic objects (pedestrians, other robots, animals, etc.)
  - A map can contain both static and dynamic information

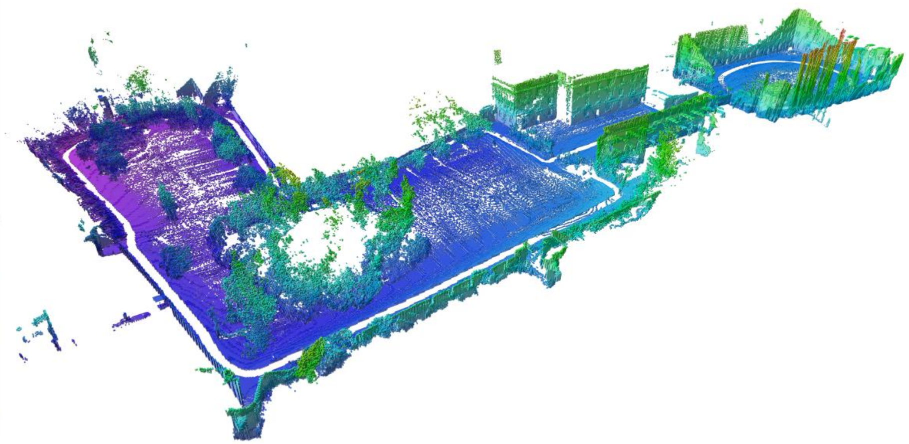
- Tends to grow exponentially in size with the area you want to map
  - They become very resource/memory demanding

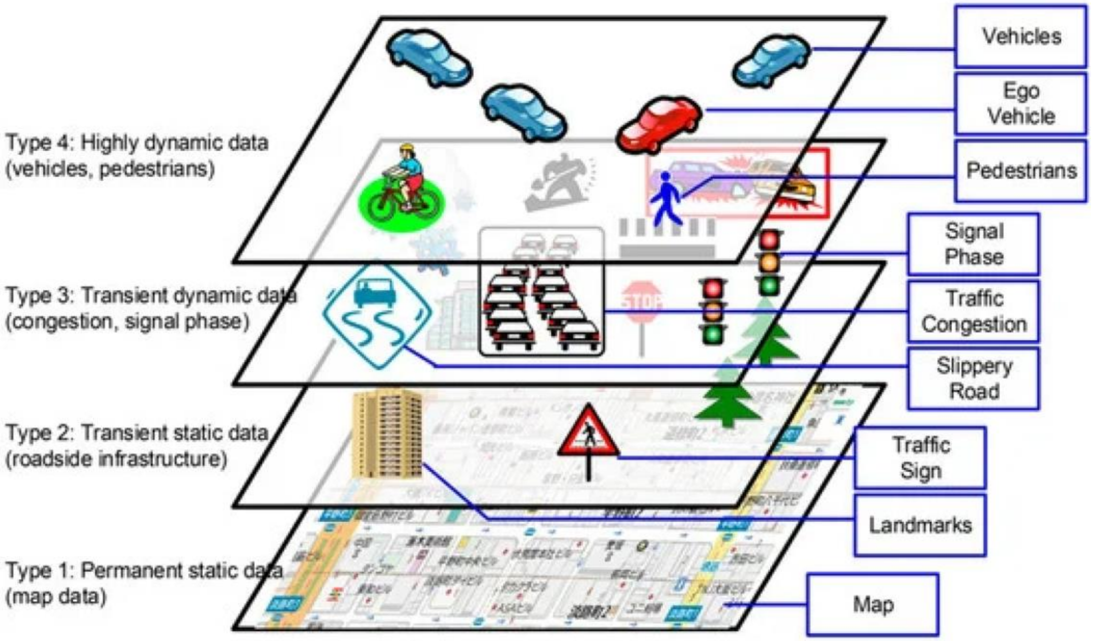
- They can be difficult to implement efficiently for large maps
  - We are updating our position w.r.t. the map multiple times per second, so getting local map information efficiently is critical to realtime operation

Map types

- Simplified map representations
  - – Topological maps
  - – Metric maps
  - – Geometric maps

- High-accuracy map representations
  - – Digital maps
  - – Enhanced digital maps
  - – High-definition maps



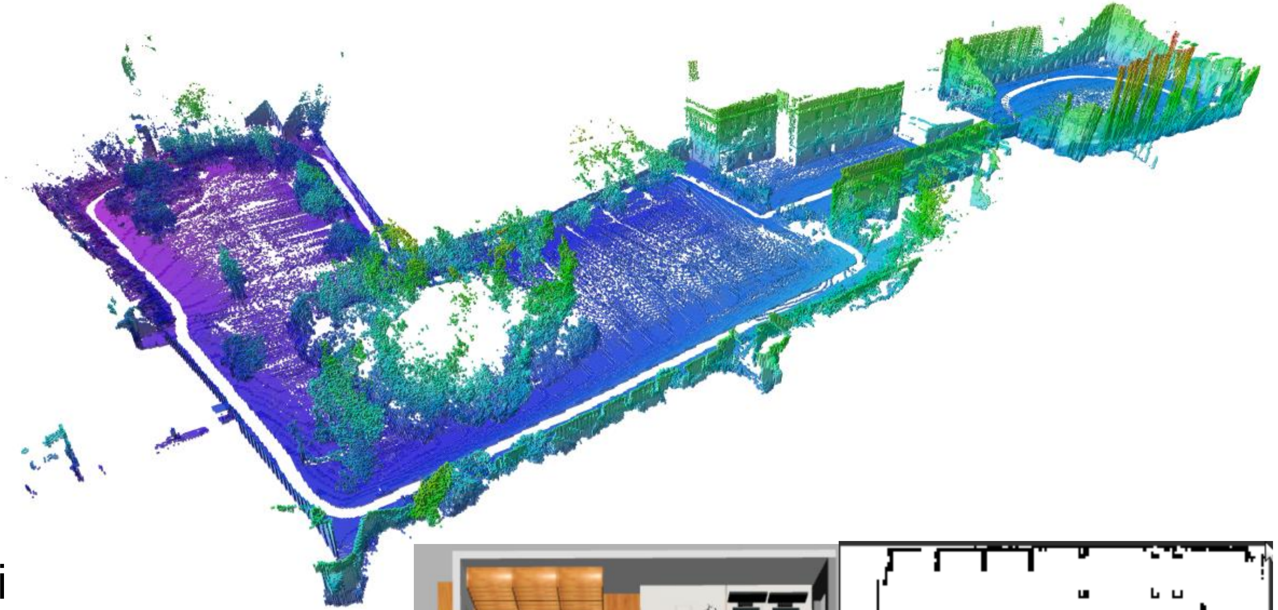


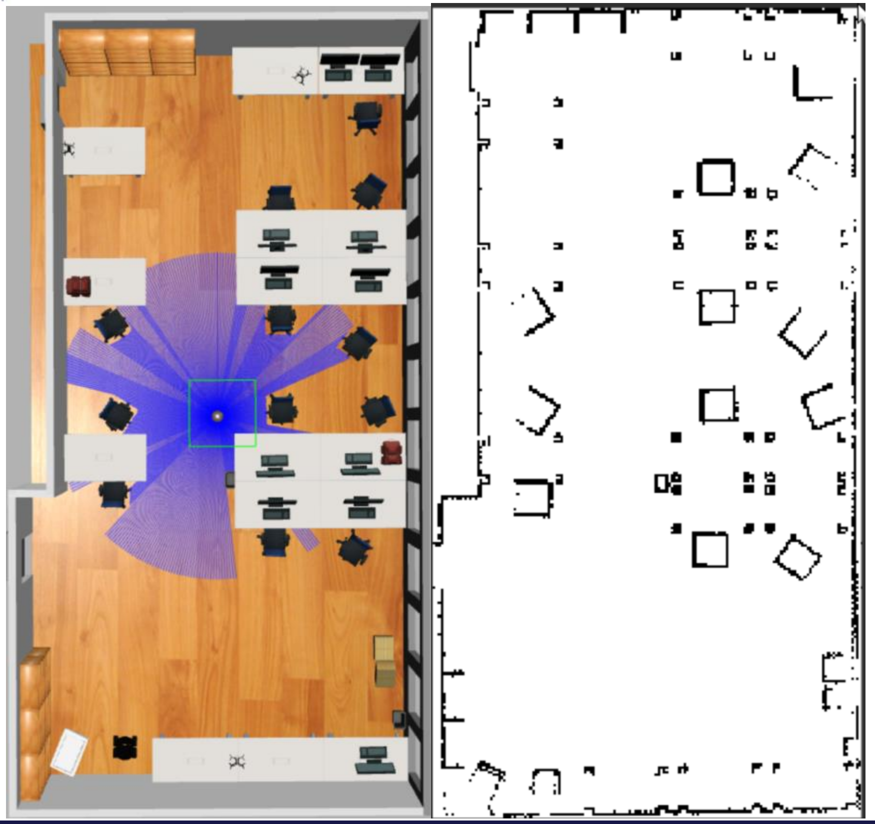
Map types

- Simplified map representations
  - **Topological maps**
    - Graph-based
    - Edges between nodes/landmarks
    - Edges can contain some information like transition probabilities
    - No sense of scale
    - Easy to do path planning in
  - Metric maps
  - Geometric maps

Map types

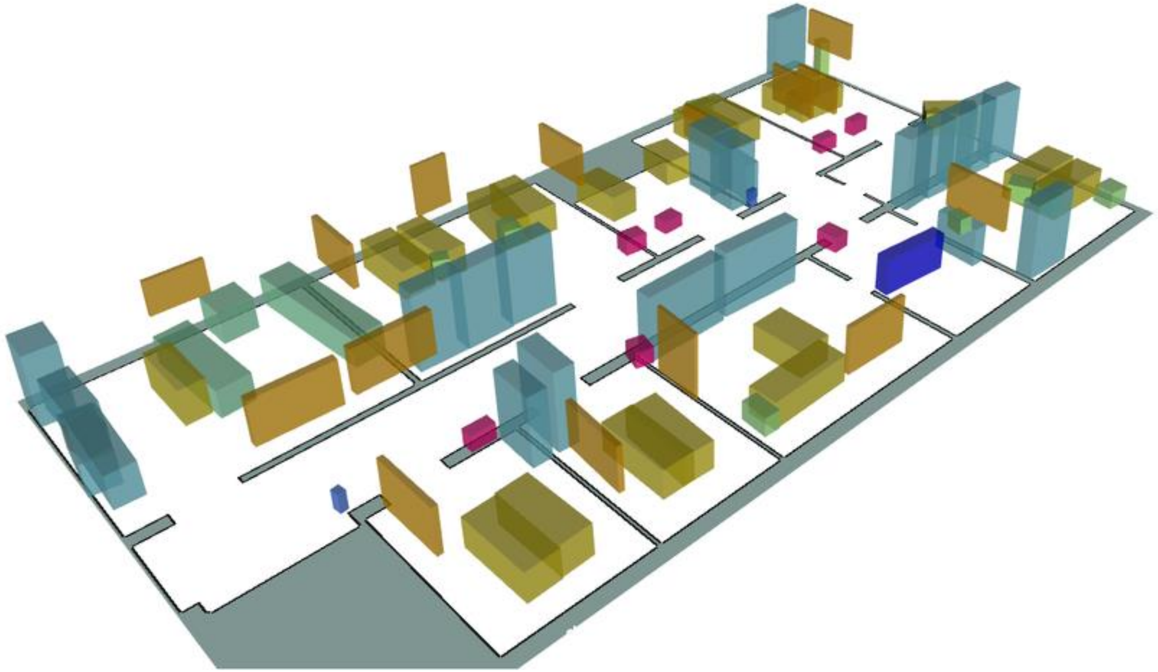
- Simplified map representations
  - Topological maps
  - **Metric maps**
    - Full 2D/3D representation of the envi
    - Contains all information to do path planning, navigation, mapping, etc.
    - Map size is directly proportional to the environment (computational heavy)
    - Landmark-based maps
      - Like topological maps, but here the landmarks are unique and incorporate scale
    - Occupancy grid maps
      - A grid of cells that contains occupancy information
  - Geometric maps





Map types

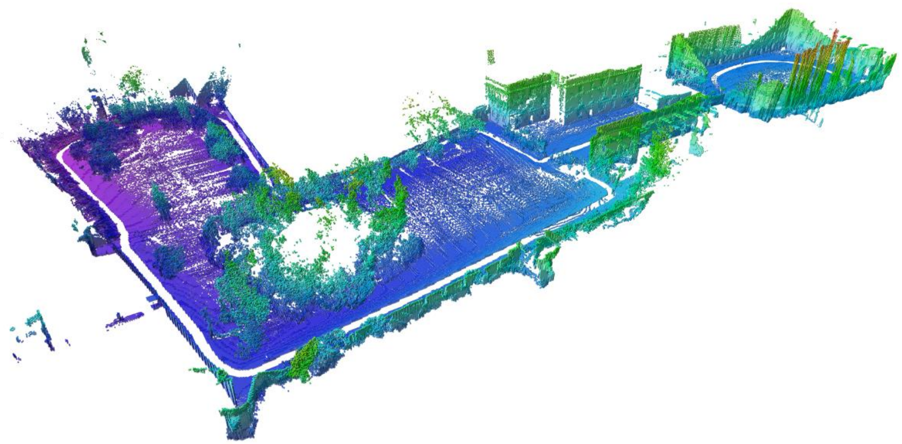
- Simplified map representations
  - Topological maps
  - Metric maps
  - **Geometric maps**
    - The environment is represented by geometric shapes instead of occupancy grids
    - Computationally efficient to generate
    - Computationally inefficient to use for path planning

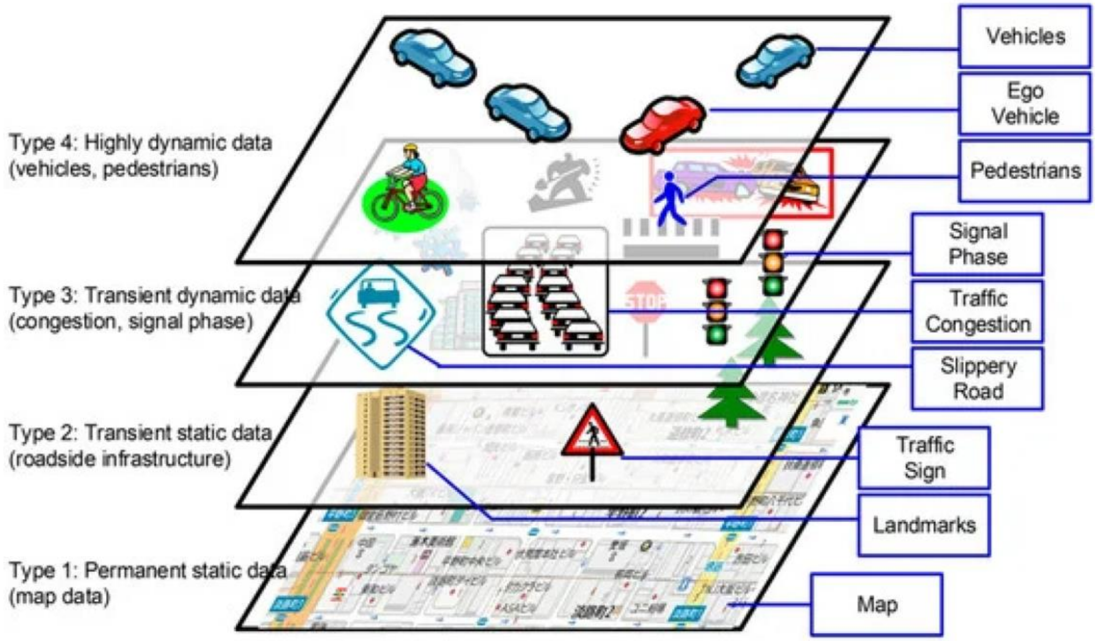


Map types

- Simplified map representations
  - Topological maps
  - Metric maps
  - Geometric maps

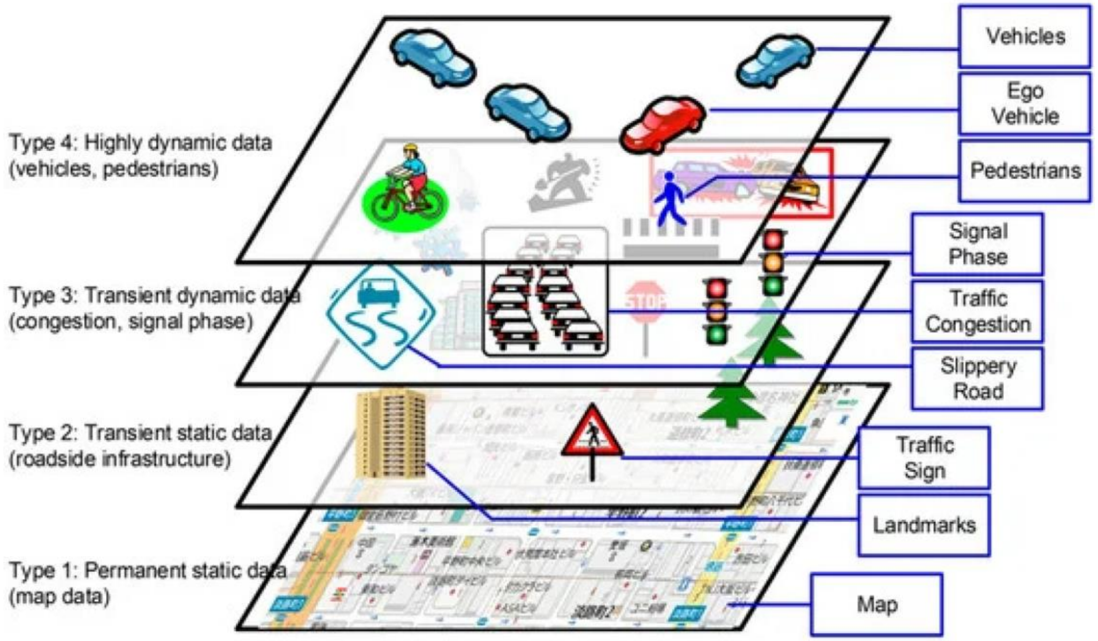
- **High-accuracy map representations**
  - Digital maps
  - Enhanced digital maps
  - High-definition maps





What information does an HD map contain?

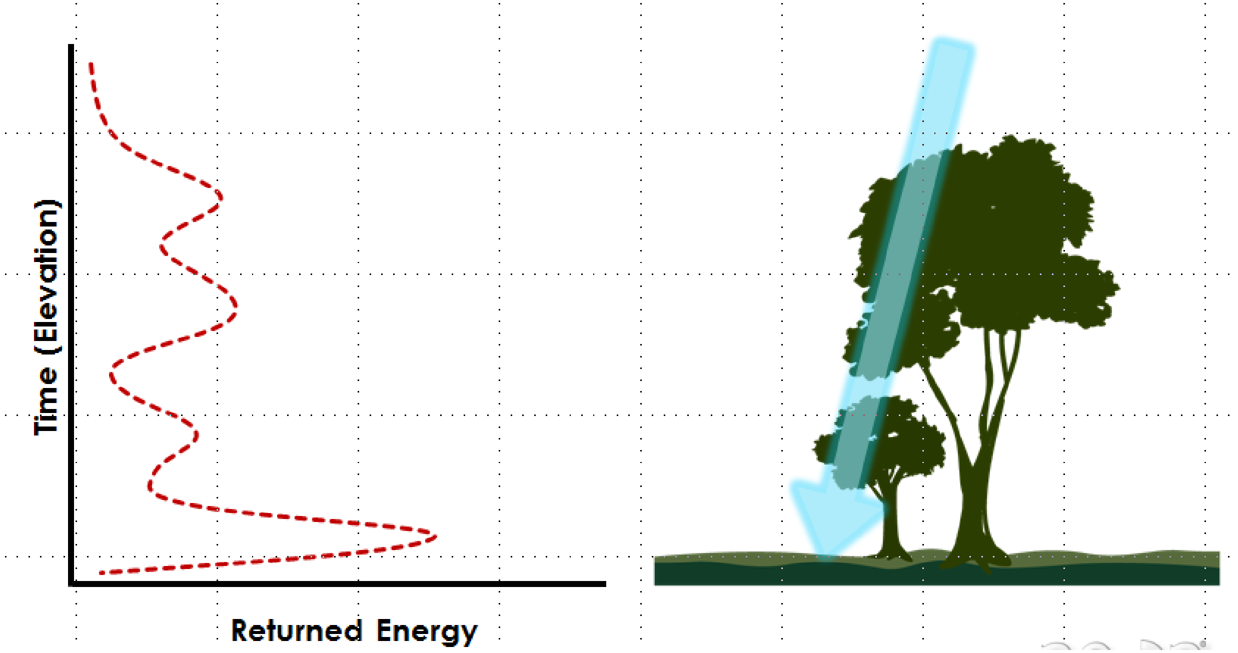
- Permanent static data
  - Everything that doesn’t change in the environment
  - Doesn’t indicate if we need to consider special conditions
- Transient static data
  - Gives us queues about the environment – things we may need to consider when moving
- Transient dynamic data
  - Dynamic data that changes at larger scales (Weather conditions, congestions, etc.)
  - Does not change quickly over time
- Highly transient dynamic data
  - Everything that moves dynamically (People, robots, animals)
  - Changes quickly over time



- At which level would our own robot be?

Using a LIDAR for mapping

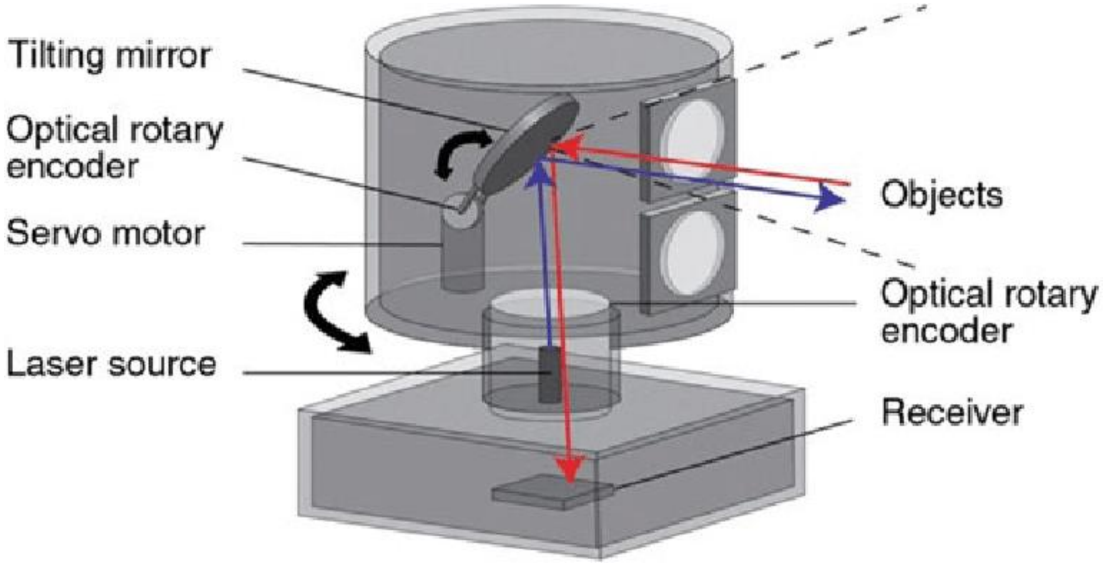
- The lidar ‘shoots’ a laser pointer out and measures the return signal
  - The stronger a return the more probable there is an object
  - Discrete vs waveform:
    - Discrete systems thresholds or selects the return to align with the modes of the wave
    - Waveform returns the hole signal
  - Thus, waveform systems require more processing but also provides more complete information
  - In robotics, discrete systems are often used for their implementation simplicity



Dealing with sensor noise

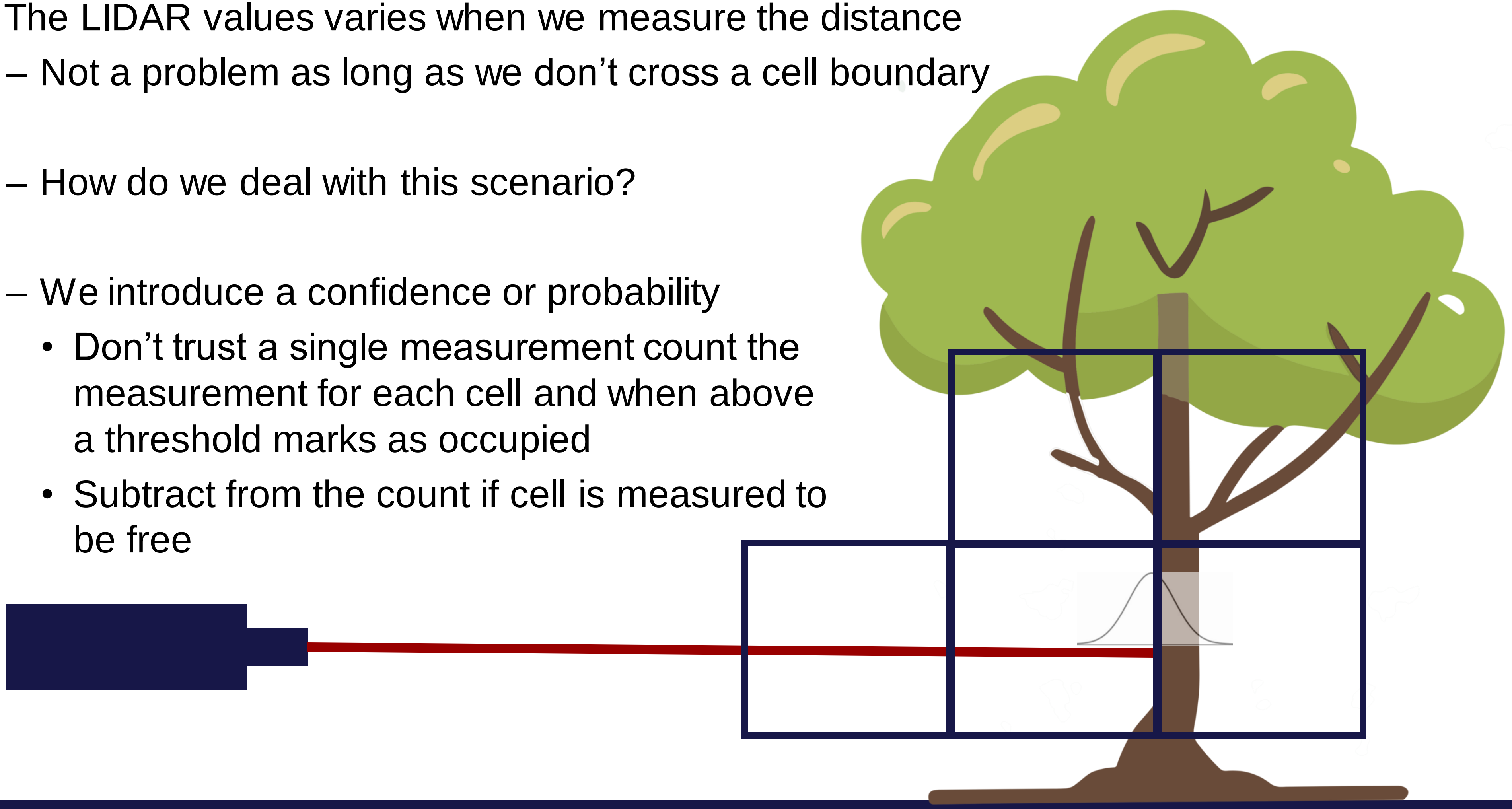
- There can be multiple reasons for a LIDAR measurement to exhibit ‘randomness’
  - The laser was shot at transparent or extremely reflective objects
  - LIDARS are often based on spinning mirrors, that can have imperfections
  - The LIDAR is subjected to vibrations
  - Shifting air temperature/humidity can alter the return signal of the laser

- Aleatoric uncertainty
  - Captures noise inherent in the observations, i.e. from our sensors



Dealing with sensor noise

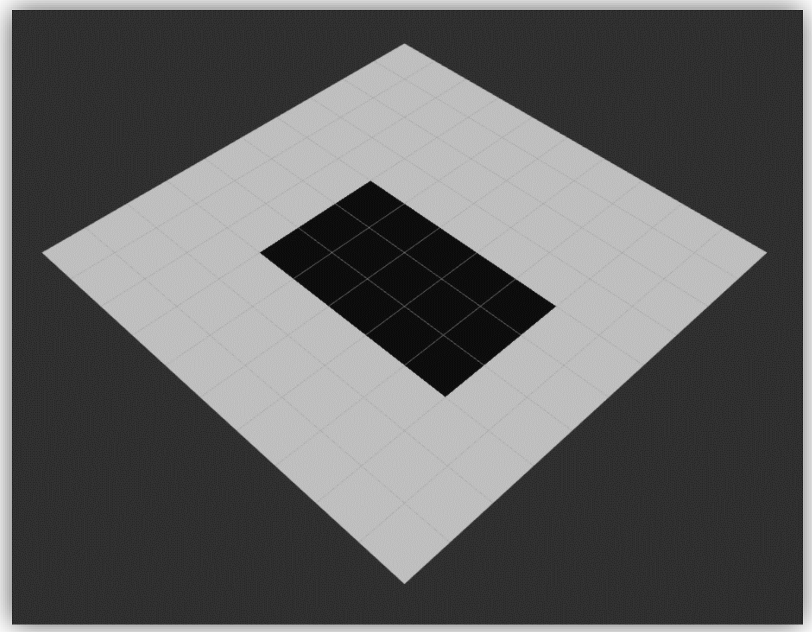
- The LIDAR values varies when we measure the distance
  - Not a problem as long as we don't cross a cell boundary
  - How do we deal with this scenario?
  - We introduce a confidence or probability
    - Don't trust a single measurement count the measurement for each cell and when above a threshold marks as occupied
    - Subtract from the count if cell is measured to be free



Representing an occupancy grid map

- What would be the simplest way to implement an occupancy grid?

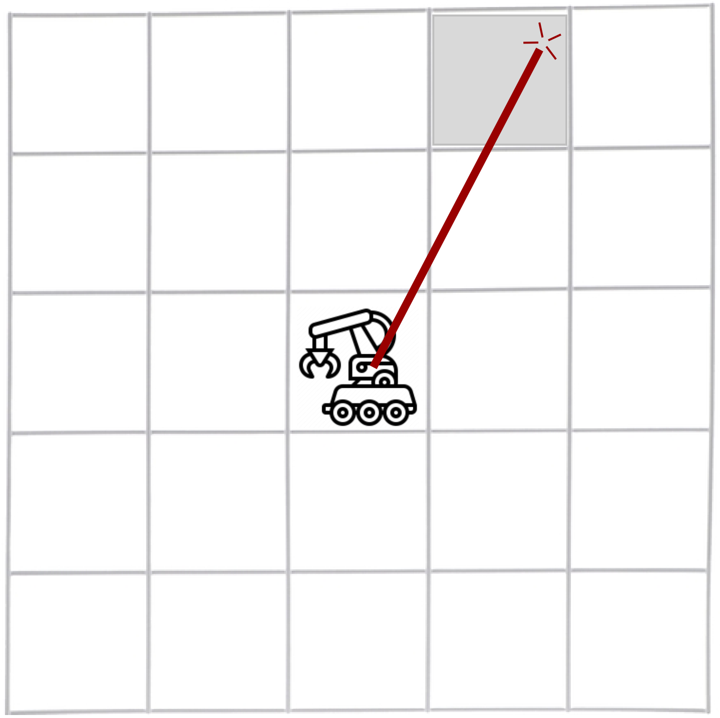
- Discretize the environment into cells/voxels
  - A 2D or 3D grid of binary values
    - 1 = Occupied cell
    - 0 = Free cell
  - Grid size and resolution is constant
  - We specify the origin of the map



- We need to define the information/interface for a map
  - ROS2 OccupancyGrid (https://docs.ros2.org/foxy/api/nav_msgs/msg/OccupancyGrid.html)
  - The data is just a reshaped vector of your 2D/3D grid

Using a LIDAR for mapping

- Discretizing a lidar scan
  - Convert the LIDAR scan to Euclidian coordinates and transform them to the map frame
  - The cell/voxel that the scan ‘ended’ in is marked as an obstacle



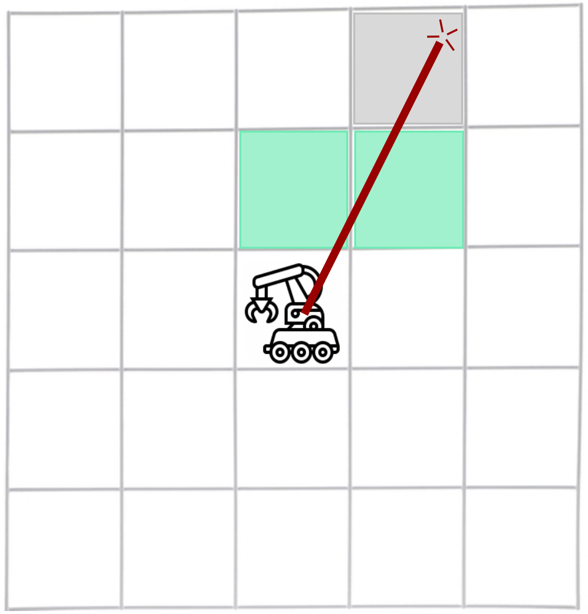
Occupied vs un-occupied

- It's only an assumption that the environment is free unless we have observed otherwise
  - When we perform exploration, we don't know the state of the cells/voxels
    - E.g. instead of only marking the voxels 0 or 1, -1 can mean "unknown" or "unexplored"

- What about the voxels that we observe indirectly
  - Can we say something about them?

  - $r(t) = \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix} * t$ where α is the angle and $0 < t \leq l$ is a scaling factor up to the length l returned by the LIDAR

  - Ray box intersections - check for intersections at all intermediate cells/voxels
    - Iteratively increase t with $\frac{\min(H,W)}{2}$ of grid cell size



Occupied vs un-occupied

- Use the fact you have a grid of perpendicular lines
  - Compute the intersections for all rays with each grid line
  - If there is an intersection, the ray passes through the cell/voxel

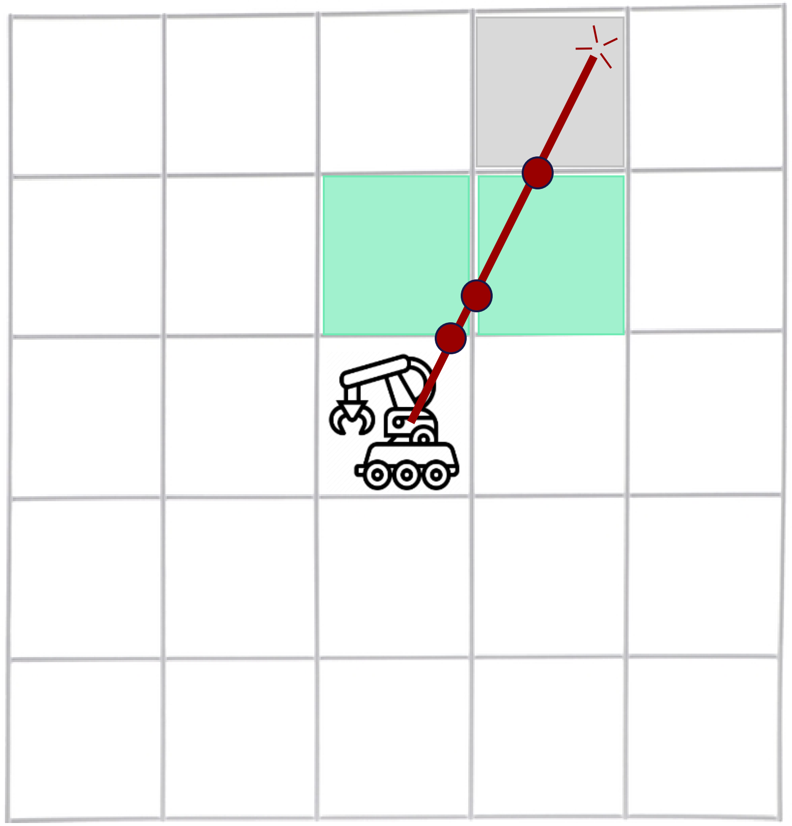
  - Equation of our ray: $r(t) = \boldsymbol{p} + t\boldsymbol{d}$
  - Equation of our line: $Ax + By + C = 0$
  - Compute the intersection by substitution:

$$A(p_x + td_x) + B\left(p_y + td_y\right) + C = 0$$

  - Solving for t

$$t = \frac{-Ad_x - Bd_y - C}{Ad_x + Bd_y}$$

  - Check the constraint $0 < t \leq l$, if upheld the cell is not occupied



- More advanced methods, like Bresenham's line algorithm can greatly increase performance

Using a ROS2 interface to publish maps

nav_msgs/msg/OccupancyGrid Message

File: nav_msgs/msg/OccupancyGrid.msg

Raw Message Definition

```
# This represents a 2-D grid map

std_msgs/Header header

# MetaData for the map

MapMetaData info

# The map data, in row-major order, starting with (0,0).

# Cell (1, 0) will be listed second, representing the next cell in the x
direction.

# Cell (0, 1) will be at the index equal to info.width, followed by (1, 1).

# The values inside are application dependent, but frequently,

# 0 represents unoccupied, 1 represents definitely occupied, and

# -1 represents unknown.

int8[] data
```

Compact Message Definition

```
std_msgs/msg/Header header
nav_msgs/msg/MapMetaData info
int8[] data
```

nav_msgs/msg/MapMetaData Message

File: nav_msgs/msg/MapMetaData.msg

Raw Message Definition

```
# This hold basic information about the characteristics of the OccupancyGrid


# The time at which the map was loaded

builtin_interfaces/Time map_load_time

# The map resolution [m/cell]

float32 resolution

# Map width [cells]

uint32 width

# Map height [cells]

uint32 height

# The origin of the map [m, m, rad]. This is the real-world pose of the

# bottom left corner of cell (0,0) in the map.

geometry_msgs/Pose origin
```

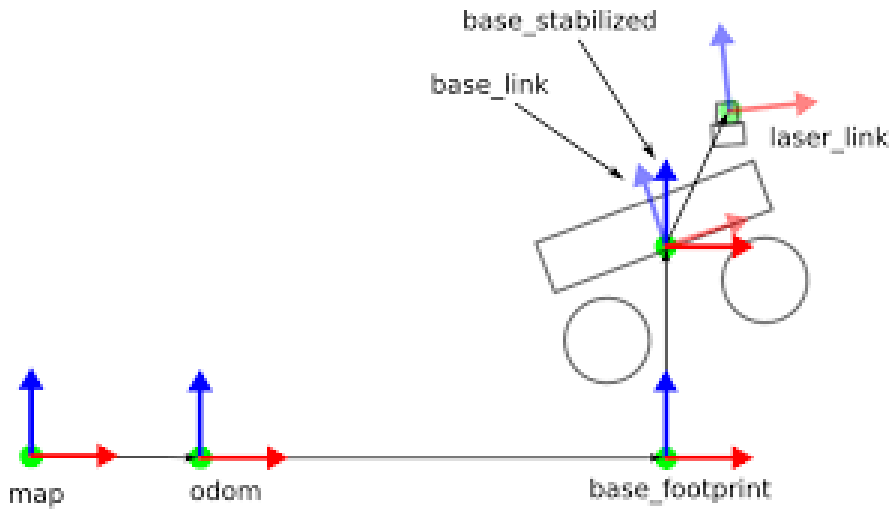
Compact Message Definition

```
builtin_interfaces/msg/Time map_load_time
float resolution
uint32 width
uint32 height
geometry_msgs/msg/Pose origin
```

Placing yourself in the map

- You need at least two transforms before you can publish the map
  - This can either be the (dynamic) location of your robot or a static frame which the robot moves with relation to

- To publish a static transform from the terminal run
  - *ros2 run tf2_ros static_transform_publisher x y z rx ry rz <from frame> <to frame>*
  - E.g. *ros2 run tf2_ros static_transform_publisher 0 0 0 0 0 0 map odom*



Exercises

- Publishing a map
  - Create a 2D occupancy grid using a 2D matrix
  - Fill random cells in the matrix with the value 100 and the rest with 0
  - Create an OccupancyGrid message and fill in the information (along with your map)
  - Publish the map on the topic /map with a frequency of 0.5Hz
  - Remember to add a transform that the map can live in (either static or dynamic)

- Overlaying laser scans
  - Create an empty 2D map
  - Subscribe to the LIDAR topic and convert the LIDAR scan to Euclidean coordinates
  - Add them to your internal map representation
  - Publish the updated map

- Moving the laser scan around in the map
  - Use the odometry you developed last time to move the pointcloud as the robot moves

- **You can use rviz to visualize the map**