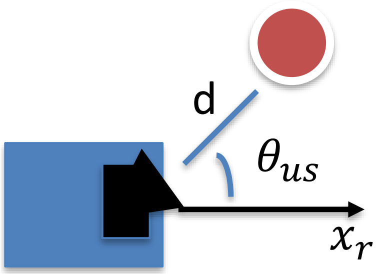
Making a Map

- When the robot sees an object, it needs to localize it within its own reference frame:



The ultrasonic sensor gives us distance-to-object (d)

The pose of the sensor gives us the angle to the object (θ_{us}) off the robot's x axis (x_r)

With d and θ_{us} you can compute the object's pose (x,y) in the robot's reference frame!

- Once the robot has the object's coordinates relative to itself, it needs to transform them to world coordinates:

$$^{A}Q = \, ^{A}_{B}R * \, ^{B}Q + \, ^{A}P$$

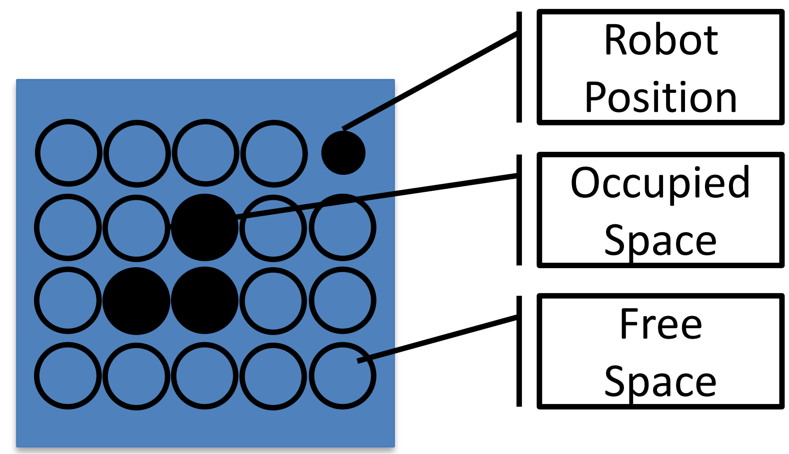
Representing the Map

- 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 simple 4-connected (N,E,S,W) grid representation as a 2D array of Boolean values
  - grid[j][i] = False if occupied, True if free space
  - Grid size is up to you, but I would recommend at least 10x10!
- 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 use design and use functions that maps world pose coordinates (x,y) to grid coordinates (i,j).

Representing the Map

To track the map's progress and make sure everything's working, we will also display a visualization of the map:

- You are free to visualize this however you wish, so long as it's thematically similar to the example on the right →



Sparki ROS Interface

To read robot state:

- Subscribe to '/sparki/state'
  - Type 'std_msgs/String'
  - Contains a JSON encoded dictionary
  - Can use JSON.loads() function to turn a JSON encoded object into a Python object
- Dictionary includes:
  - Servo theta for Ultrasonic Sensor
  - IR Sensor value array (5 values)
  - Ping distance (if ping was requested)

Sparki ROS Interface

To read robot odometry:

- Subscribe to '/sparki/odometry'
  - Type 'geometry_msgs/Pose2D'
  - Contains member variables: x, y, theta

To send motor commands:

- Publish to '/sparki/motor_command'
  - Type 'std_msgs/Float32MultiArray'
  - Send two values in the message, each between [-1.0, 1.0] to determine left and right wheel speed

Sparki ROS Interface

To send a ping command:

- Publish to '/sparki/ping_command'
  - Type 'std_msgs/Empty'
  - Tells Sparki to ping as soon as possible
  - Will include the result from the ping command in the next state broadcast over '/sparki/state'

To send a servo command:

- Publish to '/sparki/set_servo'
  - Type 'std_msgs/Int16'
  - Tells Sparki to set the servo motor to an angle in range [-80, 80] degrees

To set robot odometry:

- Publish to '/sparki/set_odometry'
  - Type 'geometry_msgs/Pose2D'
  - Tells Sparki to set the odometry (useful for loop closure)

Sparki ROS Tips

Your subscribers for Odometry and State will be getting data *asynchronously*!

- I recommend storing the last received values in a global variable within your program.

Your line following code needs to be re-implemented in Python:

- Your Python program doesn't have the same structure as the Arduino (e.g., "setup()" and "loop()") by default, so you'll want to incorporate a loop in your program.

Since you're sending commands over a Serial connection, you will have to be careful to rate-limit your Python loop

- Otherwise you'll oversaturate the communication channel by sending commands too frequently (especially with ping!)