Computer Vision-Guided Sphero Swarm Navigation via Probabilistic Localization, Planning, and Feedback-Based Movement

Phi Bui

This paper presents a swarm navigation framework integrating global camera-based localization, Monte Carlo Localization (MCL), Probabilistic Roadmaps (PRM), and a feedback-based movement algorithm to coordinate multiple Sphero Bolt robots in complex environments. Using a single overhead camera for vision-based positioning and mapping, the system identifies obstacles and goals while maintaining robust real-time localization. PRM enables dynamic path planning, and feedback-based adjustments refine robot trajectories to address noisy measurements and varying terrain. Experiments demonstrate collision-free navigation and scalability, highlighting potential applications in search and rescue, warehouse automation, and decentralized multirobot systems.

1 Introduction

Localization, mapping, coordinated navigation, and accurate movement are central challenges in swarm robotic systems operating in complex, real-world environments. This project develops a swarm navigation framework using Sphero Bolt robots guided by an overhead camera for real-time localization and mapping. Leveraging computer vision and probabilistic methods—including Monte Carlo Localization (MCL) and Probabilistic Roadmaps (PRM)—the system demonstrates precise navigation and efficient swarm coordination. Such capabilities have significant applications in real-world scenarios. In search and rescue missions, an overhead perspective, akin to a drone's view, can map hazardous environments, localize robots, and guide them to survivors or critical areas. Similarly, swarm robotics can optimize navigation in cluttered spaces, such as warehouses or disaster zones, by coordinating multiple agents to avoid obstacles and complete tasks collaboratively.

This swarm navigation framework showcases the potential of integrating advanced localization and mapping techniques with swarm robotics to enable dynamic, efficient, and scalable operations in challenging environments.

2 Background Related Work

This swarm navigation framework builds upon foundational work in single-robot SLAM, such as Hu's [4] exploration of Kalman filters and set-theoretic methods for a lone Sphero Bolt. While effective for basic localization, these methods were constrained by limited sensor capabilities. Subsequently, researchers have enhanced navigation techniques: Hasani et al. [2] introduced NavFormer, a transformer-based architecture enabling robust, camera-only navigation under dynamic conditions. In parallel, Kegeleirs et al. [5] advanced swarm experimentation platforms with the Mercator system, providing richer sensing and control frameworks for multi-robot applications.

Building on these foundations, Arslan et al. [1] developed SwarmPRM, a probabilistic roadmap technique enabling decentralized, probabilistic path planning for swarms, while Havangi et al. [3] demonstrated that multi-swarm particle filters could yield precise localization in complex, changing environments. Wustrau [6] validated the feasibility of swarm algorithms using Sphero Bolt robots and highlighted practical challenges in scalability and emergent behaviors.

By integrating camera-based global localization, MCL, and PRM, our swarm navigation framework synthesizes these insights to deliver a scalable, robust solution for Sphero Bolt-based swarm robotics in real-world conditions.

3 Technical Approach

3.1 Hardware

We employed five Sphero Bolt robots, each controlled by a Python server. An overhead Akaso Brave 4 camera provided a 4K (2880×2160) view of the environment, enabling real-time localization and mapping. The server infrastructure, consisting of three separate servers (detailed in Section 3.3), was hosted on an Asus Zenbook 13 UX325EA-DH51 laptop.

The navigation environment was simulated using a large sheet of paper, marked with colored tape, with purple presenting a goal and orange representing obstacles.

3.2 Libraries

The implementation of the swarm navigation framework relied on several Python and JavaScript libraries, each serving a critical role in various aspects of the swarm navigation framework. The following libraries were utilized:

- **opencv-python:** Used for image processing and computer vision tasks, including camera-based localization, mapping, and display functionalities.
- **websockets:** Enabled real-time communication between the WebSocket server and other components of the swarm navigation framework.
- **scikit-learn:** Provided tools for data analysis and algorithm development.

- **numpy:** Used for numerical computations and array manipulation.
- **scipy:** Supplemented `numpy` with advanced scientific computing capabilities, such as optimization and signal processing.
- **spherov2:** Facilitated interaction and control of Sphero Bolt robots.
- **bleak:** Supported Bluetooth Low Energy (BLE) communication, necessary for interfacing with Sphero Bolt robots.

These libraries collectively enabled the development of a robust, efficient, and modular system, providing essential tools for tasks ranging from robot control to data processing.

3.3 Server Architecture

The server architecture consists of three micro-service servers, with the data flow outlined in Figure 1. Each server has a specific role, as detailed below:

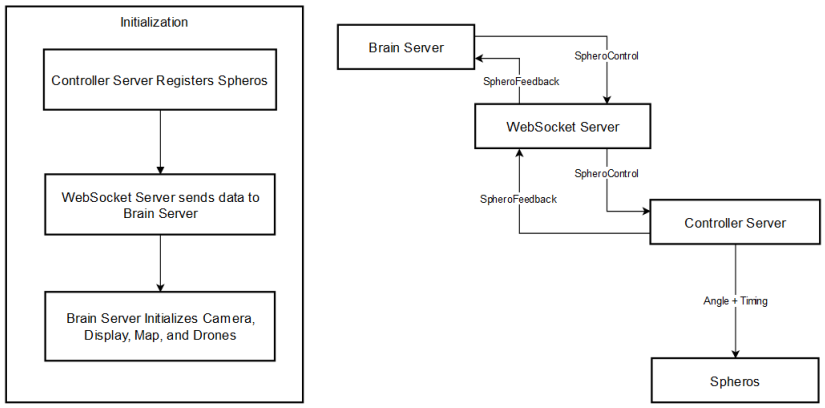

Figure 1: Server architecture data flow.

- **WebSocket Server:** Implemented in JavaScript, this server acts as a centralized communication hub. The server facilitates bidirectional data exchange between the control server and the brain server, ensuring seamless communication across the swarm navigation framework.
- **Control Server:** Implemented in Python, the control server is responsible for managing the Sphero Bolt robots. Its main functions include:
  - Initializing by sending a list of Spheros and their associated colors to the WebSocket server.
  - Receiving direction and angle commands for each Sphero from the WebSocket server and executing them.

– Sending feedback messages to the WebSocket server to confirm the successful completion of actions.

To handle multiple robots efficiently, the control server runs a separate process for each Sphero. Communication between these processes is managed using a message queue, enabling parallel control and scalability.

- **Brain Server:** Implemented in Python, the brain server manages high-level operations, including:
  - Initializing a camera and display process.
  - Managing a map system and localization for each Sphero, encapsulated in a `Drone` class.
  - Utilizing the camera for data acquisition, which is processed by the Map and Localizer.
  - Displaying processed data via the display process.

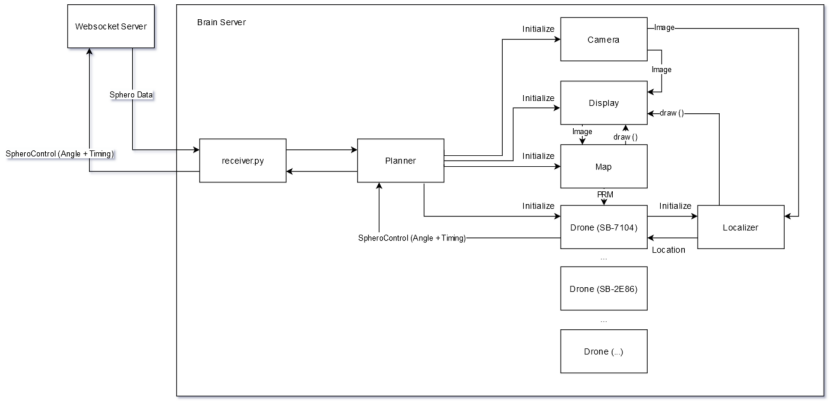

Figure 2: Brain server data flow.

The data flow specific to the brain server is shown in Figure 2. Detailed descriptions of the mapping and localization algorithms are provided in Section 3.4.

3.4 Algorithm Overview

The proposed algorithm facilitates coordinated localization, path planning, and movement for a swarm of Sphero Bolt robots in dynamic environments. It integrates **MCL** for precise position estimation, **PRM** for efficient path planning, **Feedback-Based Movement Algorithm** for feedback-based controls, and **Minimum Piecewise Euclidian Distance** for collision avoidance.

3.5 Swarm Localization, Path Planning, and Movement Algorithms

Inputs:

- R: Set of Sphero Bolt robots
- Environment with colored tape (purple for Goal, orange for Obstacles)
- Sensor Data (Images from Camera, Angles from Onboard Compass)

Outputs:

- Angle and Timing necessary to reach a target point for a given robot

Algorithm 1 MCL Algorithm

Initialize:

Generate N particles $\{p_1, p_2, \ldots, p_N\}$ uniformly distributed across the environment.

Assign equal weights to all particles: $w_i \leftarrow \frac{1}{N}, \forall i \in \{1, \ldots, N\}$.

Perception:

Capture an image of the environment using the overhead camera.

Generate a binary mask isolating the target region based on the robot's color.

Extract the pixel coordinates of the detected region and fit a Gaussian Mixture Model (GMM) to the data.

Retrieve the GMM mean (g_x, g_y) and covariance Σ.

Update Particle Weights:

for each particle p_i with position (x_i, y_i) **do**

Compute the particle's weight using the Gaussian likelihood:

$$w_i \leftarrow \frac{\exp\left(-\frac{1}{2}(z-\mu)^T \Sigma^{-1}(z-\mu)\right)}{2\pi\sqrt{\det(\Sigma)}}$$

where $z = (x_i, y_i)$ and $\mu = (g_x, g_y)$.

end for

Normalize Weights:

Normalize all particle weights such that $\sum_{i=1}^{N} w_i = 1$.

Resample Particles:

Resample N particles from the current particle set with replacement, proportional to their weights.

for each resampled particle **do**

Add Gaussian noise to the position to maintain diversity.

end for

Output Estimate:

Compute the weighted mean position of the particles:

$$\text{Estimated Position} \leftarrow \sum_{i=1}^{N} w_i \cdot (x_i, y_i)$$

Compute the confidence score from the GMM covariance:

$$\text{Confidence} \leftarrow \exp(-\det(\Sigma))$$

End Algorithm.

Algorithm 2 PRM Generation Algorithm

Input:
    Number of nodes, N.
    Initial connection radius, r_{initial}.
    Maximum connection radius, $r_{\max}$.
Output:
    PRM nodes and edges.
Step 1: Preprocess Environment
Capture an image of the environment and convert it to HSV color space.
Detect obstacles using color masks and store them as rectangular regions.
Detect the goal region and calculate its center point (g_x, g_y).
Step 2: Generate Random Nodes
Initialize an empty list of nodes: Nodes $\leftarrow \{(g_x, g_y)\}$.
Repeat until $|\text{Nodes}| = N$ or a maximum number of attempts is reached:
for each random point (x, y) in the environment **do**
  if (x, y) is not near any obstacle **then**
    Add (x, y) to Nodes.
  end if
end for
Step 3: Connect Nodes
for each node $n_i = (x_i, y_i) \in$ Nodes **do**
  Set the connection radius $r \leftarrow r_{\text{initial}}$.
  Initialize $c \leftarrow 0$ (number of connections for n_i).
  while $c < 3$ and $r \leq r_{\max}$ **do**
    for each node $n_j = (x_j, y_j) \neq n_i$ **do**
      Calculate distance $d \leftarrow \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$.
      if $d \leq r$ and line segment $\overline{n_i n_j}$ does not intersect any obstacle **then**
        Add edge (n_i, n_j) to Edges.
        Increment $c \leftarrow c + 1$.
      end if
    end for
    Increase $r \leftarrow r + 50$.
  end while
end for
Step 4: Prune Unreachable Nodes
Use Depth-First Search (DFS) starting from the goal node to find all reachable nodes.
Remove nodes and edges not connected to the goal node.
Step 5: Build KDTree
Construct a KDTree from the final set of nodes for efficient nearest neighbor queries.
Output:
The final PRM with nodes and edges.

Algorithm 3 Feedback-Based Movement Algorithm

Initialize:
    Set initial parameters:
        last_location ← None,
        speed ← 0,
        angle_offset ← None.
    Target position from next node along the PRM.
Position Estimation:
Current position (current_y, current_x) is estimated via MCL with confidence.
Check for First Move:
if this is the first move (i.e., last_location = None) **then**
    Set last_location ← (current_y, current_x).
    Return initial angle and timing.
end if
Calculate Movement Vector:
Compute actual movement vector $\Delta x_{\text{actual}}, \Delta y_{\text{actual}}$ from previous to current position.
Compute the actual movement angle actual_angle and distance moved.
Initialize Angle Offset (if needed):
if angle_offset = None **then**
    Set angle_offset ← actual_angle.
end if
Update Speed:
Adjust movement speed using confidence-weighted interpolation:

$$\text{speed} \leftarrow (1 - \text{confidence}) \cdot \text{speed} + \text{confidence} \cdot \frac{\text{distance\_moved}}{\text{timing}}$$

Calculate Target Angle and Timing:
Compute movement vector to the target position $(\Delta x_{\text{target}}, \Delta y_{\text{target}})$.
Calculate target angle relative to offset.
Compute corrected timing based on distance to target and current speed:

$$\text{corrected\_timing} \leftarrow \frac{\text{distance\_to\_target}}{\text{speed} + 10^{-6}}$$

State Update:
Update angle, timing, and last location to current position.
Return angle and timing required to reach target position.
End Algorithm.

Algorithm 4 Minimum Piecewise Euclidean Distance Algorithm

Input:
    Trajectories $\mathcal{T}_1 = \{p_1^{(1)}, p_2^{(1)}, \ldots, p_n^{(1)}\}$ and $\mathcal{T}_2 = \{p_1^{(2)}, p_2^{(2)}, \ldots, p_m^{(2)}\}$.
Output:
    Risk score R indicating potential collision risk.
Step 1: Initialize Minimum Distance
Set $d_{\min} \leftarrow \infty$.
Step 2: Compute Pairwise Distances
for each point $p_i^{(1)} \in \mathcal{T}_1$ **do**
  for each point $p_j^{(2)} \in \mathcal{T}_2$ **do**
    Compute the Euclidean distance $d_{ij} \leftarrow \|p_i^{(1)} - p_j^{(2)}\|_2$.
    Update $d_{\min} \leftarrow \min(d_{\min}, d_{ij})$.
  end for
end for
Step 3: Calculate Risk Score
Compute the risk score using an exponential decay function:

$$R \leftarrow \exp(-d_{\min})$$

Output Risk Score:
Return R, where a higher value indicates greater collision risk.

3.6 Algorithm Steps

The swarm navigation system operates through a pipeline that integrates object detection, map construction, localization, path planning, movement parameter calculation, and collision avoidance. These steps ensure that each robot (Sphero Bolt) can autonomously navigate towards a goal while dynamically adapting to changing conditions (seen in Figure 3).

1. **Environment Processing and Object Detection:**
    - Capture an overhead image and convert it to HSV color space.
    - Detect obstacles by isolating designated color ranges. Large detected regions are subdivided into smaller rectangular areas, and those containing a sufficient number of target-colored pixels are marked as obstacles.
    - Identify the goal region as a bounding box and designate its center as the target node. A specialized color calibration tool ensures adaptability to varying lighting conditions.
2. **PRM Generation:**
    - Sample candidate nodes across the environment, avoiding obstacle regions derived from the object detection step.

- Connect nodes using a growing radius approach, starting from 100 pixels and expanding to 500 pixels until each node has at least three connections.
- Employ a breadth-first search (BFS) to prune nodes that cannot ultimately reach the goal.
- Store the final set of nodes in a KD-tree structure, enabling efficient nearest-node queries. This PRM offers a flexible, probabilistic representation of the environment's navigable space.

3. **MCL:**
    - Each robot maintains a set of weighted particles distributed across the environment.
    - Using camera-based observations, a Gaussian Mixture Model (GMM) is fitted to the region corresponding to the robot's color.
    - Particle weights are updated based on their proximity to the GMM mean, and then normalized. Particles are resampled, incorporating Gaussian noise to maintain exploration.
    - The weighted mean of particles provides the robot's estimated position and a confidence measure derived from the covariance.
    - This localization step is performed at every move, ensuring robust, real-time position tracking despite sensor noise or dynamic conditions.
4. **Feedback-Based Movement:**
    - On the first movement, the robot calibrates its heading. It compares the Sphero's internal reference (e.g., 0 degrees) to the global reference established by the overhead camera view. An angle offset is calculated using $\arctan 2$, aligning the robot's local orientation with the global coordinate system.
    - Subsequent moves rely on MCL outputs. The last known location and speed estimates are combined with the current MCL-derived position and confidence.
    - The algorithm computes the movement vector between the last and current positions, determining actual angle changes and distance traveled. It adjusts speed using a confidence-weighted update, allowing the robot to compensate if it was previously stuck or moving too slowly.
    - The robot then identifies the nearest PRM node to its current position and plans a path to the goal using algorithms such as A*. The next movement angle and timing are calculated, ensuring that the robot advances efficiently toward its target node.
5. **Collision Risk Assessment and Path Adjustment:**

- Before executing movements, each robot submits its intended trajectory to a centralized Planner.
- Using a minimum piecewise Euclidean distance evaluation, the Planner calculates a risk score. An exponential decay function ensures that closer trajectories yield higher collision risk values.
- If the risk is too high, the Planner requests re-planning to avoid potential collisions. Adjusted PRM-based paths are computed to circumvent shared nodes or spatial conflicts.
- This step ensures that multiple robots can navigate simultaneously without unsafe proximity, even in crowded or dynamically changing environments.

6. **Execution and Feedback Loop:**

   - Once a safe trajectory is established, angle and timing parameters are sent to the Sphero Bolt robot via a WebSocket server.
   - The robot moves to the next way-point and sends feedback confirming its action completion. This triggers the next cycle of localization, path-finding, and movement parameter calculations.
   - Over multiple iterations, the swarm navigation framework continuously refines position estimates, adapts speeds, and reroutes paths as needed to maintain efficient and collision-free navigation.

3.7 Integration

All components—MCL for localization, PRM for path planning, feedback-based movement for fine-tuned navigation, and minimum piecewise Euclidean distance for collision avoidance—operate as a cohesive pipeline. Camera-based object detection establishes the environment's navigable space, PRM provides feasible routes, MCL refines each robot's position estimates, and feedback-based adjustments ensure efficient motion. This seamless integration enables autonomous, coordinated behavior in complex, real-world conditions.

3.8 Communication and Feedback

A centralized WebSocket server facilitates communication, relaying commands from the Brain Server to the Controls Server, which commands the Sphero Bolt robots. Upon finishing an action, a Sphero Bolt robot sends a message back across the WebSocket server to the Brain Server, where it begins to calculate the next move.

3.9 Theoretical Justification

This approach draws on established probabilistic and swarm methodologies. PRM's probabilistic sampling, as exemplified by SwarmPRM [1], efficiently handles complex, obstacle-rich terrains without exhaustive search. Similarly, MCL's

particle-based reasoning offers robust localization in noisy, unpredictable environments. Furthermore, the use of an exponential decay function for collision risk evaluation aligns with these probabilistic principles, prioritizing close-range interactions while allowing distant trajectories to coexist with minimal interference. This computationally efficient method reduces conflicts in multi-robot systems, reinforcing the suitability of our chosen algorithms for dynamic swarm applications. Together, these theoretical foundations underpin the robustness, scalability, and adaptability demonstrated in our real-world implementation.

4 Technical Demonstration

4.1 Figures and Visualization

Figure 3 demonstrates the HSV color masking capabilities of OpenCV can be seen through the color mask tweaking tool developed for this swarm navigation framework.

Figure 3: Color mask adjustment tool.

Figure 4 illustrates the MCL process, with the larger circles representing a Gaussian Mixture Model of where the swarm navigation framework thinks the robot is.

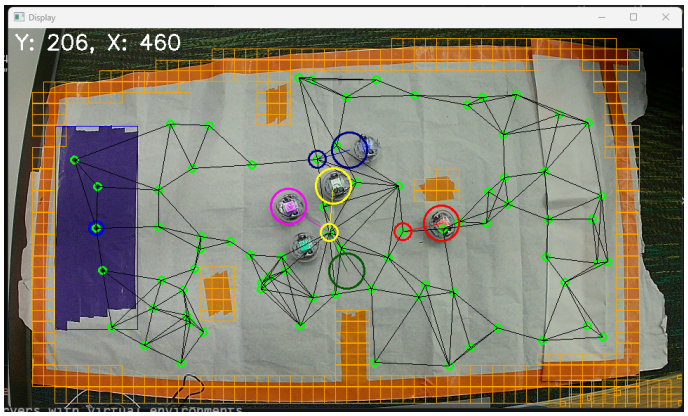

Figure 4: Three Spheros being located.

Figure 5 shows an example PRM-generated path.

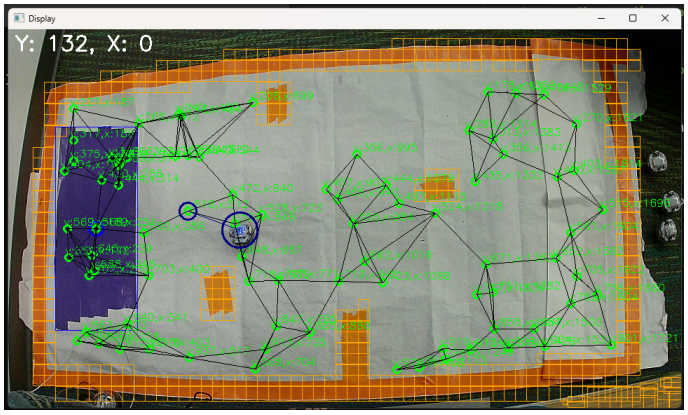

Figure 5: Probabilistic Road-map Generation.

4.2 Proof of Scalability and Real-World Feasibility

The scalability of the swarm navigation framework was tested by incrementally increasing the number of robots from one to three. Initially, the swarm navigation framework was implemented and tested with a single Sphero Bolt to refine localization, mapping, and movement algorithms, which can also be seen on YouTube (`https://youtu.be/COBFwmSk2Rg`) During the video, you can see the Sphero Bolt robot's perceived 0°(green line), which is different for each Sphero Bolt robot, and the coordinate system's 0°(red line). Subsequent trials with three robots demonstrated the swarm navigation framework's capability to manage multiple agents simultaneously, coordinating their paths without collisions, which can also be seen on YouTube (`https://youtu.be/y_hWa-OE-xM`).

This proof of concept highlights the potential for real-world scalability. While limited by the processing power of the server laptop and the overhead camera's field of view, the swarm navigation framework's modular architecture and computational efficiency suggest it could be extended to larger teams of robots with upgraded hardware. Such scalability is essential for applications like search and rescue missions, where swarms of robots must collaborate over large areas.

5 Discussion

5.1 Error and Limitation Analysis

Several challenges were encountered during the swarm navigation framework's implementation. The internal measurements of the Sphero Bolt robots were prone to noise, particularly in angle measurements. To address this, an angle offset calculation was introduced, aligning the Sphero Bolt robot's internal reference frame with the global coordinate system derived from the overhead camera. This calibration step mitigated inconsistencies caused by the noisy compass measurements.

Furthermore, the speed calculation leveraged confidence-based weighting to dynamically adjust robot movement timing. If a previous move was over- or under-compensated, the swarm navigation framework adjusted timing accordingly, allowing the robot to traverse different terrains effectively. This adaptive approach proved crucial in scenarios with uneven surfaces or high-friction zones, where simple deterministic movement strategies would fail.

While these mitigations improved robustness, further testing with higher-quality sensors or additional calibration steps could enhance performance. Addressing localization drift in larger environments also remains an area for future exploration.

5.2 Unique Contributions

This swarm navigation framework demonstrates a novel combination of MCL and PRM tailored to low-cost, off-the-shelf hardware like Sphero Bolt robots. Unlike previous works that relied on sophisticated sensors or computationally intensive methods, our system achieved robust swarm navigation using an affordable camera and consumer-grade robots.

The integration of localization-confidence-based movement timing adjustment and angle offset calibration further distinguishes this swarm navigation framework, enabling the swarm navigation framework to adapt to noisy measurements and challenging terrains. These innovations provide a foundation for scalable, practical swarm robotics solutions that can operate in dynamic, real-world environments, such as disaster response or warehouse automation, without requiring specialized hardware.

5.3 Conclusion and Future Work

The proposed framework integrates MCL, PRM, feedback-based movement adjustments, and minimum piecewise Euclidean distance collision avoidance into a unified swarm robotics system. Using a single overhead camera and consumer-grade robots, the system efficiently navigates complex environments with multiple agents, recalibrating paths and speeds in real-time to avoid collisions. This approach demonstrates clear potential for practical applications, including search and rescue missions and industrial automation.

Despite these promising results, several limitations remain. The processing speed and capacity of the server laptop restricted both the number of Spheros that could be controlled simultaneously and the frequency at which new plans could be generated. Additionally, the reliance on a single overhead camera limited the scalability of the swarm navigation framework to larger environments. In future work, these constraints can be mitigated by upgrading computing equipment, employing more powerful workstations, or utilizing distributed computing architectures. Incorporating additional sensors, multiple cameras, or more advanced localization technologies will further enhance the swarm navigation framework's coverage and reliability. In this way, we can rapidly improve upon the current proof of concept, which—despite its short development period—has already yielded a highly functional and adaptable swarm navigation framework. This progress underscores the potential of leveraging existing, readily available technology to create systems with immediate practical applications in both research and real-world operations.

References

[1] R. Arslan, M. Alimohammad, and M. Zare. Swarmprm: Probabilistic roadmap motion planning for swarm robotic systems. *arXiv preprint*, 2024.

[2] R. Hasani, R. Tanno, T. Davies, and L. Daniel. Navformer: A transformer architecture for robot target-driven navigation in unknown and dynamic environments. *arXiv preprint*, 2024.

[3] R. Havangi, M. A. Nekoui, and M. Teshnehlab. A multi swarm particle filter for mobile robot localization. *International Journal of Computer Science*, 7(3):15–22, 2010.

[4] X. Hu. Slam with the sphero robot. Master's thesis, Universitat Politècnica de Catalunya, Automatic Control and Robotics, 2018.

[5] M. Kegeleirs, R. Todesco, D. G. Ramos, G. L. Herranz, and M. Birattari. Mercator: Hardware and software architecture for experiments in swarm slam. Technical Report TR/IRIDIA/2022-012, IRIDIA, ULB, 2022.

[6] C. Wustrau. Building a scalable swarm application using sphero robots, 2019.