Bayesian Search and Robotics: Locating a Lost Submarine

Problem Definition

Bayesian statistics has been successfully applied in multiple fields. Due to its ability to update beliefs and make decisions iteratively, rather than providing a single static answer or prediction, Bayesian statistics serves as a cornerstone of control AI. It is widely used in time-dependent fields such as trading, healthcare, self-driving cars, robotics, and many others. In this project, we focus on the use of Bayesian statistics in robotics.

One notable historical example of Bayesian search theory was its application in the search for the U.S. Navy's nuclear submarine, USS Scorpion. In May 1968, the submarine failed to arrive at its home port of Norfolk, Virginia. Bayesian search theory, which employs probabilistic reasoning to update beliefs based on evidence, played a crucial role in locating the submarine. Initially, experts used prior knowledge—such as the Scorpion's last known location and trajectory—to create a probability map of its likely positions. This "prior belief" was iteratively updated as new evidence, including acoustic signals and ocean current data, became available.

In this project, we aim to recreate an ocean environment represented as a $n \times n$ grid with m sound wave radius, where a robot searches for a submarine. The submarine generates sound signals, and based on these signals, the robot updates its beliefs regarding the submarine's location. The goal is to find the submarine using Bayesian inference. Before proceeding, we will first provide a detailed explanation of Bayesian search theory.

Bayesian Search Theory

Bayes Theorem

We're starting from foundation of Bayes reasoning. We use conditional probability of an event to prior beliefs and the likelihood of evidence.

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

where:

- Hypthesis H_i: The object is at location i
- Evidence E: Information collected from a sensor
- $P(H|E)$ is posterior probability of the hypothesis H (e.g., the object is at location H), given evidence E
- $P(H)$ is prior probability of H (our belief before observing evidence)
- $P(E|H)$ is likelihood of evidence E, assuming H is true
- $P(E)$ is normalizing factor (total probability of evidence E)

Updating Probabilities

For a search grid with N cells, each cell i has:

- Prior probability $P(H_i)$, representing the initial belief the object is in cell i.
- A likelihood $P(E|H_i)$, representing how likely evidence E would be observed if the object is in cell i.

The posterior probability for each cell is given by:

$$P(H_i|E) = \frac{P(E|H_i)P(H_i)}{\sum_{j=1}^{N} P(E|H_j)P(H_j)}$$

Normalization, ensures that the probabilities sum to 1 after each update.

Search Process

The search process involves:

1. Defining Prior Beliefs:

The search begins by defining prior beliefs based on historical data or the last known position. In our example, this corresponds to assigning initial probabilities to each cell in the grid. Since there is no prior knowledge about the submarine's exact location, the most logical assumption is to assign a uniform prior probability, starting with the middle of the grid as the robot's focus.

2. Collecting Evidence:

Evidence is gathered using sensors, observations, or measurements. In this project, we focus on sound waves produced by the submarine as the primary evidence. However, this method could accommodate any type of signal. The robot's efficiency depends on the signal's strength and its ability to detect it accurately.

3. Updating Probabilities:

New evidence is incorporated into the initial beliefs using Bayes' Theorem, refining the probabilities for each possible location of the submarine.

4. Deciding Search Strategy:

Based on the updated probabilities, the robot directs its search toward the locations with the highest posterior probabilities, optimizing the chances of finding the submarine efficiently.

5. Iterating:

The process is repeated iteratively as new evidence becomes available. The robot continues this cycle of belief updates and directional adjustments until it locates the submarine.

Expected Gain

Detection Probability q_i is the probability of detecting the object in location i given a search effort:

$$q_i = 1 - e^{-\alpha e_i}$$

Where:

- α is effectiveness of the search or simply normalization constant:

$$\alpha = \frac{1}{\sum_j P(E|H_j) \cdot P(H_j)}$$

- e_i is effort expended in searching cell i

Then robot prioritizes areas where the product of posterior probability and detection probability is highest:

$$\text{Expected Gain} = P(H_i|E) \cdot q_i$$

Submarine Example

For clarification, let's outline what each part of Bayes' Theorem actually means in terms of submarine search.

- Prior $P(H_i)$: We assume the submarine is equally likely to be in any grid cell.
- Likelihood $P(E|H_i)$: Model evidence, such as the probability of detecting a sound at a sensor, given the submarine's location.
- Posterior $P(H_i|E)$: Update beliefs after observing sensor data.

Environment Implementation

grid_size and sound_radius

First, we define the search area for the submarine and the radius of its sound signals. For this project, we will use a 30x30 grid to represent the ocean and a 30-cell sound radius to simulate the range of the submarine's sound signals. These parameters can be adjusted to explore different scenarios.

However, it's important to note that the ratio of grid_size to sound_radius significantly affects the efficiency of the search. A larger grid_size/sound_radius ratio means fewer opportunities to detect the submarine quickly, as the sound signals cover a smaller fraction of the grid.

For demonstration purposes, we will keep the grid_size/sound_radius ratio at a manageable level:

$$\text{Ratio} = \frac{\text{grid size}}{\text{sound radius}} = \frac{30}{30} = 1$$

So, since we rely solely on sound wave there is no point to make sound wave radius lower than $\frac{\text{grid size}}{2}$. However, our sound radius is counted in Manhattan Distance, which will cause that sound field will be a square. Therefore we should pick sound radius that's ideally is 2 times larger than grid size.

Beliefs

We need to create matrix with initial beliefs with shape $(grid\ size, grid\ size)$. Since we need uniform beliefs we divide initial values $\frac{1}{grid\ size^2}$:

$$P(H_i) = \frac{1}{N}, \text{where } N = grid\ size^2$$

Submarine position

After we define initial `submarine_position` which is random cell from whole world, i.e. grid.

Generating the Sound Field

Sound field is a function of the distance from the submarine's position. Intensity decreases linearly with distance:

$$I(x, y) = 1 - \frac{distance}{sound\ radius}$$

Where:

- $distance = |x - x_s| + |y - y_s|$, the Manhattan distance from the submarine's position (x_s, y_s).

To sum up, this function models how sound weakens as it propagates through the environment.

Model

Bayesian Theorem Implementation

Code

```
# bayesian belief update function
def calculate_likelihood(sound_field):
    likelihood = np.exp(-0.5 * (1 - sound_field)**2)  # Gaussian
likelihood
    likelihood /= np.sum(likelihood)  # Normalize
    return likelihood

# update beliefs based on sound field using Bayesian theory
def update_beliefs(beliefs, sound_field):
    likelihood = calculate_likelihood(sound_field)  # likelihood
from the sound field
    updated_beliefs = beliefs * likelihood  # element-wise
multiplication of priors and likelihoods
    updated_beliefs /= np.sum(updated_beliefs)  # normalize to
maintain total probability of 1
    return updated_beliefs
```

Formula

$$P(position|evidence) = \frac{P(evidence|position) \cdot P(position)}{P(evidence)}$$

Where:

- Prior: $P(position)$
- Likelihood: $P(evidence|position)$, based on sound field intensity
- Posterior: $P(position|evidence)$

Prior

At first we guess where the submarine is located. Since we don't have any previous information we use uniform distribution for all cells in the grid:

$$\text{beliefs} = \frac{1}{grid\ size^2}$$

Likelihood

Sound Field

For sound field we create simple function, of course it's not actual function of sound waves dispersion in the water. However, for our task it's enough.

$$sound\ field[x, y] = 1 - \frac{distance}{sound\ radius}$$

For:

- submarine position at $(x_s, y_s) = (15, 15)$
- $sound\ radius = 30$
- cell that is 10 units away from submarine cell in Manhattan Distance

We get:

$$sound\ intensity = 1 - \frac{10}{30} = 0.67$$

Then, likelihood is given by:

$$\text{Likelihood} \propto exp\Big(- \frac{(x - \mu)^2}{2\sigma^2}\Big)$$

where:

- x: The observed sound intensity at the cell.
- μ: The expected sound intensity for the submarine position, so simply 1, because sound at submarine position has highest possible value. In other words sound is

highest where the submarine is located.
- σ: Parameter that controls how sharply the likelihood drops as the observed sound intensity devaites from the expected value. For simplification we assume $\sigma = 1$.

The formula is derrived from PDF of Gaussian Distribution. Since we look for proportionality we can drop scaling parameter.

Eventually our likelihood is given by:

$$\text{Likelihood} = exp(-0.5 \cdot (x - \mu)^2)$$

For:

- Submarine Position at (15, 15)
- sound radius = 30
- Sound Field for cell (10, 15), obeserved sound intensity is given by:

$$x = 1 - \frac{Manhattan\ distance}{sound\ radius} = 1 - \frac{5}{30} = 0.83$$

- Expected Intensity μ at submarine's position, we assume $\mu = 1$

Then let's plug it into the formula:

$$\text{Likelihood} = exp(-0.5 \cdot (x - \mu)^2) = exp(-0.5 \cdot (1 - 0.83)^2) = exp(-0.5 \cdot 0.0289$$

Posterior

$$belief[x, y] = prior[x, y] \cdot likelihood[x, y]$$

For:

- Submarine Position at (15, 15)
- sound radius = 30
- Sound Field for cell (10, 15)
- Expected Intensity μ at submarine's position, we assume $\mu = 1$

$$belief[x, y] = \frac{1}{900} \cdot 0.9856 = 0.001096$$

In other words, the updated belief for the cell (10, 15) is approximately 0.0011.

Entropy Implementation

Entropy quantifies the level of uncertainty associated with a variable's potential states. In other words, entropy indicates how much uncertainty exists in the system's

beliefs about all possible options. For example:

High entropy means that all potential locations of the submarine have similar probabilities, indicating high uncertainty. Low entropy means that one of the spots (cells) has a much higher probability than the others, indicating greater confidence about where the submarine might be located.

Code

```
def calculate_entropy(beliefs):
    return -np.sum(beliefs * np.log(beliefs + 1e-9))  # add small value to avoid log(0)
```

Formula

$$H = -\sum_{x} p(x) \cdot log(p(x))$$

Where:

- H: entropy value
- $p(x)$: probability of x
- $log(p(x))$: log probability, which measures "weight of information"

Now, we can translate it for our task:

$$H(beliefs) = -\sum belief[x, y] \cdot log(belief[x, y])$$

Where:

- $belief[x, y]$ is probability that submarine is in cell (x, y)

Example

Let's say that we've already calculated posterior and normalized the data, eventually we received grid of probabilities or preprocessed sound intensities. We will use exemplary 5x5 grid:

$$\text{Updated Beliefs} = \begin{bmatrix} 0.07 & 0.09 & 0.11 & 0.09 & 0.07 \\ 0.09 & 0.12 & 0.14 & 0.12 & 0.09 \\ 0.11 & 0.14 & 0.16 & 0.14 & 0.11 \\ 0.09 & 0.12 & 0.14 & 0.12 & 0.09 \\ 0.07 & 0.09 & 0.11 & 0.09 & 0.07 \end{bmatrix}$$

In [ ]:
```
import numpy as np

beliefs = np.array([
```

```
    [0.07, 0.09, 0.11, 0.09, 0.07],
    [0.09, 0.12, 0.14, 0.12, 0.09],
    [0.11, 0.14, 0.16, 0.14, 0.11],
    [0.09, 0.12, 0.14, 0.12, 0.09],
    [0.07, 0.09, 0.11, 0.09, 0.07]
])

# calculate entropy
def calculate_entropy(beliefs):
    return -np.sum(beliefs * np.log(beliefs + 1e-9))  # add small value t


entropy = calculate_entropy(beliefs)
entropy
```

Out[ ]: 5.861477316518562

We see that the entropy is huge, that means that we have huge uncertainty. Now, let's see what will happen if we would have a bit different probabilties:

$$\text{Updated Beliefs (lower entropy)} = \begin{bmatrix} 0.001 & 0.002 & 0.003 & 0.002 & 0.001 \\ 0.002 & 0.005 & 0.01 & 0.005 & 0.002 \\ 0.003 & 0.01 & 0.2 & 0.01 & 0.003 \\ 0.002 & 0.005 & 0.01 & 0.005 & 0.002 \\ 0.001 & 0.002 & 0.003 & 0.002 & 0.001 \end{bmatrix}$$

Here, we can see that only cell $(3, 3)$ has quite big probability, rest of cells are very small. Therefore, we are pretty confident about where the submarine may be located.

In [ ]:
```
updated_beliefs = np.array([
    [0.001, 0.002, 0.003, 0.002, 0.001],
    [0.002, 0.005, 0.01, 0.005, 0.002],
    [0.003, 0.01, 0.2, 0.01, 0.003],
    [0.002, 0.005, 0.01, 0.005, 0.002],
    [0.001, 0.002, 0.003, 0.002, 0.001]
])

calculate_entropy(updated_beliefs)
```

Out[ ]: 0.8088351788317616

As we can see we got much lower entropy than in first example, because we're much more confident where the submarine is located.

Expected Gain

Code

```
def calculate_expected_gain(beliefs, actions, sound_field,
robot_position):
```

```
...

# bew potential position
        new_x, new_y = x + dx, y + dy
        if 0 <= new_x < grid_size and 0 <= new_y < grid_size:
            # base gain from sound intensity
            sound_intensity_gain = sound_field[new_x, new_y]

            # add penalty if moving away from the current sound
intensity
            penalty = 0.1 * max(0, sound_field[x, y] -
sound_intensity_gain)
            gain = sound_intensity_gain - penalty

            # entropy reduction based on beliefs
            simulated_beliefs = simulate_belief_update(beliefs,
action, sound_field)
            entropy_after_action =
calculate_entropy(simulated_beliefs)

            gain -= entropy_after_action

            gains[action] = gain

    # return the action with the highest calculated gain
    best_action = max(gains, key=gains.get)
    return best_action
```

Entropy

Formula

$$Gain(A) = H(B) - H(B'|A)$$

Where:

- H(B): Entropy of the current belief state.
- H(B'|A): Entropy of the belief state after taking action A.

or in other words:

$$\text{Information Gain} = H_{before\ action} - H_{after\ action}$$

Now, we iterate through every possible action in all available actions. Due to the fact that we're operating on grid with 4 possible moves, we have 4 possible actions (if robot is not on bound).

Penalty for Moving Away from Higher Intensity

Formula

$$penalty = 0.1 \cdot max(0, sound\ field[x, y] - sound\ intensity\ gain)$$

Penalizes actions that move from higher sound intensity (current cell) to lower sound intensity (proposed cell).

Simulate belief Update for Action

Formula

$$simulated\ beliefs = belief\ update(beliefs, action, sound\ field)$$

simulate_belief_update shifts the robot's belief based on the proposed action, effectively predicting how the robot's certainty about the submarine's location would change. The robot doesn't just consider the sound field but evaluates how its certainty (beliefs) would evolve after moving. This makes its decision process more strategic.

Calculate Entropy After Action

Formula

$$entropy\ after\ action = -\sum P(x, y) \cdot log(P(x, y))$$

The robot calculates the entropy of the updated belief distribution to measure how uncertain it would be after the action.

Add Penalty for High Entropy

Formula

$$gain = gain - entropy\ after\ action$$

Subtract the entropy directly from the gain, penalizing moves that result in higher uncertainty.

Store Gain for the Action

Formula

$$gains[action] = gain$$

Select action with the highest gain

Formula

$$best\ action = max(gains)$$

After evaluating all possible actions, the robot will select the one with the highest gain.

Test

Let's now see how the model performs on 4 different examples.

Example 1

$$grid\ size = 30$$

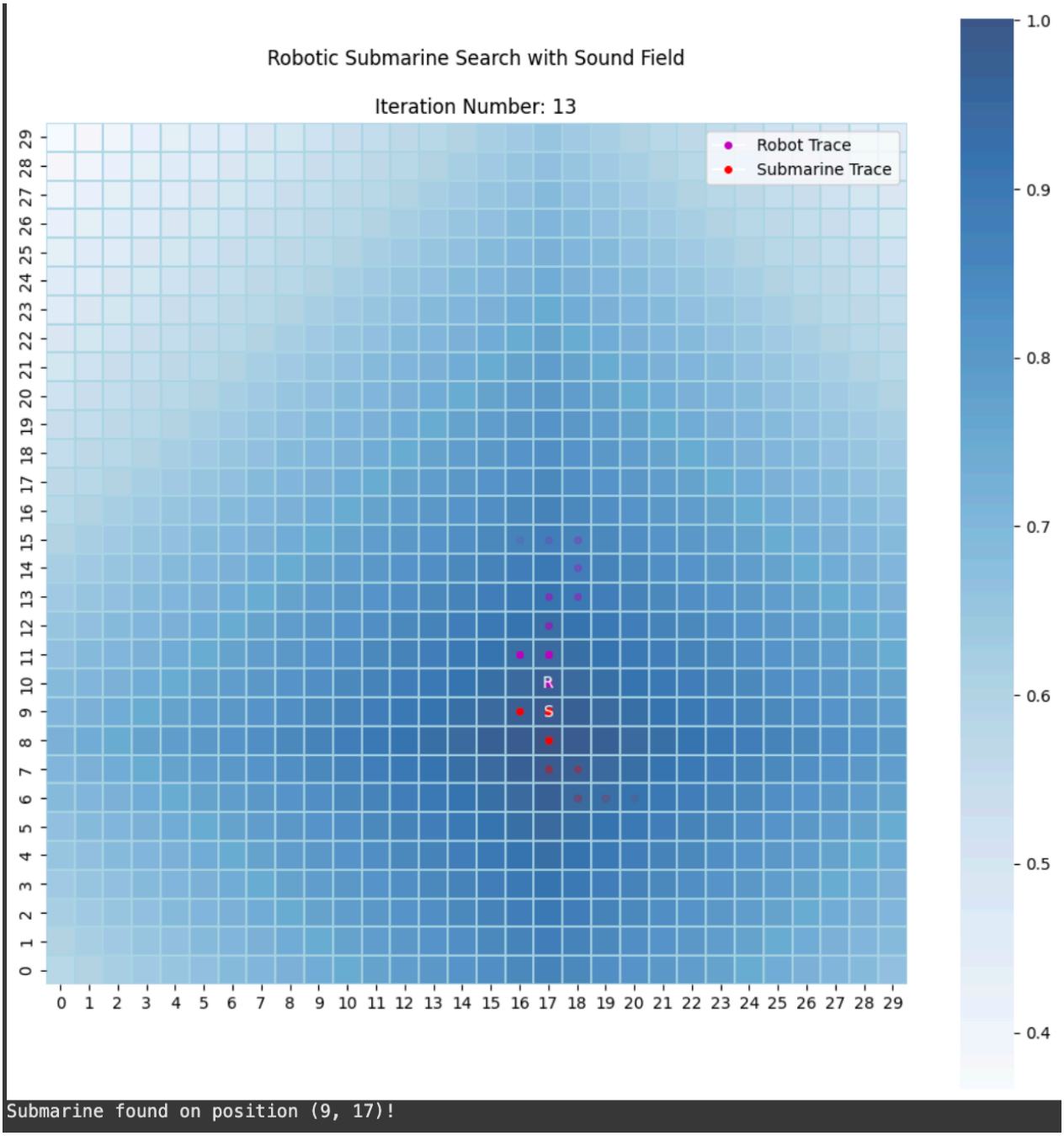
$$submarine\ sound\ radius = 60$$

$$robot\ prior\ belief = [15, 15]$$

In [ ]:
```
from PIL import Image
import IPython.display as display

def show(dir):
    img = Image.open(dir)
    display.display(img)
```

In [ ]:
```
show('/Users/adriankazi/Desktop/omscs/Bayesian Statistics/Final Project/S
```



Model was able to find the submarine after 13 iterations. This is simplest example of how model works. The core idea is that sound wave is spread all around the map otherwise robot is not able to get any evidence to work on.

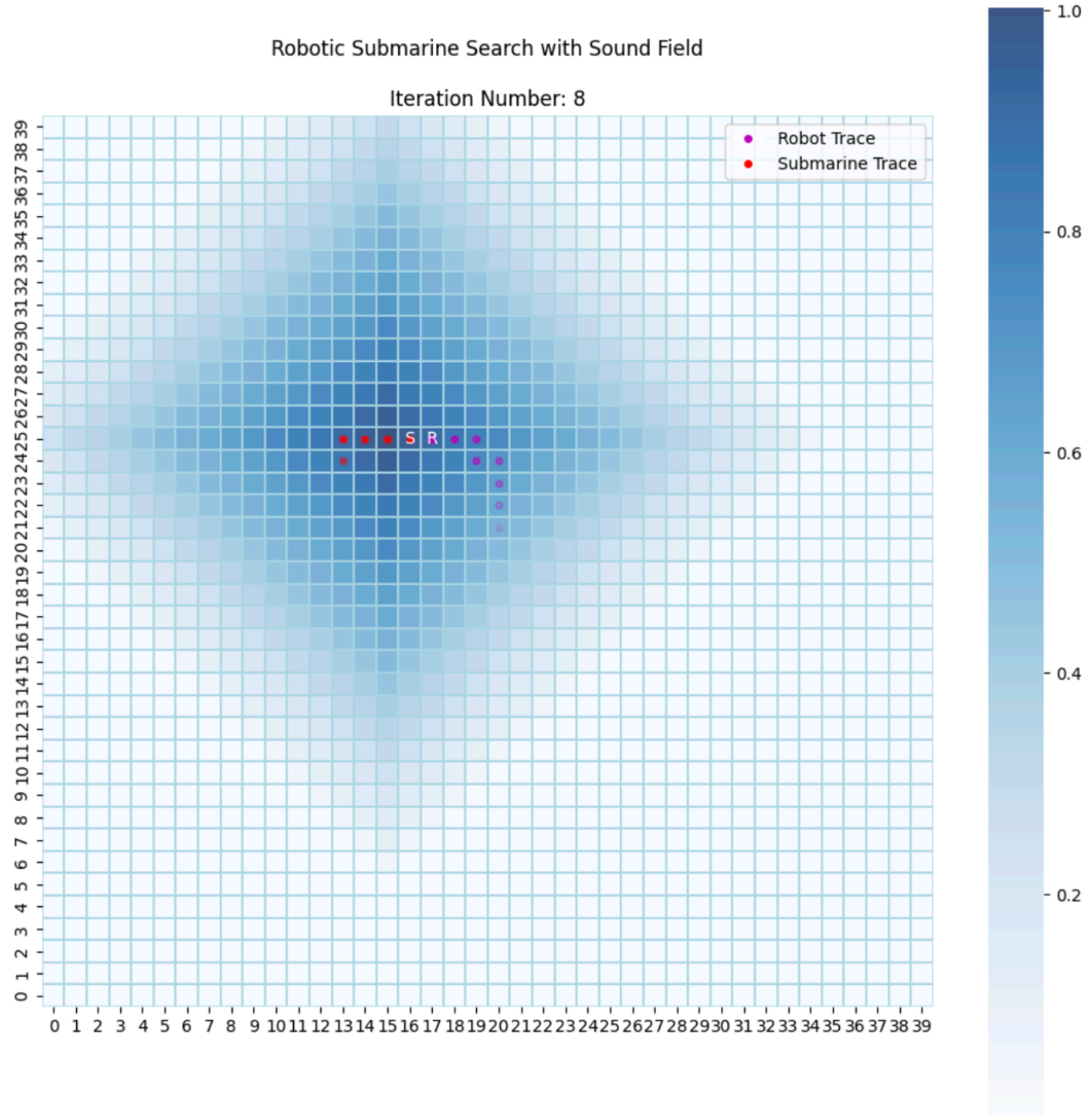
Example 2

$$grid\ size = 40$$

$$submarine\ sound\ radius = 20$$

$$robot\ prior\ belief = [20, 20]$$

In [ ]:
```
show('/Users/adriankazi/Desktop/omscs/Bayesian Statistics/Final Project/S
```




```
Submarine found on position (25, 16)!
```

Robot was able to find submarine after 8 iterations. However, we were a bit lucky due to the fact submarine was placed close enough to get the sound wave

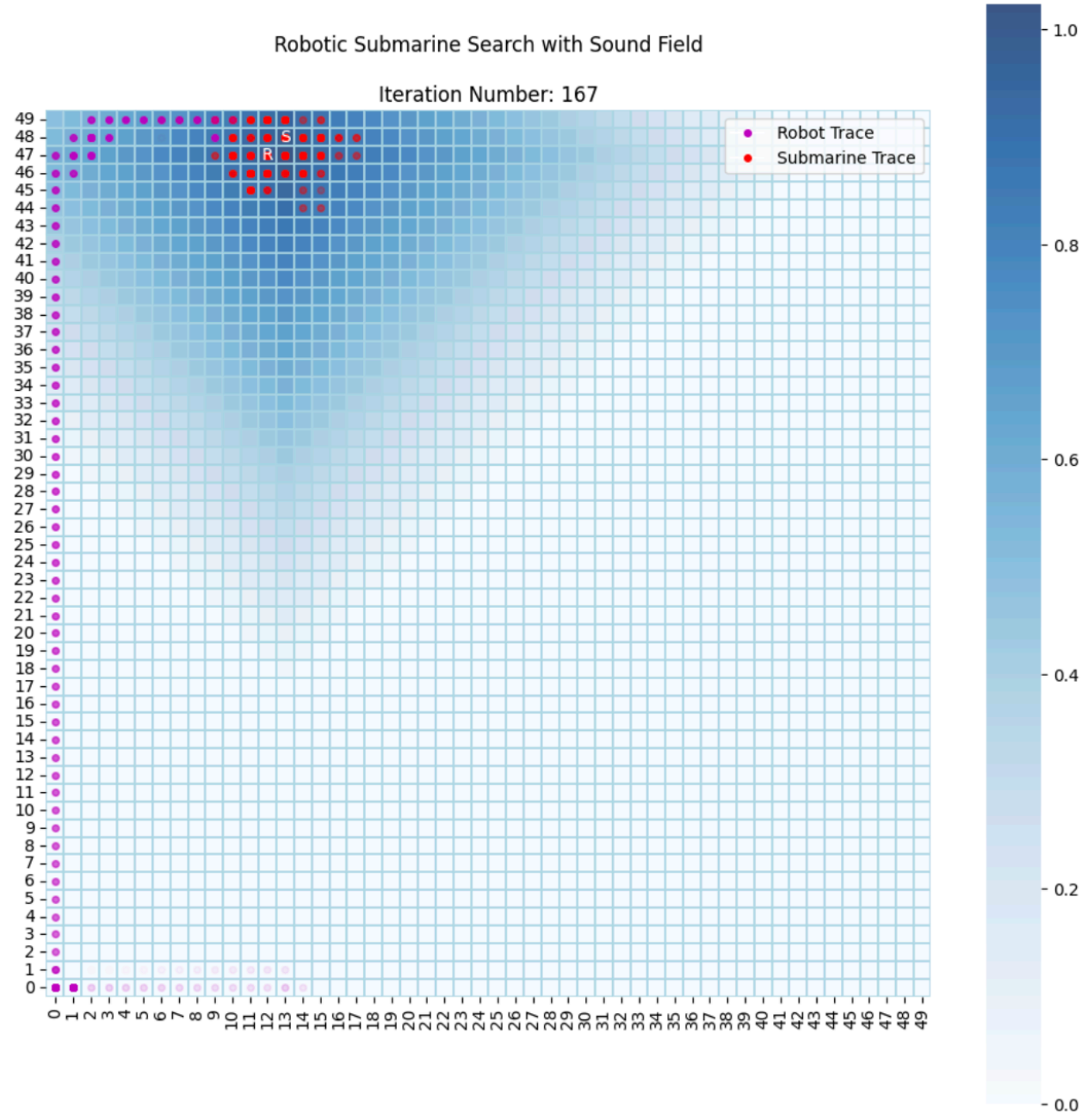
Example 3

$$grid\ size = 50$$

$$submarine\ sound\ radius = 30$$

$$robot\ prior\ belief = [1, 1]$$

```
In [ ]: show('/Users/adriankazi/Desktop/omscs/Bayesian Statistics/Final Project/S
```




```
Submarine found on position (48, 13)!
```

As we can see robot was able to find the submarine after 167 iterations and thats only becuse robot randomly got close enough to the submarine sound field. Sound field is the only evidence robot can get if its out of the field only luck can help. Let's now see same example but with much higher sound field.

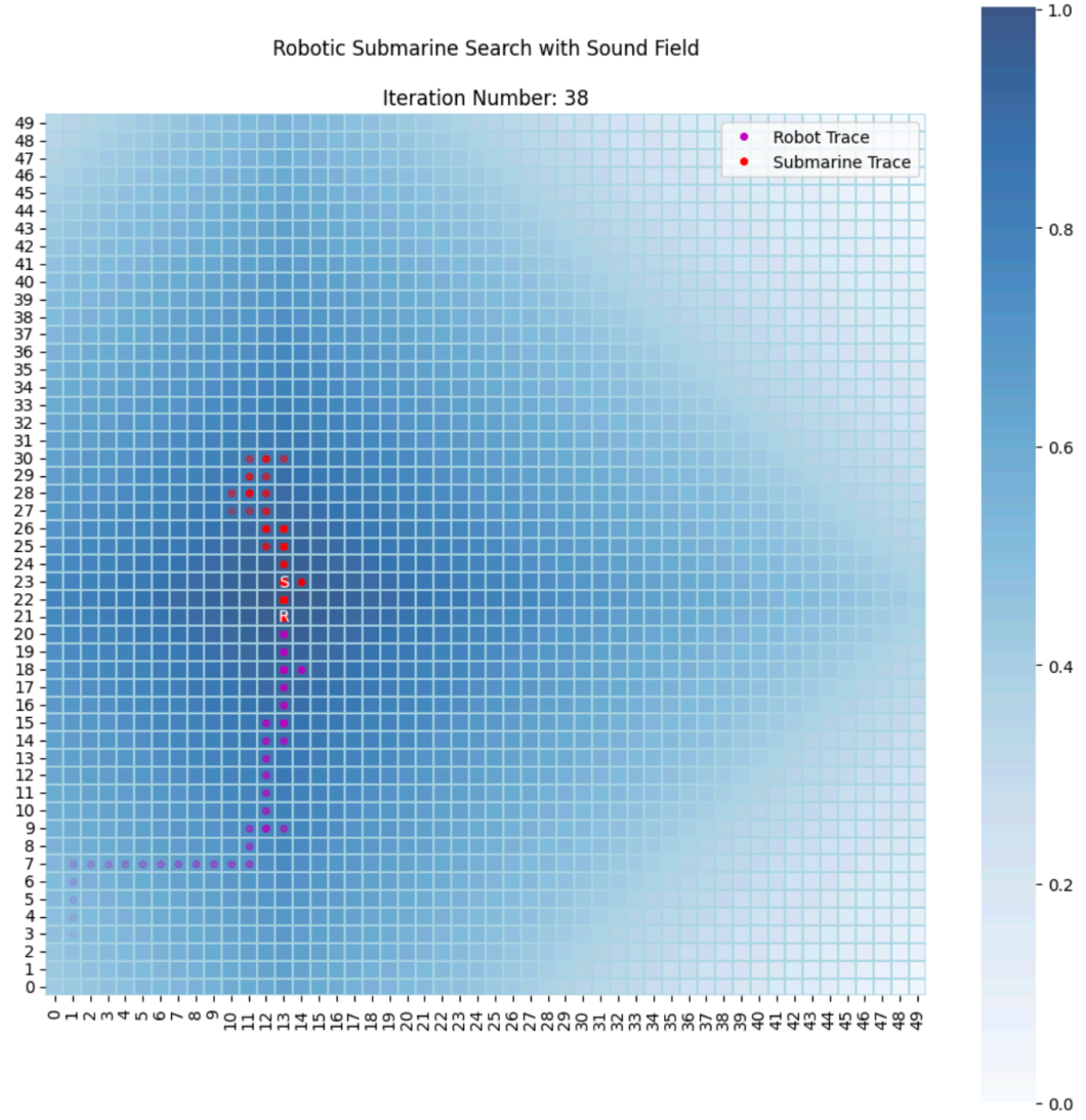
Example 4

$$grid\ size = 50$$

$$submarine\ sound\ radius = 60$$

$$robot\ prior\ belief = [1, 1]$$

In [ ]:
```
show('/Users/adriankazi/Desktop/omscs/Bayesian Statistics/Final Project/S
```




```
Submarine found on position (22, 13)!
```

Due to the higher sound field robot was able to find submarine in 38 iterations.