# TABLE OF CONTENTS

# EXECUTIVE SUMMARY

This project implements a comprehensive robot navigation system in the Gazebo simulation environment, utilizing three distinct artificial intelligence approaches: Fuzzy Logic, Behavior Trees, and Reinforcement Learning (Q-Learning). The robot is tasked with autonomous navigation from a starting position (-4, -4) to a goal location (-3.5, 3.5) while avoiding multiple static obstacles including walls, boxes, cylinders, and narrow passages within a bounded 10m × 10m arena. Each AI technique was implemented using ROS (Robot Operating System) Noetic, with the robot equipped with a 360° LIDAR sensor providing distance measurements up to 10 meters with 360 samples per scan, odometry system for position tracking, and differential drive control for movement.

The Fuzzy Logic controller uses fifteen inference rules with two input variables (front distance and heading error) and two output variables (speed and turn rate) to produce smooth velocity commands, enhanced with escape mechanisms for stuck detection and recovery. The Behavior Tree approach uses a hierarchical Selector-Sequence structure with goal checking, obstacle detection with dynamic side-distance evaluation, and fixed-duration avoidance maneuvers combining 90-degree turns and forward motion. The Q-Learning agent discretizes the continuous state space into 108 states derived from front distance, angle to goal, distance to goal, and side obstacle configurations, with 5 discrete actions, learning through 30 training episodes with epsilon-greedy exploration ($\varepsilon$: 0.9 → 0.05) and temporal difference updates with $\alpha$=0.2 and $\gamma$=0.95.

The project successfully demonstrates how different AI paradigms handle decision-making, obstacle avoidance, and goal-directed behavior in realistic robotic scenarios, with comprehensive implementation in Python using libraries including scikit-fuzzy for fuzzy inference, py_trees for behavior tree execution, and numpy for numerical computations, all deployed within a Docker containerized environment using the osrf/ros:noetic-desktop-full image for reproducibility and cross-platform compatibility.

# MOTIVATION

The motivation for this project stems from the fundamental challenge of autonomous robot navigation, where robots must intelligently navigate from a start point to a goal while avoiding obstacles in real-time. Traditional hardcoded approaches fail in complex, uncertain environments, requiring AI-driven solutions that can perceive sensor data, make decisions, and adapt to different situations.

Understanding how different AI techniques approach this problem is crucial for advancing robotics applications in warehouses, healthcare, autonomous vehicles, and smart homes where robots must operate safely alongside humans. By comparing three distinct AI paradigms—Fuzzy

Logic for approximate reasoning under uncertainty, Behavior Trees for hierarchical reactive control, and Reinforcement Learning for experience-based adaptation—in a controlled simulation environment, we gain practical insights into the strengths and tradeoffs of each approach. This comparative analysis helps bridge the gap between theoretical AI concepts and real-world robotic systems that can perform useful tasks autonomously.

## TECHNICAL REQUIREMENTS

The project technical infrastructure consists of several integrated software components:

- **Operating System & ROS**: Ubuntu 20.04 LTS with ROS Noetic Ninjemys distribution providing middleware for robot control and sensor communication
- **Simulation Platform**: Gazebo 11 simulator for physics engine and 3D visualization with realistic dynamics
- **Development Environment**: Docker Desktop for containerization using osrf/ros:noetic-desktop-full image and MobaXterm for X11 forwarding from container to Windows host
- **Programming Language**: Python 3.8 with libraries including rospy (ROS integration), numpy (numerical computations), scikit-fuzzy (fuzzy logic implementation), and py_trees (behavior tree execution), pickle (Q-table persistence), and math (mathematical operations)
- **ROS Messages**: geometry_msgs for Twist velocity commands, sensor_msgs for LaserScan data, nav_msgs for Odometry
- **Robot Model** (model.sdf): Differential drive base (0.5m × 0.4m × 0.2m body, 0.2m diameter wheels), 360° LIDAR sensor (360 samples, -3.14 to 3.14 rad range, 0.1-10m distance, 10Hz update rate on /robot/laser/scan topic), and bumper contact sensor
- **Simulation World** (simple_world.world): 10m × 10m arena with boundary walls (0.2m thick), multiple static obstacles including red box (1×1×1m at position 2,1), yellow cylinder (0.5m radius at -1,-2), narrow passage walls (0.3×2×0.6m), purple obstacle (0.2×0.2×1m at -3.5,0.5), orange checkpoint sphere (0.3m radius at 0,-1), and green target zone cylinder (0.5m radius, 0.2m height at goal -3.5, 3.5), using ODE physics engine with 0.001s time step, real-time factor 1.0, and 1000Hz update rate

# MAIN CONSTRUCTS

The project architecture comprises three independent AI navigation implementations:

**Fuzzy Logic Controller (fuzzy_brain.py, controller.py):**

- Input data: Four sensor readings - front/left/right distances from LIDAR and heading error to goal
- Fuzzy inference inputs: Two variables used by fuzzy system - front distance (0-5m with 'close', 'medium', 'far' memberships), heading error (-180° to +180° with 'hard_left', 'left', 'straight', 'right', 'hard_right' memberships)
- Two output variables: speed (0-0.5 m/s with 'slow', 'medium', 'fast' memberships), turn rate (-1.0 to +1.0 rad/s with 'hard_left', 'left', 'straight', 'right', 'hard_right' memberships)
- Fifteen fuzzy rules covering all front distance and heading error combinations
- Escape mechanism using left/right distance comparison to determine turn direction, with three-phase sequence (backup, turn toward clearer side, forward motion) triggered by stuck detection
- Scikit-fuzzy ControlSystem computes outputs via centroid defuzzification

**Behavior Tree (behavior_tree_complete.py):**

- Root Selector node with three priority-ordered children evaluated at 10Hz
- IsGoalReached: Condition checking distance to goal < 0.8m, returns SUCCESS if reached
- ObstacleAvoidance Sequence: IsObstacleNear checks front LIDAR < 0.8m, AvoidObstacle executes two-phase maneuver (4-second turn at 0.8 rad/s toward side with more clearance, 5-second forward motion at 0.35 m/s)
- MoveTowardsGoal: Action with conditional control (0.5 rad/s turn if angle error >0.5 rad, otherwise 0.35 m/s forward)
- Each node returns SUCCESS, FAILURE, or RUNNING status

**Q-Learning Agent (rl_navigation_final.py):**

- State space: 108 discrete states from 4 features - front distance (4 bins), angle to goal (3 bins), distance to goal (3 bins), side obstacles (3 bins combining left/right clearance)
- State calculation: state_id = (distance_bin × 27) + (angle_bin × 9) + (goal_dist_bin × 3) + sides_bin

- Action space: 5 discrete actions - forward fast (0.5, 0), forward-left (0.35, 0.7), forward-right (0.35, -0.7), slow-left (0.2, 0.8), slow-right (0.2, -0.8)
- Q-table: 108×5 numpy array updated with Q(s,a) ← Q(s,a) + 0.2[r + 0.95 × max Q(s',a') - Q(s,a)]
- Reward function: +25.0×distance_progress, +500.0 goal reached (<0.75m), -100.0 collision (<0.25m), -2.0 near obstacle (<0.5m), -0.3 time penalty
- Epsilon-greedy exploration: ε decays 0.9 → 0.05 over 30 episodes (decay factor 0.95)
- Escape mechanism with stuck detection based on position history (20-step window checking <0.3m movement range)

# IMPLEMENTATION

**Docker Environment Setup:**

- docker-compose.yml mounts three volumes (./models, ./worlds, ./scripts) for file sharing
- Installs dependencies: gazebo11, ROS packages (gazebo-ros-pkgs, geometry-msgs, sensor-msgs, nav-msgs), Python libraries (numpy, scipy, scikit-fuzzy, pickle)

**Robot Model (models/simple_robot/model.sdf):**

- base_link: Box (0.5×0.4×0.2m), mass 15kg
- Wheels: Two cylinders (radius 0.1m, mass 2kg each) with high friction (mu=100, mu2=50)
- Caster: Sphere (radius 0.05m) with zero friction for smooth movement
- LIDAR: Ray sensor publishes 360 samples to /robot/laser/scan at 10Hz
- Differential drive plugin controls movement via /cmd_vel and reports position via /odom

**Simulation World (worlds/simple_world.world):**

- base_link: Box (0.5×0.4×0.2m), mass 15kg
- Wheels: Two cylinders (radius 0.1m, mass 2kg each) with high friction (mu=100, mu2=50)
- Caster: Sphere (radius 0.05m) with zero friction for smooth movement
- LIDAR: Ray sensor publishes 360 samples to /robot/laser/scan at 10Hz
- Differential drive plugin controls movement via /cmd_vel and reports position via /odom

**Fuzzy Logic Implementation (controller.py):**

- Subscribes to /robot/laser/scan and /odom at 10Hz
- LIDAR zones: front (ranges 320-40), left (ranges 40-140), right (ranges 220-320), takes minimum per zone
- Heading error computed as atan2(GOAL_Y-y, GOAL_X-x) - yaw, normalized and converted to degrees
- Escape sequence triggered when stuck detected (front <0.7m, position unchanged, or very close <0.5m): backup 3s → turn toward clearer side 2.5s → forward 5s
- Stops when goal distance < 0.8m

**Behavior Tree Implementation (behavior_tree_complete.py):**

- IsGoalReached: Returns SUCCESS if distance ≤ 0.8m
- IsObstacleNear: Checks front LIDAR (ranges ±40), returns SUCCESS if < 0.8m
- AvoidObstacle: Compares left (ranges 70-110) vs right (ranges 250-290) distances, turns toward clearer side for 4s at 0.8 rad/s, then moves forward 5s at 0.35 m/s
- MoveTowardsGoal: Turns at 0.5 rad/s if angle error > 0.5 rad, otherwise moves forward at 0.35 m/s
- Tree: Selector[IsGoalReached, Sequence[IsObstacleNear, AvoidObstacle], MoveTowardsGoal]
- Executes at 10Hz

**Q-Learning Implementation (rl_navigation_fixed.py):**

- QLearningAgent: Q-table 108×5 numpy array, learning_rate=0.2, discount=0.95, epsilon 0.9→0.05 (decay 0.95)
- discretize_state(): Bins front_distance (4), angle_to_goal (3), distance_to_goal (3), sides (3) → state_id = (dist_bin×27) + (angle_bin×9) + (goal_dist_bin×3) + sides_bin
- choose_action(): ε-greedy selection from 5 actions (forward fast, forward-left, forward-right, slow-left, slow-right)
- Escape mechanism: Triggered when front < 0.6m or position stuck (20-step history, <0.3m movement range), executes 20-step backup → 25-step turn → 25-step forward
- update(): $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \times \max(Q(s',a')) - Q(s,a)]$
- Reward: +25.0×progress, +500.0 goal (<0.75m), -100.0 collision (<0.25m), -2.0 near obstacle (<0.5m), -0.3 time penalty

- RobotNavigationEnv: Downsamples LIDAR to 24 points, extracts x,y,θ from /odom
- Training: 30 episodes, 400 steps max, 0.1s action duration, saves Q-table every 5 episodes to q_table.pkl
-

# IMITATIONS

**Q-Learning Episode Reset Issue:**

- The robot was unable to reset to starting position (-4, -4) between training episodes due to technical limitations with Gazebo's reset service in the Docker environment
- Episodes ran continuously without proper resets, causing the robot to start each new episode from wherever the previous episode ended
- This prevented the agent from experiencing consistent initial conditions and learning optimal behavior from the designated start position
- The lack of goal-reaching experiences (robot starting far from goal each episode) resulted in predominantly negative rewards, hindering proper Q-value convergence and policy learning
- In proper reinforcement learning environments, agents require repeated goal-reaching experiences to learn effective policies, which was compromised by this limitation

**Fuzzy Logic Escape Mechanism Dependency:**

- The fuzzy controller relies on hardcoded escape sequences rather than adaptive learning, limiting its ability to handle novel obstacle configurations

**Behavior Tree Fixed Maneuver Durations:**

- Obstacle avoidance uses fixed time durations (4-second turn, 5-second forward) rather than sensor-feedback-based termination, which may be suboptimal for different obstacle layouts

# CONCLUSION

This project successfully concluded with the implementation of three AI-based navigation systems that enable autonomous robot movement from start to goal while avoiding obstacles.

Each approach demonstrated different decision-making strategies:

- **Fuzzy Logic** provided smooth continuous control through approximate reasoning
- **Behavior Trees** offered structured and reliable hierarchical navigation
- **Q-Learning** showed adaptive learning capability from experience over multiple episodes

Through this project, I gained practical understanding of how different AI techniques can be applied to robotic navigation problems and their respective advantages in handling obstacle avoidance and goal-directed behavior. The successful deployment in Gazebo with ROS integration demonstrates that these AI approaches are viable solutions for autonomous navigation tasks, providing a foundation for future work in more complex environments and eventual deployment on physical robot hardware