Detection of Darkships: Overview, Simulation and Testing

Violoni Gianluca[1]

*gianluca.violoni@unitn.it

Abstract: Detecting Dark Ships, vessels operating without active transponders, is a critical task for maritime surveillance and security. These ships do not transmit their position via the Automatic Identification System (AIS), making them invisible to conventional tracking methods. This project aims to implement a customizable simulation framework and test algorithms for the acoustic detection of dark ships using hydrophones, underwater microphones capable of continuously monitoring ocean sounds.

1. Introduction

Maritime security faces a critical challenge in the detection and monitoring of "dark ships" - vessels that deliberately disable or tamper with tracking systems to avoid detection by conventional monitoring methods, including Automatic Identification System (AIS) data manipulation, position spoofing, and strategic system shutdowns when entering sensitive or restricted areas. These vessels represent a significant threat to maritime surveillance and security operations worldwide, as they often engage in illicit activities such as illegal fishing, smuggling, sanctions evasion, human trafficking, and unauthorized territorial intrusions. A notable example occurred in 2023, where tankers were found to be deceiving authorities by transmitting false positions while secretly docking at terminals in Russia to transport oil to China, circumventing international sanctions.
Current approaches to dark ship detection involve multiple technologies including satellite-based monitoring (using optical imaging, Synthetic Aperture Radar, and radio frequency detection), conventional radar systems, and various signal intelligence methods. However, these methods face significant limitations including weather dependence, coverage gaps, latency issues, and the high operational costs associated with continuous wide-area surveillance. Passive acoustic monitoring offers distinct advantages as a complementary technology for the detection of dark ships. Hydrophones - underwater microphones that detect acoustic signals - can continuously monitor large maritime areas without requiring any cooperation from the vessels being observed.
The underwater acoustic channel presents unique challenges compared to terrestrial environments. Sound propagation in water is affected by various factors including temperature gradients, salinity variations, and depth-dependent changes in acoustic properties. Additionally, underwater environments are characterized by complex ambient noise from both natural sources (marine life, weather phenomena) and anthropogenic activities (other vessels, industrial operations).
This project focuses on implement and test algorithms for the passive acoustic detection of dark ships using hydrophones, which capture underwater sound patterns generated by vessel machinery, propellers, and hull interactions with the surrounding water.
To facilitate the design and evaluation of these detection algorithms, a configurable simulation environment has been specifically developed. This environment enables the modeling of realistic acoustic propagation conditions, various ship behaviors, and environmental factors such as ambient noise and bathymetry. It also supports different hydrophone configurations, including spatial layouts and sensitivity parameters, allowing the conduction of controlled experiments.

2. Noise Factors Affecting AIS Transmission

The Automatic Identification System (AIS) is a widely used method for tracking vessels, but several factors can affect its reliability. The following are the main factors influencing AIS transmission:

- **Equipment Issues**: Poor quality equipment or poor installation can cause gaps in AIS transmissions.
- **Satellite Receiver Saturation**: In high-density areas, satellite receivers can become saturated, leading to a loss of AIS data.
- **Signal Interference**: Environmental conditions such as weather and cloud coverage can shield ships from detection.
- **Intentional Disabling**: Vessels may intentionally turn off their AIS transponders to avoid observation, often linked to illegal activities.
- **AIS Message Collision**: In congested waters, signal collision and interference can cause lost signals.
- **Small Vessel Regulations**: Smaller vessels may not be legally required to transmit AIS, making their detection more difficult.
- **Spoofing**: Ships can transmit incorrect AIS information, including their identity or position, often to deceive monitoring systems.

3. Potential Approaches for Detecting Dark Ships

The following approaches have been explored for identifying dark ships in maritime surveillance:

3.1. Spatial Statistical Models

These models help identify gaps in AIS transmission and calculate the probability that a gap is due to intentional disabling. For example, a Generalized Additive Model (GAM) can model space-time variations in AIS gaps to distinguish between intentional disabling and other issues.

3.2. Multi-Sensor Data Fusion

Combining data from various sources, such as satellite imagery and radio frequency sensors, can help detect vessels that are not transmitting AIS. Common techniques include:

- **SAR (Synthetic Aperture Radar)**: SAR does not require cooperation from ships and can detect dark ships under all weather conditions.
- **Electro-Optical/Infrared (EO/IR) Imagery**: Optical imagery can detect ships regardless of their AIS status but may be limited by weather conditions and darkness.
- **Radio Frequency (RF) Sensors**: RF sensors can detect vessels that have turned off their AIS transponders.

3.3. Anomaly Detection Algorithms

Anomaly detection techniques, such as trajectory analysis and machine learning models, can help identify suspicious vessel behavior. Common methods include:

- **Trajectory Analysis**: Using machine learning to detect deviations from normal vessel behavior, based on kinematic data such as speed and course.
- **Clustering Techniques**: Identifying abnormal data by comparing AIS data with kinematic-based estimations.

- **Machine Learning**: Self-supervised deep learning models can predict whether an AIS message should be received, flagging deviations as suspicious.
- **Pattern-of-Life (PoL) Analysis**: This method analyzes vessel behavior patterns over time to detect unusual activities.

3.4. AIS Signal Analysis

Several methods can analyze AIS signal strength and patterns to identify intentional on/off switching:

- **Received Signal Strength Indicator (RSSI)**: Analyzing RSSI at AIS base stations can help detect intentional AIS transponder switching.
- **Channel Memory**: Hidden Markov Models (HMMs) can detect switching patterns by analyzing signal transmission behavior.

3.5. Cross-Referenced Data

Combining AIS data with other sensor information, such as radar data, can enhance detection reliability:

- **Radar Data**: Radar can provide real-time vessel trajectories, which can be cross-referenced with AIS data to identify spoofing or missing transmissions.

4. Specific Detection Techniques

There are several specific techniques used to detect dark ships, including advanced machine learning algorithms and image processing methods:

4.1. YOLO (You Only Look Once) and SAHI

Combining YOLO with the SAHI module improves the detection of small ships in large SAR images. SAHI works by slicing large images into smaller regions, making small targets more prominent.

4.2. CFAR (Constant False Alarm Rate) Detectors

CFAR algorithms are used to detect ships in SAR images by modeling background clutter. An improved version, AIS-RCFAR, uses AIS data to trim outliers and improve detection accuracy.

4.3. Dead Reckoning (DR)

Comparing a GNSS position with a position calculated using DR can help detect GNSS spoof attacks and identify vessels that do not transmit AIS.

5. Challenges and Considerations

Detecting dark ships involves addressing several challenges, including:

- **False Positives vs. False Negatives**: Balancing the risk of incorrectly identifying a normal vessel as a dark ship (false positives) versus missing a dark ship (false negatives) is crucial for accurate detection.
- **Small Ship Detection**: Smaller vessels are often missed in both imagery and AIS data, making their detection more difficult.

- **Data Quality**: AIS data can be unreliable, with errors in vessel information such as size and type.
- **Adaptive Behavior**: Malicious actors may adjust their behavior to avoid detection, requiring adaptable detection systems.
- **Privacy Concerns**: AIS surveillance raises concerns about privacy, as it allows continuous tracking of ships and their activities.
- **Standardized Datasets**: A lack of labeled datasets with ground truth information makes it difficult to compare and validate different detection methods.

6. Passive Acoustic Monitoring with Hydrophones

Hydrophones are underwater microphones that capture acoustic signals in the marine environment. They enable passive acoustic monitoring by recording ambient noise, ship engine sounds, and other underwater activities without the need for active emissions. In the context of dark ship detection, hydrophones offer a complementary source of information to satellite and radar-based systems, particularly when AIS signals are missing or intentionally disabled.

6.1. Advantages of Using Hydrophones

- **Passive Detection**: No active emission; stealthy and non-intrusive.
- **All-Weather Capability**: Unaffected by atmospheric or sea surface conditions.
- **Continuous Monitoring**: Hydrophones can record data 24/7 without interruption.
- **Long-Range Detection**: Low-frequency sounds can travel long distances underwater, allowing early detection of approaching vessels.

6.2. Challenges in Hydrophone-Based Detection

- **Ambient Noise**: Natural sounds (e.g., waves, rain, marine fauna) and anthropogenic noise (e.g., other ships) can mask vessel signals.
- **Multipath Propagation**: Acoustic signals may reflect off the sea surface or seabed, complicating signal analysis.
- **Source Localization**: Estimating the exact position of a vessel and differentiating between vessel types requires multiple hydrophones and complex algorithms.

7. Simulation Framework

The developed simulation framework is organized into three distinct stages, each addressing a specific phase of the dark ship detection pipeline:

- **Simulation Module** — This component generates a dynamic virtual maritime environment. It models ship trajectories, environmental conditions, and the propagation of underwater acoustic signals. As the simulation progresses, relevant data such as ship positions, hydrophone observations, and acoustic pressure fields are computed and stored in an output file. This file serves as the ground truth dataset for subsequent processing stages.
- **Tracking Module** — In this stage, the data produced by the simulation module is processed by acoustic detection and tracking algorithms. The tracking module analyzes the hydrophone observations to infer ship positions and behaviors over time. The results, including both raw observations and tracking outputs, are compiled into a structured output file used for visualization and further evaluation.

- **Visualization and Analysis Tool** — This component processes the output produced by the tracking module and generates a variety of informative plots and statistics. It allows users to visualize ship trajectories, hydrophone positions, detection events, and error metrics through time-series plots, spatial maps, and summary charts. The tool supports in-depth exploration and evaluation of the tracking algorithms in a flexible and scriptable environment, making it ideal for research and prototyping purposes.

This modular architecture allows for flexibility in testing different algorithmic approaches, simulation scenarios, and visualization strategies. It also supports reproducibility and systematic evaluation by separating data generation from inference and rendering.

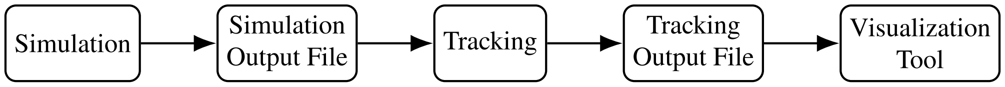

Fig. 1. Simulation-to-Visualization processing pipeline.

7.1. Configuration

The simulation is initialized through a YAML configuration file, which defines all parameters required to generate the environment, simulate ship movements, and specify sensor behavior. The main configuration sections are as follows:

- **Paths and general configuration**:
  - `output_path`: the output path where the simulation will store its results.
  - `name`: identification name for the simulation.
- **Environment parameters**:
  - `area`: the geographic region to be simulated, defined as `[lat_min, lat_max, lon_min, lon_max]`.
  - `bathymetry_path`: the path to a NetCDF file containing bathymetric data, which is used to simulate realistic underwater sound propagation influenced by seabed topography.
  - `toa_variance`: array of variance values for the time of arrival, used to simulate different levels of noise in time measurements.
- **Hydrophone configuration**:
  - `hydrophones`: a list of manually placed hydrophones, each defined by geographic coordinates and depth.
  - `noise_level`: the ambient underwater noise level, expressed in dB re 1 μPa, which affects the detectability of ships.
  - `num_random`: the number of additional hydrophones to be generated randomly.
  - `max_range_range`: range for the maximum detection range of randomly generated hydrophones, in km.
  - `depth_range`: depth interval for randomly generated hydrophones, in meters.
- **Ship configuration**:
  - `ais_ships`: manually defined ships with active AIS, each specified by position, speed, depth, and heading.

- dark_ships: manually defined "dark" ships, which do not transmit AIS, defined with the same attributes.
- num_random_ais_ships: number of AIS ships to be generated randomly.
- num_random_dark_ships: number of dark ships to be generated randomly.
- speed_range: speed interval for random ship generation, in knots.
- depth_range: depth interval for random ship generation, in meters.

An example of configuration file is reported below.

```yaml
output_path: "/Users/gean/Documents/Unitn/Thesis/darkships_detection/output"
name: "test"

environment:
  area: [10, 12, 83.5, 85.5]  # lat_min, lat_max, long_min, long_max
  bathymetry_path: "./bathymetry/bathymetry_sample.nc"
  toa_variance: [1.0e-5, 1.0e-4, 1.0e-3, 1.0e-2, 1.0e-1, 1.0]

hydrophones_config:
  # DATA FOR MANUAL HYDROPHONES GENERATION
  hydrophones:
      - coordinates: [11.0, 84.0]      # Hydrophone 1
        depth: 20                      # [m]
      - coordinates: [11.03, 84.0]     # Hydrophone 2
        depth: 20                      # [m]
      - coordinates: [11.03, 84.03]    # Hydrophone 3
        depth: 20                      # [m]
      - coordinates: [11.0, 84.03]     # Hydrophone 4
        depth: 20                      # [m]
  noise_level: 2.0                     # [dB re 1 µPa]

  # DATA FOR RANDOM HYDROPHONES GENERATION
  num_random: 0                        # Generate random hydrophones
  max_range_range: [30, 50]            # [km]
  depth_range: [0, 35]                 # [m]

ships_config:
  dark_ships:                          # Manual dark ships
    - coordinates: [11.032, 84.015]    # Dark Ship 1
      speed: 10                        # kt
      depth: 20                        # [m]
      heading: 180                     # [°]

  # DATA FOR RANDOM SHIPS GENERATION
  num_random_ais_ships: 0              # Generate random ais ships
  num_random_dark_ships: 0             # Generate random dark ships
  speed_range: [5,20]                  # [kt]
  depth_range: [0, 25]                 # [m]

```

This configuration approach provides a high degree of flexibility, allowing for both precise control through manual definition and broad scenario testing through random generation. The structure ensures reproducibility and scalability for experiments that involve different maritime scenarios and detection strategies.

7.2. Underwater Sound Propagation

To simulate acoustic propagation and model how sound attenuates with distance and depth, the simulation uses the Bellhop acoustic ray-tracing model. Bellhop takes into account bathymetry and other environmental parameters to calculate the *transmission loss* and *time of arrival* between each ship and hydrophone pair. This allows the system to estimate the *received sound pressure level* in each hydrophone, given the level and position of the source of the ship.
The simulation runs with a range of TOA variance values to analyze how measurement errors impact the detection and localization accuracy under different noise conditions, providing a more realistic assessment of the system's performance in real-world environments.

The simulated readings at the hydrophones are influenced by the following.

- The number and proximity of vessels (both AIS and dark).
- Their source levels.
- Environmental factors such as depth, shape of the seabed, and background noise.

8. DARKSHIP TRACKING APPROACHES USING HYDROPHONES

8.1. Weighted Centroid Localization (WCL)

Weighted Centroid Localization is a geometric technique for source localization based on the assumption that signal strength (or in this case, pressure differential) diminishes with distance from the source. Each hydrophone contributes to the estimated position of the source proportionally to the measured acoustic intensity.

Given a set of N hydrophones with coordinates (ϕ_i, λ_i) and corresponding measured pressure differences Δ_i, the estimated position $(\hat{\phi}, \hat{\lambda})$ is computed as:

$$\hat{\phi} = \frac{\sum_{i=1}^{N} \phi_i \cdot \Delta_i}{\sum_{i=1}^{N} \Delta_i} \tag{1}$$

$$\hat{\lambda} = \frac{\sum_{i=1}^{N} \lambda_i \cdot \Delta_i}{\sum_{i=1}^{N} \Delta_i} \tag{2}$$

This estimation can be implemented as follows in Python:

```python
1  def weighted_centroid_localize(hydrophones: list[Hydrophone]):
2      deltas = [h.compute_pressure_delta() for h in hydrophones]
3      delta_tot = sum(deltas)
4
5      if delta_tot == 0:
6          return (0.0, 0.0)
7
8      weighted_lat = sum(
9          h.coord.latitude * d for h, d in zip(hydrophones, deltas)
10     ) / delta_tot # as (1)
11
12     weighted_lon = sum(
13         h.coord.longitude * d for h, d in zip(hydrophones, deltas)
14     ) / delta_tot # as (2)
15
16     return weighted_lat, weighted_lon
```

In this implementation, lines `[8-10]` and `[12-14]` respectively compute the latitude and longitude of the estimated source position by applying equations (1) and (2).

Advantages. The main advantages of WCL are its simplicity and speed, making it suitable for real-time applications or as an initialization step for more complex tracking algorithms (e.g., particle filters).

Limitations. The reliability of Weighted Centroid Localization significantly decreases in complex underwater environments where signal propagation is strongly affected by factors such as bathymetry, thermoclines, seabed composition, and multipath reflections. In such conditions, the measured pressure deltas are no longer directly correlated with distance to the source, violating the core assumption of this method. As a result, the estimated position may become highly inaccurate or biased. Therefore, WCL is best suited for relatively homogeneous environments or as an initial estimate to be refined by more robust methods.

8.2. Time Difference of Arrival (TDOA) Localization

Time Difference of Arrival (TDOA) is a well-established technique for passive source localization based on the differences in signal arrival times across a set of spatially distributed sensors. In this context, hydrophones record the acoustic pressure waveform emitted by a ship, and the algorithm estimates the source's position by solving a system derived from the time delays between sensors.

The implementation follows the closed-form localization algorithm described in [1], which assumes that the positions of the hydrophones and the speed of sound in water v are known. One hydrophone (typically the first) is used as a reference. For each other hydrophone, the time difference $\tau_i = t_i - t_1$ is used to build a linear system based on the following equations:

$$A_i = \frac{1}{v\tau_i}(-2x_1 + 2x_i) - \frac{1}{v\tau_2}(-2x_1 + 2x_2) \tag{3}$$

$$B_i = \frac{1}{v\tau_i}(-2y_1 + 2y_i) - \frac{1}{v\tau_2}(-2y_1 + 2y_2) \tag{4}$$

$$D_i = v(\tau_i - \tau_2) + \frac{1}{v\tau_i}(x_1^2 + y_1^2 - x_i^2 - y_i^2) - \frac{1}{v\tau_2}(x_1^2 + y_1^2 - x_2^2 - y_2^2) \tag{5}$$

By stacking the coefficients A_i, B_i and D_i into matrices, the final source position (x, y) in UTM coordinates is estimated by solving the system:

$$\begin{bmatrix} A_3 & B_3 \\ \vdots & \vdots \\ A_i & B_i \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -D_3 \\ \vdots \\ -D_i \end{bmatrix} \tag{6}$$

using the Moore-Penrose pseudoinverse.

The algorithm is implemented in Python as follows:

```python
def tdoa_localize(self, v_water=None):
    """
    Implement TDOA localization using the algorithm from the paper [1].
    Returns the estimated position in geographic coordinates (lat, lon).
    """
    # Use class-level sound speed if none provided
    if v_water is None:
        v_water = self.v

    s = self.get_utm_coords()

    # First hydrophone (reference)
    s1 = s[0]
    x1, y1 = s1

    # Second hydrophone
    if len(s) < 2:
        raise ValueError("Need at least two hydrophones for TDOA")

    s2 = s[1]
    x2, y2 = s2

    # Calculate time differences (TDOA)
    # Reference time is from first hydrophone
    t1 = self.hydrophones[0].observed_pressure[-1]["toa"]

    # Pre-calculate tau_2 (time difference between second and first hydrophone)
    tau_2 = self.hydrophones[1].observed_pressure[-1]["toa"] - t1

    A = []
    D = []

    # Process hydrophones from index 2 onwards
    for i in range(2, len(self.hydrophones)):
        xi, yi = s[i]

        # Calculate time difference between current hydrophone and reference
        tau_i = self.hydrophones[i].observed_pressure[-1]["toa"] - t1

        # Guard against division by zero
        if abs(tau_i) < 1e-10 or abs(tau_2) < 1e-10:
            continue

        # Computing A_i, B_i, D_i as per equations (3)-(5)
        Ai = (1 / (v_water * tau_i)) * (-2 * x1 + 2 * xi) - (
            1 / (v_water * tau_2)
        ) * (-2 * x1 + 2 * x2)

        Bi = (1 / (v_water * tau_i)) * (-2 * y1 + 2 * yi) - (
            1 / (v_water * tau_2)
        ) * (-2 * y1 + 2 * y2)

        Di = (
            v_water * tau_i
            - v_water * tau_2
            + (1 / (v_water * tau_i)) * (x1**2 + y1**2 - xi**2 - yi**2)
            - (1 / (v_water * tau_2)) * (x1**2 + y1**2 - x2**2 - y2**2)
        )

        A.append([Ai, Bi])
        D.append(-Di)

    # Check if we have enough measurements to compute position
    if len(A) == 0:
        raise ValueError("Not enough valid measurements to compute position")
```

```
67        # Convert to numpy arrays
68        A = np.array(A)
69        D = np.array(D)
70
71        # Moore-Penrose pseudoinverse to solve A @ [x, y] = D as in equation (6)
72        position_utm = np.linalg.pinv(A) @ D
73
74        # Convert UTM coordinates back to geographic coordinates
75        lon, lat = self.utm_to_geo(position_utm[0], position_utm[1])
76
77        return (lat, lon)
```

Lines `[44-61]` correspond to equations (3), (4) and (5), while line `[72]` solves the linear system (6).

Advantages. TDOA offers high localization accuracy when time-of-arrival estimates are precise and hydrophones are well-distributed spatially. It does not assume a simple attenuation model and instead leverages time synchronization, making it more robust in complex acoustic environments.

Limitations. The accuracy of this method strongly depends on synchronization between hydrophones and the precision of time-of-arrival estimates. Environmental noise, signal reflections, and multipath propagation can distort TOA measurements, introducing error. Moreover, the algorithm requires at least three hydrophones (with two providing valid TOA deltas), which may not always be available.

8.3. TDOA Acoustical Location Based on Majorization-Minimization Optimization (T-MM)

The T-MM localization algorithm employs an iterative optimization approach based on the Majorization-Minimization framework to solve the non-convex TDOA localization problem. This method aims to find a robust estimate of the source position by iteratively minimizing a cost function derived from the squared differences between measured and predicted distances.

Initial Point Selection (Algorithm 2) Finding a suitable initial point is crucial for the convergence of the MM algorithm. Following Algorithm 2 from [1], the initial point $\mathbf{x}_0$ is selected by minimizing the cost function

$$f(\mathbf{x}) = \sum_{i=1}^{N} \left(\|\mathbf{x} - \mathbf{s}_i\| - d_i\right)^2 , \tag{7}$$

where $\mathbf{s}_i$ are the known hydrophone coordinates in UTM and the distances d_i represent the difference in distance between the acoustic source and each hydrophone relative to a reference hydrophone (typically the first one). These values are derived from the Time Difference of Arrival (TDOA) measurements.

Specifically, given the measured time of arrival t_i at each hydrophone i, and t_1 at the reference hydrophone, the time differences are calculated as:

$$\text{TDOA}_i = t_i - t_1$$

Multiplying these time differences by the speed of sound in water v converts them into distance differences:

$$d_i = \text{TDOA}_i \times v = (t_i - t_1) \times v$$

The procedure consists of:

1. Evaluating $f(\mathbf{s}_k)$ for each hydrophone position $\mathbf{s}_k$ and selecting the k that minimizes it (equation (8) in [1]).
2. Computing the gradient $\mathbf{g}_k(\mathbf{s}_k)$ (as in equation (29) in [1]ì) to determine the search direction.
3. Performing a backtracking line search along this direction to find an optimal step size t, ensuring $f(\mathbf{s}_k + t\mathbf{v}_0) < f(\mathbf{s}_k)$.

This process returns the initial estimate $\mathbf{x}_0$ used in the iterative MM algorithm.

```python
def _find_initial_point(self):
    """
    Algorithm 2 from the paper: Find an optimal initial point for the MM algorithm
    Reference: Minimizing equation (7)
    """

    s = self.get_utm_coords()
    distances = self._compute_distances()

    # Step 1: Find index k for which f(s_k) is minimum
    # Equation (8) from paper [1]: f(x) = sum_i (||x - s_i|| - d_i)^2
    min_f_val = float("inf")
    min_idx = 0

    for k in range(len(s)):
        f_val = 0
        for i in range(len(s)):
            # d_i is distances[i], ||s_k - s_i|| is Euclidean distance
            dist = np.linalg.norm(s[k] - s[i])
            f_val += (dist - distances[i]) ** 2

        if f_val < min_f_val:
            min_f_val = f_val
            min_idx = k

    # Step 2: Calculate gradient g_k(s_k)
    # Equation (29) from paper [1]
    g_k = np.zeros(2)
    for i in range(len(s)):
        if i != min_idx:
            s_k_minus_s_i = s[min_idx] - s[i]
            dist = np.linalg.norm(s_k_minus_s_i)

            if dist > 1e-10:  # Avoid division by zero
                g_k += 2 * (dist - distances[i]) * s_k_minus_s_i / dist

    # Step 3: Determine direction vector v_0
    # Use equation in Algorithm 2 from paper [1]
    if np.linalg.norm(g_k) > 1e-10:
        v_0 = -g_k / np.linalg.norm(g_k)
    else:
        # If gradient is zero or very small, use arbitrary unit vector
        v_0 = np.array([1.0, 0.0])

    # Step 4: Search for step size t with binary search
    t = 1.0  # Initial value
    x_0 = s[min_idx] + t * v_0

    # Function to evaluate f(x) at position x
    def evaluate_f(x):
        f_val = 0
        for i in range(len(s)):
            dist = np.linalg.norm(x - s[i])
            f_val += (dist - distances[i]) ** 2
        return f_val

    # f(s_k) at starting point
    f_s_k = evaluate_f(s[min_idx])

    # Binary search to find optimal t
    max_iterations = 20
    for iter in range(max_iterations):
        x_0 = s[min_idx] + t * v_0
        f_x0 = evaluate_f(x_0)

        # Check condition in Algorithm 2: f(s_k + t*v_0) < f(s_k)
        if f_x0 < f_s_k:
            break

        # Reduce t and try again
        t /= 2.0

        # If t becomes too small, use s_k as initial point
        if t < 1e-8:
            x_0 = s[min_idx]
            break

    # If we reach maximum iterations without finding a valid point
    if iter == max_iterations - 1:
        x_0 = s[min_idx]

    return x_0
```

Iterative Localization (Algorithm 3) Algorithm 3 applies the MM iterative scheme to refine the source position estimate. Starting from $\mathbf{x}_0$, the algorithm updates the estimate using the rule (equation (26) in [1]):

$$\mathbf{x}^{(r+1)} = \frac{\sum_{i=1}^{N} \left(\mathbf{s}_i + d_i \frac{\mathbf{x}^{(r)} - \mathbf{s}_i}{\|\mathbf{x}^{(r)} - \mathbf{s}_i\|} \right)}{\sum_{i=1}^{N} 1}, \tag{8}$$

where r denotes the iteration index.

The iterations continue until convergence criteria are met, specifically when the change in position between iterations falls below a defined tolerance, or the maximum number of iterations is reached.

```
def tmm_localize(self, max_iterations=100, tolerance=1e-8):
    """
    Algorithm 3 from the paper: T-MM underwater acoustic-based location algorithm.
    Reference: equations (28)-(29)
    """

    s = self.get_utm_coords()
    tdoa_diffs = self._compute_distances()  # d_i = v_water * _i (d_i - d_1)

    # Step 1: Initial x est
    x_init = self._find_initial_point()

    d1_init = np.linalg.norm(x_init - s[0])
    di_fixed = [d1_init + tdoa_diffs[i] for i in range(len(s))]

    def compute_objective(x):
        return sum(
            (np.linalg.norm(x - s[i]) - di_fixed[i]) ** 2 for i in range(len(s))
        )

    # Step 3: MM iterations with fixed d_i
    x_current = x_init.copy()
    for _ in range(max_iterations):
        x_prev = x_current.copy()

        numerator = np.zeros(2)
        denominator = 0
        for i in range(len(s)):
            direction = x_current - s[i]
            norm_dir = np.linalg.norm(direction)
            if norm_dir < 1e-10:
                continue
            numerator += s[i] + di_fixed[i] * (direction / norm_dir)
            denominator += 1
        x_current = numerator / denominator  # Eq. (26)

        if np.linalg.norm(x_current - x_prev) < tolerance:
            break

    lon, lat = self.utm_to_geo(x_current[0], x_current[1])
    return lat, lon
```

Advantages. The main advantage of the T-MM algorithm is its robustness in realistic scenarios where time-of-arrival (TOA) data are corrupted by noise and environmental effects. In such conditions, the T-MM algorithm tends to outperform classical TDOA methods by providing more accurate and stable localization results. Its iterative majorization-minimization framework also ensures monotonic convergence and improved stability through the initial point selection step.

Limitations. The method requires knowledge of accurate distance measurements and may be computationally heavier than closed-form solutions. Convergence to a global optimum is not guaranteed if the initial point is poorly chosen.

8.4. Squared Range-based Least Squares (SR-LS)

The Squared Range-based Least Squares (SR-LS) approach offers an alternative methodology for underwater acoustic source localization. Unlike standard TDOA methods, SR-LS optimizes squared range measurements between the source and sensors, as proposed in [2].

The SR-LS method formulates the localization problem as:

$$(\text{SR-LS}) : \quad \min_{\mathbf{x}} \sum_{i=1}^{m} \left(\|\mathbf{x} - \mathbf{a}_i\|^2 - r_i^2 \right)^2 \tag{9}$$

where $\mathbf{x}$ is the source coordinate vector to be estimated, $\mathbf{a}_i$ are the known coordinates of the i-th hydrophone, and r_i is the measured distance from the i-th hydrophone to the source.

Although this optimization problem is non-convex, it can be solved globally and efficiently. The solution approach transforms the problem into a Generalized Trust Region Subproblem (GTRS), which involves minimizing a quadratic function subject to a single quadratic constraint:

$$\min_{\mathbf{y} \in \mathbb{R}^{n+1}} \{ \|\mathbf{A}\mathbf{y} - \mathbf{b}\|^2 : \mathbf{y}^T \mathbf{D} \mathbf{y} + 2\mathbf{f}^T \mathbf{y} = 0 \} \tag{10}$$

The optimal solution is given by $\hat{\mathbf{y}}(\lambda) = (\mathbf{A}^T\mathbf{A} + \lambda \mathbf{D})^{-1}(\mathbf{A}^T\mathbf{b} - \lambda \mathbf{f})$, where λ is the unique solution of a secular equation that can be efficiently solved using bisection methods.

One key advantage of SR-LS is that, despite being non-convex, it provides an exact global solution rather than an approximation. This contrasts with other approaches that rely on relaxation techniques or discarding constraints, which often lead to suboptimal results. Numerical experiments have shown that SR-LS significantly outperforms approximate methods such as unconstrained squared-range based LS (USR-LS) and semidefinite relaxation (SDR) approaches, especially in high-noise scenarios.

It's worth noting that the SR-LS approach is suboptimal in the maximum likelihood sense when compared to range-based least squares (R-LS) formulations, particularly under Gaussian noise assumptions. This is because the covariance matrix of squared errors in the squared range domain is not proportional to the identity matrix. However, the ability to compute the exact global solution of SR-LS efficiently makes it particularly attractive for underwater acoustic localization applications.

This method is implemented in Python as follows:

```python
def _construct_matrix_A(self):
    s = self.get_utm_coords()
    m = len(s)

    A = []

    # compute matrix A as (17) in [2]
    for i in range(m):
        A.append([s[i][0] * (-2), s[i][1] * (-2), 1])

    A = np.array(A)

    return A

def _construct_vector_b(self):
    # compute vector b as (17) in [2]
    coords = np.array(self.get_utm_coords())
    r_squared = np.array(
        [
            (hydro.observed_pressure[-1]["toa"] * self.v) ** 2
            for hydro in self.hydrophones
        ]
    )

    a_norm_squared = np.sum(coords**2, axis=1)

    return r_squared - a_norm_squared

def _construct_matrix_D(self):
    # compute matrix D as (18) in [2]
    s = self.get_utm_coords()
    n = len(s[0])

    D = np.zeros((n + 1, n + 1))

    D[:n, :n] = np.eye(n)

    return D

def _construct_vector_f(self):
    # compute vector f as (18) in [2]

    n = len(self.get_utm_coords()[0])
    f = np.zeros(n + 1)
    f[n] = -0.5

    return f

def sr_ls_localize(self):
    # Compute matrix A and vector b as (17) in [2]
    A = self._construct_matrix_A()
    b = self._construct_vector_b()

    # Compute matrix D and vector f as (18) in [2]
    D = self._construct_matrix_D()
    f = self._construct_vector_f()

    # Find optimal lambda as (24) in [2]
    lambda_opt = self.find_optimal_lambda(A, b, D, f)

    # Compute position estimation
    source_position = self.calculate_position_estimate(lambda_opt, A, b, D, f)

    # return coord in (lat, lon) format
    lon, lat = self.utm_to_geo(source_position[0], source_position[1])
    return lat, lon

def determine_lambda_interval(self, A, D):
    # Compute lower bound for interval I for lambda as eq. (26) in [2]

    # Compute A^T A
    ATA = A.T @ A

    # Compute lowest eigenvalue
    eigvals = scipy.linalg.eigvals(D, ATA)
    lambda_min = min(abs(eigvals))

```

```
        # Compute lower bound as (26) in [2]
        lower_bound = -1 / lambda_min + 1e-10

        return lower_bound

    def phi_function(self, lambda_val, A, b, D, f):
        # Compute value of () as eq. (25) in [2]

        try:
            # Compute A^T A + D
            ATA_plus_lambda_D = A.T @ A + lambda_val * D

            # Compute A^T b - f
            ATb_minus_lambda_f = A.T @ b - lambda_val * f

            # Compute ŷ() = (A^T A + D)^(-1)(A^T b - f)
            y_lambda = np.linalg.solve(ATA_plus_lambda_D, ATb_minus_lambda_f)

            # Compute () = ŷ()^T D ŷ() + 2f^T ŷ()
            phi_val = y_lambda.T @ D @ y_lambda + 2 * f.T @ y_lambda

            return phi_val
        except np.linalg.LinAlgError:
            return float("inf")

    def find_optimal_lambda(self, A, b, D, f):
        # Find optimal value of lambda as () = 0, eq. (24) in [2]

        lower_bound = self.determine_lambda_interval(A, D)
        upper_bound = 1e6

        def phi_for_bisect(lambda_val):
            return self.phi_function(lambda_val, A, b, D, f)

        lower_val = phi_for_bisect(lower_bound)
        upper_val = phi_for_bisect(upper_bound)

        if lower_val * upper_val > 0:
            if abs(lower_val) < abs(upper_val):
                return lower_bound
            else:
                return upper_bound

        optimal_lambda = scipy.optimize.bisect(
            phi_for_bisect, lower_bound, upper_bound, rtol=1e-6
        )

        return optimal_lambda

    def calculate_position_estimate(self, lambda_opt, A, b, D, f):
        # Calculate pos estimation given optimal lambda as eq. (23) in [2]

        # Compute A^T A + D
        ATA_plus_lambda_D = A.T @ A + lambda_opt * D

        # Compute A^T b - f
        ATb_minus_lambda_f = A.T @ b - lambda_opt * f

        # Compute ŷ() = (A^T A + D)^(-1)(A^T b - f)
        y_lambda = np.linalg.solve(ATA_plus_lambda_D, ATb_minus_lambda_f)

        n = len(self.get_utm_coords()[0])
        source_position = y_lambda[:n]

        return source_position
```

9. Numerical Simulation

9.1. Simulation Setup

To evaluate the performance and robustness of the three acoustic localization algorithms (TDOA, T-MM, and SR-LS), has been conducted extensive numerical simulations using the developed framework. The simulation environment was configured as follows:

- **Geographic Area**: A region defined by coordinates [10-12° latitude, 83.5-85.5° longitude]
- **Bathymetry**: Realistic seabed topography data from the General Bathymetric Chart of the Oceans (GEBCO) database, loaded from a NetCDF file
- **Acoustic Propagation**: The Bellhop ray-tracing model was employed to simulate realistic underwater sound propagation, with 10 bathymetric points sampled between each hydrophone-source pair to account for depth variations
- **Acoustic Error Modeling**: Six levels of TOA variance were tested [1.0e-5, 1.0e-4, 1.0e-3, 1.0e-2, 1.0e-1, 1.0], corresponding to progressively higher levels of measurement noise

9.2. Hydrophone Configuration

Four hydrophones were strategically positioned to form a square array:

- Hydrophone 1: (11.0, 84.0) at 20m depth
- Hydrophone 2: (11.03, 84.0) at 20m depth
- Hydrophone 3: (11.03, 84.03) at 20m depth
- Hydrophone 4: (11.0, 84.03) at 20m depth

The ambient noise level was set at 2.0 dB re 1 µPa; no random hydrophones were generated for this simulation.

9.3. Target Configuration

The simulation included a single dark ship with the following parameters:

- Position: (11.032, 84.015)
- Speed: 10 knots
- Depth: 20 meters
- Heading: 180 degrees

9.4. Methodology

For each of the six TOA variance levels, has been conducted 100 independent simulation runs, recording the performance of each localization algorithm (TDOA, T-MM, and SR-LS) in terms of mean squared error (MSE) between the estimated and actual ship position. This approach allowed to systematically analyze how the performance of each algorithm degrades with increasing noise levels, providing insights into their robustness and suitability for different operational conditions.

9.5. Results and Analysis

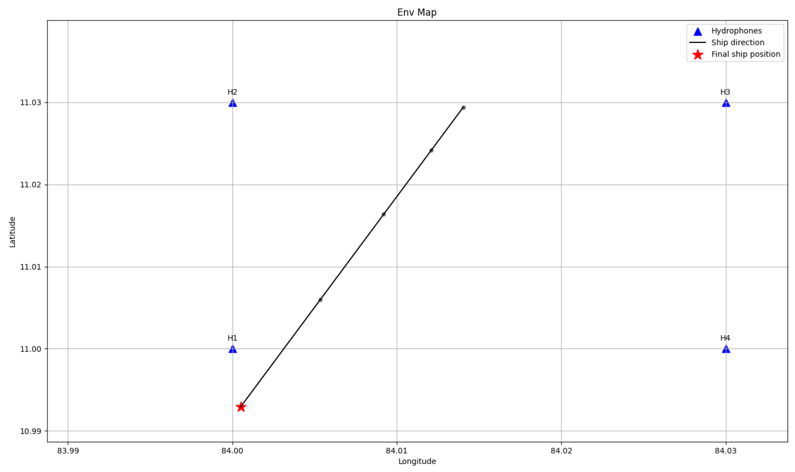

Fig. 2. Spatial configuration of the simulation showing the positions of the four hydrophones (blue markers) and the dark ship (red marker) in the defined geographic area.

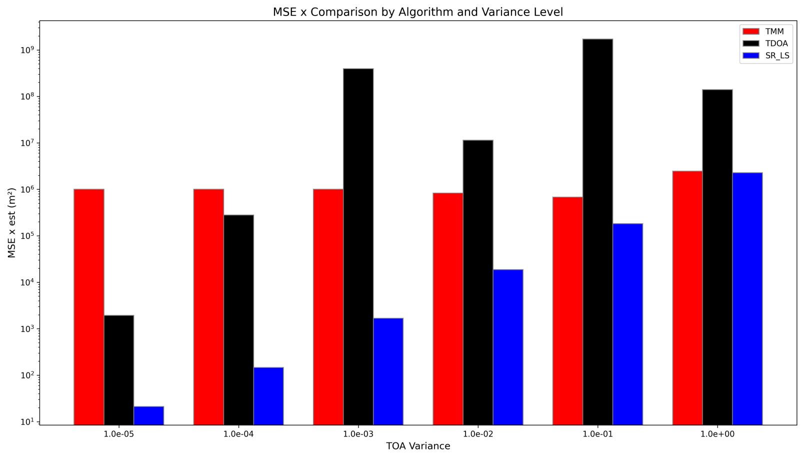

Fig. 3. Comparative Mean Square Error (MSE) of TDOA, T-MM, and SR-LS algorithms across different TOA variance levels.

Figure 4 shows the error progression as a function of increasing noise levels, providing insight into how rapidly each algorithm's performance degrades with higher measurement uncertainty.

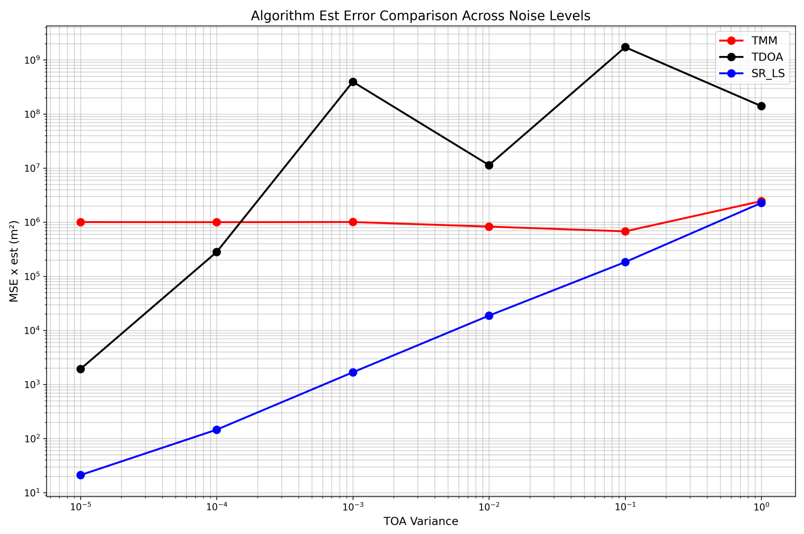

Fig. 4. Performance degradation trends of the three localization algorithms with increasing measurement noise. The logarithmic scale highlights the exponential relationship between noise levels and localization error.

The line chart format emphasizes the trend patterns and reveals critical threshold points where algorithm performance significantly deteriorates.

The results from the simulations demonstrate distinct performance characteristics for each algorithm:

- The conventional TDOA algorithm shows acceptable performance at lower noise levels (TOA variance < 10^{-3}) but degrades rapidly as noise increases, making it suitable only for relatively quiet underwater environments or scenarios with high-quality hydrophones.
- The T-MM algorithm exhibits significantly improved resilience to noise compared to standard TDOA. The majorization-minimization approach effectively mitigates the impact of measurement errors, demonstrating the value of this optimization technique.
- The SR-LS algorithm demonstrates the best overall robustness, particularly at higher noise levels. By formulating the problem in terms of squared ranges and employing global optimization techniques, this approach maintains acceptable localization accuracy even under challenging acoustic conditions where the other methods fail.

References

1. H. S. Shuangshuang Li and H. Esmaiel, “Underwater tdoa acoustical location based on majorization-minimization optimization,” Sensors **20**, 4457 (2020).
2. A. Beck, P. Stoica, and J. Li, “Exact and approximate solutions of source localization problems,” IEEE Transactions on Signal Processing **56**, 1770–1778 (2008).