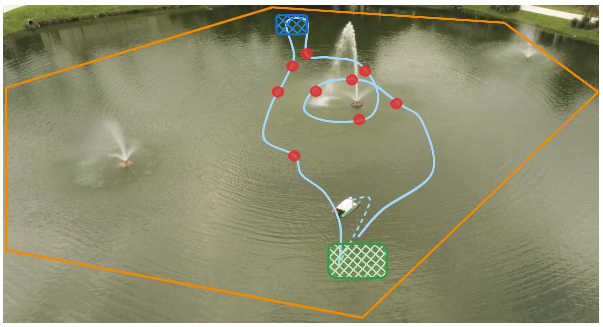
APPENDIX

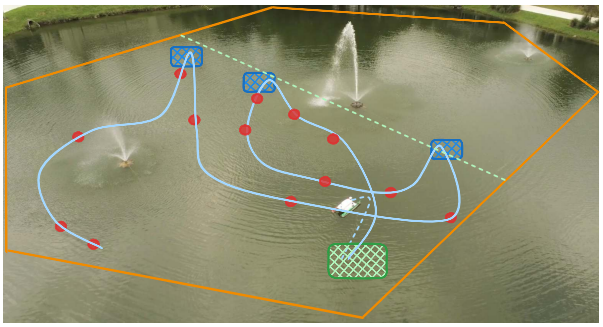
This supplementary material provides:

- *Additional Experimental Results*: Extended evaluations of GUIDE on various tasks.
- *Implementation Details*: Descriptions of the dataset, model architectures and hyperparameters for GUIDE and all baselines.

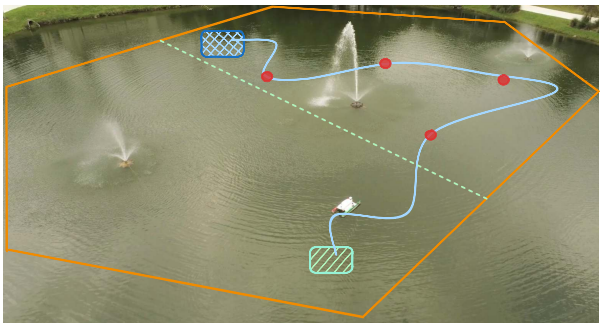
I. ADDITIONAL EXPERIMENTAL RESULTS



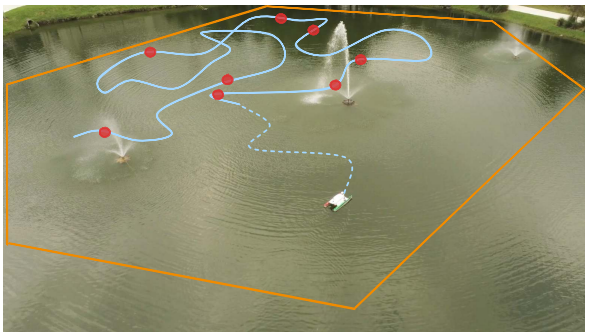
(a) "Go to point [80,90] and then go around the central fountain and return to the dock."



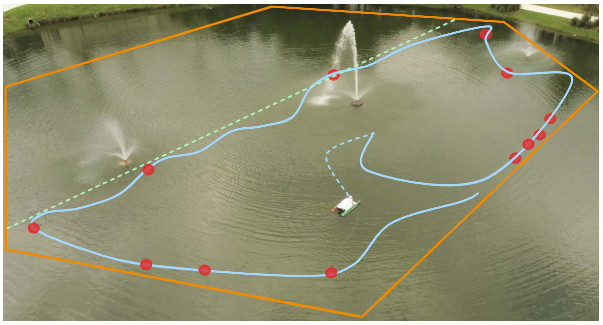
(b) "Go to [80,60] and then go to [80, 20] and then go to [80, 70] and finally return to the dock. This task has to be completed by avoiding the right half of the area."



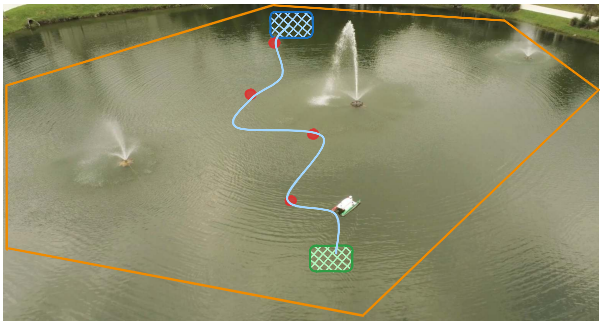
(c) "Go to [40,60] while avoiding the left half of the environment."



(d) "Explore the top half of the environment."



(e) "Navigate the perimeter of the bottom half of the lake."



(f) "Go to [40,60]."

Fig. 1: Illustrations of various navigation tasks performed by GUIDEd SAC in the environment. In all images, the light blue line represents the trajectory, red dots indicate areas where uncertainty was reduced, the green rectangle represents the dock, and brown rectangles represent waypoints that need to be reached. The specific tasks are described below each figure.

Note: the coordinates mentioned in the illustrations and the examples are dummy values to maintain anonymity.

A. Implementation Details

1) Task Categories and Examples: The tasks in the dataset are grouped into six categories, each designed to test specific aspects of navigation and uncertainty management. Below, we describe each category in detail and provide representative examples.

a) Goal Reaching: Waypoint: This category involves tasks where the ASV is instructed to navigate to specific coordinates. The focus is on reaching designated points in the environment.

Examples:

- `Navigate to waypoint (12.0, -7.5).`
- `Proceed to the coordinates (8.5, 15.0).`
- `Go to the location at (5.0, -10.0).`

b) Goal Reaching: Contextual Landmarks: This category includes tasks where the ASV is instructed to navigate to locations identified by contextual landmarks rather than explicit coordinates. This tests the ability to interpret semantic information and associate it with spatial positions.

Examples:

- `Go to the dock.`
- `Proceed to the central fountain.`
- `Navigate to the area in front of the left fountain.`

c) Avoidance Tasks: These tasks instruct the ASV to avoid certain points or areas, emphasizing obstacle detection and path planning to circumvent specified locations.

Examples:

- `Avoid the coordinates (10.0, -5.0).`
- `Steer clear of the submerged rock at (3.5, 4.0).`

d) Perimeter Navigation Tasks: In this category, the ASV is tasked with navigating around the perimeter of a specified area. This requires maintaining a certain distance from boundaries.

Examples:

- `Navigate around the perimeter of the bottom-right quadrant.`
- `Circumnavigate the central fountain.`
- `Traverse the boundary of the entire lake.`

e) Exploration Tasks: These tasks involve exploring a specified area for a fixed duration of 5 minutes, testing the ASV's ability to stay withing an area and cover parts.

Examples:

- `Explore the top-half of the lake.`
- `Conduct an exploration of the top-right quadrant.`

f) Restricted Area Navigation: Tasks in this category require the ASV to navigate while avoiding specified regions.

Examples:

- `Go to waypoint (6.0, -3.0) while avoiding the right half of the lake.`
- `Navigate to the right fountain, avoiding the exclusion zone.`
- `Proceed to the dock without passing through the left half of the lake.`

2) Natural Language Processing and Embedding Generation: To process the natural language task descriptions, we utilized a fine-tuned RoBERTa language model, which captures contextual nuances and effectively handles synonyms and varied phrasings. This enables the model to interpret different expressions of similar tasks, ensuring robustness to linguistic variations. For example, phrases like ``proceed to'', ``navigate to'', and ``go to'' are recognized as equivalent in intent.

The semantic embeddings generated by RoBERTa are paired with spatial embeddings derived from the associated coordinates or landmarks, allowing the model to learn meaningful associations between language and location. This approach ensures that even when new tasks are presented with different wording or synonyms, the model can generalize and generate appropriate Task-Specific Uncertainty Maps.

3) Dataset and Data Processing: The dataset used for training our TSUM generation model consists of overhead imagery collected from various marine environments and simulation scenes, carefully curated to capture a wide range of operational scenarios. The data collection process spanned multiple weather conditions to ensure robustness and generalizability of the trained model.

For the labeling process, we employed experts with experience in autonomous navigation. These experts followed an annotation protocol to maintain consistency across the dataset. The protocol involved examining each 224×224 pixel patch of overhead imagery and assigning binary labels indicating relevance to predefined subtasks and constraints. The labeling criteria were established through iterative refinement to ensure practical applicability.

To address class imbalance issues in our dataset, we implemented a comprehensive sampling strategy. Many critical navigation scenarios, such as complex docking maneuvers or navigation around certain obstacles, were naturally underrepresented in the raw data collection. We addressed this through strategic oversampling of these crucial but rare scenarios. Specifically, patches containing these underrepresented classes were duplicated in the training set, while maintaining cross-validation splits to prevent overfitting. For negative examples, we employed a systematic sampling approach, randomly selecting patches that domain experts had deemed irrelevant to specific subtasks or constraints, while ensuring a balanced representation across different environmental conditions.

The environmental variations in our dataset were curated to capture the diverse conditions encountered in real-world marine operations. Our collection includes patches from both real-world lake environments and high-fidelity simulated scenarios. The real-world data encompasses variations in weather conditions (clear, overcast, and light rain), lighting conditions (morning, midday, and evening), and seasonal changes affecting water conditions. Shoreline features vary from natural boundaries to man-made structures, and obstacle density ranges from sparse open waters to crowded marina environments. The simulated data complements these real-world scenarios by providing additional coverage of rare but critical scenarios that are difficult to capture in real-world data collection.

For data storage and organization, we implemented a metadata management system. Each image patch is stored in a georeferenced index using a custom CSV format, which maintains associations between image patches, their geographical coordinates, and corresponding labels. This schema ensures efficient retrieval during training and enables seamless integration with existing robotic navigation systems. The metadata includes essential information such as timestamp, environmental conditions, and annotation confidence scores from domain experts.

Our data augmentation pipeline was designed to enhance model robustness while preserving critical visual cues for marine navigation. The primary augmentations include random rotations (in 90-degree increments) and horizontal flips, which reflect the rotational invariance of navigation tasks while maintaining the natural appearance of water features. Notably, we deliberately avoided color jittering and intensity transformations, as these could distort important water-related visual cues that are crucial for accurate navigation. The augmentation parameters were tuned through extensive experimentation to find the optimal balance between increasing data diversity and maintaining task-relevant features. All image patches undergo consistent preprocessing to ensure uniformity in the training data. This includes standardization to zero mean and unit variance, computed across the training set, and resizing to maintain consistent spatial resolution across different source imagery. The preprocessing pipeline also includes automated quality checks to identify and filter out patches with excessive noise, sensor artifacts, or poor visibility conditions that could negatively impact model training.

4) Technical Infrastructure and Hardware Setup: Our technical infrastructure was designed to efficiently handle the computational demands of processing large-scale overhead imagery and training complex neural networks. The foundation of our data processing pipeline centers on the conversion of geographical coordinates to discrete 224×224 pixel patches, which serve as the basic unit of analysis for our TSUM generation system. The coordinate-to-patch mapping system employs a gridding algorithm that subdivides large overhead maps into a regular grid of fixed-size patches. Each patch is assigned center coordinates that correspond to real-world geographical positions. The mapping process utilizes a custom spatial indexing structure that enables efficient lookup of relevant patches given any arbitrary location in the operational space. To handle edge cases where locations fall between patch boundaries, we implemented a nearest-center assignment strategy that ensures every point in the operational space maps to exactly one patch while maintaining spatial continuity.

To maximize processing efficiency, we developed a parallelized data generation pipeline. The system utilizes a custom-built multithreaded tiling script that leverages all available CPU cores to concurrently process large imagery datasets. This parallelization is implemented using Python's multiprocessing library, with careful attention to memory management to prevent resource exhaustion when handling particularly large maps. The tiling process is coordinated by a master thread that manages work distribution and ensures balanced load across all available processors. Each worker thread independently processes assigned regions of the input imagery, generating patches and associated metadata in parallel.

The runtime environment is built on Ubuntu 20.04 LTS with CUDA 11.8 and cuDNN 8.7, optimized for deep learning workloads. We utilize Docker containers to ensure consistency across different compute nodes and to simplify deployment. The container images are based on NVIDIA's NGC PyTorch container, customized with additional dependencies required for our specific workload. This containerized approach ensures reproducibility and enables easy scaling across different hardware configurations. Data movement between storage and compute nodes is optimized using a custom data loading pipeline built on top of PyTorch's DataLoader class. We implemented prefetching mechanisms that load and preprocess data for upcoming batches while the current batch is being processed on the GPUs. This approach effectively hides I/O latency and ensures near-continuous GPU utilization. The data loading pipeline includes automatic checkpointing to enable recovery from system failures without losing progress.

The typical processing workflow for a complete dataset involves approximately 2-3 hours of initial data tiling and preprocessing, followed by 10 epochs of model training. The training process is distributed using PyTorch's DistributedDataParallel, with gradient synchronization optimized for our specific network architecture and batch sizes. Our implementation achieves

approximately 71% GPU utilization during training, with the remaining overhead primarily attributed to necessary data loading operations.

Resource monitoring and system health checks are performed using a combination of Prometheus for metrics collection and Grafana for visualization. This monitoring infrastructure allows us to track system performance, identify bottlenecks, and optimize resource utilization in real-time. Additionally, we maintain comprehensive logs of all processing steps, enabling detailed analysis of system performance and facilitating debugging when necessary.

A crucial component of our technical infrastructure is our high-fidelity simulation environment, built using Unity3D (2022.3.16f1) in conjunction with ROS2 Humble. This simulator serves as a vital tool for both data generation and policy validation. The Unity environment provides physically accurate water dynamics for fluid simulation, capable of modeling complex wave patterns, water resistance, and hydrodynamic forces. We implemented custom shaders to accurately render water surface properties, ensuring visual fidelity crucial for training vision-based navigation systems.

The simulator is tightly integrated with ROS2 through a custom bridge that enables bidirectional communication between Unity and ROS2 nodes. This integration allows for seamless transfer of sensor data, control commands, and state information. We implemented the full suite of sensors found on our physical ASV, including simulated GPS (with configurable noise patterns), IMU (with drift characteristics matching real hardware), and cameras (with accurate lens distortion and environmental effects). The simulator also includes detailed models of environmental factors such as wind effects, current patterns, and varying lighting conditions.

For training data generation, the simulator can be run in a distributed headless manner across instances, each generating different scenarios and environmental conditions. We used an off-shelf scenario generation system that creates diverse training situations, including varying obstacle configurations, weather conditions, and traffic patterns. The simulator supports both synchronous and asynchronous operation modes, allowing for rapid data collection when generating training datasets and real-time operation when validating policies.

B. Hyperparameters

Hyperparameter Name	**Brief Explanation**	**Value**
TSUM Aggregation		
w_Φ	Weight for subtask relevance function $\Phi^\tau(l)$	0.5
$w_\mathcal{C}$	Weight for constraint relevance $\mathcal{C}^\tau(l)$	0.3
$w_\mathcal{E}$	Weight for environmental factors $\mathcal{E}(l)$	0.2
CLIP Fine-Tuning		
CLIP embedding dimension d	Output dimension of ViT-B/32 encoders (text/image)	512
Number of frozen layers	Initial layers in CLIP left frozen to retain general cross-modal knowledge	6
Fine-tuning epochs	Epochs used to adapt the unfrozen CLIP layers	10
Learning rate for CLIP	LR for training the unfrozen CLIP layers	1×10^{-5}
Alignment weighting λ_{align}	Weight on additional alignment loss $\mathcal{L}_{\text{align}}$	1.0
Contrastive temperature ζ	Temperature hyperparameter for CLIP contrastive loss	0.07
Reinforcement Learning (SAC)		
Discount factor γ	Discount factor in the RL objective	0.99
Initial temperature α	Initial entropy temperature in SAC	0.2
Target entropy $\bar{\mathcal{H}}$	Target policy entropy term in SAC	-2
Soft target update τ	Polyak averaging coefficient for target Q-networks	0.005
Replay buffer size $\lvert\mathcal{D}\rvert$	Maximum capacity of the experience replay buffer	1×10^6
Batch size	Number of samples per RL training batch	256
Learning rate for policy network	LR for optimizing the policy π_θ	3×10^{-4}
Learning rate for Q-value networks	LR for optimizing the Q-value functions Q_{ϕ_i}	3×10^{-4}
Learning rate for temperature α	LR for adjusting the SAC temperature	3×10^{-4}
Policy network architecture	MLP for π_θ with hidden layers and activation	2 layers, 256 units each, ReLU
Q-value networks architecture	MLP for Q_{ϕ_i}, identical structure	2 layers, 256 units each, ReLU
Maximum steps per episode	Upper bound on steps in each training episode	1000

TABLE I: Hyperparameters for the CLIP-based TSUM generation and SAC policy learning.

Hyperparameter Name	Brief Explanation	Value
1. Standard RL without TSUMs (SAC, Ablation 1)		
State representation	The agent uses only the original state s	
2. GUIDEd PPO (G-PPO, Ablation 2)		
Policy optimization algorithm	Proximal Policy Optimization (PPO) is used instead of SAC	
Clip ratio ϵ	Clipping parameter for PPO policy updates	0.2
Number of epochs per update	Number of epochs per policy update iteration	10
Learning rate for policy and value networks	Learning rate for updating network parameters	3×10^{-4}
Generalized Advantage Estimation lambda λ_{GAE}	Smoothing parameter for advantage estimation	0.95
Batch size	Number of samples per training batch	64
State representation	Augmented state $\tilde{s} = [s, U^{\tau}(s), u(s)]$	
3. RL with Uncertainty Penalization (SAC-P)		
Penalty weight ζ	Weighting factor for uncertainty penalization in the reward function $R_{\text{SAC-P}} = R_{\text{base}} - \zeta u(s)$	0.4
State representation	Original state s	
4. Bootstrapped Uncertainty-Aware RL (B-SAC)		
Number of bootstrap heads	Number of Q-value networks used to estimate epistemic uncertainty	10
Bootstrap sampling probability	Probability of selecting each head during training	0.8
Uncertainty estimation method	Standard deviation across bootstrap heads	
State representation	Original state s	
5. Heuristic Policy (HEU)		
Switching threshold distance	Distance to obstacles or task-critical regions at which the agent switches to exact position estimation	3.5 meters
Planning algorithm	Base planning using SAC	
Exact position estimation mode	The mode η used for precise localization when close to obstacles	$\eta = 1$
State representation	Original state s	
6. Risk-Aware RL (CVaR)		
Risk level α_{CVaR}	Conditional Value at Risk (CVaR) level, determining the quantile of worst-case outcomes considered	0.05
CVaR optimization method	Optimization of the expected value over the worst α_{CVaR} fraction of outcomes	CVaR objective
State representation	Original state s	
7. Uncertainty-Aware Motion Planning (RAA)		
Acceptable collision probability $p_{\text{collision}}$	Threshold probability of collision acceptable during planning	0.01
Risk assessment horizon	Number of steps ahead considered for risk assessment	50

TABLE II: Hyperparameters for the baseline and abalation methods. Unless specified, methods use the same hyperparameters as in GUIDEd SAC (see Table I). Only differences from GUIDEd SAC are listed for each method.