Simulation Video:  
<https://drive.google.com/file/d/1Mhb0yLLjEHfgJpvX5pTDY3UKXcjI9GmR/view?usp=sharing>

1. A data structure distances between all graph nodes would be prohibitively large and difficult to process. Since the robot is working in continuous space, there are an infinite number of points between any two locations, so it is impossible to create a graph that can account for every point. Even if one were to enforce a minimum resolution, the large number edges would make it very difficult to calculate the path distances. Thus, discretizing the graph into a grid format allows us to heavily compress this information into a more computation friendly format.
2. Subscribers
   1. odometry [Pose2D]
      1. The odometry subscriber receives Pose2D objects from the server that specify the robot’s pose (x, y, θ). The callback function can then pushes the new state to the clients global state variable, allowing it to be accessed anywhere in the code.
   2. state [string]
      1. The state subscriber gets a JSON dictionary of robot sensor values. The dictionary always includes the line sensor readings. Additionally, the dictionary will include ultrasonic ping distances when values are requested
3. Publishers
   1. motor\_command [Array<float>]
      1. Transmits an array of motor speed values to the server. Two dimensional array [LeftMotor, RightMotor] controls wheel speed.
   2. set\_odometry [Pose2D]
      1. Resets the robot’s calculated position using the specified Pose2D (x, y, θ). Used establish closed-loop behavior by resetting the pose to (0, 0, 0) when the robot passes the starting line.
   3. ping\_command [empty]
      1. Request an ultrasonic ping distance to be included in the next state dictionary.
   4. set\_servo [short]
      1. Set the ultrasonic sensor angle to the specified angle (degrees).
   5. render\_sim [empty]
      1. Request a simulator frame render.
4. 

There is a slight mismatch between the obstacles and the resulting map. We believe this is in large part due to our resolution. We could improve this by increasing our resolution.

1. I understand the value of ROS and its ability to simplify complex system management, but I think the complexity is difficult to justify when working with something like an Arduino. The pub/sub message model can lead to dirty reads and synchronization issues if not properly accounted for. Additionally, the message broker would put a significant strain on the already limited processor resources.
2. Roughly 6-10 hours