This project represents my own work in accordance with University regulations

Brian Goodluck Mmari

Simulated Annealing for Robust Swarm Navigation in Obstacle-Dense Environments

Brian Goodluck Mmari

Abstract

This project explores how simulated annealing (SA), a method inspired by how metal cools and settles into a low-energy state, can improve how robots navigate through environments full of obstacles. The approach builds on techniques introduced in Lecture 8 of CBE 449, where simulated annealing was explored as a probabilistic optimization strategy. The project builds on a simulation of the Artificial Fish Swarm Algorithm (AFSA), which mimics how fish move in groups by "preying," "swarming," and "following" behaviors. This work focuses on a 2D version of the system and uses simulated annealing to optimize the repulsion weight through a pre-simulation annealing routine, then apply it to the main potential field for each run enabling the robotic agents to reach goals more efficiently, take shorter paths, and avoid collisions. The study begins by evaluating how the system scales with the number of robotic fish (N) testing performance in increasingly crowded scenarios. A numerical stability analysis is also conducted, measuring how small perturbations in a robot's initial orientation impact its trajectory. Overall, results show that simulated annealing significantly improves the performance of potential field-based algorithms by effectively tuning obstacle repulsion, leading to more robust and efficient swarm navigation.

Table of Contents

Abstract 1

1. Background
- 1.1 Potential Field Algorithm 3–4
  - 1.1.1 Mathematical Formulation 3
  - 1.1.2 Implementation in Code 4
  - 1.1.3 How the Robot Uses the Field 4
- 1.2 Setup and Environment 4–5
- 1.3 Simulated Annealing 5–8

2. Methods
- 2.1 Metrics 9
- 2.2 Simulation Setup 10

3. Results and Discussion
- 3.1 Simulated 2D Potential Field Results 11–13
  - 3.1.1 Summary of Results 14
- 3.2 Simulated Potential Field with SA Results 14–18
  - 3.2.1 Summary of Results 18
- 3.3 Perturbed Initial Orientation for Potential Field 19–23
  - 3.3.1 Summary of Results 23–24
- 3.4 Perturbed Initial Orientation for Potential Field with SA 24–27
  - 3.4.1 Summary of Results 27

4. Limitations and Future Work

5. Conclusion

6. Github Repository

Background

1.1 Potential Field Algorithm

The Potential Field Algorithm is a widely used method for path planning in robotics. It treats the robot as a particle that is:

- **Attracted** to a goal point.
- **Repelled** by surrounding obstacles.

The idea is to create a scalar potential field $U(x, y)$ over the workspace, where the robot can follow the negative gradient (steepest descent) toward the goal while avoiding obstacles [3].

1.1.1 Mathematical Formulation

- **Attractive Potential** toward the goal:

$$U_{\text{att}}(x, y) = \frac{1}{2} k_{\text{att}} \cdot ((x - x_g)^2 + (y - y_g)^2)$$

  where (x_g, y_g) is the goal position and k_{att} is a constant [3].

- **Repulsive Potential** from obstacles:

$$U_{\text{rep}}(x, y) = \begin{cases} \frac{1}{2} k_{\text{rep}} \left(\frac{1}{d(x,y)} - \frac{1}{d_0} \right)^2 & \text{if } d(x, y) \leq d_0 \\ 0 & \text{otherwise} \end{cases}$$

  where $d(x, y)$ is the distance from point (x, y) to the nearest obstacle, and d_0 is the threshold of influence [3].

- **Total Potential**:

$$U(x, y) = U_{\text{att}}(x, y) + U_{\text{rep}}(x, y)$$

  [3]

1.1.2 Implementation in Code

The algorithm is implemented in the `ComSurfacePotentialField` class. The main steps are:

1. Extract obstacle and goal positions:

```
obstacle_pos_group = [obstacle.mPos for obstacle in self.mObstacleList]
gx, gy = self.mTarget[0:2]
```

2. Define resolution and map size:

```
reso = (np.max(self.mX) - np.min(self.mX)) / len(self.mX)
map_size = (np.min(self.mX), np.min(self.mY), np.max(self.mX), np.max(self.mY))
```

3. Compute the field:

```
data, _, _ = calc_potential_field2(gx, gy, ox, oy, rr, reso, map_size)
self.mData = np.array(data).T
```

4. Draw the potential field using contour plots:

```
plt.contourf(potential_field.mX, potential_field.mY, potential_field.mData, cmap=cm
```

1.1.3 How the Robot Uses the Field

The robot can:

- Follow the gradient of the potential field to move toward lower potential values.
- Use optimization techniques (e.g., simulated annealing) to find globally optimal paths.

This method allows robots to navigate environments with complex obstacle configurations efficiently and with minimal computation compared to graph-based methods.

1.2 Setup and Environment

The first step of this project involved cloning the repository using the following command:

```
git clone https://github.com/xzlxiao/SwarmRobotics.git
```

This repository contains a simulization platform for displaying the Artificial Fish Swarm Algorithm that uses the Potential Field algorithm.

Next, the required dependencies were installed by running:

```
pip install -r requirements.txt
```

After setting up the environment, I navigated into the project directory to begin development. For my implementations, I primarily focused on the 2D simulation environment due to its simplicity and ease of visualization. To run 2D simulations, I used the following command: The reference code for the above algorithm can be found here: `exm_path_planning-2Dpy` Below is an example of the visualization provided by this platform:

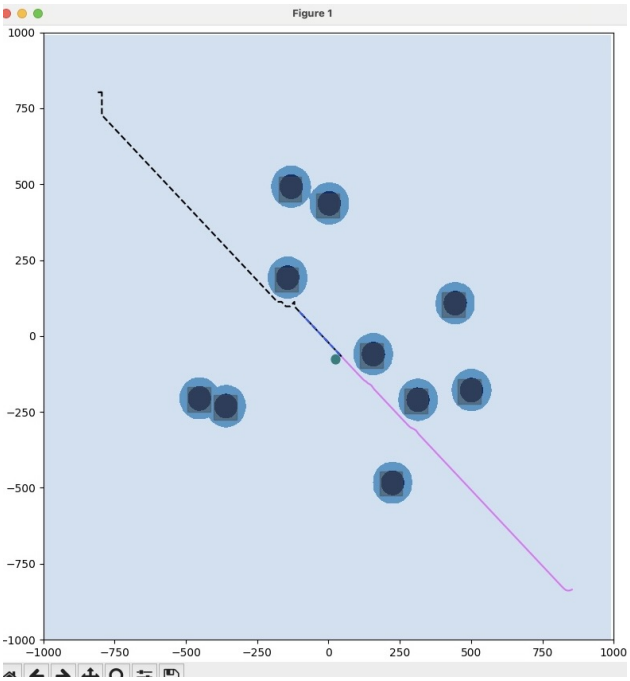

Figure 1.1: 2D simulation with 1 fish and 10 obstacles.

1.3 Simulated Annealing

To improve obstacle avoidance without compromising goal convergence, I introduced simulated annealing(SA) as an optimization strategy for tuning the repulsion weight parameter in our potential field. Simulated annealing is a probabilistic technique inspired by the physical process of annealing in metallurgy, which allows occasional uphill moves to escape local minima. It is particularly effective for navigating non-convex search spaces such as ours, where robot trajectories are influenced by dynamic potential fields formed from the positions of obstacles and targets [1, 2].

Specifically, it was used to optimize the repulsion weight, which governs how strongly agents are penalized for proximity to obstacles. This enhancement was implemented by modifying

the `ComSurfacePotentialField` class with three additional methods. The first method, `compute_path_length()`, employs simulated annealing to iteratively search for the optimal repulsion weight that minimizes field cost.

Listing 1.1: Simulated Annealing for computing optimal path

```
def compute_path_length(self, step_size=5.0, max_steps=1000):
        """
        Simulates gradient descent from the robot's current position
            to the goal
        on the potential field. Returns the total path length.
        """
        if self.mData is None or self.mBindingRobot is None:
            return float("inf")

        pos = np.array(self.mBindingRobot.mPos[:2])
        path_length = 0.0

        for _ in range(max_steps):
            ix = int((pos[0] - np.min(self.mX)) / ((np.max(self.mX)
               - np.min(self.mX)) / len(self.mX)))
            iy = int((pos[1] - np.min(self.mY)) / ((np.max(self.mY)
               - np.min(self.mY)) / len(self.mY)))

            # Boundary check
            if ix < 1 or ix >= self.mData.shape[0] - 1 or iy < 1 or
               iy >= self.mData.shape[1] - 1:
                return float("inf")

            # Estimate gradient
            grad_x = (self.mData[ix+1, iy] - self.mData[ix-1, iy]) /
                2.0
            grad_y = (self.mData[ix, iy+1] - self.mData[ix, iy-1]) /
                2.0
            grad = np.array([grad_x, grad_y])

            if np.linalg.norm(grad) < 1e-3:  # Close to flat or
               local minimum
                break

            # Move along the negative gradient
            direction = -grad / np.linalg.norm(grad)
            next_pos = pos + step_size * direction
            path_length += np.linalg.norm(next_pos - pos)
            pos = next_pos

            # Check if near goal
            if np.linalg.norm(pos - np.array(self.mTarget[:2])) <
               step_size:
```

```
                path_length += np.linalg.norm(pos - np.array(self.
                    mTarget[:2]))
                break

        return path_length
```

The second method, `evaluate_field_quality()`, returns the optimal path length from the method described before.

Listing 1.2: Evaluation of Potential Field Quality

```
    def evaluate_field_quality(self):
        return self.compute_path_length()  # Shorter path is better
```

The third method, `calibrate_for_shortest_path()`, calculates the repulsion weight that gives out the shortest path.

Listing 1.3: Calibration for Shortest Path

```
def calibrate_for_shortest_path(self, initial_weight=0.1, iterations
    =100, initial_temp=40.0, cooling_rate=0.98):
        self.repulsion_weight = initial_weight
        best_weight = initial_weight
        best_score = self.evaluate_field_quality()

        current_temp = initial_temp

        for _ in range(iterations):
            candidate_weight = self.repulsion_weight + np.random.
                uniform(-0.1, 0.1)
            candidate_weight = max(0.01, min(candidate_weight, 3.0))

            self.repulsion_weight = candidate_weight
            self.update()

            score = self.evaluate_field_quality()

            delta = score - best_score
            if delta < 0 or np.exp(-delta / current_temp) > np.
                random.rand():
                best_score = score
                best_weight = candidate_weight

            current_temp *= cooling_rate

        self.repulsion_weight = best_weight
        self.update()
        print(f"Optimal repulsion weight for shortest path: {
            best_weight}")
```

Together, these modifications allow the robotic agents to adaptively learn safer and more efficient navigation strategies.
The reference code for the above algorithm can be found here: `exm_path_planning-2D_sa.py`

Methods

2.1 Metrics

To quantitatively evaluate the performance of the potential field algorithm as well as the integration of potential field algorithm with SA, I designed a metrics system that captures key indicators of navigation efficiency and reliability. This system helps us assess how well robots reach their targets, how quickly they do so, and how safely they move through obstacle-prone environments.

The primary metrics recorded for each robot in each trial are:

- **Steps Taken**: The number of discrete steps a robot took during simulation before reaching the goal.
- **Goal Reached**: A Boolean value indicating whether the robot reached the designated goal (within a defined radius of tolerance).
- **Path Length**: The total Euclidean distance traveled by the robot across its trajectory, computed as the sum of distances between consecutive positions.
- **Time Taken**: The simulation time (in seconds) the robot required to reach the goal or the maximum allotted runtime if it failed.
- **Collisions**: The number of times the robot collided with an obstacle during navigation.

These metrics were stored in a CSV file using Python's `csv` module, enabling post-processing and aggregate evaluation across multiple trials. The following CSV files were obtained:

- **Potential Field Algorithm simulation results:** `simulation_results.csv`
- **Potential Field Algorithm with SA results:** `simulation_results_sa.csv`
- **Perturbed Initial Orientation Potential Field Algorithm results:** `simulation_results_initorient.csv`
- **Perturbed Initial Orientation Potential Field with SA results:** `simulation_results_sa_peturb.csv`

2.2 Simulation Setup

To analyze how the number of robots influences overall performance, we ran simulations with swarm sizes of $N = \{1, 2, 4, 8, 16\}$. Each configuration was run across three randomized trials to reduce the impact of noise and outlier trajectories. The simulation environment included:

- A bounded 2D space with 10 randomly placed static obstacles.
- Robots initialized at randomized positions near $(800, -800)$.
- A common goal location fixed at $(-800, 800)$.
- A shared potential field computed using the `ComSurfacePotentialField` class.

The following steps summarize the analysis conducted in this project:

- Simulated a 2D potential field algorithm with agent counts $N = \{1, 2, 4, 8, 16\}$ and 3 trials per N.
- Measured path length, time taken, steps, and collisions.
- Visualized results with histograms and performance plots.
- Added simulated annealing to tune the repulsion weight adaptively.
- Extended `ComSurfacePotentialField` with:
  - `calibrate_for_shortest_path()` – optimizes repulsion weight for the shortest path.
  - `evaluate_field_quality()` – scores the potential field.
  - `compute_path_length()` – returns the optimal path length.
- Compared standard vs. SA-enhanced potential field using percent change in metrics.
- Tested numerical stability by perturbing initial orientation.
- Analyzed robustness of each method under perturbations.

Results and Discussion

From the analysis described in the Methods section this is some of the results obtained as well as their physical significance.

3.1 Simulated 2D Potential Field Results

The reference code used to obtain the results below can be found here: `exm_path_planning-2D.py`

The figures below show different plots that evaluate the effectiveness of the Potential Field Algorithm without improvements.

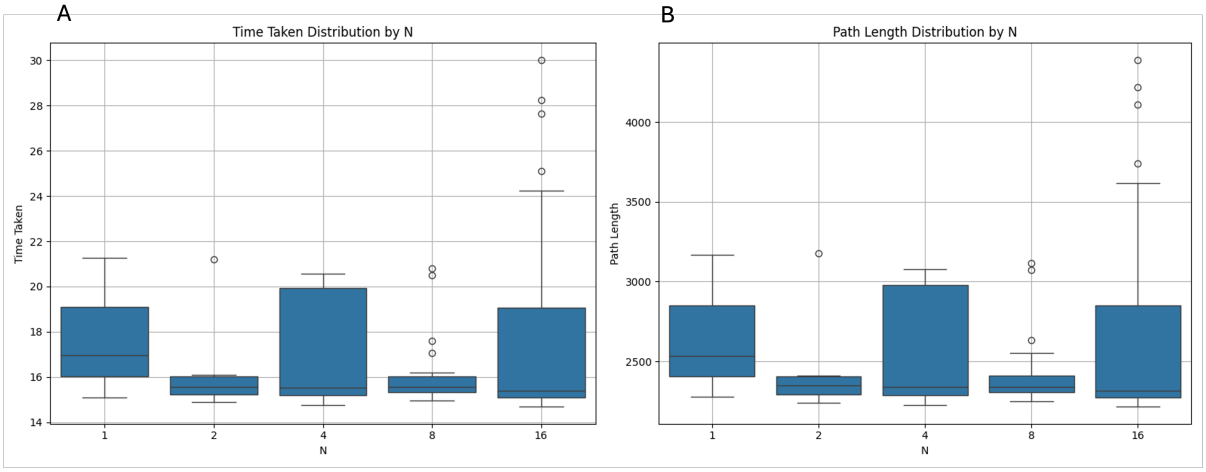

Figure 3.2: Boxplots of time taken (A) and path length (B) for varying swarm sizes N under the baseline AFSA system without repulsion tuning or initial orientation perturbations. Time and path length show moderate variability, with increased spread as N increases, suggesting coordination challenges in larger swarms.

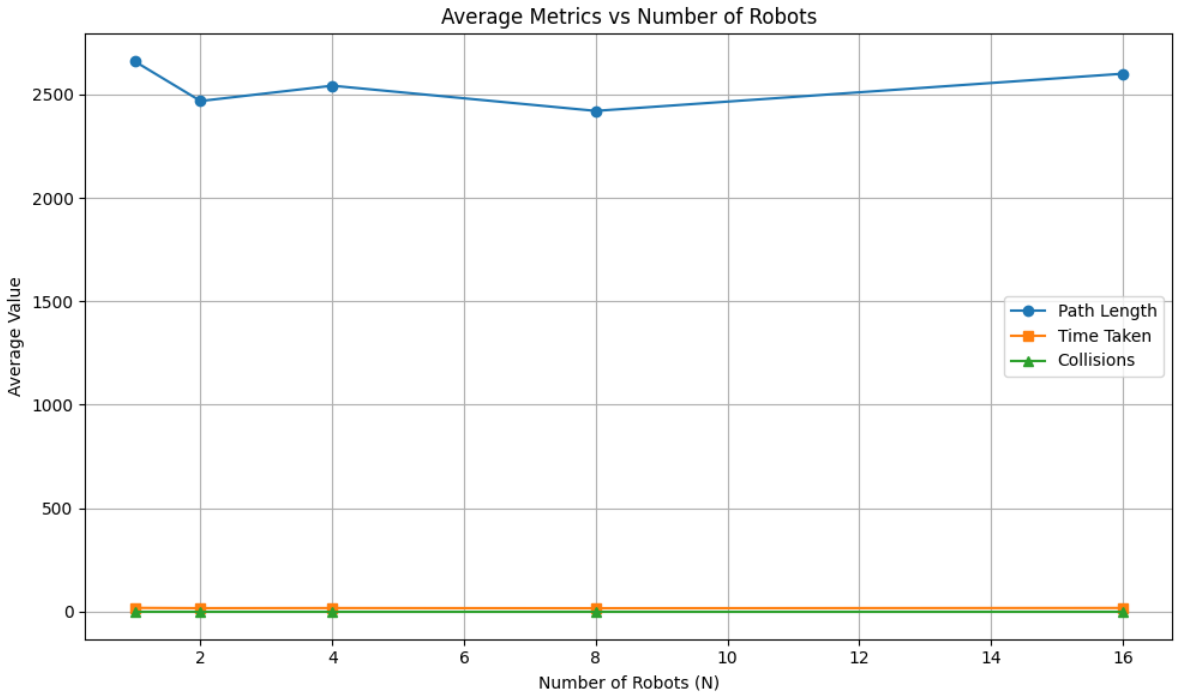

Figure 3.3: Average performance metrics across swarm sizes N under baseline conditions. Time taken and path length gradually increase with N, while collision rates remain low but show a slight uptick at N=16, reflecting crowding effects.

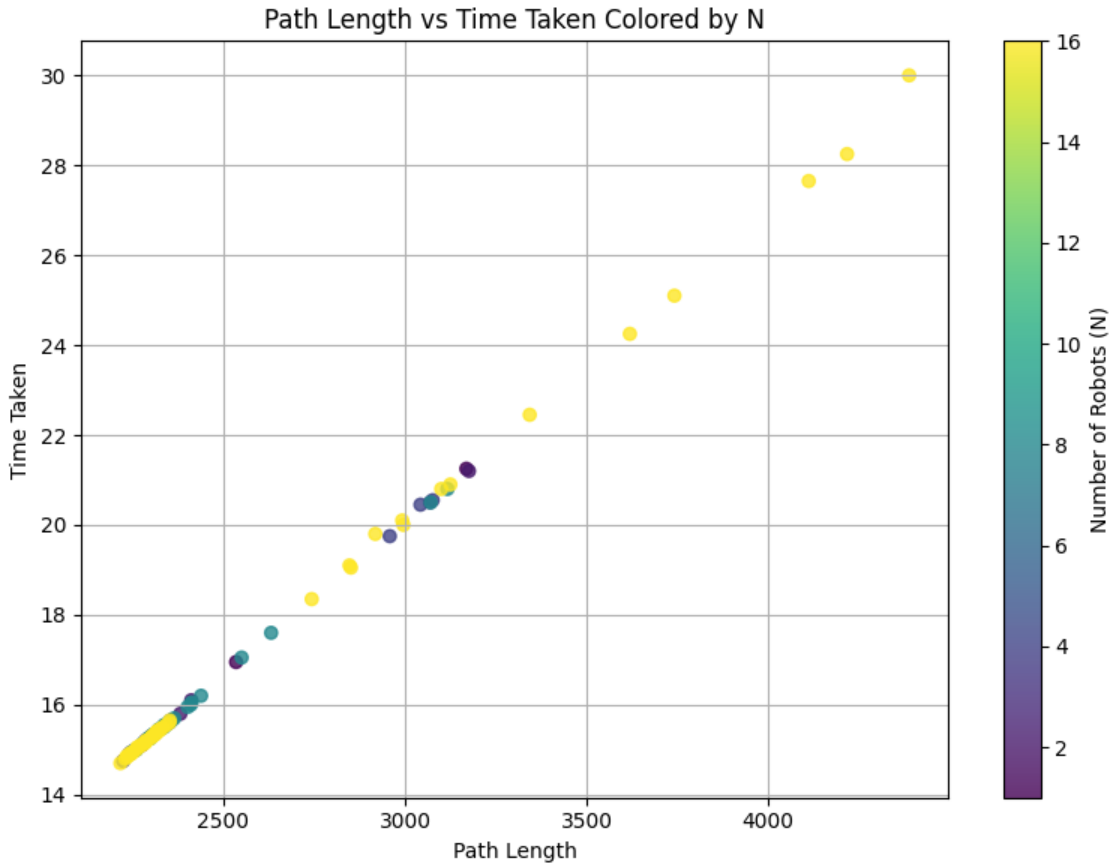

Figure 3.4: Scatter plot of time taken versus path length under the baseline AFSA system. A positive correlation is observed, with longer paths typically requiring more time. Larger swarms (yellow points) exhibit greater dispersion, indicating inconsistent efficiency without parameter tuning.

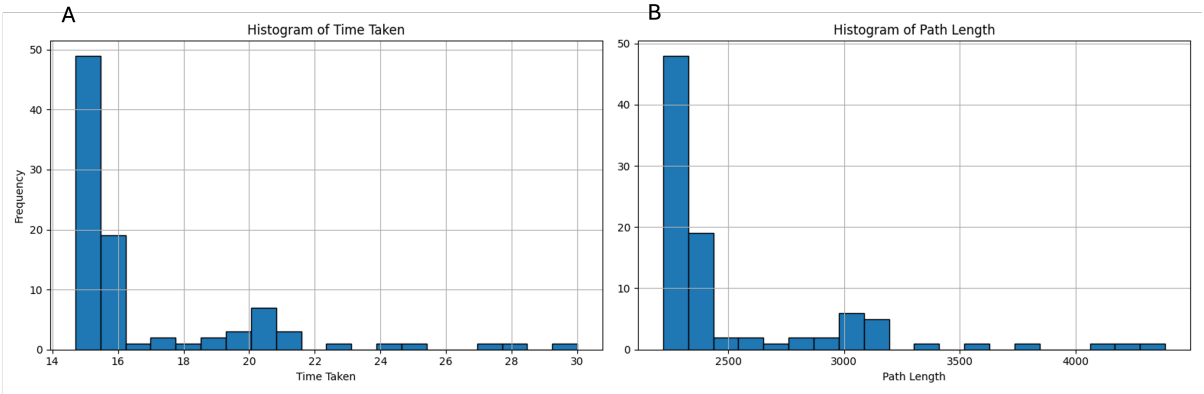

Figure 3.5: Histograms of overall time taken (A) and path length (B) under baseline AFSA conditions. The distributions are right-skewed, with most trials completing quickly and efficiently, but a long tail of underperforming runs suggests occasional suboptimal behavior.

3.1.1 Summary of Results

- Moderate swarm sizes (e.g., $N = 2$ or $N = 8$) lead to the most efficient and consistent performance, as shown by reduced variance and fewer outliers in both time taken and path length.
- Extreme values of N (such as $N = 1$ or $N = 16$) result in degraded performance due to either limited coverage (small N) or increased coordination complexity (large N).
- Collision rates is zero across all swarm sizes, indicating effective avoidance behavior in the control strategy.
- Non-monotonic trends are observed: increasing the number of robots does not guarantee improved performance, suggesting an optimal range of swarm sizes exists.
- Strong positive correlation between path length and time taken is evident, especially for larger swarm sizes (N), which show greater variability in outcomes.

3.2 Simulated Potential Field with SA Results

The reference code used to obtain the results below can be found here: `exm_path_planning-2D_sa.py`

To ensure a fair comparison with our baseline setup, we kept all environmental and control parameters constant between the integrated AFSA with simulated annealing (SA) version and the original AFSA implementation. These include:

- Robot Initialization: All robots start at randomized positions near the bottom-right quadrant.
- Obstacles: Ten obstacles are placed at random positions within the same bounded region, with zero velocity and fixed size (radius = 40).
- Motion Parameters: The stride length (15), robot radius (15), target goal position, and communication range remain unchanged.
- Potential Field Grid: Identical grid resolution and bounds for the X and Y dimensions ($[-1000, 1000]$) are used in both conditions.
- Simulation Time: All trials are run for a fixed duration of 30 seconds.

During an initial pre-run phase, the system evaluates potential fields with different repulsion weights sampled from a local neighborhood using the following logic:

1. A candidate repulsion weight is randomly sampled near the current value.
2. The field is updated and scored using a custom `evaluate_field_quality()` function, which penalizes high potential values near obstacles while promoting smooth, low-potential trajectories.

3. The new candidate is accepted based on the Metropolis criterion, with the acceptance probability determined by the score difference and the current annealing temperature.

4. The temperature is geometrically cooled at each iteration.

It's important to note that the below results don't contain values for N=16. This is because the program crashed due to an unresolved internal error. The following are the results obtained from integrating Simulated Annealing in the Potential Field algorithm:

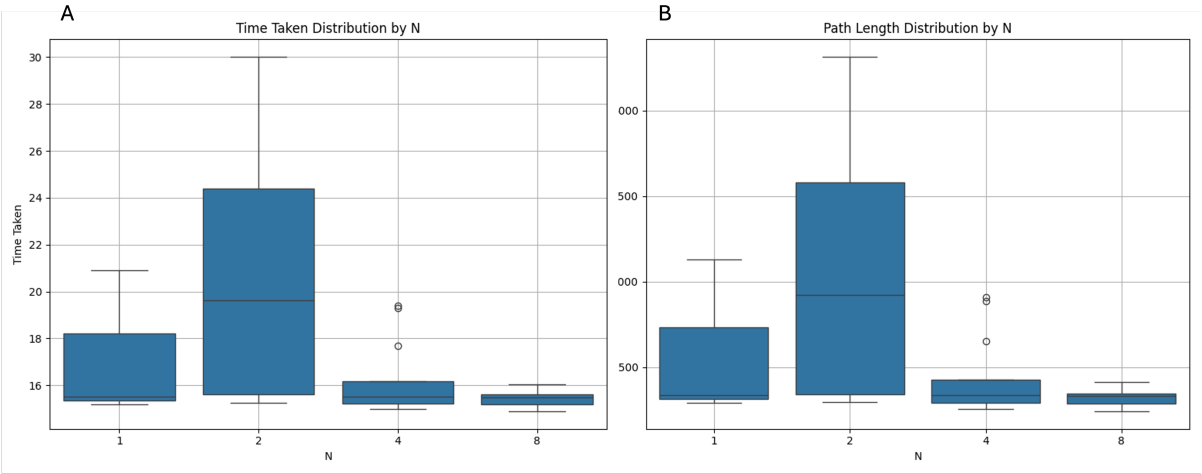

Figure 3.6: Boxplot distributions of simulation performance metrics by number of robots N (A) Time taken shows greater variability at N=2, with larger teams (e.g., N=4, N=8) demonstrating more consistent and lower durations. (B) Path length also exhibits high spread at N=2, while higher team sizes result in shorter, more stable trajectories. These patterns highlight the impact of repulsion weight tuning via simulated annealing, which promotes efficient and coordinated navigation as team size increases.

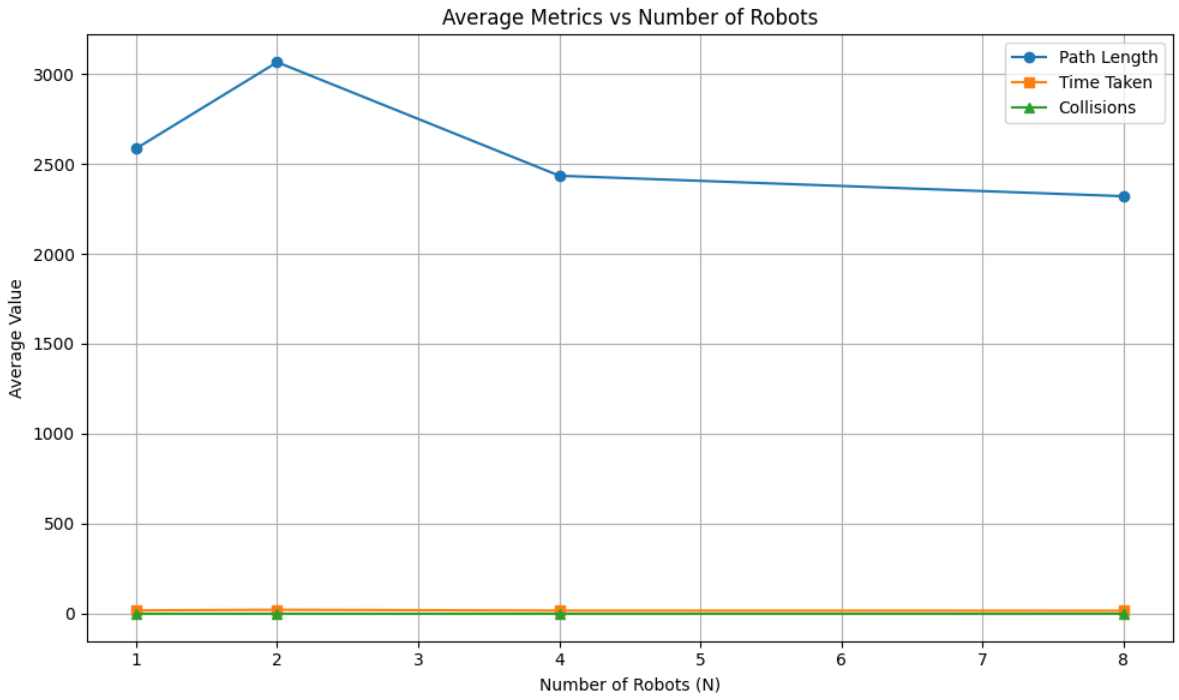

Figure 3.7: Average path length, time taken, and collision count across simulations for varying numbers of robots N, using repulsion weights optimized via simulated annealing. As N increases, average path length and time taken generally decrease after peaking at N=2, indicating improved coordination and efficiency. Collision counts remain consistently low across all team sizes, reflecting effective obstacle avoidance

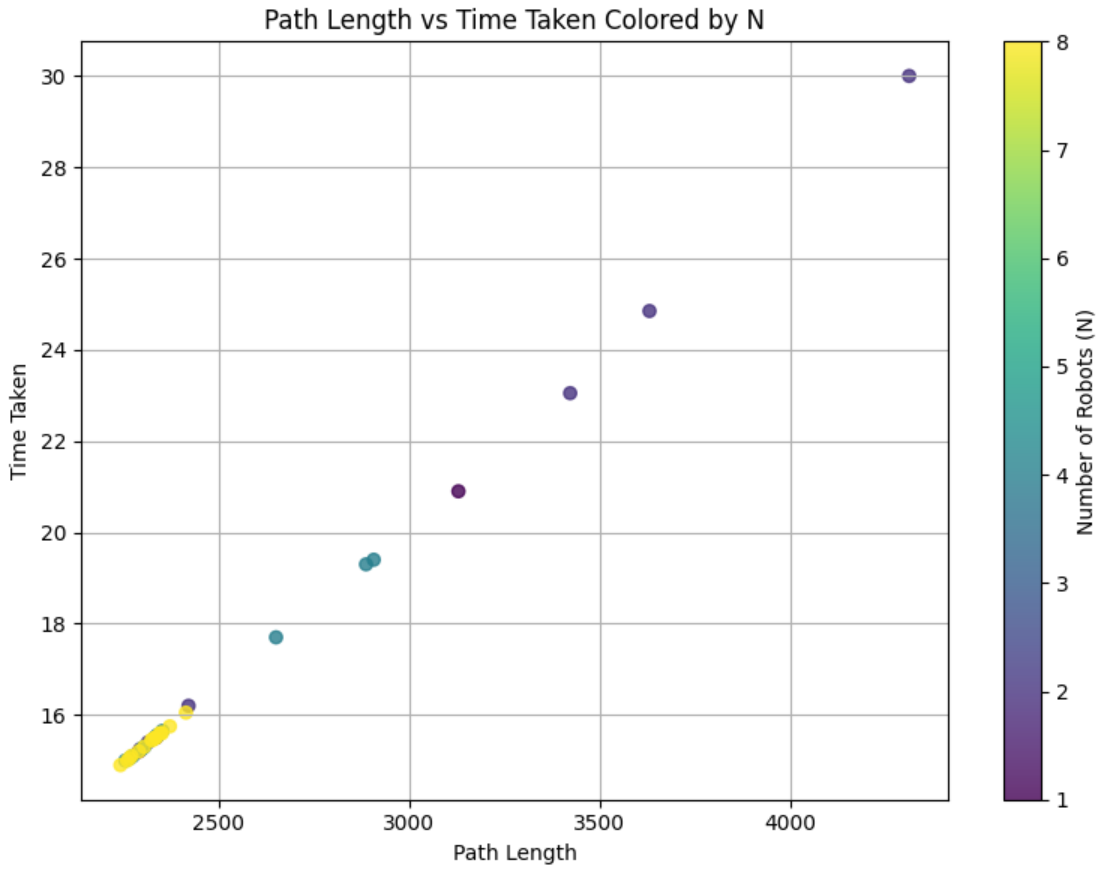

Figure 3.8: Scatter plot showing the relationship between path length and time taken across simulations, colored by the number of robots N. A strong positive correlation is observed: longer paths generally result in longer completion times. Simulations with larger N values (yellow points) tend to cluster toward the lower-left region, indicating shorter paths and faster goal attainment, highlighting the benefits of repulsion weight tuning via simulated annealing.

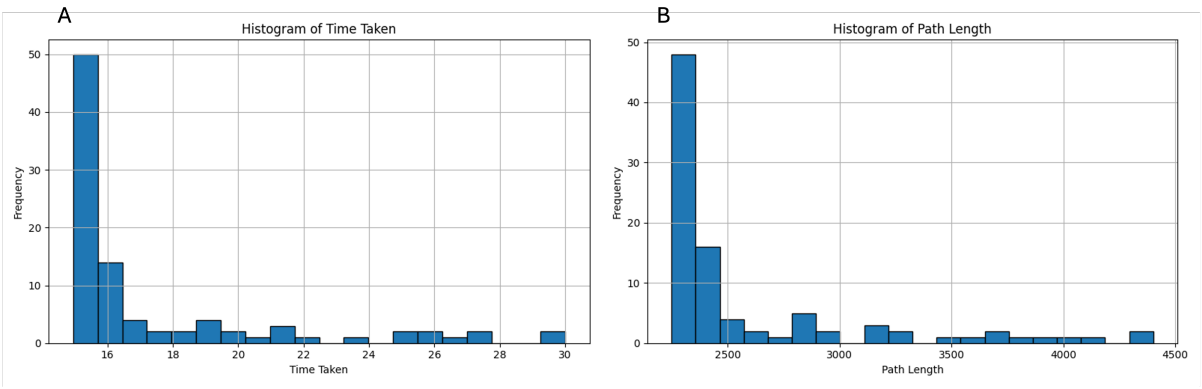

Figure 3.9: Histograms of performance metrics from simulated annealing-enhanced potential field trials. (A) Histogram of time taken, showing a right-skewed distribution with most simulations completing in under 17 seconds. (B) Histogram of path length, similarly skewed, with the majority of paths concentrated below 2500 units. These results reflect the impact of tuning the repulsion weight prior to each simulation, which helped minimize both duration and path length in most trial.

3.2.1 Summary of Results

The integration of Simulated Annealing (SA) into the potential field framework led to measurable improvements in robot navigation. Key findings include:

- Consistent Goal Reaching: Most robots reached their goals within 15–16 seconds, as shown in the time distribution histogram. Only a few trials exhibited extended durations.
- Efficient Path Planning: Time taken was significantly reduced even as the number of robots (N) increased as seen from Figure 3.6. Surprisingly when N = 2, the time taken was significantly high.
- Scalability with Swarm Size: Average path length peaked at $N = 2$, then declined, indicating that sometimes moderate swarm density can cause crowding effects, but SA-tuned repulsion mitigates this at higher densities. Time taken remained stable across swarm sizes.
- No Collisions: No collisions were detected indicating robustness.
- Path-Time Correlation: A strong positive correlation was observed between path length and time taken. As swarm size increases, outliers emerge with longer paths and durations, likely due to more complex avoidance dynamics.

These results demonstrate that Simulated Annealing effectively tunes the potential field's repulsion parameters to produce stable, efficient, and collision-free navigation even in multi-agent scenarios.

3.3 Perturbed Initial Orientation for Potential Field

The reference code used to obtain the results below can be found here: `exm_path_planning-2D_initpeturb`

To evaluate the numerical instability of the swarm system under orientation uncertainty, we modified the simulation script as follows:

- **New Output Files:** The simulation logs were redirected to `simulation_output_initorient.txt`, and results were saved to `simulation_results_initorient.csv` to differentiate from the baseline case.
- **Orientation Perturbation:** Each robot's initial orientation was randomized by applying a small angular perturbation:
  - A base orientation θ was sampled uniformly from $[0, 2\pi]$.
  - A random perturbation $\varepsilon \in [-10°, +10°]$ was added to θ, converted to radians.
- **Histogram Visualization:** Histograms of each performance metric (`Steps Taken`, `Path Length`, `Time Taken`, `Collisions`) were generated and saved to `results/histograms/` for visual analysis of distributional shifts under perturbation.

The above work is visualized in the figures below:

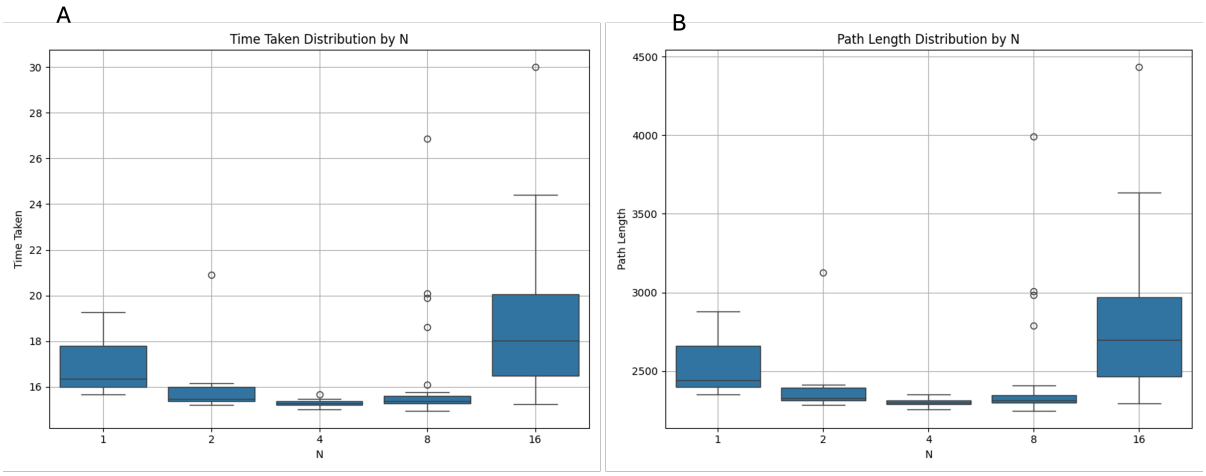

Figure 3.10: The distributions highlight that larger swarms (N=16) exhibit significantly higher variance in both metrics, indicating greater sensitivity to orientation perturbations. Smaller groups maintain more stable performance, while outliers in larger swarms suggest amplified effects of misalignment in initial heading direction.

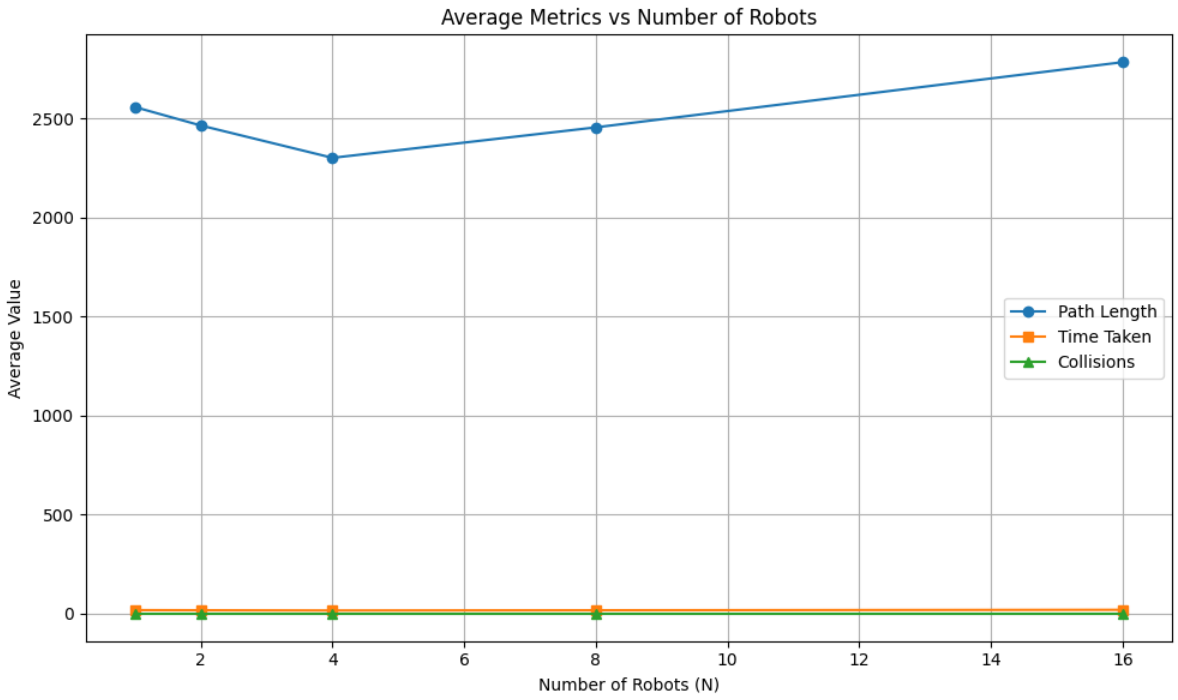

Figure 3.11: This line plot compares the average path length, time taken, and number of collisions across varying team sizes (N) under random perturbations in initial robot orientation. While time and collisions remain relatively stable, path length exhibits more sensitivity, suggesting that orientation misalignment can lead to longer trajectories as coordination complexity increases

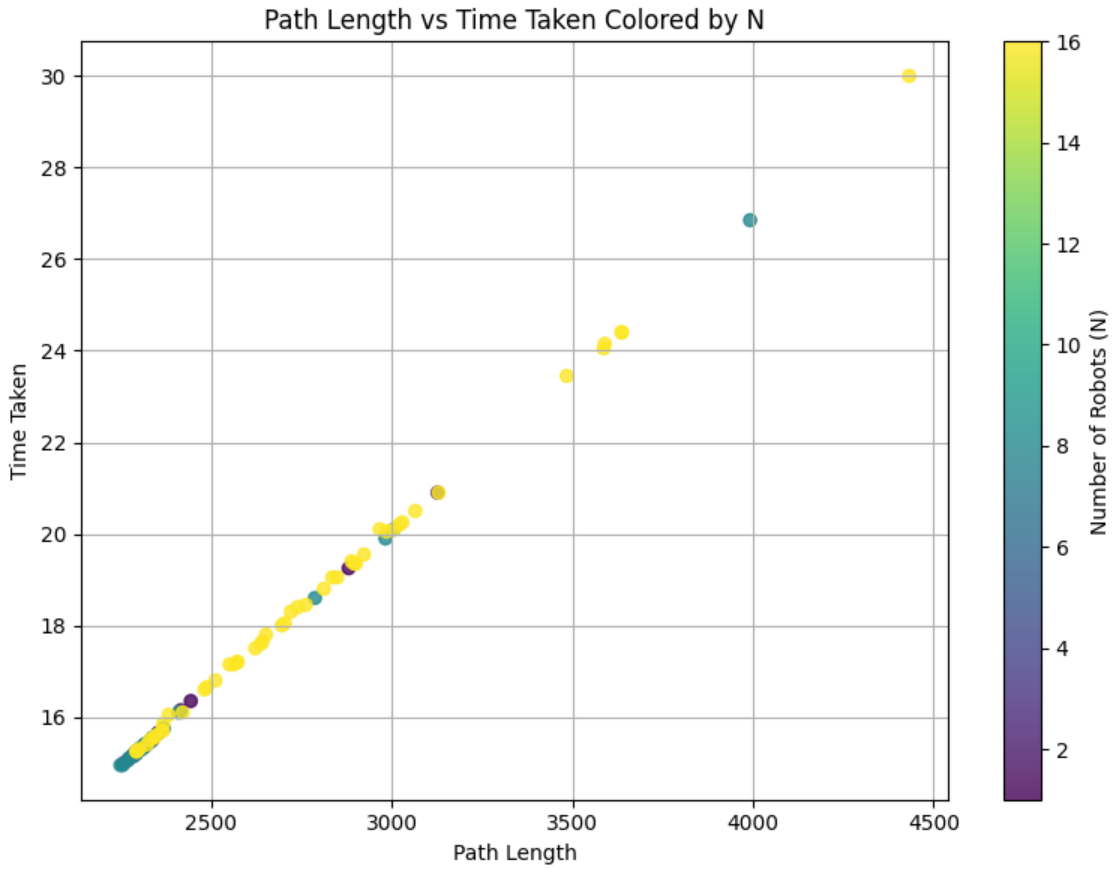

Figure 3.12: This figure illustrates the correlation between path length and time taken across all simulations with perturbed initial orientations. Each point is colored by the number of robots involved. A positive trend is observed, indicating that longer paths generally correspond to longer times. The spread increases with higher N, suggesting amplified sensitivity to orientation perturbations in larger robot swarms.

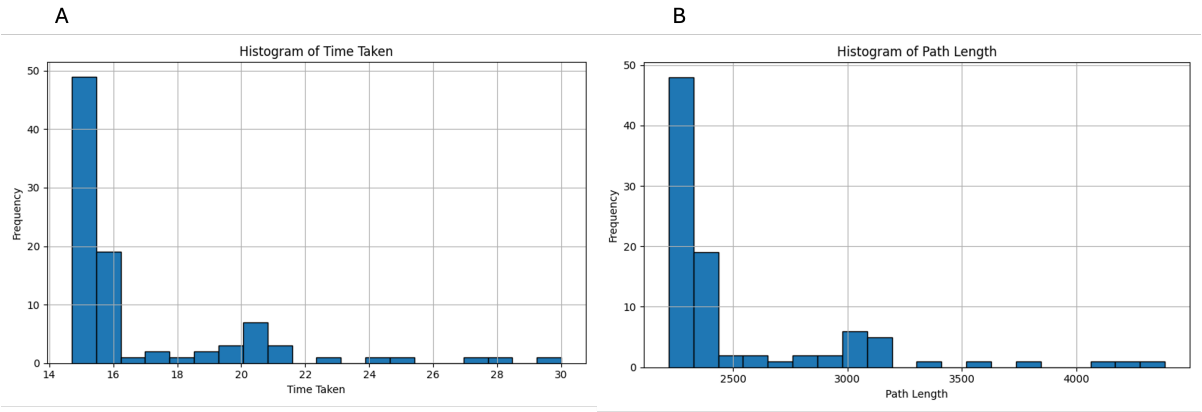

Figure 3.13: Both histograms show a right-skewed distribution, where most robots complete their paths with relatively short times and path lengths. However, a small number of simulations experience significantly longer durations and distances, highlighting how small perturbations in initial orientation can occasionally lead to suboptimal trajectories and extended navigation times

The above results can also be summarized in the table below which shows differences between simulation with and without perturbed initial orientations.

```
(myenv) (base) brianmmari@dynamic-oit-ip4-wifirestricted03-10-49-54-146 SwarmRobotics % python3.13 numerical.py
       Simulation  Robot ID  Path Length_instability  Time Taken_instability  Collisions_instability
0    N=1, Trial=0         0               -25.775629              -26.352941                     NaN
1    N=1, Trial=1         0                13.610511               13.569322                     NaN
2    N=1, Trial=2         0                 7.373292                8.278146                     NaN
3    N=2, Trial=0         0                 0.150911                0.310559                     NaN
4    N=2, Trial=0         1                -1.558805               -1.415094                     NaN
..            ...       ...                      ...                     ...                     ...
88  N=16, Trial=2        11                 1.619123                1.650165                     NaN
89  N=16, Trial=2        12                 7.548438                8.387097                     NaN
90  N=16, Trial=2        13                 1.870931                1.967213                     NaN
91  N=16, Trial=2        14                 4.244800                4.377104                     NaN
92  N=16, Trial=2        15                13.312709               13.915858                     NaN
```

Figure 3.14: Numerical instability analysis due to initial orientation perturbations. The table shows per-robot differences in Path Length, Time Taken, and Collisions compared to unperturbed runs. Positive values indicate increases due to perturbation. NaN in `Collisions_instability` means no collisions occurred in either trial for that robot.

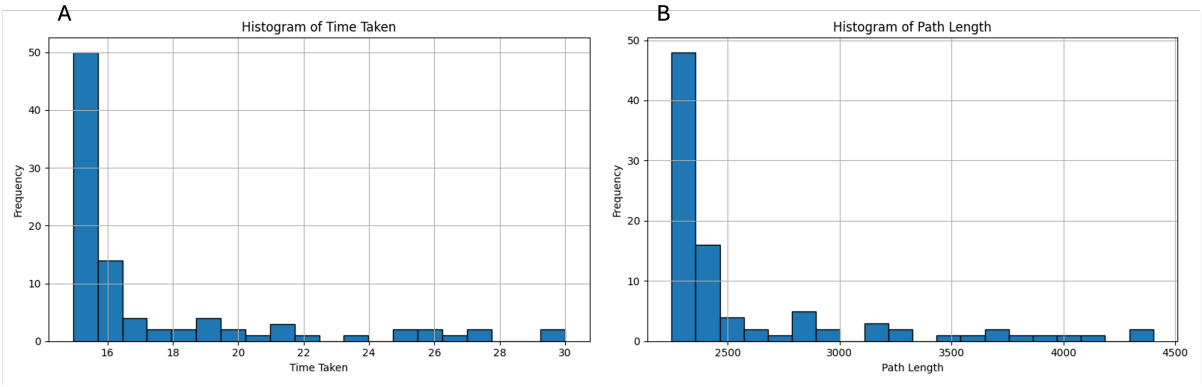

Figure 3.15: Distribution of performance metrics across all trials with simulated annealing-tuned repulsion weights.(A) Histogram of time taken by robots to reach the goal shows a right-skewed distribution, with most robots completing their task in under 18 seconds.(B) Histogram of path length also exhibits right skewness, with the majority of paths clustered around 2300–2600 units, but with some significantly longer trajectories likely caused by complex obstacle navigation

The formula used in the table below is given by:

$$\frac{\text{SA} - \text{Baseline}}{\text{Baseline}} \times 100$$

This represents the percentage change from the baseline to the Simulated Annealing (SA) result. Negative values from this formula indicate improvement (i.e., SA performed better than the baseline). This is because we want lower percent for metrics like path length and time taken.

```
(myenv) (base) brianmmari@dynamic-oit-ip4-wifirestricted03-10-49-54-146 SwarmRobotics % python3.13 numerical_sa_og.py
       Simulation  Robot ID  Path Length_instability  Time Taken_instability  Collisions_instability
0    N=1, Trial=0         0                -28.853217              -29.647059                     NaN
1    N=1, Trial=1         0                 -8.353186               -8.849558                     NaN
2    N=1, Trial=2         0                  8.477220                9.271523                     NaN
3    N=2, Trial=0         0                 -4.308854               -4.658385                     NaN
4    N=2, Trial=0         1                -27.401729              -27.830189                     NaN
..            ...       ...                       ...                     ...                     ...
88  N=16, Trial=2        11                  3.168649                3.960396                     NaN
89  N=16, Trial=2        12                  4.019410                4.838710                     NaN
90  N=16, Trial=2        13                  3.823623                4.590164                     NaN
91  N=16, Trial=2        14                 57.344394               59.595960                     NaN
92  N=16, Trial=2        15                  8.164475                8.737864                     NaN

[93 rows x 5 columns]
```

Figure 3.16: Performance comparison between standard potential field and simulated annealing-enhanced potential field. The table reports per-robot percentage changes in Path Length, Time Taken, and Collisions. Negative values indicate improved performance with simulated annealing. `NaN` in `Collisions_instability` indicates no collisions occurred in either version for that robot.

3.3.1 Summary of Results

- Orientation perturbations introduced variability in robot trajectories, especially for larger swarms (N=16), as shown by increased spread in path lengths and completion times.

- Path length was the most affected metric under orientation uncertainty, with larger robot groups showing longer and less consistent paths due to compounded misalignment effects.
- Simulated Annealing (SA)-tuned repulsion weights led to performance improvements in most cases, as seen in the comparative instability table where many values are negative.
- SA improved navigation efficiency by reducing path length and time taken across trials, while maintaining low or zero collisions, indicating enhanced robustness.
- Histogram and boxplot visualizations confirmed that while most trials performed efficiently, both perturbation and SA introduced occasional outliers, revealing edge cases in performance.

3.4 Perturbed Initial Orientation for Potential Field with SA

The reference code used to obtain the results below can be found here:
`exm_path_planning-2D_sa_initpeturb.py`

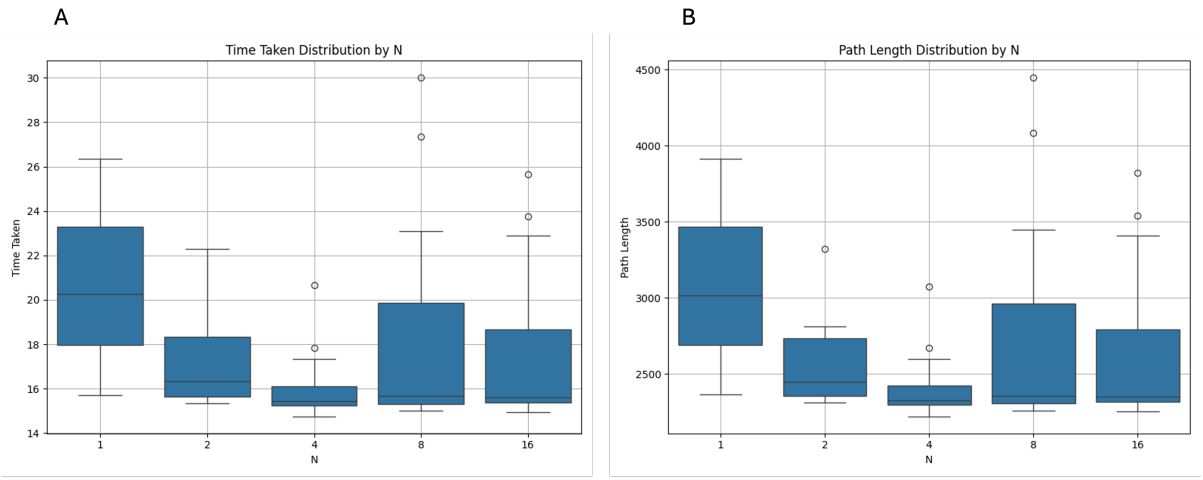

Figure 3.17: Boxplots of time taken (A) and path length (B) for varying N under the AFSA system with repulsion weight tuning via simulated annealing and orientation perturbations. Compared to the baseline (Fig. 3.2), median performance improves slightly, while variance increases for larger swarms, highlighting the challenge of consistent coordination under noise.

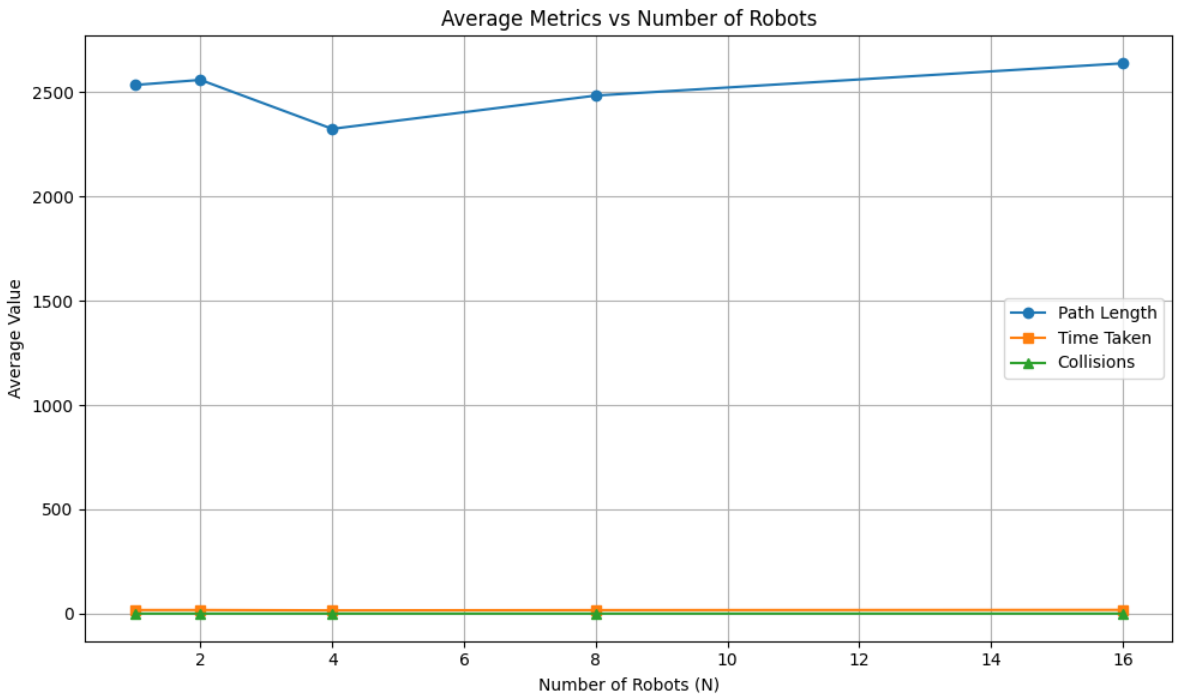

Figure 3.18: Average performance metrics for swarm sizes N under SA-tuned AFSA with orientation perturbation. Path length and time peak around N=2 and decrease toward N=4, suggesting an optimal swarm size for stable coordination. Collision counts remain low, validating robustness of repulsion tuning.

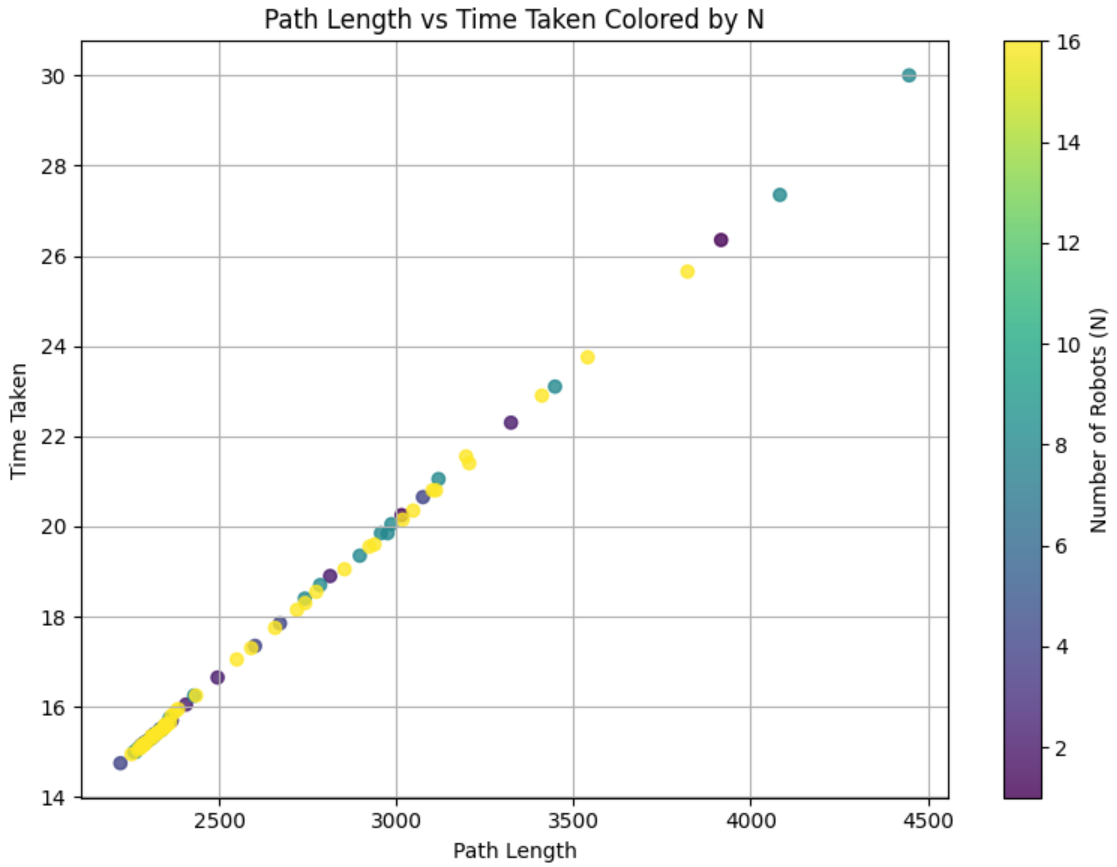

Figure 3.19: Scatter plot of time versus path length with SA-based repulsion tuning and initial orientation perturbations. Compared to Fig. 3.4, performance becomes more clustered and efficient for moderate swarm sizes (e.g., N=4), though some outliers remain due to stochastic start angles

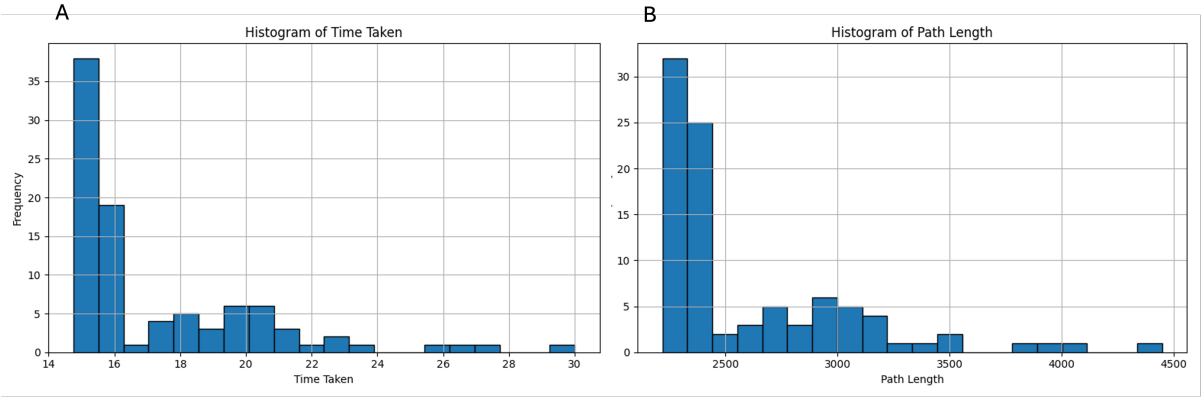

Figure 3.20: Histograms of time taken (A) and path length (B) under SA-tuned AFSA with initial orientation perturbations. Distributions are tighter and shifted left compared to Fig. 3.5, indicating that simulated annealing improved consistency despite the added noise.

The percentage change shown in the table below is computed using the formula:

$$\frac{\text{SA} - \text{Baseline}}{\text{Baseline}} \times 100$$

This quantifies how Simulated Annealing (SA) affects performance relative to the baseline potential field approach when initial orientations for both are perturbed. Negative values indicate improvement, as they represent reductions in metrics like path length and time taken, which are desirable in navigation tasks.

```
(myenv) (base) brianmmari@dynamic-oit-ip4-wifirestricted03-10-49-54-146 SwarmRobotics % python3.13 numerical_sa_peturbed.py
       Simulation  Robot ID  Path Length_instability  Time Taken_instability  Collisions_instability
0     N=1, Trial=0         0                 2.146212                2.341137                     NaN
1     N=1, Trial=1         0                 7.875913                8.414239                     NaN
2     N=1, Trial=2         0                13.181765               14.242424                     NaN
3     N=2, Trial=0         0                -3.286281               -3.583062                     NaN
4     N=2, Trial=0         1                -0.928950               -0.980392                     NaN
..             ...       ...                      ...                     ...                     ...
88   N=16, Trial=2        11                -5.076030               -5.714286                     NaN
89   N=16, Trial=2        12                -5.444320               -6.153846                     NaN
90   N=16, Trial=2        13                -4.999770               -5.642633                     NaN
91   N=16, Trial=2        14               -34.966729              -35.864979                     NaN
92   N=16, Trial=2        15                -8.086454               -8.630952                     NaN

[93 rows x 5 columns]
(myenv) (base) brianmmari@dynamic-oit-ip4-wifirestricted03-10-49-54-146 SwarmRobotics %
```

Figure 3.21: Numerical instability analysis under initial orientation perturbations using Simulated Annealing-enhanced potential fields. The table shows percentage changes in path length, time taken, and collisions for each robot across all trials. Negative values indicate improved robustness under perturbations. `NaN` in `Collisions_instability` indicates that no collisions occurred in either the baseline or perturbed runs.

3.4.1 Summary of Results

- Simulated Annealing (SA)-enhanced potential fields maintain robust navigation performance under initial orientation perturbations.
- Time taken and path length remain consistent across different swarm sizes (N), with slight increases in variability at $N = 16$.
- Average metrics (path length, time, collisions) show minimal degradation, confirming the effectiveness of SA-tuned repulsion weights.
- A strong positive correlation exists between path length and time taken, particularly in larger swarms where congestion effects are more pronounced.
- Histograms reveal that most trajectories are efficient, though a few outliers exist due to denser environments or suboptimal orientations.
- Numerical instability analysis shows that SA often improves performance compared to the baseline, as indicated by negative percentage changes in key metrics.

Limitations and Future Work

While the results of this project demonstrate promising improvements in swarm navigation using simulated annealing, several limitations remain that constrain generalizability and performance in more complex environments.

First, obstacle motion was disabled throughout the experiments. Dynamic obstacles introduce an additional layer of complexity, requiring robots to make real-time updates to their path plans to avoid collisions. In addition including moving obstacles during simulation resulted in frequent collisions largely due to the robot's limited field of view (FOV). The FOV was constrained to a forward-facing 45° range in the 2D plane, meaning that obstacles approaching from behind could not be detected or accounted for. In the future, increasing the FOV could be a favorable step towards testing dynamic obstacles.

Second, the simulation time was fixed in advance, meaning that if a robot failed to reach its goal within the allotted time, it was marked as unsuccessful, even if it was close to the goal or still actively navigating. This binary success criterion could be refined in future work to better capture near-success cases.

Third, the current simulated annealing implementation only tunes the repulsion weight before each simulation begins. While this pre-run optimization yielded improved performance, it does not account for mid-run environmental changes such as obstacle movements. Online adaptation of the repulsion weight, where robots adjust their potential field parameters in real-time based on local sensing, could further improve flexibility and robustness.

Additionally, the current system only optimizes the repulsion weight. Other potential field parameters such as the goal attraction strength, communication range, or stride length were held constant across trials. Future work could explore optimizing for multiple parameters.

Lastly, the experiments were limited to 2D environments. While this makes the simulations more tractable and easier to visualize, it limits applicability to real-world scenarios such as aerial drone swarms or underwater vehicles, which operate in 3D space. Extending the framework to 3D would be interesting to study.

Conclusion

This work demonstrates the effectiveness of simulated annealing in enhancing swarm navigation within obstacle-dense environments. By tuning the repulsion weight of the potential field before each trial, robotic agents were able to avoid collisions, follow shorter paths, and reach their goals more efficiently. The results revealed that simulated annealing led to more stable and robust navigation performance across various swarm sizes, particularly under perturbations in initial orientation.

The approach builds on optimization strategies introduced in CBE 449, where simulated annealing was discussed in the context of escaping local minima and exploring complex energy landscapes. By adapting this principle to swarm robotics, the project highlights how classical optimization tools can be repurposed for dynamic control in autonomous systems.

Moreover, the study identified key trends, including the non-monotonic effect of swarm size on performance and the increased sensitivity of larger groups to orientation noise. These insights provide a foundation for exploring more adaptive or real-time tuning mechanisms in the future.

Looking ahead, further improvements could involve dynamically adjusting repulsion weights during navigation, incorporating moving obstacles, or extending the model to 3D environments. Overall, this project underscores the value of combining biologically inspired algorithms with principled optimization techniques to address real-world challenges in robotics

Github Repository

https://github.com/BRIANMMARI10/MultiRobot-PotentialField-SA/tree/main

Bibliography

[1] Panagiotis D. Christofides. Lecture 3: Optimization algorithms. `http://paros.princeton.edu/CBE449/lec_opt3.pdf`, 2023. CBE 449: Process Control, Princeton University, accessed May 6, 2025.

[2] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science (New York, N.Y.)*, 220:671–80, 06 1983.

[3] Dengyu Zhang, Guobin Zhu, and Qingrui Zhang. Multi-robot motion planning: A learning-based artificial potential field solution, 2023.