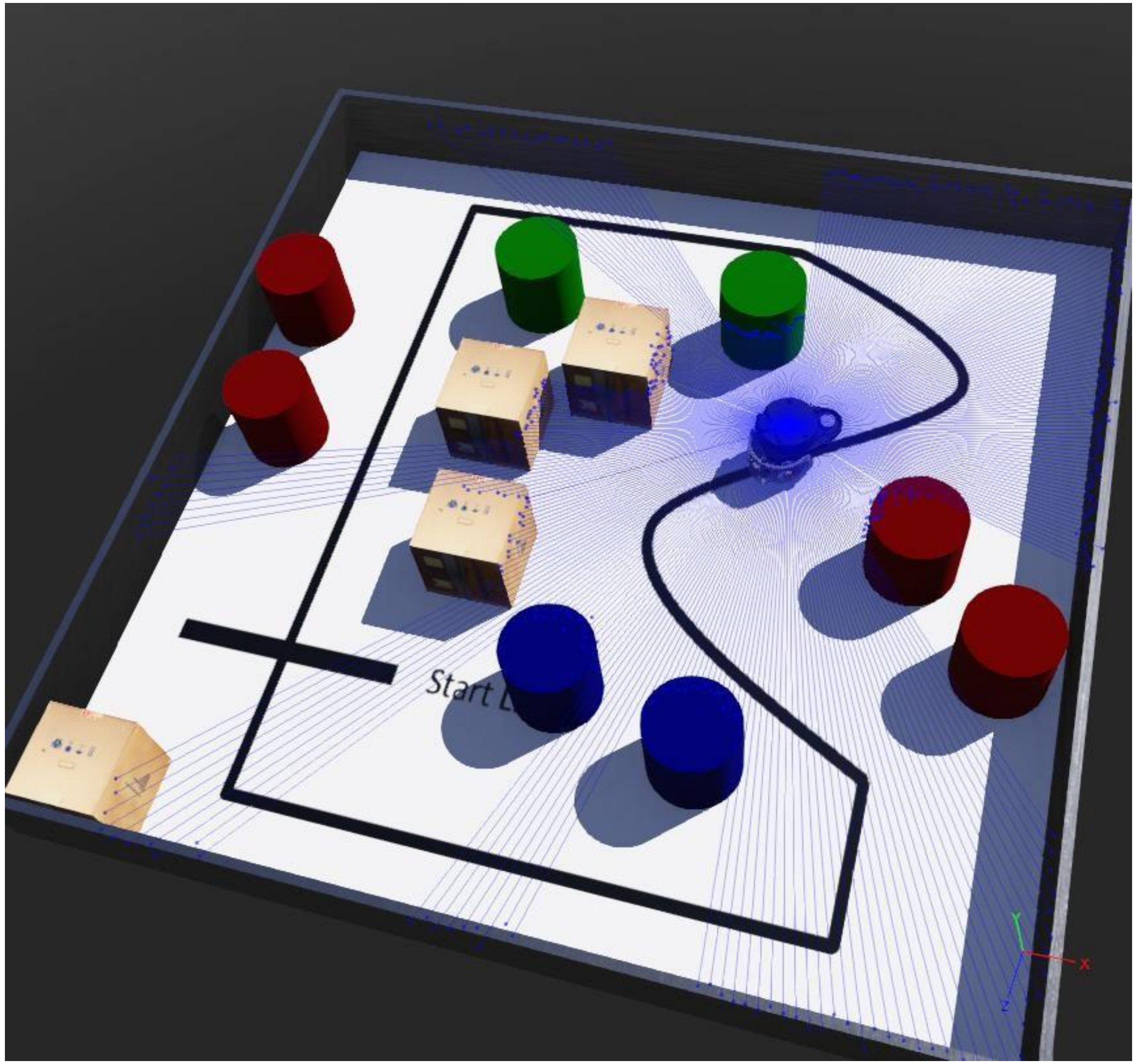
# Lab 4
## Mapping

# Lidar




$196

| | |
|---|---|
| **Distance Range** | 120 ~ 3,500mm |
| **Distance Accuracy (120mm ~ 499mm)** | ±15mm |
| **Distance Accuracy(500mm ~ 3,500mm)** | ±5.0% |
| **Distance Precision(120mm ~ 499mm)** | ±10mm |
| **Distance Precision(500mm ~ 3,500mm)** | ±3.5% |
| **Scan Rate** | 300±10 rpm |
| **Angular Range** | 360° |
| **Angular Resolution** | 1° |

# Learning an Environment

- With a sensor and some memory, a robot can generate a map of an environment

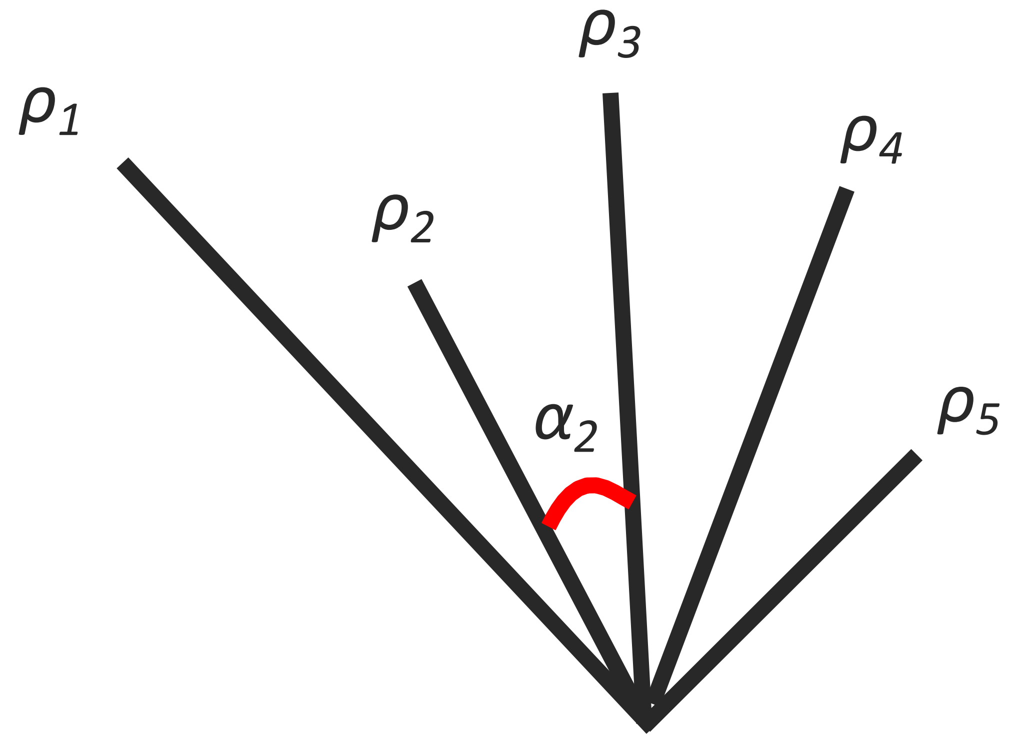
- What does a robot need to know to make a map?

# Learning an Environment

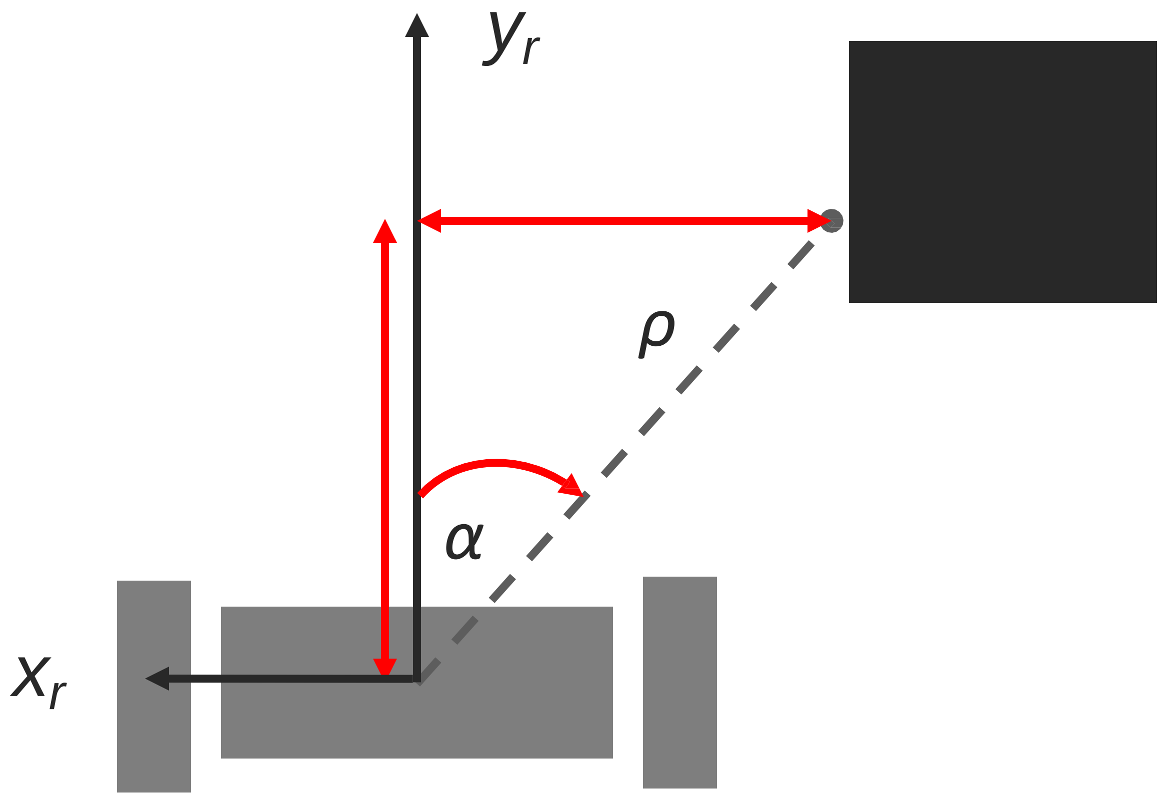
- **With a sensor and some memory, a robot can generate a map of an environment**

- **What does a robot need to know to make a map?**
  - Where obstacles are relative to the robot
    - LIDAR sensor readings!
  - Where the robot and obstacles are relative to some reference frame
    - Odometry and homogenous transforms

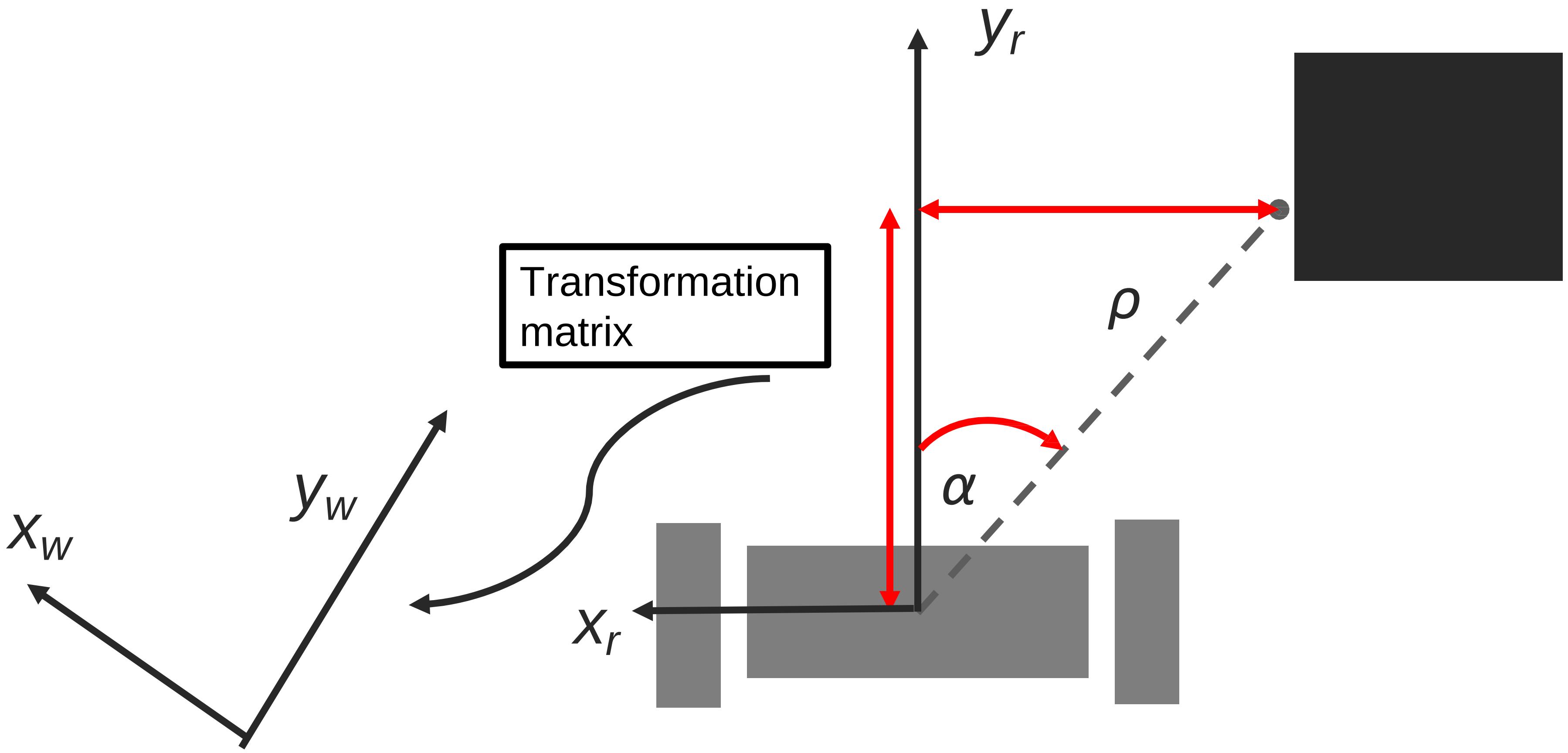
# LiDAR rays are discreet and at uniform interval



# Turn measurements into robot coordinates



# Turn robot coordinates into world coordinates

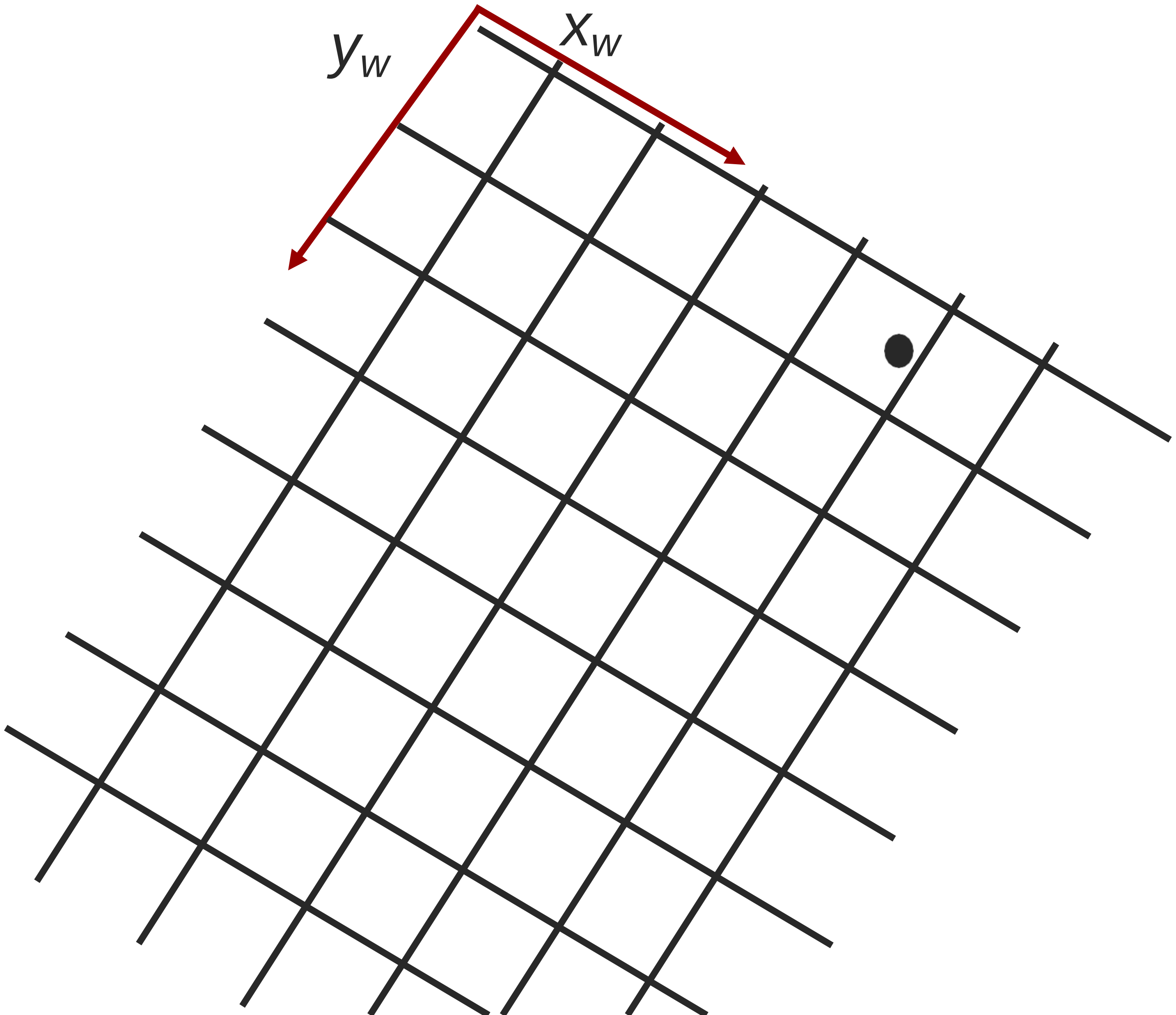


# Recall: Homogenous Transform

Instead of ${}^{A}Q = {}^{A}_{B}R * {}^{B}Q + {}^{A}P$, we can express the transformation as a single matrix multiplication

$$\begin{bmatrix} {}^{A}Q \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} & {}^{A}_{B}R & & {}^{A}P \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} {}^{B}Q \\ 1 \end{bmatrix}$$

# Turn into world coordinates into map pixels



# Representing the Map

- Once we know where obstacles are, we need to **store** them in a representation that allows for planning around them.
- For Lab 4, we'll implement a grid representation as a 2D array of Boolean values. The display can do the job for this **grid** in this lab's case.
  - For row j and column i, **grid[j][i] = 1** if occupied, **0** if free space
- Since we're using a grid, each cell will represent a region of the world space instead of a single point.
  - If any obstacle is within the cell, we mark it as occupied.
  - To figure out where the robot is, we will design and use functions that map world **pose coordinates (x,y) to grid coordinates (i,j)**.

# Algorithm

1. Calculate robot coordinates for each ray that is not ***inf***
2. Turn robot coordinates into world coordinates
3. Draw obstacle (pixel) and free space onto the map
4. Draw robot location onto the map
5. Move forward

- **This is a 1-week lab. Due Next Friday evening.**
- **Hints:**
  - View -> Optional Rendering -> Show Lidar point cloud
  - pip3 install numpy


Questions?