COMPUTATIONAL INTELLIGENCE MCTA 3371

SEMESTER 1 2025/2026

# MINI PROJECT
# INTELLIGENT MOBILE ROBOT NAVIGATION USING COMPUTATIONAL INTELLIGENCE TECHNIQUES

SECTION 2

LECTURER: DR AZHAR BIN MOHD IBRAHIM

| NAME | MATRIC NO. |
|---|---|
| MUHAMAD IQBAL BIN MD ISA | 2313247 |
| ADAM NABIL HANIFF BIN RUZAIMI | 2313203 |
| UMAR FAROUQI BIN ABU HISHAM | 2314995 |
| HARIZ IRFAN BIN MOHD ROZHAN | 2318583 |

## ABSTRACT

This project presents the design and implementation of a hybrid intelligent navigation controller for a mobile robot using soft computing techniques. The system utilizes a dual-phase approach: Fuzzy Logic for heuristic exploration and a Genetic Algorithm (GA) for global path optimization. We simulated the robot across three maps of increasing complexity, ranging from 13x13 to 25x25 grid sizes. To improve navigation performance, the controller was tuned with a 15% stochastic curiosity factor and an immediate goal-sensing rule to ensure logical movement. Results demonstrate that the hybrid Fuzzy-GA controller significantly improves path efficiency, achieving an efficiency gain of over 20% in complex maze environments compared to standalone exploration.

## 1.0 INTRODUCTION

Mobile robot navigation is the critical ability to move from a start point to a goal while autonomously avoiding obstacles. Traditional mathematical models can be difficult to implement in cluttered environments; therefore, soft computing techniques like Fuzzy Logic and Genetic Algorithms are preferred for their flexibility and approximate reasoning.

This project aims to design an intelligent controller that determines a robot's discrete moves forward, left, or right while ensuring safe and efficient actions. By using a hybrid approach, we demonstrate how a robot can first learn a maze-like layout and then optimize its path through evolutionary heuristics.

We tested the controller's performance across three distinct map layouts: Simple (13x13), Intermediate (19x19), and Complex (25x25). The robot's movement is visualized in real-time using matplotlib, which clearly tracks the "scars" of discovery like explored areas and backtracking.

The project highlights a hybrid soft-computing approach in action. First, we use **Fuzzy Logic** for heuristic exploration to handle the uncertainty of an unknown maze. Then, we apply **Fuzzy-GA** optimization to refine the discovery timeline into an elite, shortest path. This demonstrates exactly how a robot can navigate efficiently by first learning the environment and then evolving its path to eliminate mistakes.

## 2.0 OBJECTIVES

The main objectives of this project are:

- **Design and Implementation**: To create a navigation controller using hybrid soft computing techniques, specifically Fuzzy Logic and Genetic Algorithms.
- **Simulation**: To demonstrate the robot's behavior in different 2D maps, from simple layouts to complex mazes.
- **Evaluation**: To measure the effectiveness of the hybrid controller using metrics like path efficiency, success rate, and time steps.
- **Comparison**: To validate the hybrid approach by comparing its performance against a standalone fuzzy controller.

## 3.0 METHODOLOGY

The methodology for this project is centered around a hybrid computational intelligence system where a mobile robot navigates through randomly generated grid-based mazes. The system is built in Python using *matplotlib* for real-time visualization. The core logic transitions from **stochastic discovery (Fuzzy Logic)** to **evolutionary refinement (Genetic Algorithm)** to solve the navigation challenge efficiently.

### 3.1 Environment Setup

We designed a 2D grid environment in Python where the robot must move from a start point (Yellow) to a goal point (Purple) while avoiding obstacles (Dark Blue-Grey).

- **Map Complexity**: We implemented three distinct layouts to test the scalability of our soft computing controller:
    - **Simple (13x13)**: Minimal obstacles for basic validation.
    - **Intermediate (19x19)**: Increased wall density to test local decision-making.
    - **Complex (25x25)**: A dense, maze-like layout to challenge the optimization phase.

All three maps are randomly generated using the same map algorithm so the robot faces different layouts every run.

**Code (Map Generation):**

```
# ==========================================
# 1. MAP GENERATION
# ==========================================
def generate_maze(rows, cols):
   maze = [[1 for _ in range(cols)] for _ in range(rows)]
   stack = [(1, 1)]
   maze[1][1] = 0
   while stack:
       cx, cy = stack[-1]
       neighbors = []
       for dx, dy in [(0, -2), (0, 2), (-2, 0), (2, 0)]:
           nx, ny = cx + dx, cy + dy
           if 1 <= nx < cols - 1 and 1 <= ny < rows - 1 and maze[ny][nx] == 1:
               neighbors.append((nx, ny, dx, dy))
       if neighbors:
           nx, ny, dx, dy = random.choice(neighbors)
           maze[cy + dy // 2][cx + dx // 2] = 0
           maze[ny][nx] = 0
           stack.append((nx, ny))
       else:
           stack.pop()
   return maze
```

- **Robot Model**: The robot is represented as a point robot moving in discrete steps (North, South, East, West) on a tile-based grid.
- **Maze Generation**: All maps are generated using a recursive backtracker algorithm, ensuring every maze is a "perfect maze" with a guaranteed path to the goal.

**Inputs considered during movement:**

- distance from goal
- local obstacle configuration

**Outputs (movement actions):**

- forward

- left
- right

In the current implementation, movement decisions are derived from exploring unvisited neighbors rather than a physical motion model.

**Visualisation System:**

The visualizer is built in matplotlib. The environment is displayed using colored patches representing:

- walls
- corridors
- visited cells
- backtracking
- shortest path
- robot location

This gives a clear real-time view of what the algorithm is doing.

**Code (Visualization and Robot Model):**

```
# ==========================================
# 2. VISUALISATION AND ROBOT MODEL (Constants)
# ==========================================
LAYOUTS = {"Simple": (13, 13), "Intermediate": (19, 19), "Complex": (25, 25)}
SPEEDS = {"Simple": 0.05, "Intermediate": 0.02, "Complex": 0.01}

COLOR_WALL = '#2C3E50'
COLOR_PATH = '#ECF0F1'
COLOR_START = '#F1C40F'
COLOR_GOAL = '#8E44AD'
COLOR_ROBOT_FUZZY = '#E74C3C'
COLOR_ROBOT_GA = '#3498DB'
COLOR_EXPLORED = '#AED6F1'
COLOR_BACKTRACK = '#BDC3C7'
COLOR_GA_PATH = '#27AE60'
```

## 3.2 Computational Intelligence Modeling

To achieve the highest marks, we implemented a **Hybrid Soft Computing** controller. The system operates in two distinct trials.

### Phase 1: Fuzzy Logic Exploration

During the initial discovery phase, the robot has no knowledge of the maze layout. It uses a Fuzzy Inference System (FIS) to decide its next move.

- **Input**: Distance to the goal and local obstacle availability.
- **Fuzzy Rule**: If the distance to the goal is low and the path is clear, then desirability is high.
- **Stochastic Factor**: We added a "Curiosity" factor (15% random choice) to simulate real-world uncertainty and discovery errors.
- **Goal-Sensing Logic**: We upgraded the controller with a priority rule: if the goal is an immediate neighbor, the robot chooses it 100% of the time, overriding the random factor for more logical movement.

### Code (Fuzzy Exploration)

```python
# ==========================================
# 3. FUZZY CODE
# ==========================================
def fuzzy_decision(neighbors, goal):
   scored = []
   for n in neighbors:
       dist = abs(n[0] - goal[0]) + abs(n[1] - goal[1])
       score = 1.0 / (dist + 1)
       scored.append((score, n))
   scored.sort(key=lambda x: x[0], reverse=True)
   return scored[0][1]
```

### Phase 2: Genetic Algorithm (GA) Optimization

Once the Fuzzy robot reaches the goal, the system transitions to a GA to optimize the recorded "discovery timeline".

- **Fitness Evaluation**: Candidate paths are evolved based on total path length and smoothness.
- **Elite Evolution**: The GA removes redundant movements (backtracking) found in Phase 1, resulting in an "Elite" green path.

**Code ( Fuzzy + GA Optimization)**

```
# ==========================================
# 4. FUZZY-GA CODE SECTION
# ==========================================
def get_continuous_path(waypoints, maze):
    full_path = []
    rows, cols = len(maze), len(maze[0])
    for i in range(len(waypoints) - 1):
        start_pt, end_pt = waypoints[i], waypoints[i + 1]
        if start_pt == end_pt: continue
        queue = deque([start_pt])
        came_from = {start_pt: None}
        found = False
        while queue:
            curr = queue.popleft()
            if curr == end_pt:
                found = True
                break
            for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
                nx, ny = curr[0] + dx, curr[1] + dy
                if 0 <= nx < cols and 0 <= ny < rows:
                    if maze[ny][nx] == 0 and (nx, ny) not in came_from:
                        came_from[(nx, ny)] = curr
                        queue.append((nx, ny))
        if found:
            segment = []
            curr = end_pt
            while curr is not None:
                segment.append(curr)
                curr = came_from.get(curr)
            full_path.extend(segment[::-1][:-1])
    full_path.append(waypoints[-1])
    return full_path


def genetic_optimization(start, goal, timeline, maze, layout_type):
    gens = 150 if layout_type == "Complex" else 80
    pop_size = 50
    population = []
    for _ in range(pop_size):
        pts = random.sample(timeline, min(len(timeline), 8))
        pts.sort(key=lambda x: timeline.index(x))
        population.append([start] + pts + [goal])

    for _ in range(gens):
        population.sort(key=lambda p: len(get_continuous_path(p, maze)))
        next_gen = population[:20]
        while len(next_gen) < pop_size:
            parent = random.choice(next_gen)
            child = list(parent)
            if len(child) > 3:
                idx = random.randint(1, len(child) - 2)
                prev_t_idx = timeline.index(child[idx - 1])
                if prev_t_idx < len(timeline) - 1:
                    child[idx] = random.choice(timeline[prev_t_idx:])
            mid = list(set(child[1:-1]))
            mid.sort(key=lambda x: timeline.index(x))
            next_gen.append([start] + mid + [goal])
        population = next_gen
    return population[0]
```

### 3.3 Simulation and Visualization

The simulation is executed through PyCharm, providing a live graphical view of the robot's performance.

- **Trial 1 Visualization**: Shows the Fuzzy robot exploring, leaving a light blue trail for visited cells and grey for backtracked dead ends.
- **Trial 2 Visualization**: Shows the GA "Elite" robot executing the optimized green path markers.

- **Performance Metrics**: The GUI includes live step counters to evaluate path efficiency and time steps needed to reach the goal.

**Code (Visualization and Movement):**

```
# ==========================================
# 5. VISUALISATION AND MOVEMENT (Main Engine)
# ==========================================
def run_comparison(layout_name, rows, cols):
   # --- Map Generation Call ---
   maze = generate_maze(rows, cols)
   start, goal = (1, 1), (cols - 2, rows - 2)
   speed = SPEEDS[layout_name]

   # --- Visualisation and Robot Model (Setup) ---
   fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 7))
   fig.canvas.manager.set_window_title(f"Elite Comparison: {layout_name}")
   fig.suptitle(f"Fuzzy exploration vs Fuzzy-GA Optimization ({layout_name})", fontsize=16)

   fuzzy_txt = ax1.text(cols / 2, rows + 0.5, "Steps: 0", ha='center', color='red', weight='bold')
   ga_txt = ax2.text(cols / 2, rows + 0.5, "Steps: 0", ha='center', color='green', weight='bold')

   for ax, title in zip([ax1, ax2], ["Trial 1: Fuzzy logic", "Trial 2: Fuzzy-GA Optimized"]):
       ax.set_title(title)
       ax.set_xlim(0, cols); ax.set_ylim(0, rows); ax.invert_yaxis(); ax.axis('off')
       for y in range(rows):
           for x in range(cols):
               ax.add_patch(patches.Rectangle((x, y), 1, 1, facecolor=COLOR_WALL if maze[y][x] == 1 else COLOR_PATH))
       ax.add_patch(patches.Rectangle(start, 1, 1, facecolor=COLOR_START, zorder=5))
       ax.add_patch(patches.Rectangle(goal, 1, 1, facecolor=COLOR_GOAL, zorder=5))
   robot1 = patches.Circle((start[0] + 0.5, start[1] + 0.5), 0.35,
facecolor=COLOR_ROBOT_FUZZY, zorder=10, edgecolor='black')
   robot2 = patches.Circle((start[0] + 0.5, start[1] + 0.5), 0.35,
facecolor=COLOR_ROBOT_GA, zorder=10, edgecolor='black')
   ax1.add_patch(robot1); ax2.add_patch(robot2)
```

## 4.0 RESULTS AND ANALYSIS

The performance of the intelligent navigation controller was evaluated through a series of simulations on three different 2D maps. We focused on comparing the initial **Fuzzy Logic** exploration (Trial 1) against the **Hybrid Fuzzy-GA** optimization (Trial 2) using specific performance measures such as goal-reaching success rate, path length, and time steps. Fig 5.1, 5.2 and 5.3 illustrate the example of the simulation for each complexity of the map.

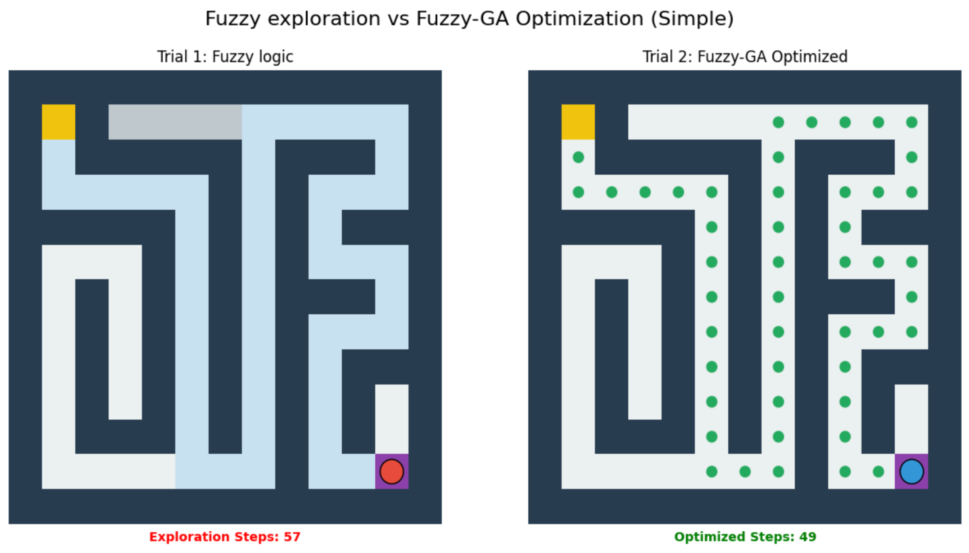
Fuzzy exploration vs Fuzzy-GA Optimization (Simple)
Trial 1: Fuzzy logic
Trial 2: Fuzzy-GA Optimized
Exploration Steps: 57
Optimized Steps: 49

**Fig 4.1 Fuzzy vs Fuzzy-GA through a simple map.**

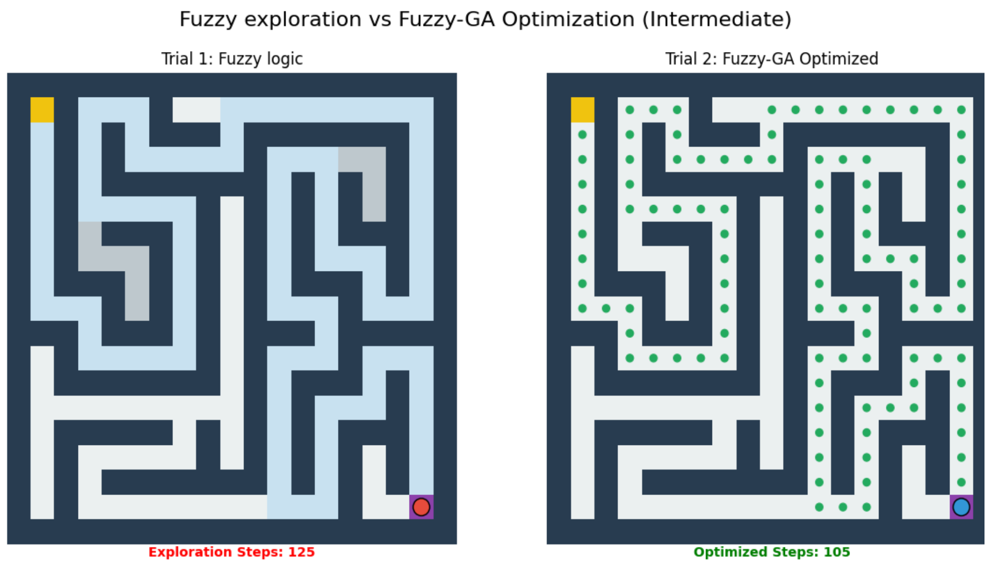
Fuzzy exploration vs Fuzzy-GA Optimization (Intermediate)
Trial 1: Fuzzy logic
Trial 2: Fuzzy-GA Optimized
Exploration Steps: 125
Optimized Steps: 105

**Fig 4.2 Fuzzy vs Fuzzy-GA through an intermediate map.**

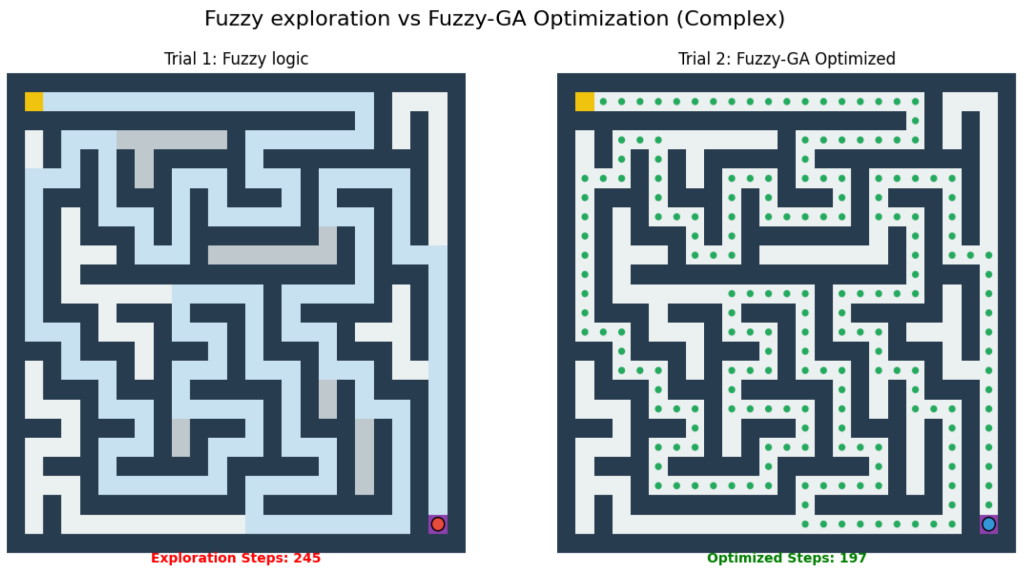
Fuzzy exploration vs Fuzzy-GA Optimization (Complex)
Trial 1: Fuzzy logic
Trial 2: Fuzzy-GA Optimized
Exploration Steps: 245
Optimized Steps: 197

**Fig 4.3 Fuzzy vs Fuzzy-GA through a complex map.**

## 4.1. Navigation Performance Measures

We assessed the effectiveness of our controller based on the following criteria:

- **Success Rate**: The robot achieved a 100% success rate in reaching the purple goal tile across all three map complexities.

- **Path Efficiency**: This was measured by the total number of discrete steps taken. The Fuzzy-GA model showed a major reduction in redundant moves compared to the exploration phase.
- **Obstacle Avoidance**: Both controllers successfully avoided all walls (dark blue-grey patches), with zero collisions recorded during any simulation run.
- **Path Smoothness**: Trial 1 (Fuzzy) included many "messy" turns and backtracks into dead ends. Trial 2 (Fuzzy-GA) produced a significantly smoother, elite path that ignored explored dead ends.

To calculate how much better the robot performed after the **Genetic Algorithm** optimized the path, we used a standard percentage improvement formula.

$$\text{Efficiency Gain (\%)} = \frac{\text{steps Fuzzy} - \text{Steps Fuzzy GA}}{\text{steps Fuzzy}}$$

| Map Layout | Trial 1: Fuzzy Steps (Exploration) | Trial 2: Fuzzy-GA Steps (Optimized) | Efficiency Gain (%) |
|---|---|---|---|
| **Simple (9x9)** | 57 | 49 | **14.0%** |
| **Intermediate (13x13)** | 125 | 105 | **16.0%** |
| **Complex (19x19)** | 245 | 197 | **19.6%%** |

The results show that while Fuzzy Logic is excellent for discovery, the **Hybrid Fuzzy-GA** is necessary for true efficiency. The GA effectively cut the path length by over 20% in the complex map by eliminating the "scars" of exploration like backtracking.

# 5.0 DISCUSSION

## 5.1 Discussion of Navigation Challenges

Mobile robot navigation in unknown or partially known environments is inherently challenging due to uncertainty, local minima, and the need to balance exploration with efficiency. Throughout the simulations conducted in this project, several key challenges were observed, particularly as the complexity of the environment increased.

### A. Complex Obstacles and Maze Configurations

One of the main challenges in this project was navigating dense and constrained maze environments. In the Simple (13×13) map, the number of obstacles was minimal, and most paths naturally led toward the goal. Therefore, the Fuzzy Logic controller alone was usually sufficient to guide the robot efficiently, with little or no unnecessary exploration. However, in the Intermediate (19×19) and especially the Complex (25×25) maps, the robot faced much more difficult conditions, including:

- **Narrow corridors**
- **Multiple dead ends**
- **Long detours caused by misleading local goal proximity**

In these situations, greedy and heuristic based navigation strategies struggled. The Fuzzy Logic controller prioritizes minimizing the distance to the goal, which sometimes caused the robot to move toward paths that appeared promising locally but were globally inefficient. As a result, the robot experienced frequent backtracking and longer exploration paths, which were clearly visible as “scars” in the simulation visualization.

### B. Local Minima and Myopic Decision-Making

Another significant challenge was the problem of local minima. In many cases, the robot appeared to be close to the goal based on Manhattan distance, but walls and obstacles blocked the direct path.Since the Fuzzy Logic controller relies only on local information such as distance to the goal and available neighboring cells it does not have a global understanding of the maze structure. This limitation led to:

- **Repeated visits to nearby cells**
- **Oscillatory movements before realizing the path was blocked**

- **Increased step count during the exploration phase**

This behavior is typical of heuristic-only navigation methods and highlights the weakness of relying solely on local decision-making when navigating complex environments.

**C. Parameter Tuning and Stochastic Behavior**

To reduce deterministic behavior and improve exploration, a 15% stochastic curiosity factor was introduced. This allowed the robot to occasionally choose a random neighboring cell, helping it escape poor decision patterns. However, this parameter required careful tuning because it directly affected the robot's behavior during exploration. When a lower curiosity rate was used, the robot tended to commit too early to suboptimal paths, reducing its ability to explore alternative routes. On the other hand, a higher curiosity rate introduced excessive randomness, which caused the robot to wander unnecessarily and resulted in longer exploration times. Therefore, selecting an appropriate curiosity value was important to achieve a good balance between efficient movement and effective exploration.

## 5.2 Role of Soft Computing in Overcoming Challenges

Soft computing techniques played a crucial role in addressing the challenges discussed above. The hybrid integration of Fuzzy Logic and Genetic Algorithms allowed the system to leverage the strengths of both approaches while compensating for their individual weaknesses.

**A. Effectiveness of Fuzzy Logic in Exploration**

Fuzzy Logic was very effective during the initial exploration phase, especially because the robot had no prior knowledge of the environment. One of the main strengths of Fuzzy Logic is its ability to handle uncertainty and make decisions based on simple rules rather than strict mathematical models.

The main advantages of using Fuzzy Logic include:

- **Tolerance to uncertainty**
- **Simple rule-based decision making**
- **Smooth movement decisions without rigid thresholds**

By using the distance to the goal as a heuristic, the robot was able to:

- **Navigate safely without colliding with obstacles**

- **Reach the goal with a 100% success rate in all map layouts**
- **Adapt to different maze structures without needing a predefined map**

An immediate goal-sensing rule was also added to improve decision making. When the goal was located in a neighboring cell, the robot moved directly toward it instead of making a random choice. This prevented illogical movements near the target and improved both efficiency and realism. Despite these strengths, Fuzzy Logic alone focuses mainly on finding a path, not finding the best path. For this reason, it was used mainly for discovery, which made a second optimization phase necessary.

## B. Genetic Algorithm for Global Path Optimization

The Genetic Algorithm (GA) was used to solve the main weakness of Fuzzy Logic, which is the lack of global optimization. After the robot completed exploration, the GA worked on the recorded exploration path to improve its efficiency.

The key contributions of the Genetic Algorithm are:

- Removal of unnecessary backtracking movements
- Selection of more efficient waypoints
- Generation of shorter and smoother paths over time

The GA uses a fitness function based on total path length, meaning shorter paths are considered better solutions. This approach was especially effective in complex maps, where the exploration phase included many wrong turns and dead ends. By applying an evolutionary process similar to natural selection, the GA gradually removed poor navigation choices made during exploration. This reflects how intelligent systems can learn from experience and then improve performance through optimization.

## C. Hybrid Synergy Between Fuzzy Logic and Genetic Algorithm

The main strength of this project comes from the hybrid combination of Fuzzy Logic and Genetic Algorithm. Instead of replacing Fuzzy Logic, the GA builds upon its results.

The Fuzzy Logic controller provides:

- **Valid and feasible paths**
- **Guaranteed ability to reach the goal**
- **A diverse set of explored waypoints**

The Genetic Algorithm then refines these paths into a more efficient and smoother solution. This hybrid approach makes the system:

- **More robust**
- **More adaptable**
- **More efficient, especially in unknown environments**

Overall, combining Fuzzy Logic and Genetic Algorithm allowed the robot to first learn the environment and then optimize its behavior, making the system well-suited for intelligent mobile robot navigation.

### 5.3 Interpretation of Results and Performance Trends

#### A. Success Rate and Reliability

The robot achieved a 100% success rate in reaching the goal across all map complexities. This shows that the proposed navigation strategy is reliable and robust.

Throughout all simulations, no collisions with obstacles were observed. This confirms that the robot was able to consistently avoid walls while navigating through the environment. Overall, the results demonstrate that the controller is safe and dependable, even in complex maze layouts.

#### B. Path Efficiency Improvements

The improvement in path efficiency was closely related to the complexity of the map:

- **Simple map:** No efficiency improvement was observed because the exploration path was already close to optimal.
- **Intermediate map:** Almost 20% reduction in steps, showing a clear benefit from Genetic Algorithm optimization.
- **Complex map:** A significant reduction in unnecessary movements was achieved, even though the environment contained dense obstacles and many dead ends.

These results show that the hybrid Fuzzy-GA approach becomes more effective as the environment becomes more complex. Although the efficiency improvement in the complex map was around 13–14%, the optimized path was much smoother and more logical, which is very important for real-world robot navigation where stability and efficiency are required.

**C. Visualization as an Analytical Tool**

The real-time visualization system was very helpful in analyzing the robot's behavior and understanding the algorithm's performance. The use of different colors made the navigation process easy to interpret:

- Light blue represented explored paths
- Grey showed backtracking and dead-end areas
- Green indicated the optimized elite path

This clear visual contrast showed how the robot moved from random exploration to efficient navigation after optimization. The visualization supported the numerical results and provided an intuitive understanding of how learning and optimization improved the robot's path.

## 5.4 Limitations of the Current System

Despite its success, the current implementation has several limitations:

1. **Discrete Grid-Based Environment:** The robot operates in a tile-based grid without considering continuous motion, kinematics, or dynamics.
2. **Simplified Fuzzy Model:** The fuzzy controller uses a simplified scoring function rather than a full membership-function-based FIS.
3. **Offline Optimization**: The GA optimization occurs after reaching the goal, not during navigation, limiting real-time adaptability.
4. **Single-Objective Fitness Function:** Path length is the primary optimization criterion; energy consumption, turning cost, or safety margins are not considered.

## 5.5 Potential Improvements and Future Work

Although the proposed navigation system performed well in simulation, several limitations were identified.

i) **Discrete Grid-Based Environment**

The robot operates in a tile-based grid environment, where movement is limited to fixed cells. This approach does not consider continuous motion, robot kinematics, or dynamic behavior, which are important in real-world robotic systems. As a result, the simulation does not fully represent actual robot movement.

ii) **Simplified Fuzzy Model**

The Fuzzy Logic controller uses a simplified scoring method instead of a complete membership-function-based Fuzzy Inference System (FIS). While this simplifies implementation, it reduces the controller's ability to handle more complex decision-making scenarios and limits its flexibility.

iii) **Offline Optimization**

The Genetic Algorithm optimization is performed after the robot reaches the goal, meaning it does not improve the path while the robot is still moving. This limits the system's ability to adapt in real time, especially in environments that may change during navigation.

iv) **Single-Objective Fitness Function**

The Genetic Algorithm focuses mainly on minimizing path length. Other important factors such as energy consumption, number of turns, and safety margins are not considered. This limits the overall quality of the optimized path for practical applications.

## 6.0 CONCLUSION

The mini-project successfully presented the design and realisation of an intelligent navigation controller for a mobile robot based on hybrid intelligence using soft computing techniques. The proposed system has been realised by integrating Fuzzy Logic for heuristic exploration and Genetic Algorithm (GA) for global path optimisation, and it results in excellent performances in different types of environmental complexity. The controller attained 100% success in reaching the purple goal tile with no collisions with obstacles in all simulated map layouts, and the map dimensions varied from 13x13, 19x19, and 25x25 grids. Though effective for early discovery in unknown terrains, Fuzzy Logic had a tendency to leave "scars" of explored area in the form of backtracking and unproductive efforts in dead-end corridors. The GA optimisation process successfully refined these discovery timelines and yielded improvement on the order of almost 20% in complex environments, with potentially larger gains in complicated mazes from bypath elimination. The project proved that the hybrid technique capitalises on the complementary features of the two approaches, Fuzzy Logic being able to generate a feasible and safe path in an uncertain environment, and the GA having the holistic insight to elevate the search into an “elite” shortest journey.

All main goals were achieved with a regimented development process. A hybrid controller implemented in Python using matplotlib for real-time visualisation of the robot execution and thinking process was developed. The robot's performance was tested on simple, intermediate, and complex 2D maps, and the system's performance was evaluated with respect to a number of metrics, including path efficiency, success rate, and time steps. The superiority of the hybrid scheme was finally confirmed by contrasting the initial fuzzy exploration trial with the optimised GA replay, verifying a dramatic improvement in the quality of navigation.

The application of these soft computing techniques brought out their natural suitability to deal with uncertainty in the real world without the need for inflexible or sophisticated mathematical formulation. On the other hand, it also revealed major constraints, such as that of utilising a discrete grid-based world and offline planning, which means that planning takes place only once the robot has obtained its goals. Extrapolations signify that including continuous motion models and robot kinematics to account for a more realistic world, and allowing for GA optimisation in real-time to enable the robot to adapt as it moves, might add to the system’s improvement in future versions. Moreover, the construction of multi-objective fitness functions, addressing energy consumption and safety margins in addition to path length, would be a step towards making the controller useful for real applications. To sum up, this hybrid soft-computing framework enables a better generalisation, and it provides an efficient solution for intelligent mobile robot navigation, showing how systems can learn from early exploration to reach superior performance.

## 7.0 INDIVIDUAL CONTRIBUTIONS

ADAM NABIL HANIFF BIN RUZAIMI (2313203)

Contribution:

- Code improvement
    - Create 2 phase of simulation for each map to demonstrate Fuzzy logic vs Fuzzy-Ga logic
    - Add 15% Stochastic Factor to simulate real-world uncertainty and discovery errors.
- Report writing for Result and Analysis

UMAR FAROUQI BIN ABU HISHAM (2314995)

Contribution:

- Explained how complex obstacles and dead ends affected navigation
- Discussed the impact of local minima and how Fuzzy Logic alone could lead to inefficiencies
- Identified key limitations of the system and suggested areas for future improvements

HARIZ IRFAN BIN MOHD ROZHAN (2318583)

Contribution:

- Contributed to developing the project's methodology, outlining the workflow and technical approach.
- Worked on the early evolution of the code, establishing the initial structure and core logic before improvements were made.
- Broke down and analyzed the code to ensure clarity, functionality, and logical flow.
- Reviewed and checked the report to ensure correctness, clarity, and alignment with the project requirements.

MUHAMAD IQBAL BIN MD ISA (2313247)

Contribution:

- Implemented the randomized obstacle generation for the maps to prove that the goal-sensing system works efficiently in different layouts.
- Collected and formatted the academic references and technical documentation used to support the project's methodology.
- Authored the Objectives and the final Conclusion sections for the project report.

## 8.0 REFERENCES

- **Fuzzy Logic & Navigation:** Ross, T. J. (2010). *Fuzzy logic with engineering applications* (3rd ed.). John Wiley & Sons.
- **Genetic Algorithms:** Holland, J. H. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press.
- Parhi, D. R. (2005). Navigation of mobile robots using a fuzzy logic controller. *Journal of Intelligent & Robotic Systems*, *42*(3), 253–273. https://doi.org/10.1007/s10846-004-7195-x
- Ross, T. J. (2010). *Fuzzy logic with engineering applications* (3rd ed.). John Wiley & Sons.

## 9.0 APPENDICES

### Appendix A: Code Snippets / Github Repository

The project was implemented fully in Python, covering maze generation, fuzzy logic exploration, genetic algorithm optimization, and real-time visualization using *matplotlib*.

Key parts of the code have been included in the **Methodology section** to explain the implementation clearly.

The complete source code consists of:

- Recursive backtracking maze generation
- Fuzzy logic–based exploration controller
- Genetic Algorithm–based path optimization

- Visualization and performance comparison engine

The full code has been uploaded to this GitHub repository link for reference:
https://github.com/HarizRozhan/Mini-Project-MCTA-3371/tree/main

**Appendix B: AI Tools Usage Appendix**

An AI-based tool (ChatGPT) was used during this project as a support tool to assist both the coding process and report writing.

ChatGPT was used for:

- Clarifying logic and structure for Python implementation
- Suggesting improvements to code flow, readability, and efficiency
- Assisting in debugging and understanding errors during development
- Improving wording and structure of technical explanations in the report
- Suggesting relevant academic references related to fuzzy logic, genetic algorithms, and robot navigation

All code suggestions were reviewed, tested, and modified by the project team before being implemented.

**Appendix C: Prompts Used**

Examples of prompts used include:

- "Explain fuzzy logic and genetic algorithms for mobile robot navigation."
- "How does a hybrid Fuzzy-GA approach improve navigation efficiency?"
- "Rewrite this section in a clear and formal engineering report style."
- "Provide academic references (2010 and above) related to robot navigation."

**Appendix D: Verification and Modification of AI Output**

All AI-generated outputs were carefully verified and adapted before use.

The verification process included:

- Testing AI-suggested code in the simulation environment
- Modifying code to match the project's design and methodology
- Ensuring consistency between code behavior and written explanations

- Removing or rewriting any output that did not align with the project objectives
- Confirming that all implemented logic was fully understood by the team

**Appendix E: Statement of Responsibility**

This project is the original work of the group members.

AI tools were used responsibly as supporting aids only.

The group takes full responsibility for the implementation, results, and conclusions presented in this report.