

Pre-class assignment



<https://kaset.sart/art/dbEFWQ>

SLAM

Noppanut Thongton

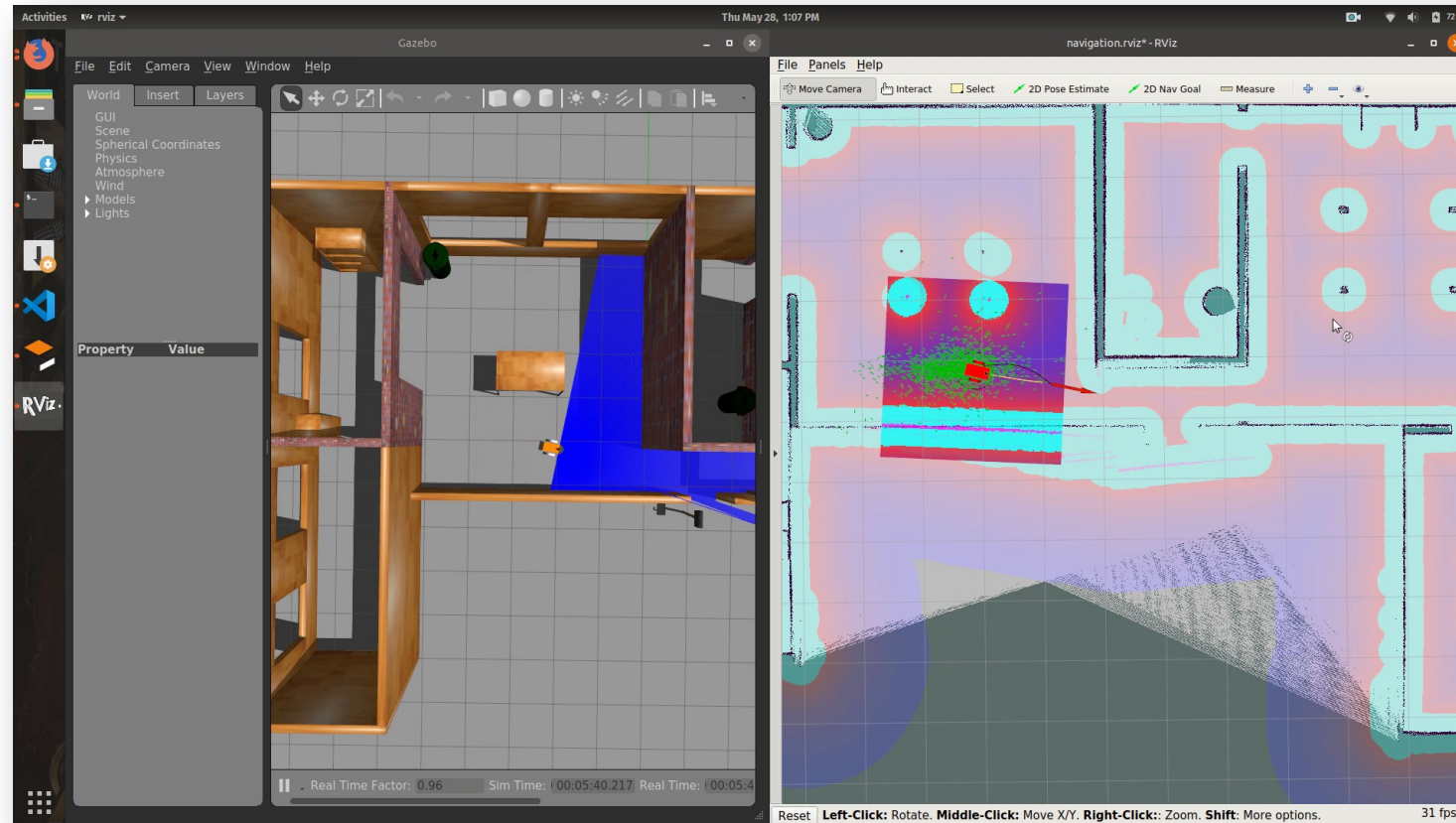
SLAM

Simultaneous localization and mapping (SLAM)

What is SLAM?

SLAM (simultaneous **localization** and **mapping**) is a method used for autonomous vehicles that lets you **build a map** and **localize your vehicle in that map** at the **same time**. SLAM algorithms allow the vehicle to map out unknown environments. Engineers use the map information to carry out tasks such as *path planning* and *obstacle avoidance*.

What is SLAM?



Why SLAM Matters?

SLAM has been the subject of technical research for many years. But with vast improvements in computer processing speed and the availability of low-cost sensors such as cameras and laser range finders, SLAM is now used for practical applications in a growing number of fields.

SLAM Examples

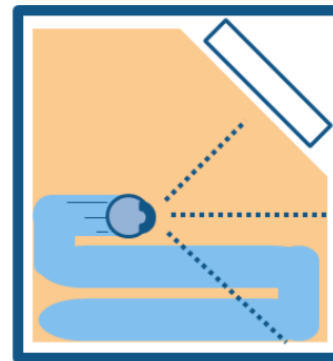
Consider a home robot vacuum. Without SLAM, it will just move **randomly** within a room and may not be able to clean the entire floor surface. In addition, this approach uses **excessive power**, so the battery will run out more quickly. On the other hand, robots with SLAM can use information such as the **number of wheel revolutions** and **data from cameras** and other imaging sensors to determine the amount of movement needed. This is called **localization**. The robot can also simultaneously use the camera and other sensors to ***create a map*** of the obstacles in its surroundings and ***avoid*** cleaning the same area twice. This is called **mapping**.

SLAM Examples

SLAM is useful in many other applications such as **navigating** a fleet of mobile robots to arrange shelves in a warehouse, **parking a self-driving car** in an *empty spot*, or **delivering** a package by navigating a drone in an *unknown environment*.



Without SLAM:
Cleaning a room randomly.

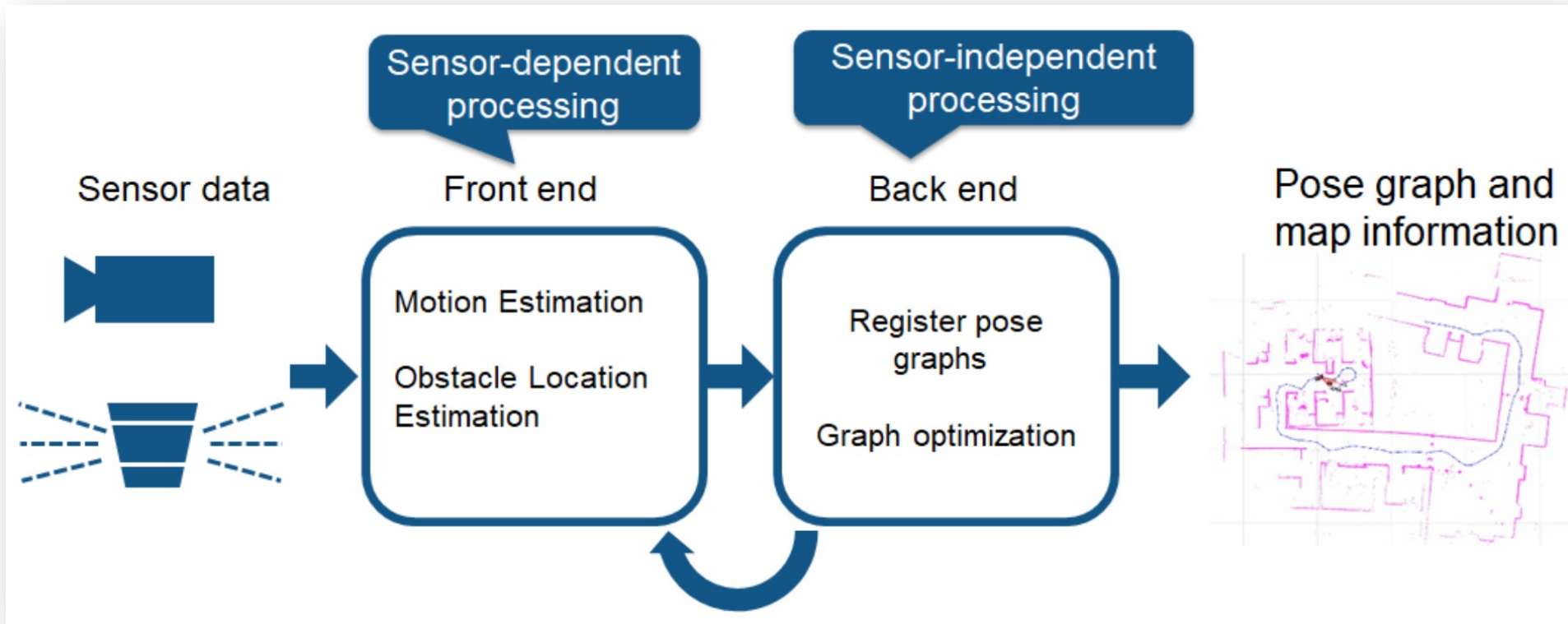


With SLAM:
Cleaning while understanding the room's layout.

How SLAM Works?

Broadly speaking, there are **two types** of technology components used to achieve SLAM. The first type is **sensor signal processing**, including the front-end processing, which is largely dependent on the sensors used. The second type is **pose-graph optimization**, including the back-end processing, which is sensor-agnostic.

How SLAM Works?



Visual SLAM

As the name suggests, **visual SLAM** (or vSLAM) uses images acquired from cameras and other image sensors. Visual SLAM can use **simple** cameras (wide angle, fish-eye, and spherical cameras), **compound eye** cameras (stereo and multi cameras), and **RGB-D** cameras (depth and ToF cameras).

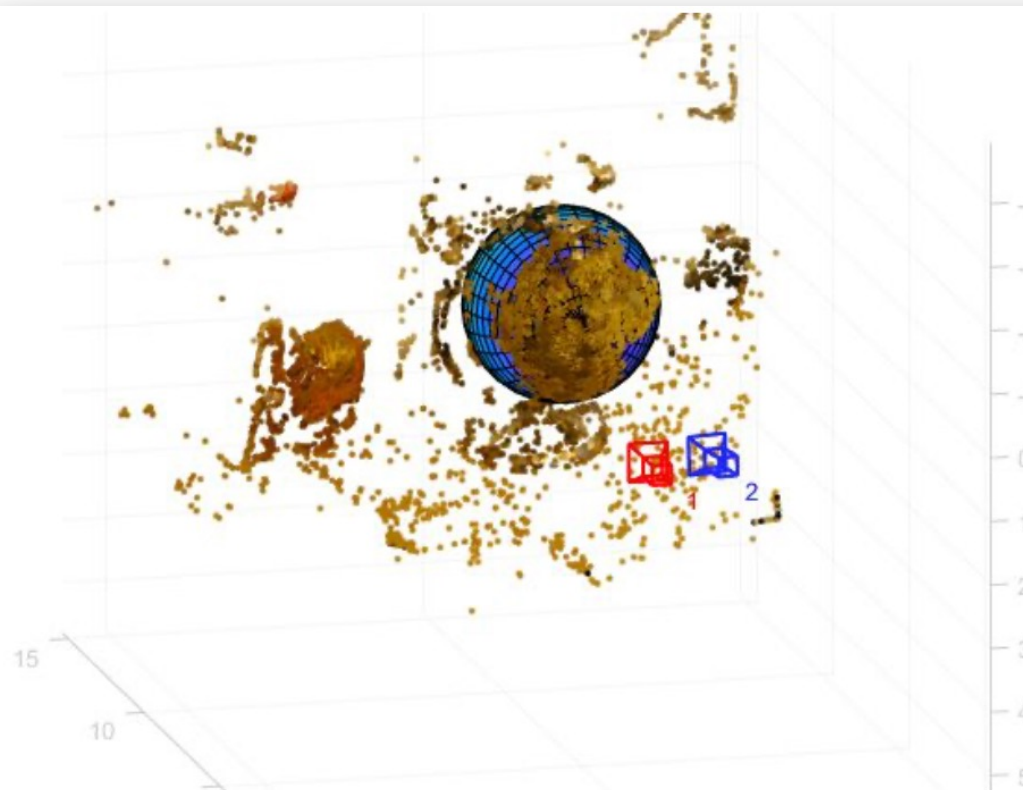
Visual SLAM can be implemented at **low cost** with relatively inexpensive cameras. In addition, since cameras provide a large volume of information, they can be used to detect **landmarks** (previously measured positions). Landmark detection can also be **combined** with graph-based optimization, achieving flexibility in SLAM implementation.

Visual SLAM

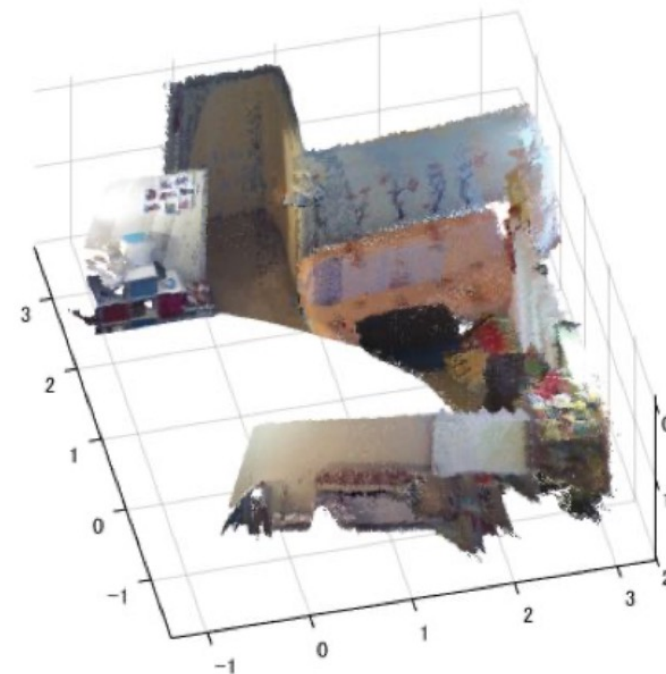
Monocular SLAM is when vSLAM uses a single camera as the only sensor, which makes it challenging to ***define depth***. This can be solved by either detecting AR markers, checkerboards, or other known objects in the image for localization or by fusing the camera information with another sensor such as Inertial Measurement Units (IMUs), which can measure physical quantities such as ***velocity*** and ***orientation***. Technology related to vSLAM includes structure from motion (SfM), visual odometry, and bundle adjustment.

Visual SLAM algorithms can be broadly classified into two categories. Sparse methods **match feature points of images** and use algorithms such as PTAM and ORB-SLAM. Dense methods use the overall brightness of images and use algorithms such as DTAM, LSD-SLAM, DSO, and SVO.

Visual SLAM



Structure from motion.



Point cloud registration for RGB-D SLAM

LiDAR SLAM

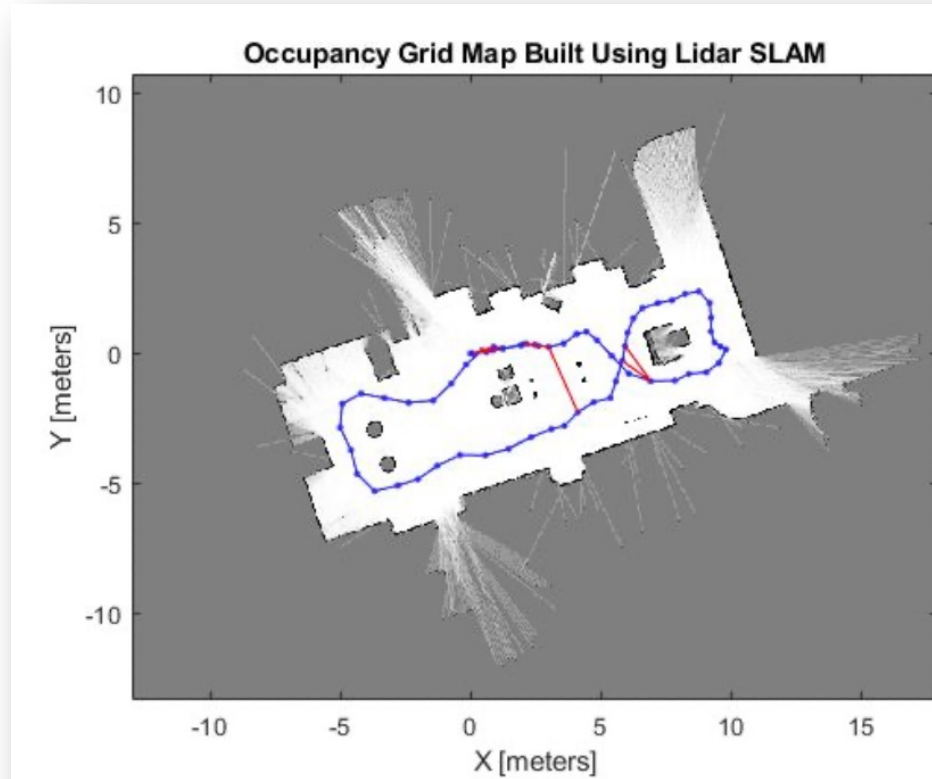
Light **D**etection **A**nd **R**anging (LiDAR) is a method that primarily uses a **laser sensor** (or distance sensor).

Compared to cameras, ToF, and other sensors, lasers are significantly more precise, and are used for applications with high-speed moving vehicles such as self-driving cars and drones. The output values from laser sensors are generally **2D** (x, y) or **3D** (x, y, z) point cloud data. The laser sensor point cloud provides high-precision distance measurements and works very effectively for map construction with SLAM. Generally, movement is estimated sequentially by matching the point clouds. The calculated movement (traveled distance) is used for localizing the vehicle. For lidar point cloud matching, registration algorithms such as **iterative closest point** (ICP) and **normal distributions transform** (NDT) algorithms are used. 2D or 3D point cloud maps can be represented as a ***grid map*** or ***voxel map***.

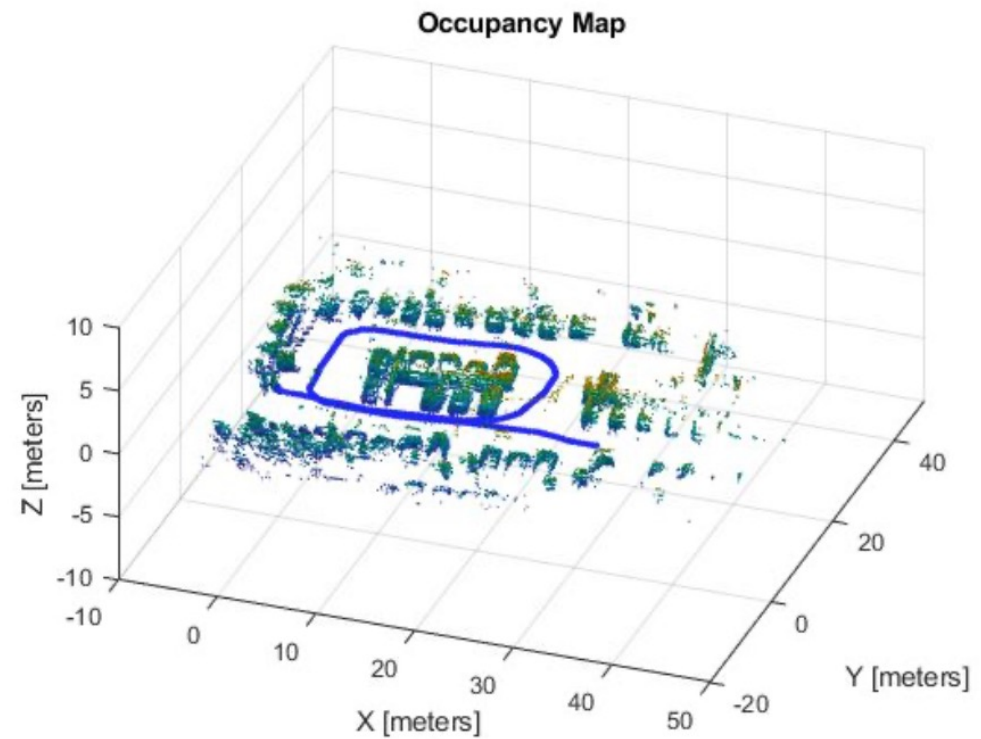
LiDAR SLAM

On the other hand, point clouds are **not** as finely detailed as images in terms of density and **do not** always provide sufficient features for matching. For example, in places where there are few obstacles, it is difficult to align the point clouds, and this may result in losing track of the vehicle location. In addition, point cloud matching generally requires high processing power, so it is necessary to optimize the processes to improve speed. Due to these challenges, localization for autonomous vehicles may involve **fusing** other measurement results such as wheel odometry, global navigation satellite system (GNSS), and IMU data. For applications such as warehouse robots, 2D lidar SLAM is commonly used, whereas SLAM using 3D lidar point clouds can be used for UAVs and automated driving.

LiDAR SLAM



SLAM with 2D LiDAR



SLAM with 3D LiDAR

Creating a Map [Known poses]

- Normally, mapping with known poses involves, given the measurements and the poses

$$d = \{x_1, z_1, x_2, z_2, \dots, x_t, z_t\}$$

- To calculate the most likely map

$$m^* = \operatorname{argmax}_m P(m \mid d)$$

Creating a Map [Known poses]

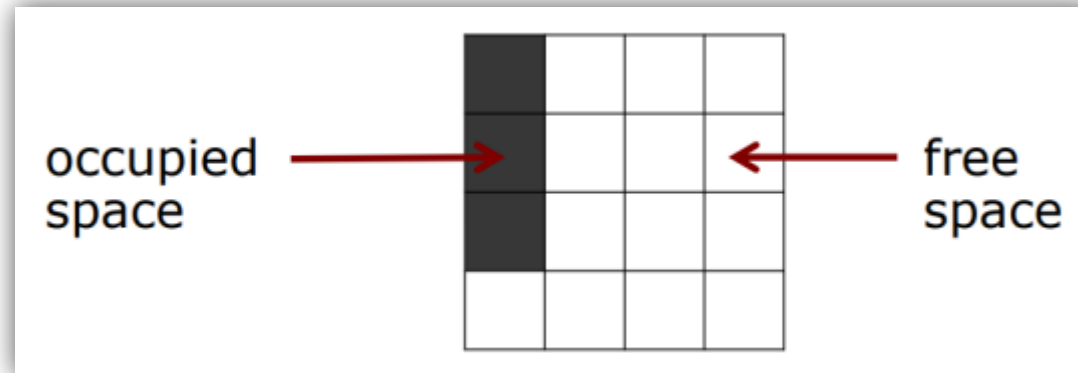
- **Grid Maps**

- We discretize the world into cells
 - The grid structure is rigid
 - Each cell is assumed to be occupied or free
 - It is a non-parametric model
 - It requires substantial memory resources
- It does not rely on a feature detector

Creating a Map [Known poses]

- **Assumption 1**

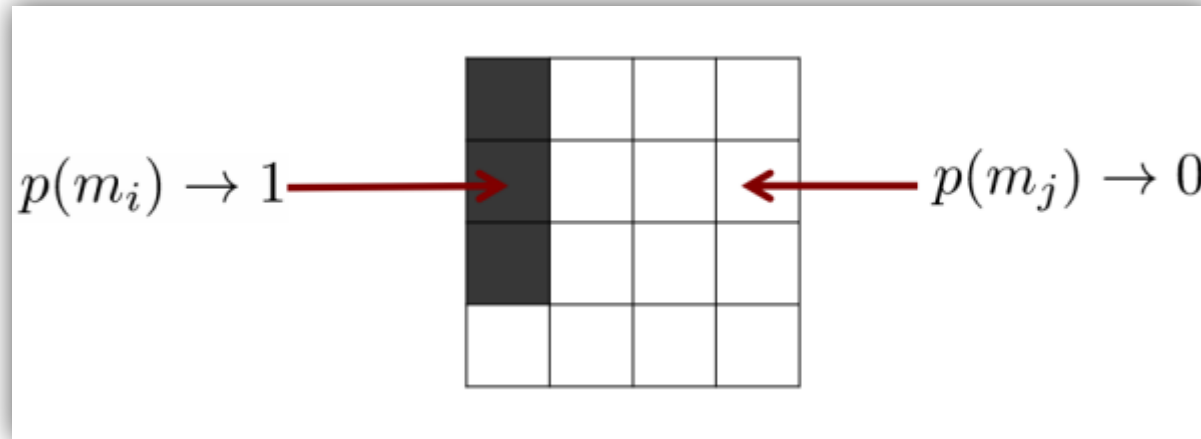
- The area that corresponds to a cell is either completely free or occupied



Creating a Map [Known poses]

- **Representation**

- Each cell is a binary random variable that models the occupancy



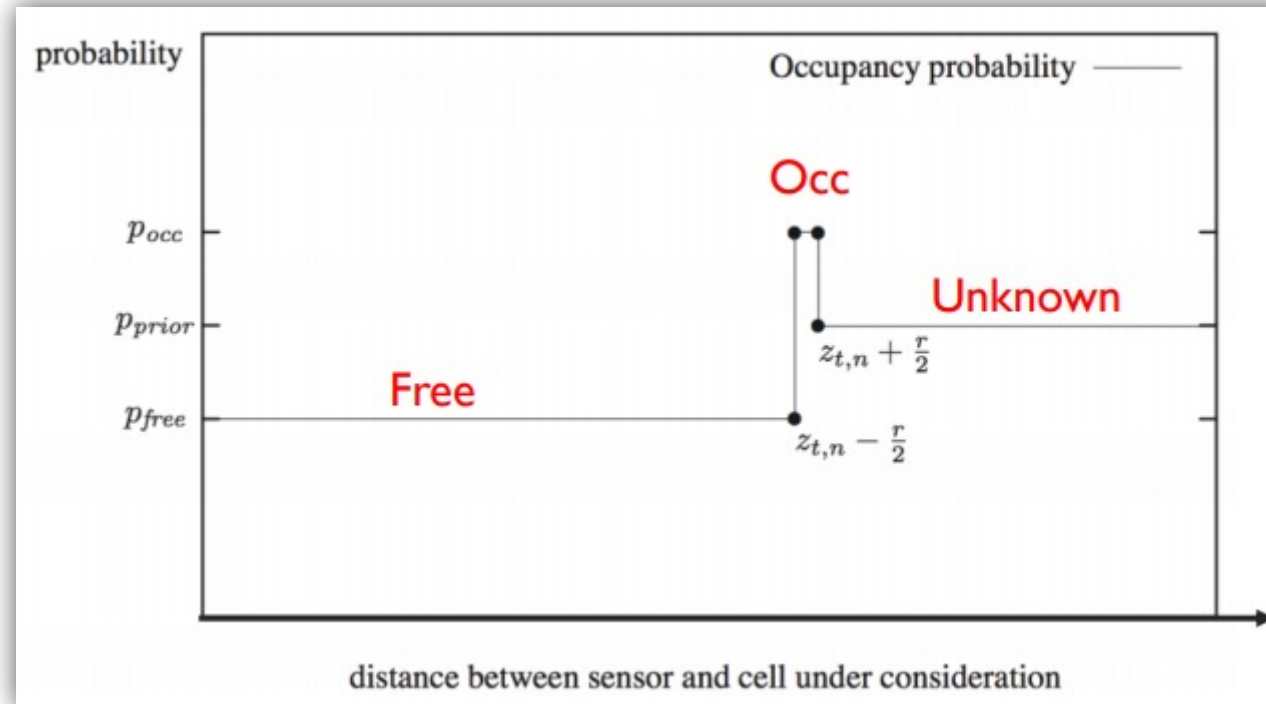
Creating a Map [Known poses]

- **Occupancy Probability**

- Each cell is a binary random
- variable that models the occupancy
- Cell is occupied $p(m_i) = 1$
- Cell is not occupied $p(m_i) = 0$
- No information $p(m_i) = 0.5$
- The environment is assumed to be static

Creating a Map [Known poses]

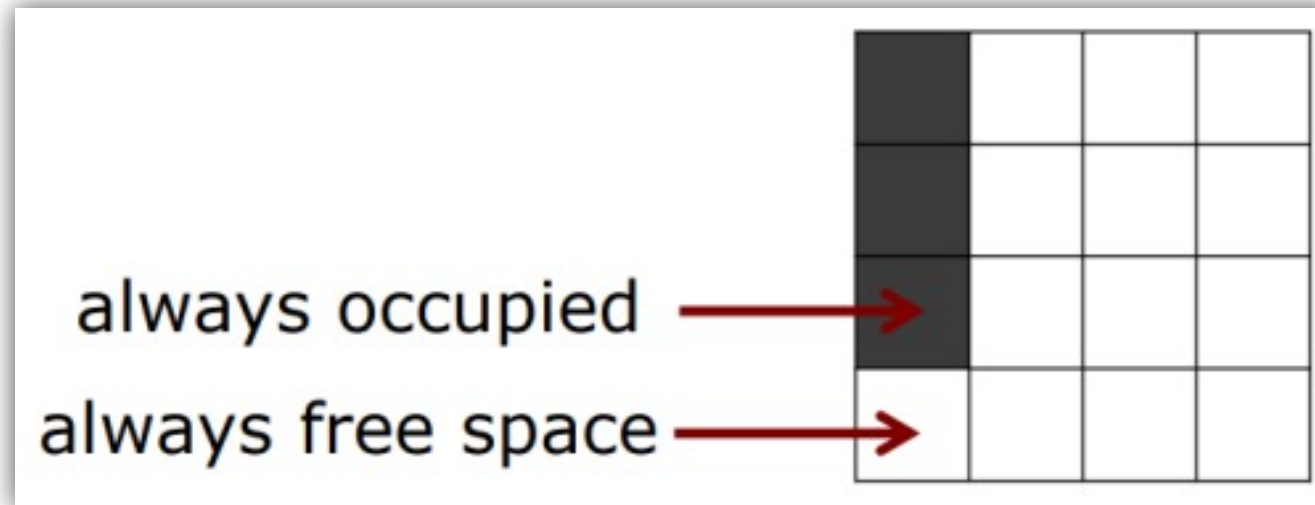
- Inverse Sensor Model for Laser Range Finders



Creating a Map [Known poses]

- **Assumption 2**

- The world is static (most mapping systems make this assumption)

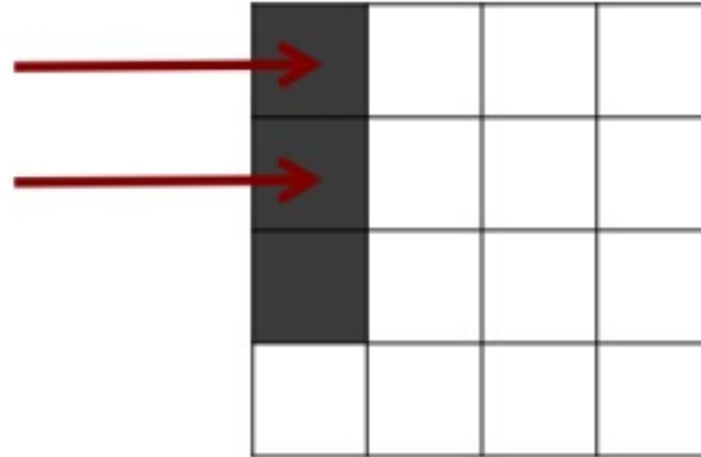


Creating a Map [Known poses]

- **Assumption 3**

- The cells (the random variables) are independent of each other

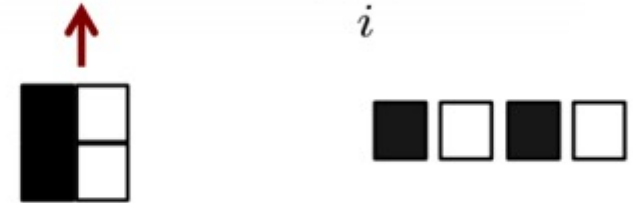
no dependency
between the cells



Creating a Map [Known poses]

- **Representation**

- The probability distribution of the map is given by the product over the cells

$$p(m) = \prod_i p(m_i)$$


The diagram shows the equation $p(m) = \prod_i p(m_i)$. Below the left side of the equation, there is a 2x2 grid of squares. The left column contains two black squares, and the right column contains two white squares. A red arrow points upwards from the top-left black square to the $p(m)$ term. Below this grid is the text "example map (4-dim state)". To the right of the equation, below the product symbol, there are four individual squares in a row: black, white, black, and white. Below this row of squares is the text "4 individual cells".

example map (4-dim state)

4 individual cells

Creating a Map [Known poses]

- **Estimating a Map From Data**

- Given sensor data $z_{1:t}$ and the poses $x_{1:t}$ of the sensor, estimate the map

$$p(m \mid z_{1:t}, x_{1:t}) = \prod_i p(m_i \mid z_{1:t}, x_{1:t})$$



binary random variable



Binary Bayes filter
(for a static state)

Creating a Map [Known poses]

- **Log Odds Notation**

- The log odds notation computes the logarithm of the ratio of probabilities

$$\frac{p(m_i | z_{1:t}, x_{1:t})}{1 - p(m_i | z_{1:t}, x_{1:t})} = \underbrace{\frac{p(m_i | z_t, x_t)}{1 - p(m_i | z_t, x_t)}}_{\text{uses } z_t} \underbrace{\frac{p(m_i | z_{1:t-1}, x_{1:t-1})}{1 - p(m_i | z_{1:t-1}, x_{1:t-1})}}_{\text{recursive term}} \underbrace{\frac{1 - p(m_i)}{p(m_i)}}_{\text{prior}}$$

$$\Rightarrow l(m_i | z_{1:t}, x_{1:t}) = \log \left(\frac{p(m_i | z_{1:t}, x_{1:t})}{1 - p(m_i | z_{1:t}, x_{1:t})} \right)$$

Creating a Map [Known poses]

- **Occupancy Mapping in Log Odds Form**
 - The product turns into a sum

$$\begin{aligned} l(m_i | z_{1:t}, x_{1:t}) \\ = \underbrace{l(m_i | z_t, x_t)}_{\text{inverse sensor model}} + \underbrace{l(m_i | z_{1:t-1}, x_{1:t-1})}_{\text{recursive term}} - \underbrace{l(m_i)}_{\text{prior}} \end{aligned}$$

- Or in short

$$l_{t,i} = \text{inv_sensor_model}(m_i, x_t, z_t) + l_{t-1,i} - l_0$$

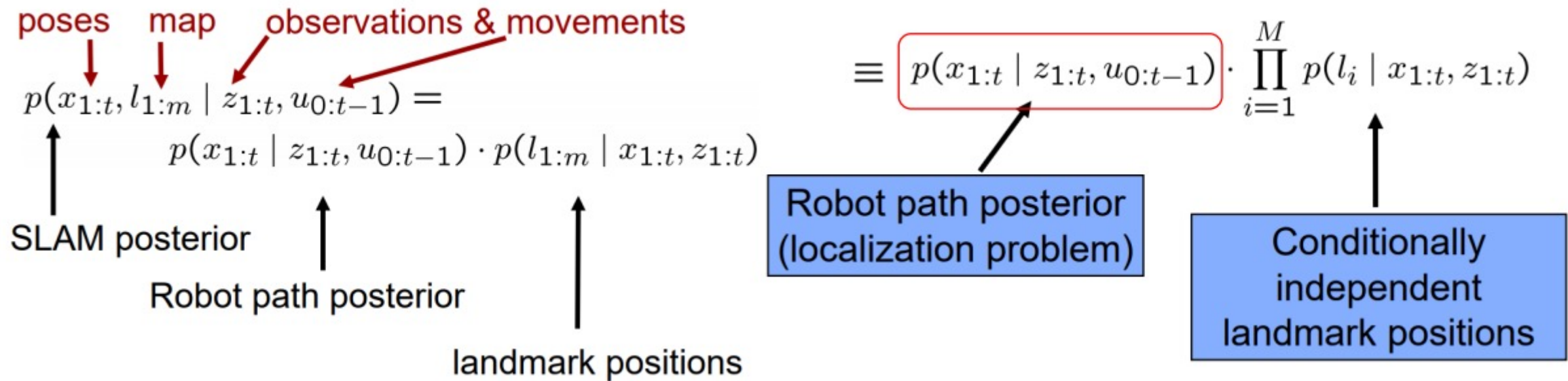
Log odds ratio is defined as

$$l(x) = \log \frac{p(x)}{1 - p(x)}$$

and with the ability to retrieve $p(x)$

$$p(x) = \frac{1}{1 + \exp(-l(x))}$$

Creating a Map [SLAM-GMAPPING]



Common Challenges with SLAM

Although SLAM is used for some practical applications, several technical challenges prevent more general-purpose adoption. Each has a countermeasure that can help overcome the obstacle.

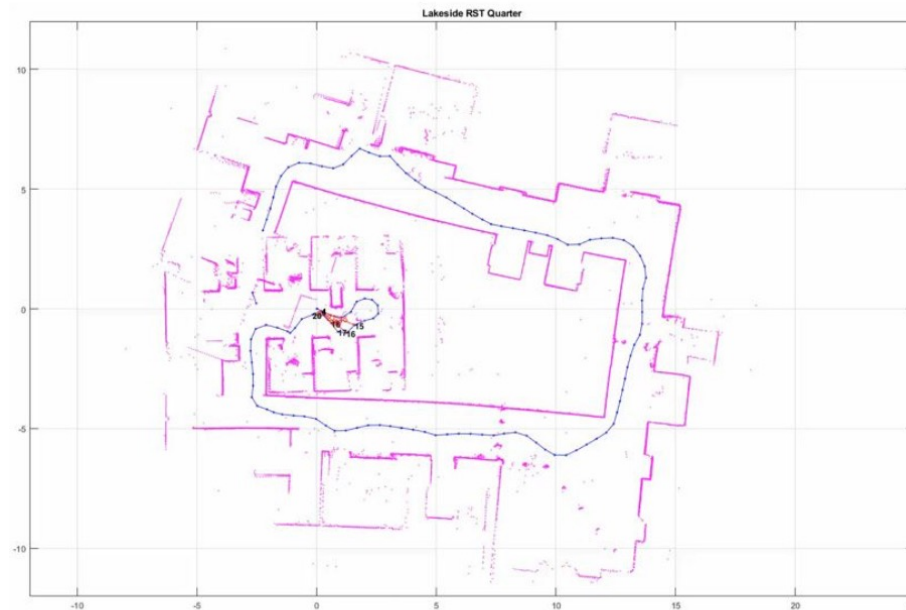
Common Challenges with SLAM

Localization errors accumulate, causing substantial deviation from actual values

SLAM estimates sequential movement, which include some **margin of error**. The error accumulates over time, causing substantial deviation from actual values. It can also cause map data to collapse or distort, making subsequent searches difficult. Let's take an example of driving around a **square-shaped** passage. As the error accumulates, robot's starting and ending point no longer match up. This is called a **loop closure problem**. Pose estimation errors like these are unavoidable. It is important to detect loop closures and determine how to correct or cancel out the accumulated error.

Common Challenges with SLAM

Localization errors accumulate, causing substantial deviation from actual values



Example of constructing a pose graph and minimizing errors.

Common Challenges with SLAM

Localization errors accumulate, causing substantial deviation from actual values

One **countermeasure** is to remember some characteristics from a previously visited place as a landmark and minimize the localization error. Pose graphs are constructed to help correct the errors. By solving error minimization as an optimization problem, more accurate map data can be generated. This kind of optimization is called bundle adjustment in visual SLAM.

Common Challenges with SLAM

Localization errors accumulate, causing substantial deviation from actual values



Example of constructing a pose graph and minimizing errors

Common Challenges with SLAM

Localization fails and the position on the map is lost

Image and point-cloud mapping **does not** consider the characteristics of a robot's movement. In some cases, this approach can generate discontinuous position estimates. For example, a calculation result showing that a robot moving at 1 m/s suddenly jumped forward by 10 meters. This kind of localization failure can be prevented either by using a recovery algorithm or by fusing the motion model with multiple sensors to make calculations based on the sensor data.

Common Challenges with SLAM

Localization fails and the position on the map is lost

There are several methods for using a motion model with sensor fusion. A common method is using **Kalman filtering** for *localization*. Since most differential drive robots and four-wheeled vehicles generally use **nonlinear motion models**, **extended Kalman filters** and **particle filters** (Monte Carlo localization) are often used. More flexible Bayes filters such as **unscented Kalman filters** can also be used in some cases. Some commonly used sensors are inertial measurement devices such as IMU, Attitude and Heading Reference System or AHRS, Inertial Navigation System or INS, accelerometer sensors, gyro sensors, and magnetic sensors. ***Wheel encoders attached to the vehicle are often used for odometry.***

Common Challenges with SLAM

Localization fails and the position on the map is lost

When localization fails, a **countermeasure** to recover is by **remembering a landmark** as a key-frame from a previously visited place. When searching for a landmark, a feature extraction process is applied in a way that it can scan at high speeds. Some methods based on image features include bag of features (BoF) and bag of visual words (BoVW). More recently, deep learning is used for comparison of distances from features.

Common Challenges with SLAM

High computational cost for image processing, point cloud processing, and optimization

Computing cost is a problem when implementing SLAM on a vehicle hardware. Computation is usually performed on compact and low-energy embedded microprocessors that have limited processing power. To achieve accurate localization, it is essential to execute image processing and point cloud matching at **high frequency**. In addition, optimization calculations such as loop closure are high computation processes. ***The challenge is how to execute such computationally expensive processing on embedded microcomputers.***

Common Challenges with SLAM

High computational cost for image processing, point cloud processing, and optimization

One **countermeasure** is to run different processes in ***parallel***. Processes such as **feature extraction**, which is preprocessing of the matching process, is relatively suitable for parallelization. Using multicore CPUs for processing, single instruction multiple data (SIMD) calculation, and embedded GPUs can further improve speeds in some cases. Also, since pose graph optimization can be performed over a relatively long cycle, lowering its priority and carrying out this process at regular intervals can also improve performance.