Problem Chosen	2024 MCM/ICM Summary Sheet	Team Control Number
B		2426376

Summary

Maritime-Cruises Mini-Submarines (MCMS) is a Greek company that manufactures human-carrying submersibles. Submersibles are underwater vehicles that are taken to a desired location by a host ship, then deployed unthethered. MCMS is interested in providing tourist excursions to see sunken ships in the Ionian Sea using their submersibles. We have been tasked with developing safety precautions in the occurrence of lost communication or malfunction of the submersibles, as well as strategies for predicting the location of the submersible over time and optimal deployment locations and search routes for search and rescue efforts.

Inspired by Bayesian Search Theory, we constructed a model that utilizes multivariate Gaussian distribution to determine the most probable location of a submersible from an aerial view, given its last known location. To take into account the geography of the seafloor, we then introduce a function that determines the likelihood of a submersible being located based on the depth of a given location. We then constructed an algorithm that simulates deep sea currents and the effect they have on the movement of a submersible. Our final function combines the three models in order to most accurately predict the location of a submersible over time.

We then utilize that model to determine the best locations for the deployment of search and rescue teams. We then construct an algorithm for determining an optimal search pattern using an expanding square search pattern. Incorporating our wave function into this algorithm, we develop a modified expanding square search pattern that adapts to the predicted movement of a submersible. This gives us a unique search pattern dependent on a submersible's last known location and the deep sea currents within the search area.

In order to decrease uncertainties with our models, we recommend that the submersible carry equipment such as pressure sensors, Doppler Velocity Log sensors, and acoustic modems. This equipment will help to ensure that the submersibles are recording important information, such as speed, location, and depth, that can aid in the refinement of our model to best predict the location of a submersible and develop an optimal search plan.

Our model is fairly versatile and can be adapted to tourist locations outside of the Ionian Sea. Given known information about the sea floor and deep sea current patterns within a given location, the model can be adapted to any search area.

Contents

1 Introduction **3**

2 Location Prediction **3**

2.1 Assumptions 3

2.2 Approach 3

2.3 Model 3

2.4 Analysis 8

3 Deployment and Search Patterns **8**

3.1 Assumptions 8

3.2 Model 8

3.3 Analysis 10

3.4 Extrapolation to Other Contexts 10

4 Simulation **11**

4.1 Assumptions 11

4.2 Simulation 11

5 Memo **14**

6 References **16**

7 Appendix **17**

1 Introduction

Maritime Cruises Mini-Submarines (MCMS), a submersible construction company, aims to introduce submersible excursion opportunities for tourists to explore sunken shipwrecks in the Ionian Sea. Submersibles are underwater vehicles that are moved to a desired location by a host ship and deployed untethered. Seeking to secure regulatory approval, MCMS has tasked us with developing a model to predict the location of a submersible over time. This initiative is necessary to ensure the safety of passengers in the occurrence of a loss of communication with the host ship or mechanical malfunctions. We will also provide recommendations for possible equipment that could aid in search efforts. Additionally, we will develop a model to determine optimal launching points and search patterns for equipment to locate a lost submersible, aiming to minimize the time it takes. Finally, we will consider how this model can be applied to other possible tourist destinations outside of the Ionian Sea.

2 Location Prediction

2.1 Assumptions

1. After the initial accident the submersible can no longer move on its own accord and send out additional telemetry.
2. The last known location of the submersible is known.
3. While underwater currents affect the likelihood of where the submersible is found they are unlikely to move the submersible long distances [3].
4. Underwater geographic features, visibility, and depth of submersible all affect the likelihood of finding a submersible and can be combined into one variable.

2.2 Approach

Our approach is to use Bayesian search theory [6] to create three probability density functions, each of which generates a probability density map of the search space that when combined creates an overall probability density map. The first density matrix is a function of time and is determined by the last seen location of the ship and the assumption of the direction it could have traveled due to underwater currents which is modeled in the last matrix. The second density matrix is static and is determined by assumption 5, or in other words, models how likely we are to find the submersible in a region given it is there. Finally, the last matrix models the likelihood of the submersible being in a location due to underwater currents and interacts with the first matrix to simultaneously shift the position of the search area as a function of time.

2.3 Model

We first began by developing a model that simply showed the probability of where a submersible could be given its last known location. This involved the creation of 2D probability distribution grid wherein the intensity is shaped by a Gaussian distribution centered around the last known coordinates and whose standard deviation along its axes is a function of time to model the diffusion of our certainty in the submersible location over time.

The Gaussian distribution equation is as follows [2]:

$$P(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left(-\frac{1}{2}\left(\frac{(x-\mu_x)^2}{\sigma(t)_x^2} + \frac{(y-\mu_y)^2}{\sigma(t)_y^2}\right)\right)$$

Where $P(x,y)$ gives us the probability density of a given point (x,y) and (μ_x,μ_y) represents the center of the distribution. $\sigma(t)_x$ and $\sigma(t)_y$ represent the standard deviations along the X and Y axes as a function of time and are proportional to the estimated radius plus a covariance variable multiplied by the time passed searching. In the context of the probable location of a submersible, $P(x,y)$ signifies the likelihood that a submersible is at a given point and (μ_x,μ_y) corresponds to the last known location of the submersible.

The location of the Gaussian blot can also be changed as a function of time to model external forces acting on the submersible by simply shifting the values in the matrix along its axes. The direction and speed of the probability distribution are determined by the directions and speed of the wave matrix discussed on Page 6.

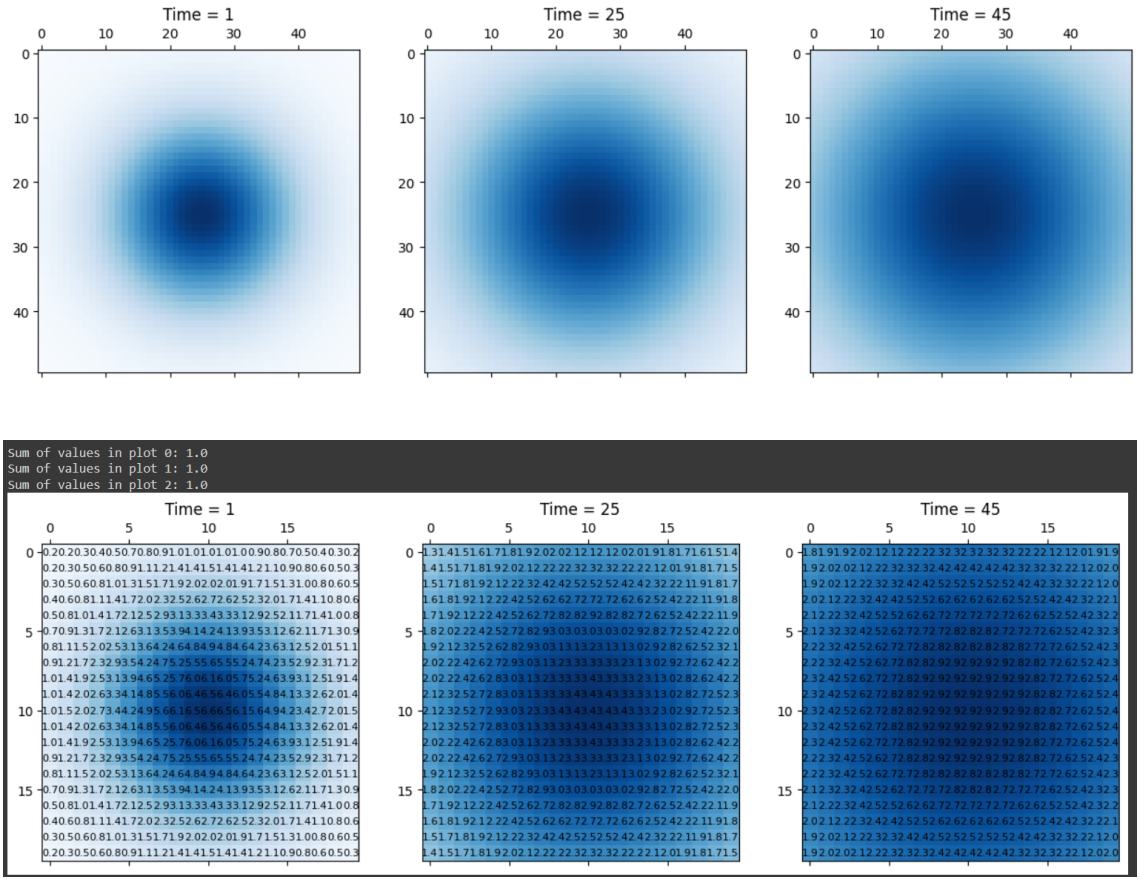


Figure 1: Size of Probability Distribution Blot Over Time (Note that as the time increases and the normal distribution variance widens the sum of all values still adds up to one since it is a probability distribution matrix.)

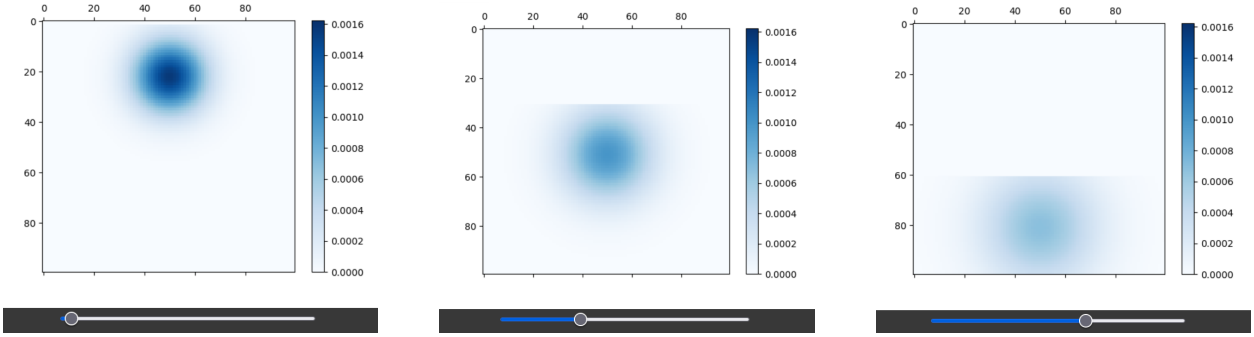


Figure 2: Movement and Diffusion of Probability Matrix over Time

To take into consideration how visibility, seafloor geography, and depth, all of which can affect the probability of being able to locate a submersible at a given point, we adjusted the model to show the probability of how likely a submersible is to be found given that it is known to be at a certain location. Whereas the first model told us the likelihood of the submersible being in a given place based on the last known location and time passed, this model tells us how likely we are able to actually find the submersible based on hypothetical obstacles that would impede the search efforts. For example, locations with greater depth will have a lower likelihood compared to locations with shallow depth, as there is a greater distance to search with more depth. Or, certain areas could have worse visibility which in turn decreases the likelihood of finding an object in that area even if the area was checked. This model can be defined as:

$$M_{ij} = \text{likelihood} \times \exp\left(-\left(\frac{\text{distance}}{\text{radius}}\right)^2\right)$$

$$\text{distance} = \sqrt{(i - \text{center\_x})^2 + (j - \text{center\_y})^2}$$

Where M_{ij} is the value of the i-th row and j-th column, likelihood is a parameter associated with a given circle, distance tells us how far a grid cell is from the center of a circle at (x,y), and radius tells how far the circle reaches. Circles represent areas within the sea that have greater depths, such as sea caverns. Each circle has an associated likelihood of not finding submersibles, with areas of greater depth having a higher likelihood. Circles are defined as (x, y, r, z) where x and y determine the position of the center, r determines the radius, and w is the likelihood of not finding a submersible within that circle.

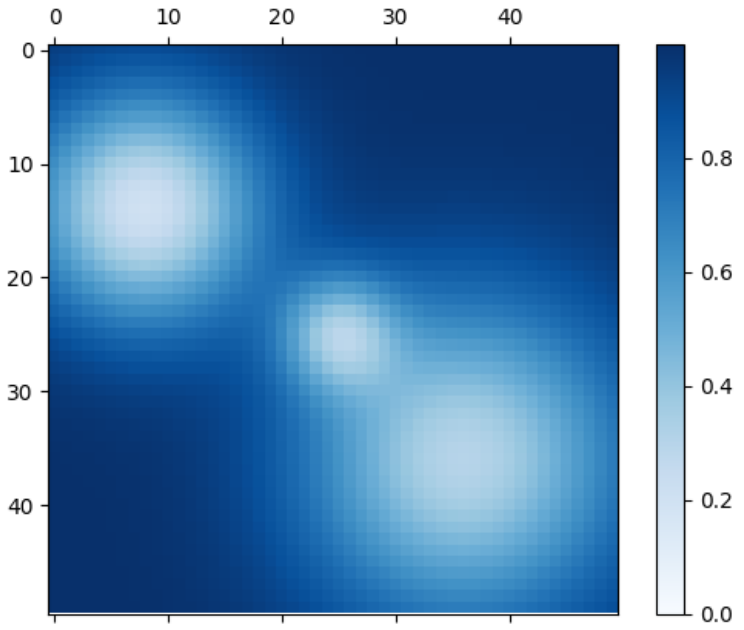


Figure 3: Likelihood of Finding a Submersible

Figure 2 shows a sample probability map given the circles (12, 12, 10, 0.5), (25, 28, 9, 0.8), and (10, 35, 8, 0.4). White spots can be seen near the coordinates of the circles provided, showing how the closer a submersible is to an area of deeper depth, the less likely we are to actually find that submersible in that specific location.

We also wanted to consider how different underwater currents or waves could impact the location of a submersible. To do this, we introduced a wave model that simulates the effects of currents acting on the submersible. We did this by considering the impact of currents in both the horizontal and vertical directions in order to determine the total impact on a submersible.

The impact in the horizontal direction is defined as:

$$\text{horizontal\_wave}(i) = \exp\left(-\frac{(i-\text{peak\_h}-\text{off\_h})^2}{20.0}\right)$$

The impact in the vertical direction is defined as

$$\text{vertical\_wave}(j) = \exp\left(-\frac{(j-\text{peak\_v}-\text{off\_v})^2}{20.0}\right)$$

Combining these, we get a cumulative model for the effect of a wave on a submersible:

$$\text{wave\_generated}(i, j) =$$
$$1 + \text{wave\_intensity} \times (\text{horizontal\_wave}(i) + \text{vertical\_wave}(j))$$

Where i is the horizontal position, j is the vertical position, peak_h is the peak position of the horizontal wave, peak_v is the peak position of the vertical wave, off_h is the horizontal offset, off_v is the vertical offset, and wave_intensity is a parameter controlling the strength of the overall wave.

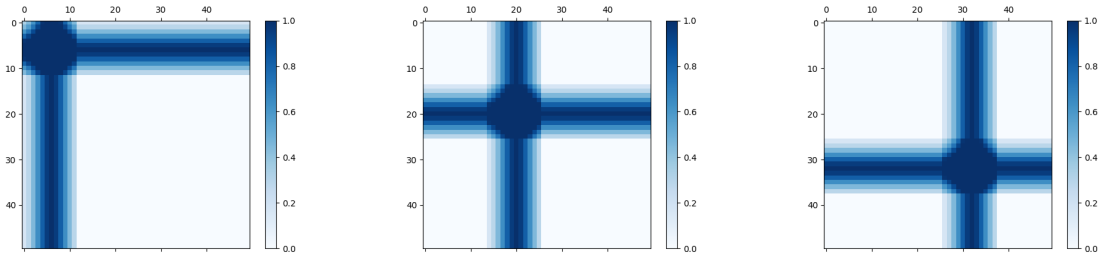


Figure 4: Sample Wave Moving Down and Right Over Time

Finally, we combined these three models together to most optimally predict the location of a submersible given its last known location. We did this by multiplying the three models together.

This combined model is defined as:

$$\text{combined\_prediction} = P(x, y) \times M_{ij} \times \text{wave\_generated}(i, j)$$

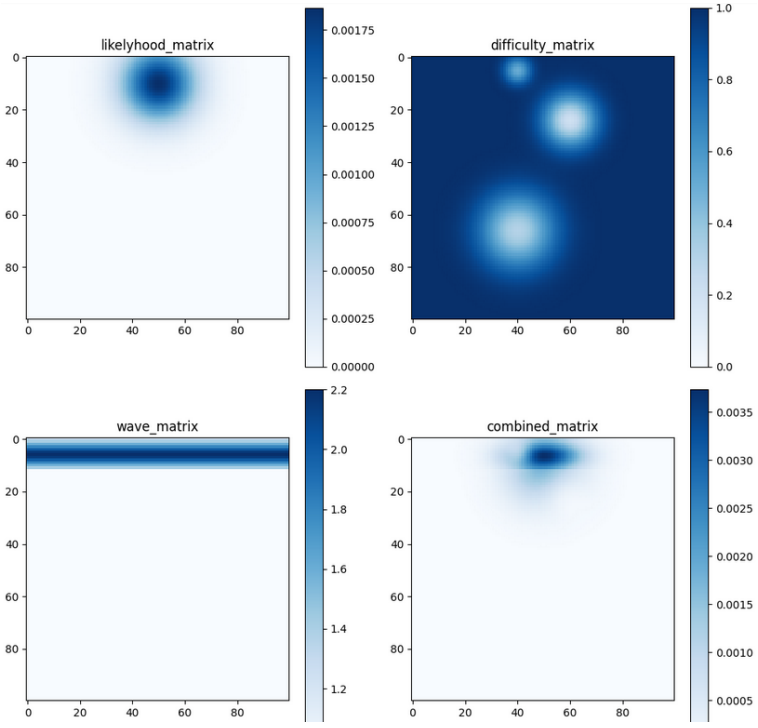


Figure 5: Combined Predictive Location Model

Figure 5 shows the three models in their initial states and their combination. The top left image is the location probability density function which will diffuse and shift down with time. The top right image shows the difficulty matrix where areas of lighter color symbolize geographical areas where the search is obstructed. The lower left image shows the initial state of a wave which will propagate down the length of the search area several times during the duration of the simulation.

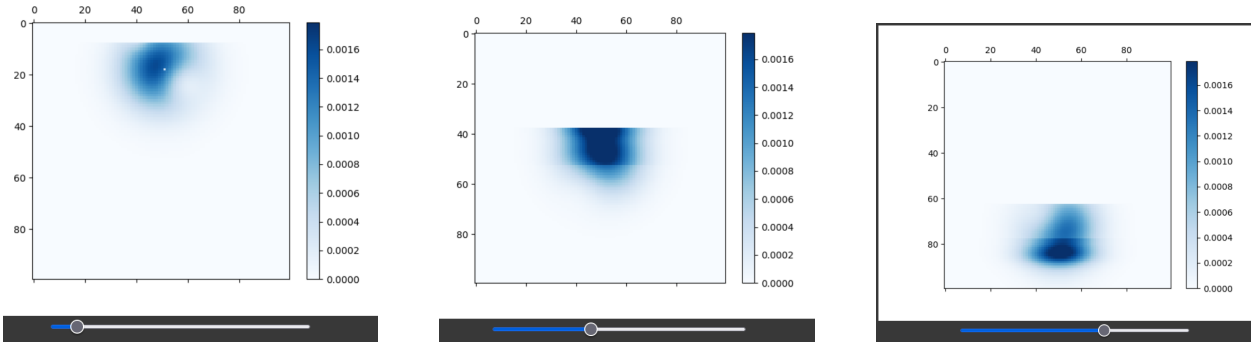


Figure 6: Combined Prediction Matrix Over Time

2.4 Analysis

Some uncertainties to consider with this model are that we do not have the exact values that can be applied to our model and are only using estimates from the data we are given. Some useful information that the submersible can periodically send back to its host ship to decrease uncertainties prior to an incident is its current location, speed, and pressure levels. Equipment that the submersible would need for this is a pressure sensor, a Doppler Velocity Log sensor, and an acoustic modem. The Doppler Velocity Log is an acoustic sensor that will estimate the velocity of the submersible relative to the sea floor [1]. An acoustic sensor can be used to determine the location of the boat and then send back that information to the host ship using sonar communication [9].

3 Deployment and Search Patterns

3.1 Assumptions

1. When the search vessel navigates over a certain point, it is not guaranteed the vessel will find the submersible.

2. There is an underwater current pushing the submersible at a near constant speed.

3.2 Model

The expanding square search pattern is sufficient to cover a large area. In the case that the submersible was immobilized under the water, this pattern would be efficient. Since the likelihood matrix strongly builds upon where the submersible was initially lost, if there were no underwater currents the search would eventually find the missing ship. However, the search fails to account for dynamically shifting environments. This includes not only oceanic settings but land areas affected

by external factors as well. As an alternative, the algorithm was modified to account for significant shifts in a single direction.

To search the area for the missing submersible, the algorithm builds upon an expanding square search pattern in which a search starts from a datum point and moves outward in a systemic square pattern [4]. Here, the datum is the point at which the boat has the highest likelihood of being found. To find each successive point, the following factors are taken into consideration:

- The **leg_length** is the starting distance for each side of the square.
- The **increment** is how much this **leg_length** increments on each turn.
- The **dir** is which distance the algorithm wants to maximize.
- **s** is how much this distance is being maximized.

The algorithm to find the new point of search is given by the following:

$$(x_{n+1}, y_{n+1}) = \begin{cases} (x_n + 0, y_n + 1 + s) \text{ for dir = 'd'} \\ (x_n, y_n - 1 - s) \text{ for dir = 'u'} \\ (x_n - 1 - s, y_n) \text{ for dir = 'l'} \\ (x_n + 1 + s, y_n) \text{ for dir ='r'} \end{cases}$$

The modification to this algorithm stems from the wave or deep water currents pushing the submersible. Instead of always incrementing distances by 1, there is a variable parameter s determined by the speed and direction of the wave. The algorithm cycles through all four cases to produce an expanding square.

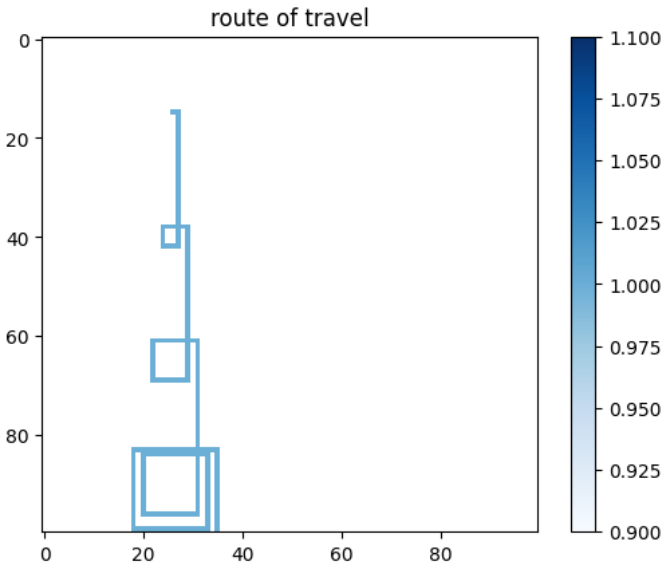


Figure 7: Route of Expanding Square Search

Figure 7 shows the path of the search as the wave moves downwards. The length of the square in the negative y direction is adjusted to account for movement of the submersible underwater in a state of neutral buoyancy.

It is assumed that when a square is searched it is not guaranteed for the lost submersible to be found. As a result, the likelihood is updated with the following:

$$\text{likelihood}(x, y) = \frac{\text{likelihood}(x, y) \times (1 - \text{difficulty(x,y)})}{(1 - \text{likelihood}(x, y)) + \text{likelihood}(x, y) \times (1 - \text{difficulty}(x, y))}$$

Where the difficulty matrix is given by natural terrain and various external factors. Since the current zone is updated all other likelihoods also need to be re-scaled to ensure validity of the probability density matrix. The re-scaling is done using:

$$\text{likelihood}(x, y) = \frac{\text{likelihood}(x, y)}{((1 - \text{likelihood}(x, y)) + \text{likelihood}(x, y)) \times (1 - \text{likelihood}(x, y)) \times \text{difficulty}(x, y)}$$

This is applied to every point in the probability matrix besides the one searched in the former equation.

This modified expanding square search model enables more environmental flexibility to find the hidden submersible. It accounts for changes in terrain and movement of the underwater vessel. The speed of the movement is also variable depending on the speed of the underwater currents.

3.3 Analysis

The search model visualizes an expanding square search over the most probable region the submersible was lost in. It performs this search while trying to maximize time in the most probable areas.

To some extent the model struggles to model rough situations because it fails to account for major shifts in multiple directions. It instead accounts for a major shift in a singular direction and minor shifts in others. This is due to the characteristics of an expanding square search, which originally accounted for covering a stationary area. However, we modified it, to cover a one-directional moving area. For example, if the vessel is supposedly moving in the $-y$ direction, the search will account for this shift, with minor movement in the $+x$ and $-x$ direction to cover as much area as possible.

3.4 Extrapolation to Other Contexts

The Modified expanding square search can be extrapolated to any context that requires a search with uncertainty over a large area. The potential of the search is not exclusive to aquatic environments and can be used in land areas as well. Any time a "Search and Rescue" type objective is needed, the flexibility of the algorithm allows for the area to be moving with specific environmental dynamics.

4 Simulation

4.1 Assumptions

1. The possible search area is 10 square kilometers, based on the geographical distances in parts of the Ionian Sea where shipwreck tourism takes place [4].

2. The last known location of the submersible is known.

3. While underwater currents affect the likelihood of where the submersible is found they are unlikely to move the submersible long distances as we assume an average speed of 6km/hr [3].

4. Underwater geographic features, visibility, and depth of submersible all affect the likelihood of finding a submersible and can be combined into one variable.

5. Possible depths do not exceed 200 meters as most shipwreck locations are located in shallower areas [4].

6. The speed of the search vehicle equipped with commercial sonar devices with a sensing depth of around 150 meters has to maintain a speed of 15-25km/hr for accurate readings, we assume a speed of 18km/hr [6].

7. We assume an initial search area with a radius of 1 kilometer. Note: the time between the submersible's immobilization and the mounting of search efforts can increase the radius of the initial search area as additional time increases uncertainty.

4.2 Simulation

The dense circle in the probability density matrix continues in this downward trend for the rest of the simulation. Furthermore, waves continuously wash over the images to simulate underwater currents. The "search" mechanism is represented by the small pixel and is moving over the surface in a Modified Expanding Square Search.

The matrix in the simulation is represented using a 100 x 100 metric, each square represents 100 meters. This was used to more closely demonstrate a submersible getting lost in the Ionian Sea. Furthermore, the geography of the area does not exceed 200 meters in depth because shipwreck locations do not typically exceed large distances.

On the difficulty matrix, lighter areas indicate regions that are more challenging to traverse through, such as zones with deeper terrain. It is also important to note that visibility and other geographic vehicles are combined into this same depth variable.

The speed of the search is also taken into account during the simulation which is assumed to be 18 km/hr while the speed of the underwater waves which can push the submersible have a speed of 6 km/hr, Please note that the speed of the waves affects but is not directly equal to the speed of the possible location of the submersible. In the simulation 2 distinct waves can be seen, moving over the search area at the same time that the submersible's probability blot only crosses the same area once.

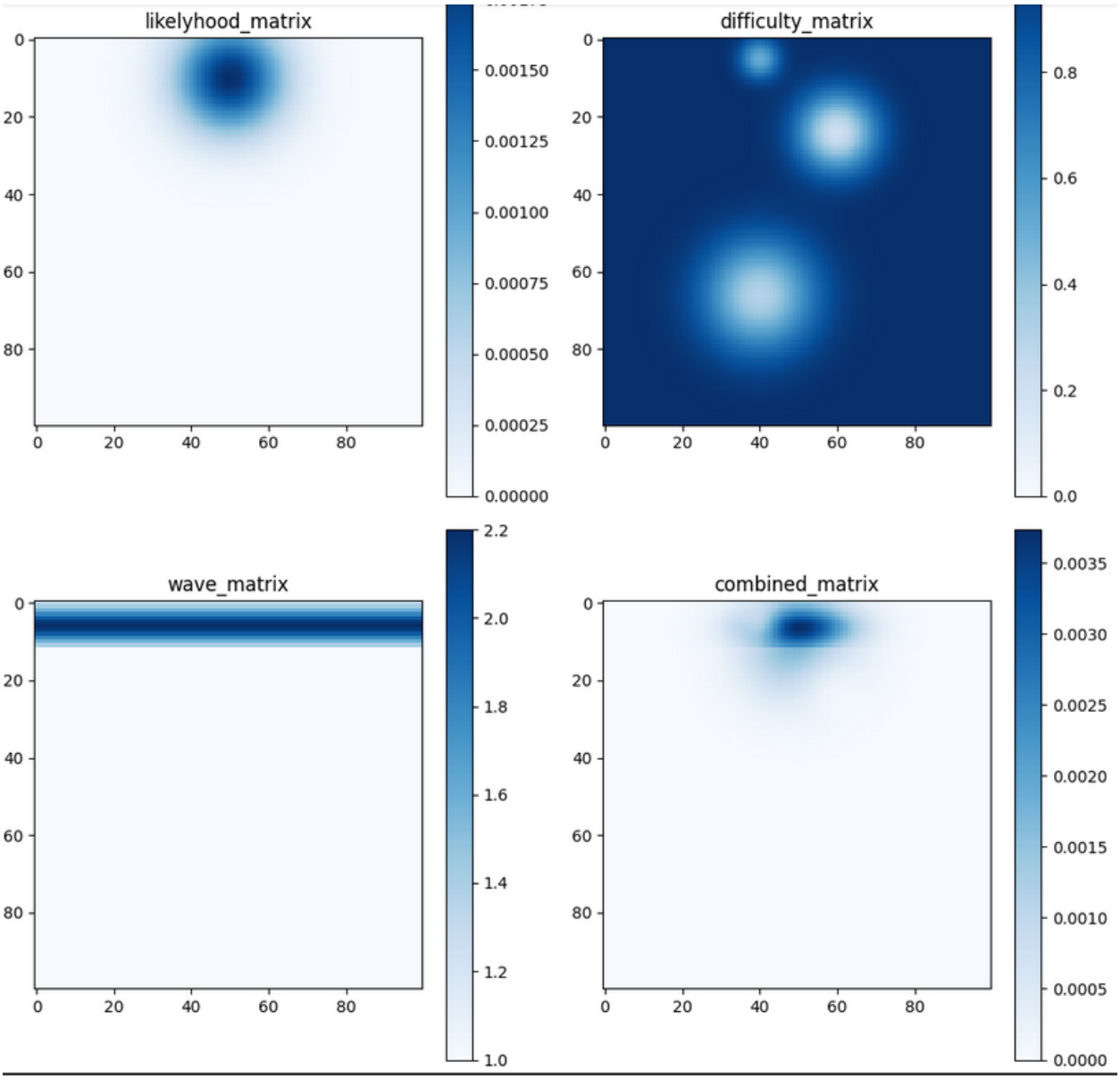
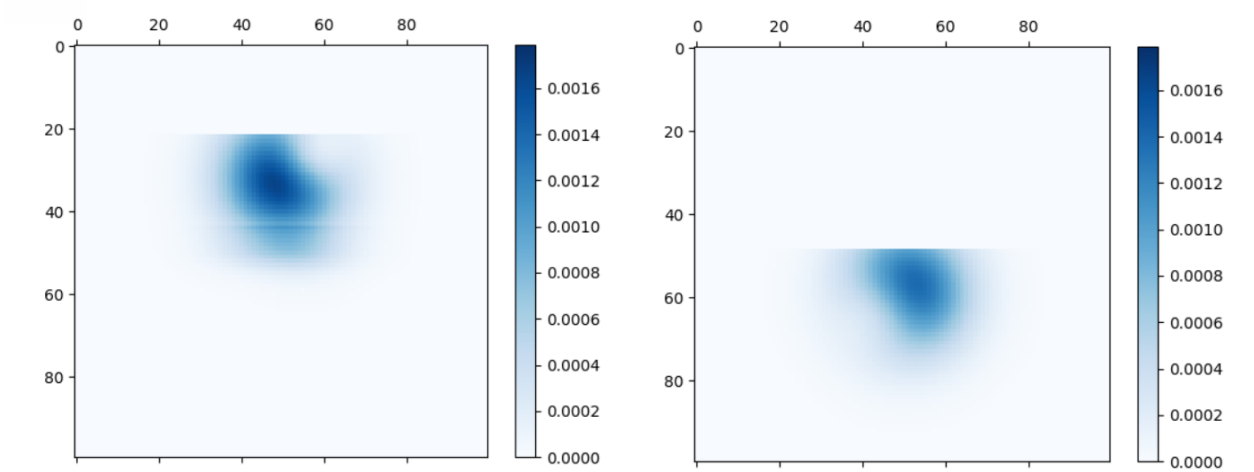


Figure 8: Components of Simulation with Realistic Values



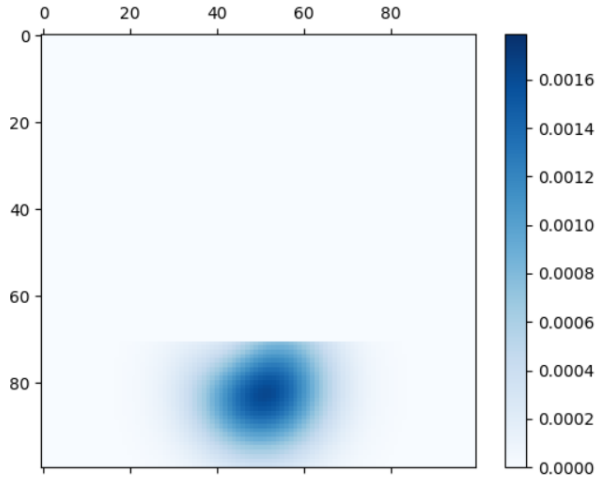


Figure 9: Results of the Simulation with Realistic Values Over Time

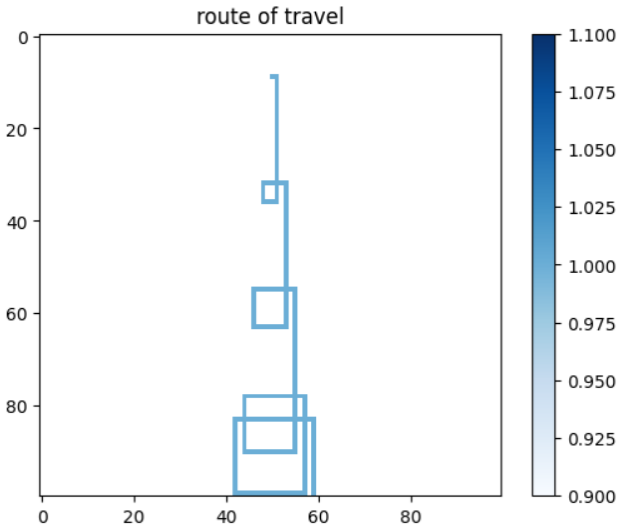


Figure 10: Realistic Search Route

5 Memo

To the Greek Government,

Our team has worked with Maritime Cruises Mini-Submersibles (MCMS) in order to develop safety precautions for the deployment of submersibles within the Ionian Sea. On behalf of MCMS, we have determined a way to predict the location of a submersible over time, determined the types of necessary search equipment in the case of an emergency, developed a model to determine the optimal search pattern and deployment locations for a search and rescue team, and considered how to utilize these methods in other vacation destinations. We will now provide an overview of the previously stated findings.

We first developed a model that would predict the location of a submersible given its last known location in the unlikely occurrence that a submersible malfunctions or loses communication with its host ship. This model is greatly influenced by Bayesian Search Theory and we use multivariate Gaussian distribution in order to determine the probability of a ship being in a certain location given its most recent known location. We also took into consideration how deep-sea search and rescue will vary greatly from search and rescue on land. In order to emulate this in our model, we accounted for how varying depths will affect the likelihood of finding a submersible in a certain location. We also considered underwater currents and incorporated how these waves can affect the movement of a submersible. Combining all these factors, our final model is able to predict where a given submersible would be located over time given its last known location, taking into consideration the location from a two-dimensional aerial perspective, depth, and underwater currents.

With this model there are some uncertainties with with the exact values that we can apply to our model, as it can be difficult to exactly gauge where a submersible is and under what condition it is in. In order to decrease these uncertainties as much as possible before an incident, it would be helpful for the submersible to periodically send back information about its current location and speed, as well as its depth and pressure to its host ship. To effectively do this, submersibles will need some kind of pressure sensor, a Doppler Velocity Log sensor, and an acoustic modem. This equipment is fairly standard for a submersible and will not cost a considerable amount.

Our model relies only on the use of sonar, the given information of the environment, and the information we receive from the submersible, which should already be included on every vessel and host ship in use. Therefore, we do not have any additional recommendations for search equipment for MCMS to carry on their host ships to deploy in case of emergency, as the system in place should already cover it. Search crews only require the technology to view the updating search pattern model and the diving equipment to actually reach the submersible. Similarly, we do not have any additional recommendations for equipment for the vessel to carry to aid in search efforts, as all the necessary equipment should be built into the submersible.

In order to aid in search efforts in the occurrence of a malfunctioning or lost submersible, we also created a model to predict the most optimal points of deployment and search pattern for search and rescue crews to utilize. Optimal points of deployment are based on our first model which predicts the location of the submersible over time. From there, we utilize an expanding square search in which search and rescue crews would start at a deployment location and travel along calculated legs, expanding every third leg in order to systematically cover an entire search

area. Our model, however, is modified to account for waves or deep sea currents pushing on the submersible. Instead of the datum of the search pattern remaining stationary, it moves over time as deep sea currents act on it. Consequentially, the expanding square pattern moves with it based on the predicted location of where the submersible is moving over time. This movement is based on known information about the deep sea currents in the area we predict the submersible to be located in.

Finally, we considered how to apply these models to other tourist destinations. Our model is very adaptable to different search areas and is not confined to the Ionian Sea. In order to extrapolate our model to a new tourist location, one would simply have to adjust the parameters based on the known information about the general sea floor and wave current patterns within a targeted region. The rest of the model relies directly on information that is unique to a given submersible, such as their last known location and direction. This makes our model fairly simple to apply to other locations, provided that we know information about the general region the submersible will be traveling in.

Ultimately, we have designed a model influenced by Bayesian Search Theory that would effectively predict the location of a submersible and a model that would find the optimal deployment locations and search pattern using a modified expanding square search pattern that adjusts based on the movement of the submersible by deep sea currents. We have also provided you with the necessary equipment needed to ensure that these models can be utilized to their fullest potential to ensure the quick rescue of a lost submersible. Finally, we have shown how these models can be extrapolated outside of this specific context and be used in tourist locations in the future. We hope you are able to see how with these plans in motion, the submersible excursions with MCMS will be a safe and secure experience for passengers.

Sincerely,

Team 2426376

6 References

References

[1] A complete guide to underwater navigation. (n.d.). *Nortek*. Nortek. https://www.nortekgroup.com/knowledge-center/wiki/new-to-subsea-navigation:˜text=Doppler

[2] Do, C. B. (2008, October 10). *The Multivariate Gaussian Distribution.* Stanford University. https://cs229.stanford.edu/section/gaussians.pdf. Accessed 2 February 2024

[3] Exploring our Fluid Earth. (n.d.). *Ocean Surface Currents.*https://manoa.hawaii.edu/exploringourfluidearth/physical/atmospheric-effects/ocean-surface-currents. Accessed 2 February 2024

[4] Iamsar search patterns. (2021, July 20). *Marine Teacher.* Marine Teacher. https://www.marineteacher.com/post/iamsar-search-patterns. Accessed 3 February 2024

[5] Geraga, M., Christodoulou, D., Eleftherakis, D., Papatheodorou, G., Fakiris, E., Dimas, X., . . . Ferentinos, G. (2020). Atlas of shipwrecks in Inner Ionian Sea (Greece): A Remote Sensing Approach. Retrieved from https://www.mdpi.com/2571-9408/3/4/67

[6] Ritvikmath. (2021, February 10). Bayesian Treasure Hunt: Data Science Code. YouTube. YouTube. https://www.youtube.com/watch? v=zp7TV6GypSA&ab_channel=ritvikmath. Accessed 2 February 2024

[7] SVB GmbH. (n.d.). Sonar for boats. Retrieved from https://www.svb24.com/en/guide/sonar-for-boats.html

[8] US Department of Commerce. (2012, October 29). Vessels: NOAA Office of Ocean Exploration and Research. *National Oceanic and Atmospheric Administration.* US Department of Commerce. https://oceanexplorer.noaa.gov/technology/subs/subs.html#: :text=Submersibles

[9] US Department of Commerce, N. O. and A. A. (2013, June 1). What is sonar? *NOAA's National Ocean Service.* Department of Commerce. https://oceanservice.noaa.gov/facts/sonar.html: :text=Active

7 Appendix

```
!apt install ffmpeg
!pip install plotly bokeh
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy
import time
from google.colab import output
import seaborn as sns
import matplotlib.animation as animation
from matplotlib import rc
from matplotlib.animation import FuncAnimation
from scipy.stats import multivariate_normal
# helper function to plot each matrix throughout the program
def plot_2d_matrix(matrix, title="graph"):
    plt.imshow( matrix, cmap="Blues", interpolation="nearest", vmin=np.min(matrix),vma
    plt.colorbar()
    plt.title(title)
    plt.show()
"""### Probability Density Function
Below we are generating a pdf that simulates the likelihood of finding the missing sub
"""
"""
n: int n defines n by n size of grid
center: int c defines the center of the circle where object was last seen
radius: int r defines the radius of of the circle, note the intensity of the circle fo
if custom_covariance is on then the size of the blot will increase with time affected
graph: bool graph determines if user wants to graph the output, defult is False
"""
def create_probability_distribution_grid(
    n, center, radius, custom_covariance=None, cov_mult=0.02, graph=False
):
    matrix = np.zeros((n, n))
    center_x, center_y = center
    x, y = np.meshgrid(np.arange(n), np.arange(n))
    # Calculate the Gaussian distribution with custom covariance if provided
    if custom_covariance is not None:
        covariance = np.eye(2) * ((radius + custom_covariance * cov_mult) ** 2)
    else:
        covariance = np.eye(2) * (radius**2)
    # Calculate the Gaussian distribution
    mv_normal = multivariate_normal(mean=[center_x, center_y], cov=covariance)
    prob_values = mv_normal.pdf(np.column_stack((x.flatten(), y.flatten())))
```

```
    prob_matrix = prob_values.reshape((n, n))
    # Normalize the matrix so that the values add up to 1
    matrix = prob_matrix / np.sum(prob_matrix)
    if graph:
        # Create a figure and axis
        fig, ax = plt.subplots()
        # Display the matrix with a heatmap and annotate with probability values
        cax = ax.matshow(matrix * 100, cmap="Blues", vmin=0, vmax=np.max(matrix) * 100
        # Add a colorbar to the plot
        cbar = fig.colorbar(cax)
        if n < 26:
            # Annotate with probability values
            for i in range(n_size):
                for j in range(n_size):
                    text = ax.text(j, i, f"{matrix[i, j]*100:.1f}",ha="center", va="ce
        # Show the plot
        plt.title("Probability Distribution")
        plt.show()
    return matrix
# Example usage with three different custom covariance values
n_size = 20
center_coords = (10, 10)
circle_radius = 5
custom_covariance_values = [1, 25, 45]
# Create subplots
fig, axs = plt.subplots(1, len(custom_covariance_values), figsize=(15, 5))
# Plot the three probability distributions with different custom covariance values
for i, custom_covariance in enumerate(custom_covariance_values):
    prob_dist = create_probability_distribution_grid(
        n_size, center_coords, circle_radius, custom_covariance=custom_covariance, cov
    axs[i].matshow(prob_dist * 100, cmap="Blues", vmin=0, vmax=np.max(prob_dist) * 100
    axs[i].set_title(f"Time = {custom_covariance}")
    for m in range(n_size):
        for n in range(n_size):
            axs[i].text(n, m, f"{prob_dist[m, n]*1000:.1f}", ha="center", va="center",
# Show the plots
plt.show()
"""
n: int for number of circles
circles: np array of the  form (x, y, r, w) where x and y determine position of center
for example a location with a higgher w value is harder to search
"""
def create_matrix(n, circles, graph=False):
    matrix = np.ones((n, n))
    for circle in circles:
```

```
        center_x, center_y, radius, likelihood = circle
        for i in range(n):
            for j in range(n):
                distance = np.sqrt((i - center_x) ** 2 + (j - center_y) ** 2)
                matrix[i, j] -= likelihood * np.exp(-((distance / radius) ** 2))
    if graph:
        plot_2d_matrix(matrix, "Likelihood of finding object if square is checked")
    return matrix
n_size = 50
circles = [(12, 12, 10, 0.5), (25, 28, 9, 0.8), (10, 35, 8, 0.4)]
difficulty_matrix = create_matrix(n_size, circles, True)
center_coords = (25, 25)
circle_radius = 10
prob_dist = create_probability_distribution_grid(
    n_size, center_coords, circle_radius, cov_mult=0.2, graph=False
)
prob_of_find_matrix = difficulty_matrix * prob_dist
plot_2d_matrix(prob_of_find_matrix, "probability of finding object")
"""### Creating Waves
These waves simulate the underwater current that pushes the submersile at an average p
"""
"""
waves
size: int n, defines n by n size grid
width_h/_v : int w, define the width of the wave
peak_h/_v : int p, define where the wave peaks relative to the width
set width and peak variable of eaither vertical or horizontal bar equal to 0 to not co
off_h/_v : int c, set close to peak values to create a wave with a more "edged" look
h_/v_dir  : char "u" for up, "d" for down for the horizontal wave, "l" for left, "r" f
"""
def waves(
    size, width_h=12, peak_h=6, width_v=12, peak_v=6, off_h=0, off_v=0,  h_dir="d", v_
    initial_matrix = np.zeros((size, size))
    # Create a figure and axis
    fig, ax = plt.subplots()
    # Display the initial matrix
    cax = ax.matshow(initial_matrix, cmap="Blues", vmin=0, vmax=1)
    # Add a colorbar to the plot
    cbar = fig.colorbar(cax)
    # Function to update the animation frames
    def update(frame):
        initial_matrix = np.zeros((size, size))
        for i in range(width_h):
            if h_dir == "d":
                initial_matrix[(frame + i) % size, :] = np.exp(-((i - peak_h - off_h)
```

```
            elif h_dir == "u":
                initial_matrix[(-frame - i) % size, :] = np.exp( -((i - peak_h - off_h
        for j in range(width_v):
            if v_dir == "r":
                initial_matrix[:, (frame + j) % size] += np.exp( -((j - peak_v - off_v
            elif v_dir == "l":
                initial_matrix[:, (-frame - j) % size] += np.exp( -((j - peak_h - off_
        # Update the data of the existing plot
        cax.set_array(initial_matrix)
    # Create an animation
    animation = FuncAnimation(fig, update, frames=range(50), interval=500)
    return animation
ani = waves(50)
# below is the part which makes it work on Colab
rc('animation', html='jshtml')
ani    # or HTML(anim.to_jshtml())
def shift_matrix_no_wrap(matrix, shift, axis=0):
    if axis == 0:  # Shift rows
        matrix = np.roll(matrix, shift, axis=axis)
        if shift > 0:  # Shift down
            matrix[:shift, :] = 0
        else:  # Shift up
            matrix[shift:, :] = 0
    elif axis == 1:  # Shift columns
        matrix = np.roll(matrix, shift, axis=axis)
        if shift > 0:  # Shift right
            matrix[:, :shift] = 0
        else:  # Shift left
            matrix[:, shift:] = 0
    return matrix
def move_likelyhood(likelyhood_matrix, frame, dir="d"):
    dynamic_shift = frame
    args = {
        "d": (dynamic_shift, 0),
        "l": (-dynamic_shift, 1),
        "u": (-dynamic_shift, 0),
        "r": (dynamic_shift, 1),
    }
    shift, axis = args[dir]
    return shift_matrix_no_wrap(likelyhood_matrix, shift, axis)
"""
wave
frame: int f, defines what frame the animation should be on
size: int n, defines n by n size grid
wave_intensity: float w_i, determines the intensity of the wave or the value on the wa
```

```
width_h/_v : int w, define the width of the wave
peak_h/_v : int p, define where the wave peaks relative to the width
set width and peak variable of eaither vertical or horizontal bar equal to 0 to not co
off_h/_v : int c, set close to peak values to create a wave with a more "edged" look
h_/v_dir  : char "u" for up, "d" for down for the horizontal wave, "l" for left, "r" f
"""
def wave( frame, size, wave_intensity=0.1, width_h=12, peak_h=6, width_v=0, peak_v=0,
):
    initial_matrix = np.zeros((size, size))
    for i in range(width_h):
        if h_dir == "d":
            initial_matrix[(frame + i) % size, :] = np.exp(-((i - peak_h - off_h) ** 2
        elif h_dir == "u":
            initial_matrix[(-frame - i) % size, :] = np.exp( -((i - peak_h - off_h) **
    for j in range(width_v):
        if v_dir == "r":
            initial_matrix[:, (frame + j) % size] += np.exp(-((j - peak_v - off_v) **
        elif v_dir == "l":
            initial_matrix[:, (-frame - j) % size] += np.exp(-((j - peak_h - off_h) **
    initial_matrix = initial_matrix + 1
    initial_matrix[initial_matrix > 1] = initial_matrix[initial_matrix > 1] * (
        wave_intensity + 1
    )
    return initial_matrix
wave_matrix = wave(
    10,50,wave_intensity=0.1,width_h=12, peak_h=6,width_v=0, peak_v=0, off_h=0,  off_v
)
plot_2d_matrix(wave_matrix, "image of a wave")
"""
This function updates the probability distribution matrix after a square is checked as
x and y define the point which is searched
important!!! returns the combination of the 3 matrices
"""
def search_update(likelyhood_matrix, difficulty_matrix, wave_matrix, x, y, plot=False)
    # checking the single point
    p_is_there_and_not_found = ( likelyhood_matrix[x][y] * (1 - difficulty_matrix[x][y

    # updating all other point values
    for xs, ys in np.ndindex(likelyhood_matrix.shape):
        if xs != x and ys != y:
            likelyhood_matrix[xs][ys] = likelyhood_matrix[xs][ys] / (
                (1 - likelyhood_matrix[xs][ys])
                + likelyhood_matrix[xs][ys]
                * (1 - likelyhood_matrix[xs][ys])
                * difficulty_matrix[xs][ys]
```

```
            )
    if plot:
        plot_2d_matrix(likelyhood_matrix, "Just the updated likelihood matrix")
    return likelyhood_matrix * difficulty_matrix * wave_matrix
"""### Searching Algorithm
We use a modified Expanding Square Search to find the missing submersile here.
"""
"""
Algorithm for searching
"""
def square_search(start, num_legs, length, increment, max_x, max_y, s=25, dir="d"):
    d = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    current = start
    yield current
    for i in range(num_legs):
        dx, dy = d[i % 4]
        change = 0
        match (dir):
            case "d":
                if dy == 1 and dx == 0:
                    change += s
            case "u":
                if dy == -1 and dx == 0:
                    change -= s
            case "l":
                if dy == 0 and dx == -1:
                    change -= s
            case "r":
                if dy == 0 and dx == 1:
                    change += s
        for _ in range((length + i * increment) + change):
            next_p = (current[0] + dy, current[1] + dx)
            if 0 <= next_p[0] < max_x and 0 <= next_p[1] < max_y:
                current = next_p
                yield next_p
"""### Model and Simulation"""
def prob_dist_update(likelyhood_matrix, difficulty_matrix, frames_n):
    p = difficulty_matrix * likelyhood_matrix
    # Create a figure and axis
    fig, ax = plt.subplots()
    # Display the initial matrix
    cax = ax.matshow(p, cmap="Blues")
    # Add a colorbar to the plot
    cbar = fig.colorbar(cax)
    start = np.unravel_index(p.argmax(), p.shape)
```

```
    find_point = square_search( start, 18, 1, 1, likelyhood_matrix.shape[0], likelyhoo
    search_matrix = np.zeros(likelyhood_matrix.shape)
    # Function to update the animation frames
    def update(frame):
        try:
            for _ in range(1, 5):
                (x, y) = next(find_point)
                search_matrix[x][y] = 1
        except:
            (x, y) = (0, 0)
        new_likelyhood_matrix = create_probability_distribution_grid( n=len(likelyhood
        wave_matrix = wave((frame * 2) % int(frames_n),len(likelyhood_matrix[0]), wave
        )
        p = search_update( move_likelyhood(new_likelyhood_matrix, frame), difficulty_m

        # Update the data of the existing plot
        cax.set_array(p)

    # Create an animation
    animation = FuncAnimation(fig, update, frames=range(frames_n), interval=1000)
    return animation, search_matrix
# Example usage:
n_size = 100
# Define circles as (center_x, center_y, radius, likelihood of not finding object)
circles = [(5, 40, 5, 0.5), (24, 60, 10, 0.8), (66, 40, 14, 0.7)]
no_circle = [(0, 0, 0, 0)]
# last know location of the boat
center_coords = (50, 10)
# inital seach space is within a one kilometer radious of the last known location
circle_radius = 10
likelyhood_matrix = create_probability_distribution_grid(
    n_size, center_coords, circle_radius
)
wave_matrix = wave(0, len(likelyhood_matrix[0]))
difficulty_matrix = create_matrix(n_size, circles)
prob_of_find_matrix = likelyhood_matrix * difficulty_matrix * wave_matrix
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
# Display the first matrix in the first subplot
axs[0, 0].imshow(likelyhood_matrix, cmap="Blues")
axs[0, 0].set_title("likelyhood_matrix")
# Display the second matrix in the second subplot
axs[0, 1].imshow(difficulty_matrix, cmap="Blues", vmin=0, vmax=1)
axs[0, 1].set_title("difficulty_matrix")
# Display the third matrix in the third subplot
axs[1, 0].imshow(wave_matrix, cmap="Blues", vmin=np.min(wave_matrix), vmax=np.max(wave
```

```
axs[1, 0].set_title("wave_matrix")
# Display the fourth matrix in the fourth subplot
axs[1, 1].imshow(
    prob_of_find_matrix, cmap="Blues", vmin=0, vmax=np.max(prob_of_find_matrix)
)
axs[1, 1].set_title("combined_matrix")
print(np.sum(prob_of_find_matrix))
# Add a colorbar to each subplot
for ax_row in axs:
    for ax in ax_row:
        cbar = fig.colorbar(ax.images[0], ax=ax, orientation="vertical")
plt.tight_layout()
plt.show()
anim, search_matrix = prob_dist_update(likelyhood_matrix, difficulty_matrix, 100)
1# below is the part which makes it work on Colab
rc('animation', html='jshtml')
anim    # or HTML(anim.to_jshtml())
masked = np.ma.masked_where(search_matrix == 0, search_matrix)
plot_2d_matrix(masked, "route of travel")
# Example usage:
n_size = 100
# Define circles as (center_x, center_y, radius, likelihood of not finding object)
circles = [(5, 40, 5, 0.5), (24, 60, 10, 0.8), (66, 40, 14, 0.7)]
# intial location of the boat
center_coords = (50, 10)
# inital seach space is within a one kilometer radious of the last known location
circle_radius = 10
likelyhood_matrix = create_probability_distribution_grid(n_size, center_coords, circle
wave_matrix = wave(0, len(likelyhood_matrix[0]))
difficulty_matrix = create_matrix(n_size, circles)
prob_of_find_matrix = likelyhood_matrix * difficulty_matrix * wave_matrix
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
# Display the first matrix in the first subplot
axs[0, 0].imshow(likelyhood_matrix, cmap="Blues")
axs[0, 0].set_title("likelyhood_matrix")

# Display the second matrix in the second subplot
axs[0, 1].imshow(difficulty_matrix, cmap="Blues", vmin=0, vmax=1)
axs[0, 1].set_title("difficulty_matrix")
# Display the third matrix in the third subplot
axs[1, 0].imshow(wave_matrix, cmap="Blues", vmin=np.min(wave_matrix), vmax=np.max(wave
axs[1, 0].set_title("wave_matrix")
# Display the fourth matrix in the fourth subplot
axs[1, 1].imshow(prob_of_find_matrix, cmap="Blues", vmin=0, vmax=np.max(prob_of_find_m
axs[1, 1].set_title("combined_matrix")
```

```
print(np.sum(prob_of_find_matrix))
# Add a colorbar to each subplot
for ax_row in axs:
    for ax in ax_row:
        cbar = fig.colorbar(ax.images[0], ax=ax, orientation="vertical")
plt.tight_layout()
plt.show()
anim = prob_dist_update(likelyhood_matrix, difficulty_matrix, 100)
# below is the part which makes it work on Colab
rc('animation', html='jshtml')
anim    # or HTML(anim.to_jshtml())
```